

Discriminant analysis using sparse graphical models

Dylon Botha

Thesis presented in partial fulfilment of the requirements for
the degree of Master of Commerce (Statistics) in the
Department of Statistics and Actuarial Science at
the University of Stellenbosch



Supervisor: Dr Francois Kamper

Co-supervisor: Dr Surette Bierman

March 2020

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

DECLARATION

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.

Date: March 2020

Abstract

The objective of this thesis is the proposal of a new classification method. This classification method is an extension of classical quadratic discriminant analysis (QDA), where the focus is placed on relaxing the assumption of normality, and on overcoming the adverse effect of the large number of parameters that needs to be estimated when applying QDA.

To relax the assumption of normality, we consider assigning to each class density a different nonparanormal distribution. Based on these nonparanormal distributions, new discriminant functions can be derived. When one considers the use of a nonparanormal distribution, the underlying assumption is that the associated random vector, can through the use of an appropriate transformation, be made to follow a Gaussian distribution. Such a transformation is based on the marginals of the distribution, which is to be estimated in a nonparametric way.

The large number of parameters in QDA is a result of the estimation of class precision matrices. To overcome this problem, penalised maximum likelihood estimation is performed by placing an L1 penalty on the size of the elements in the class precision matrices. This leads to sparse precision matrix estimates, and therefore also to a reduction in the number of estimated parameters.

Combining the above approaches to overcome the problems induced by nonnormality and a large number of parameters to estimate, leads to the following novel classification method. To each class density, a separate transformation is applied. Thereafter L1 penalised maximum likelihood estimation is performed in the transformed space. The resulting parameter estimates are then plugged into the nonparanormal discriminant functions, thereby facilitating classification. An empirical evaluation of the novel proposal shows it to be competitive with a wide array of existing classifiers. We also establish a connection to probabilistic graphical models, which could aid in the interpretation of this new technique.

Key words:

Gaussian graphical models; graphical lasso; inverse covariance matrix; nonparanormal model; quadratic discriminant analysis; regularisation.

Opsomming

Die doelwit van hierdie tesis is die voorstel van 'n nuwe klassifikasie-metode. Hierdie klassifikasie-metode is 'n uitbreiding van klassieke kwadratiese diskriminant-analise (KDA), waarin die normaliteits-aanname van KDA verslap word, en waarin die negatiewe effek van die groot aantal parameters wat beraam moet word in KDA toepassings, aangespreek word.

Ter verslapping van die normaliteits-aanname beskou ons die toekenning van verskillende nie-paranormale verdelings aan elke klas. Op grond van hierdie nie-paranormale digtheidsfunksies kan nuwe diskriminantfunksies afgelei word. Wanneer 'n nie-paranormale verdeling veronderstel word, is die onderliggende aanname dat die geassosieerde vektor van stogastiese veranderlikes na 'n normaalverdeling transformeer kan word. Hierdie transformasie is gebaseer op die marginale verdelings, wat weer op 'n nie-parametriese wyse beraam word.

Die groot aantal parameters in KDA is die gevolg van die beraming van presisiematrikse vir elke klas. Om hierdie probleem te oorkom, word gepenaliseerde maksimum aanneemlikheidsberaming toegepas, spesifiek deur L1-penaliserings op die groote van die elemente in die presisiematrikse. Dit lei tot 'n patroon van skaarsheid in die inverse kovariansiematrikse, en derhalwe ook tot 'n vermindering in die aantal beraamde parameters.

Die samevoeging van die bogaande twee benaderings ten einde die probleme veroorsaak deur nie-normaliteit en die groot aantal parameters om te beraam, te oorkom, lei tot die volgende nuwe klassifikasie-metode. Vir elke klasdigtheid word 'n aparte transformasie toegepas. Daarna word L1-gepenaliseerde maksimum aanneemlikheidsberaming in die getransformeerde ruimte toegepas. Die beramings wat sodoende gevind word, word dan by die nie-paranormale diskriminant funksies ingestel ten einde klassifikasie te doen. Empiriese evaluering van die nuwe tegniek wys dat dit goed vergelyk met bestaande klassifikasie-metodes. Ons bevestig ook 'n verwantskap met grafiese modelle, wat moontlik kan bydra tot interpretasie van die nuwe tegniek.

Sleutelwoorde:

grafiese "lasso"; grafiese model onder die normaal aanname; inverse kovariansiematriks; kwadratiese diskriminante-analise; nie-paranormale model; regulering.

Acknowledgements

I would like to express my gratitude to the following people and organisations:

The National Research Foundation for funding my studies.

Dr Kamper for his guidance, insight, support, patience and coffee discussions throughout the construction of this thesis.

Dr Bierman for her contributions to the thesis.

Stephanie and Martin Hendrie for their contributions in language editing and formatting of this document.

Anna-Marie Botha and Samantha O'Leary for their support throughout the thesis.

.

Dedications

I dedicate this thesis to my grandmother, Dorothea Regina Botha who has supported me throughout all my studies. Thank you.

Table of contents

Discriminant analysis using sparse graphical models	i
DECLARATION	ii
Abstract	iii
Opsomming	iv
Acknowledgements	v
Dedications	vi
List of figures	ix
List of tables	x
List of appendices	xi
List of abbreviations and/or acronyms	xii
CHAPTER 1 INTRODUCTION	1
1.1 Introduction	1
1.2 Research Proposal	2
1.3 Key Connections	3
1.3.1 Regularisation and the precision matrix	3
1.3.2 The precision matrix and discriminant analysis	4
1.3.3 Probabilistic graphical models	4
1.4 Thesis Outline	5
CHAPTER 2 LITERATURE REVIEW	6
2.1 Introduction	6
2.2 Gaussian Based Discriminant Analysis	7
2.2.1 Linear discriminant analysis	8
2.2.2 Quadratic discriminant analysis	9
2.2.3 Reduced-Rank LDA	11
2.2.4 Regularised discriminant analysis	12
2.3 Non-Gaussian Based Discriminant Analysis	13
2.3.1 Flexible discriminant analysis	13
2.3.2 Penalised discriminant analysis	14
2.3.3 Mixture discriminant analysis	15
2.4 Sparse Graphical Model Based Discriminant Analysis	16
2.5 Summary	17
CHAPTER 3 BASIC METHODOLOGY	18
3.1 Introduction	18
3.2 The Relationships Between Variables Found in Graphs	19
3.2.1 Marginal correlation graphs	19
3.2.2 Partial correlation graphs	20

3.2.3	Conditional independence graphs	20
3.3	Markov Random Fields for Continuous Random Variables	24
3.3.1	The Gaussian graphical model and its relation to a pairwise Markov graph	24
3.3.2	The Gaussian graphical model and its relation to partial correlation graphs	25
3.4	Estimating the Precision Matrix for the Gaussian Graphical Model	27
3.4.1	The maximum likelihood estimate	27
3.4.2	The L1 penalised estimate	28
3.4.3	The graphical lasso	30
3.4.4	The faster graphical lasso	32
3.4.5	Selecting the tuning parameter for the graphical lasso	38
3.5	Summary	40
CHAPTER 4 DISCRIMINANT ANALYSIS USING SPARSE GRAPHICAL MODELS		41
4.1	Introduction	41
4.2	QDA and The Graphical Lasso	41
4.3	The Non-Normal Case	42
4.3.1	Transforming multivariate observations	42
4.3.2	The nonparanormal model	44
4.3.3	Estimation	47
4.3.4	Kernel density estimation	49
4.3.5	The graphical lasso nonparanormal model (gINPN)	52
4.4	Summary	53
CHAPTER 5 PRACTICAL APPLICATION		55
5.1	Introduction	55
5.2	Methodology	56
5.3	The Vowel Dataset	58
5.4	The Zip Code Dataset	62
5.5	The Spam Dataset	67
5.6	Summary	70
CHAPTER 6 CONCLUSION		71
6.1	Summary	71
6.2	Future Research	72
6.3	Conclusion	73
REFERENCES		74
APPENDIX A: R CODE		77

List of figures

Figure 3.1: Undirected graphical model representation of joint distribution with conditional independence between two variables vs joint distribution with no independence.

Figure 3.2: Illustrating the conditional independencies between subgraphs.

Figure 3.3: Algorithm 1, 2 and 3 applied to the flow-cytometry data.

Figure 4.1: Natural density estimate example bandwidth ω equal to 0.1, 0.5, 1 and 2.

Figure 4.2: Gaussian kernel density estimate with ω set to 0.1, 0.5, 1 and 2.

Figure 4.3: Gaussian kernel density estimate with bandwidth set to 0.527.

Figure 5.1: Handwritten digits example.

List of tables

Table 3.1: Precision matrix obtained by applying Algorithm 1, 2 and 3 to the flow-cytometry data with $\lambda = 52$.

Table 5.1: Vowel dataset results by Hastie et al. (2009).

Table 5.2: Multivariate normality test for the Vowel data.

Table 5.3: Results of application to the Vowel data using the glQDA model.

Table 5.4: Results of application to the Vowel data using the gINPDA model.

Table 5.5: Zip code dataset results by Hastie et al. (2009).

Table 5.6: Results of applications to the MNIST dataset.

Table 5.7: Multivariate normality test for the zip code dataset.

Table 5.8: Results of application to the zip code dataset using the glQDA model.

Table 5.9: Results of application to the zip code data using the gINPDA model.

Table 5.10: Spam dataset results by Hastie et al. (2009).

Table 5.11: Multivariate normality test for the spam dataset.

Table 5.12: Results of application to the spam data using the glQDA model.

Table 5.13: Results of application to the spam data using the gINPDA model.

List of appendices

APPENDIX A: R CODE

- A.1) Algorithm 1: Graphical lasso.
- A.2) Algorithm 2: Faster graphical lasso 1.
- A.3) Algorithm 3: Faster graphical lasso 2 and Algorithm 4: Ordering variables.
- A.4) Code to draw graphs.
- A.5) Natural and Gaussian kernel densities.
- A.6) Modified huge package functions.
- A.7) glQDA function code.
- A.8) gINPDA function code.
- A.9) glQDA application to vowel data with different methods for selecting the tuning parameter.
- A.10) gINPDA application to vowel data with different methods for selecting the tuning parameter.
- A.11) glQDA application to zip code data with different methods for selecting the tuning parameter.
- A.12) gINPDA application to zip code data with different methods for selecting the tuning parameter.
- A.13) glQDA application to spam data with different methods for selecting the tuning parameter and 5-fold cross-validation.
- A.14) gINPDA application to spam data with different methods for selecting the tuning parameter and 5-fold cross-validation.

List of abbreviations and/or acronyms

BIC	Bayesian information criterion.
CDF	Cumulative distribution function.
CPU	Central processing unit.
eBIC	Extended Bayesian information criterion.
EPE	Expected prediction error.
GB	Gigabyte.
GHz	Gigahertz.
gINPDA	Graphical lasso nonparanormal discriminant analysis.
gIQDA	Graphical lasso quadratic discriminant analysis.
<i>huge</i>	High-dimensional undirected graph estimation.
LDA	Linear discriminant analysis.
MLE	Maximum likelihood estimate.
MNIST	Modified National Institute of Standards and Technology.
QDA	Quadratic discriminant analysis.
RAM	Random-access memory.
StARS	Stability approach to regularization selection.

CHAPTER 1

INTRODUCTION

1.1 Introduction

In statistical learning, all methods can generally be divided into two groups, i.e. supervised and unsupervised methods. Supervised methods use a set of features (predictors) to predict the outcome of another predefined set of features (response), whereas unsupervised methods investigate the relationships and structures in one set of features. Furthermore, supervised methods can be separated into two groups, i.e. regression models and classification models. The difference between regression and classification models is the nature of the response features. For classification models the response features are categorical, while for regression models the features are numerical. This thesis considers classification methods. The application of classification models can be found all around us, examples include credit approval, facial recognition, medical diagnosis, recommendation systems and many others.

Training a supervised statistical learning model often has a dual purpose, viz. to obtain a model which is able to generalise well to new unseen observations, and to obtain a model which facilitates inference regarding the relationships among the features. With respect to the generalisation performance or prediction accuracy of a model, note that when the training data set is used to measure the performance of a model, a function can always be found that fits the training data perfectly. Such a model is misleading and undesirable, since it can potentially identify patterns in the training data that are unique to the observed sample and that do not occur in the population, or in other samples. Using such a model to make predictions on another sample could lead to poor results, since the same patterns in the training sample are not necessarily present in other samples. On the contrary, if very little information from the training sample is used to obtain a function, the model will be missing patterns in the training sample that carry over to other samples and the population. The above phenomena are respectively known as overfitting and underfitting the training sample. The number of parameters used in fitting a model measures the complexity of a model. Complex models are often far more flexible (i.e. can fit a wide range of functions) and can easily lead to overfitting.

In regression using a squared error loss, the capability of a model to generalise to unseen cases is influenced by three factors, viz. the irreducible error, the bias and the variance of the model. For other loss functions (for example classification), this influence is not that clear, but similar decompositions can be derived, see for example Pretorius et al. (2016) regarding results for the 0-1 loss. The irreducible error is the error the model makes due to inherit randomness in the response that can not be captured by the features used as predictors in the model. The bias refers to the error the model makes by modelling complex reality using a simplified model. The variance

refers to how much the model will vary when trained on different samples from the same population. For example, consider the two extreme cases where one model is extremely flexible and can fit the training sample perfectly for every sample, while the other is extremely inflexible and selects a fixed constant as a prediction for all samples. Comparing the variability of the two models over different samples, it is intuitively obvious that the model fitting each sample perfectly will vary significantly more than the model that uses a fixed constant over the samples. Thus, the extremely flexible model has a higher variance than the inflexible model. However, the more flexible a model, the more complex the model is, and the closer it is to the complex reality. For this reason, the model that fits the samples closely will have a smaller bias than the model that selects a fixed constant as a prediction for all samples. Therefore, as the flexibility of a model increases its bias will decrease at the cost of a higher variance. At low levels of flexibility, the decrease in bias typically offsets the increase in variance, causing the model to generalise better. At some level of flexibility the optimal bias-variance trade-off is attained, whereafter a further increase in model flexibility causes the variance to increase at a higher rate than the decrease in bias. The result is a detrimental effect on generalisation performance.

A scenario in which it typically pays to fit a less flexible model is in the case where the size of the training set is small relative to the number of features. In high dimensions with few observations, the observations are spread far from one another. Fitting a complex or flexible model in such a space can easily result in overfitting and high variance in the model. To curb overfitting, regularisation can be used. Regularisation entails shrinking the estimated parameters of a model towards a constant. This in turn leads to less variability when training the model on different samples, since the estimated parameters of the model will be shrunk to the same constant. Regularisation has been combined with a wide range of statistical learning methods to improve their performance. Examples include, regularisation in linear discriminant analysis (Guo et al., 2006), and regularisation for generalised linear models (Friedman et al., 2010). In addition, regularisation can also improve the interpretability of a model. This is done by using regularisation to find the features that have the greatest influence on the response.

1.2 Research Proposal

The research proposal which forms the focus of this study is the use of regularisation in classification by means of probabilistic graphical models for continuous random variables. The objective of the study is to find a classification model with high prediction accuracy. To achieve the objective a novel method, i.e. Non-Gaussian based discriminant analysis using sparse graphical models is proposed. The above-mentioned method improves on ordinary quadratic discriminant analysis by one, transforming the input space to deal with non-Gaussian data and two, perform regularisation on the inverse covariance matrix to prevent overfitting.

Considering the numerous and diverse real-world applications of classification models, such a proposal could have significant practical advantages. Other objectives include evaluating the influence of different parameter selection techniques specific to the regularisation problem, writing R functions that can be used for future research and development, and providing different views of familiar statistical learning concepts.

1.3 Key Connections

This research proposal is motivated by the connection between regularisation on the precision matrix, the role of the precision matrix in discriminant analysis, and the use of probabilistic graphical models. These connections are discussed below.

1.3.1 Regularisation and the precision matrix

The precision matrix is defined as the inverse covariance matrix. For Gaussian distributed random variables the precision matrix captures the conditional independence between the random variables as well as the partial correlation between the random variables. The importance of the precision matrix to this study will become evident in Section 1.3.2.

Various statistical learning models are dependent on an estimated precision matrix (also known as the inverse covariance matrix). Hence, the estimated precision matrix plays an important role in the prediction accuracy realised by these models. For a set of continuous random variables, probabilistic graphical models (PGMs) often assume the joint distribution to be a multivariate Gaussian distribution. In high-dimensional settings such an assumption may yield favourable outcomes, due to the curse of dimensionality. When observations are thinly spread (often the case in high dimensions), it is easy to overfit the training set, which results in poor generalisation performance of the model. The graph structure associated with a Gaussian PGM is captured in the inverse covariance matrix, also known as the precision matrix. The reason being, under the Gaussian assumption the conditional independencies depicted by the graph are captured in the inverse covariance matrix. The most common estimator for the precision matrix is the maximum likelihood estimator (MLE). Disadvantages of this estimator include possible high variance and rare occurrence of zero entries in the precision matrix. For the latter this implies that we cannot associate a sparse graphical model with the estimated Gaussian density (a sparse graphical model is a graphical model with some missing edges in its associated graph).

These problems of maximum likelihood estimation can be overcome by means of regularisation. As noted in Section 1.1, regularisation will shrink the estimated parameter to a constant for all samples, thereby reducing the variance. A regularisation technique, the L1 penalised estimator, provides a means of reducing variance and obtaining zero entries in the precision matrix, albeit at the expense of greater estimator bias. Zero entries in the precision matrix indicate conditional

independence and are required to obtain sparse graphs. The *graphical lasso* algorithm can be used to find the L1 penalised estimator. The potential effect of a sparse precision matrix on classification is a classifier with lower variance and potentially higher prediction accuracy.

In the non-Gaussian case, the relationship between the precision matrix and the underlying probabilistic graphical model is not as clear. Applying the Gaussian assumption to non-Gaussian data introduces model bias. The novel method proposed in the thesis makes use of a method where the underlying graphical model is captured by the precision matrix in a transformed space. This reduces the model bias introduced by the Gaussian assumption.

1.3.2 The precision matrix and discriminant analysis

When a statistical learning method such as quadratic discriminant analysis (QDA), which is based on normality assumptions and which estimates the group specific precision matrices, is considered, it provides an opportunity to evaluate the results when the precision matrices in the model are regularised using the L1 penalty. In addition, relaxing the assumption of normality also provides the opportunity to obtain a regularised statistical learning method which extends to non-Gaussian distributions.

1.3.3 Probabilistic graphical models

When working with a set of random variables, the joint probability distribution of these random variables is needed for probabilistic reasoning. Probabilistic graphical models provide a mechanism for exploiting the structure in complex distributions and describe them compactly in a way that allows it to be constructed and utilized effectively. Probabilistic graphical models use graphs to compactly encode the joint distribution of features and can be used to represent dependencies among a set of variables of a distribution. In addition, the graph can be used to partition the distribution into smaller factors. The joint distribution can then be defined as a product of these factors, rendering distributions in high-dimensional spaces considerably more manageable.

Two types of PGM graph structures exist, viz. directed acyclic graphs, and undirected graphs. The directed acyclic graphs are used when the directionality of the interaction between the variables can be ascribed, otherwise undirected graphs are used (Koller and Friedman, 2012). In this thesis focus is placed on undirected graphs.

1.4 Thesis Outline

An overview of the main content of this thesis is as follows. Chapter 2 presents a review of existing literature related to the research proposal of this study. Chapter 3 provides a review of the main sources of information used in the study. It introduces key concepts with respect to probabilistic graphical models and their components. It also presents methods for estimating the precision matrix and explores different ways of selecting the optimal regularisation parameter. Chapter 4 continues a discussion of discriminant analysis using sparse graphical models. The approach for combining the regularised estimator of the precision matrix with QDA is described, followed by an investigation of the methods for relaxing the assumption of normality. Chapter 5 presents the empirical work that was done in order to compare the performance of the novel regularised classifier to that of QDA and multiple other classification techniques on three distinct datasets. In Chapter 6 a summary of key results and conclusions pertaining to this thesis is given. This chapter also includes recommendations for potential further research.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Discriminant analysis is a classification technique derived from the Bayes classifier. This technique fits a density function to each class in the classification problem and uses these estimated density functions to classify an observation to the most probable class. The loss function (the criterion to be minimised) for discriminant analysis is the zero-one loss function. The zero-one loss assigns a loss of one to observations that are misclassified, and a zero loss to observations that are correctly classified.

More formally, for classification models the expected prediction error (EPE) is defined as,

$$EPE = E[L(G, G(\mathbf{X}))], \quad (2.1)$$

where $L()$ is the loss function taking as inputs the true class G , and the estimated class $G(\mathbf{X})$. The expectation is taken with respect to the joint distribution of the inputs, \mathbf{X} , and the true classes, in other words the expectation is taken with respect to $P(G, \mathbf{X})$ (Hastie et al., 2009).

By conditioning on \mathbf{X} , Equation (2.1) can be written as

$$EPE = E_{\mathbf{X}} \sum_{k=1}^K L[\Upsilon_k, G(\mathbf{X})] P(\Upsilon_k | \mathbf{X}), \quad (2.2)$$

where Υ_k refers to the value representing class k in the set of all class values (Υ), $k=1, \dots, K$.

$G(\mathbf{X})$ may be estimated by means of pointwise minimisation of the EPE, that is

$$G^*(\mathbf{x}) = \arg \min_{g \in \Upsilon} \sum_{k=1}^K L[\Upsilon_k, g] P(\Upsilon_k | \mathbf{X} = \mathbf{x}). \quad (2.3)$$

Assuming a zero-one loss function, (2.3) becomes

$$G^*(\mathbf{x}) = \arg \min_{g \in \Upsilon} [1 - P(g | \mathbf{X} = \mathbf{x})], \quad (2.4)$$

which simply states that an observation is classified to the class which is most probable, given the observed input. The classifier in (2.4) is known as the Bayes classifier. Multiple methods try to approximate the Bayes classifier. One example is the k-nearest neighbour classifier, which attempts to directly estimate the Bayes classifier. Discriminant analysis also attempts to estimate the Bayes classifier, but differs from k-nearest neighbours by adopting structural assumptions regarding the form of the class-conditional densities.

More formally, let $f_k(\mathbf{x})$ be the class-conditional density of \mathbf{X} , in class $G = Y_k$, and let π_k be the prior probability that an observation belongs to class k , where $\sum_{k=1}^K \pi_k = 1$. A simple application of Bayes' theorem states that

$$P(G = Y_k | \mathbf{X} = \mathbf{x}) = \frac{f_k(\mathbf{x})\pi_k}{\sum_{l=1}^K f_l(\mathbf{x})\pi_l}. \quad (2.5)$$

The Bayes classifier thus classifies an observation to the class for which (2.5) is the largest. This is equivalent to classifying an observation to the class for which $f_k(\mathbf{x})\pi_k$, or any monotonic transformation of $f_k(\mathbf{x})\pi_k$, is the largest. Discriminant analysis explicitly estimates $f_k(\mathbf{x})$ and π_k for $k = 1, \dots, K$. Multiple methods for estimating the conditional densities and priors exist. These methods can be grouped into Gaussian-based methods and non-Gaussian based methods.

The aim of this chapter is to provide a literature review of techniques that attempt to achieve the same objective as the technique proposed later in this thesis. These techniques are discriminant-based classification techniques that employ regularisation to avoid overfitting. To achieve this goal, Gaussian-based discriminant analysis techniques are introduced in Section 2.2, which include linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), reduced-rank LDA, and regularised discriminant analysis. In Section 2.3, non-Gaussian based methods, such as flexible discriminant analysis (FDA), penalized discriminant analysis and mixture discriminant analysis are discussed. Finally, in Section 2.4, the methods proposed in this thesis are discussed. We refer to these methods as *sparse graphical model based discriminant analysis*. An exhaustive description of all discriminant analysis based techniques in this thesis is an impossible task due to the vast number of variations. Thus, the goal of this chapter is to introduce the most popular methods and those applicable to this thesis.

2.2 Gaussian-Based Discriminant Analysis

Gaussian-based discriminant analysis techniques assume the class conditional density of \mathbf{X} to be a Gaussian density function. As mentioned by Hastie et al. (2009), this might seem like a very restrictive assumption, but in practice it is found that the Gaussian assumption yields good results. The reason is not necessarily related to the data being normally distributed, but rather that such structural assumptions often lead to simpler models which, when compared to more flexible techniques, are ultimately more *stable* (viz., have lower variance).

2.2.1 Linear discriminant analysis

LDA is a well-known discriminant-based classification technique. It assumes that the class conditional densities of the p random variables, \mathbf{X} , are Gaussian density functions. The multivariate Gaussian density function for class k is given by

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{p}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right), \quad (2.6)$$

where Σ_k and $\boldsymbol{\mu}_k$, $k=1, \dots, K$, are the class specific covariance matrices and mean vectors, respectively. In addition, LDA assumes a common covariance matrix for the density functions corresponding to all classes, i.e. $\Sigma_k = \Sigma \forall k$. Substituting (2.6), with a common covariance matrix Σ , into (2.5) and taking the natural logarithm on both sides, yields,

$$\begin{aligned} \log(P(G = Y_k | \mathbf{X} = \mathbf{x})) &= \log(1) - \log\left((2\pi)^{\frac{p}{2}}\right) - \log\left(|\Sigma|^{\frac{1}{2}}\right) \\ &\quad - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) + \log(\pi_k) - \log\left(\sum_{l=1}^K f_l(\mathbf{x})\pi_l\right), \end{aligned} \quad (2.7)$$

$k=1, \dots, K$. For the purpose of classification, we can ignore the constants in (2.7) that do not depend on k . Hence, we can base our classification on the discriminant functions,

$$\delta_k(\mathbf{x}) = \mathbf{x}^T \Sigma^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \log(\pi_k), \quad k=1, \dots, K. \quad (2.8)$$

Note that the discriminant functions in Equation (2.8) are linear in \mathbf{x} . Linear discriminant analysis classifies an observation to the class for which (2.8) is the largest. With reference to (2.8), a strong resemblance with the Mahalanobis distance can be observed. If the class priors π_k , are all equal, then LDA is equivalent to classifying an observation to the class with the smallest Mahalanobis distance to the class centroid. In practice the parameters Σ , $\boldsymbol{\mu}_k$ and π_k , $k=1, \dots, K$ need to be estimated. Using the training data, the most common estimates are,

- $\hat{\pi}_k = \frac{N_k}{N}$ where N_k is the number of observations in class k , and $N = \sum_{k=1}^K N_k$
- $\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{g_i=Y_k} \mathbf{x}_i$
- $\hat{\Sigma} = \frac{1}{N-K} \sum_{k=1}^K \sum_{g_i=Y_k} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^T$.

For Gaussian graphical models the graph structure is captured in the inverse covariance matrix, viz. Σ^{-1} . Thus, by estimating a sparse precision matrix, each conditional class density can be represented by a sparse graphical model.

2.2.2 Quadratic discriminant analysis

QDA can be viewed as a non-linear extension of LDA. Like LDA, QDA assumes that the density $f_k(\mathbf{x})$ for each class is multivariate Gaussian, but unlike LDA, QDA does not assume a common covariance matrix over all classes. In a similar manner to LDA, substituting (2.6) with a class specific covariance matrix into the numerator of (2.5) and taking the natural logarithm, gives the discriminant functions for QDA,

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) + \log(\pi_k), \quad k = 1, \dots, K. \quad (2.9)$$

From (2.9) it can be observed that the discriminant function is quadratic in the input space. This results in decision boundaries that are quadratic, and thus more flexible than LDA.

The flexibility of QDA vs LDA can also be seen in the number of parameters that need to be estimated. For the discriminant function associated with LDA the parameters Σ , $\boldsymbol{\mu}_k$ and π_k need to be estimated for $k = 1, \dots, K$. It appears that $0.5 \times p \times (p+1) + K \times (p+1)$ parameters are needed in order to discriminate. However, in order to classify an observation we only need to consider $\tilde{\delta}_k(\mathbf{x}) = \delta_k(\mathbf{x}) - \delta_K(\mathbf{x})$ for $k = 1, \dots, K-1$. If all these are negative, we assign the observation to class K . Otherwise we assign the observation to the class with the highest value of $\tilde{\delta}_k(\mathbf{x})$. By substituting (2.8) into $\tilde{\delta}_k(\mathbf{x})$ we see that,

$$\tilde{\delta}_k(\mathbf{x}) = \mathbf{x}^T \Sigma^{-1} (\boldsymbol{\mu}_k - \boldsymbol{\mu}_K) - \frac{1}{2} \boldsymbol{\mu}_k^T \Sigma^{-1} \boldsymbol{\mu}_k + \frac{1}{2} \boldsymbol{\mu}_K^T \Sigma^{-1} \boldsymbol{\mu}_K + \log(\pi_k / \pi_K) \quad (2.10)$$

for $k = 1, \dots, K-1$. Hence, in order to classify, we effectively need $p+1$ parameters for each of the $k = 1, \dots, K-1$ class. Thus, the total number of parameters that need to be estimated for LDA is $(K-1) \times (p+1)$. For QDA the number of parameters that need to be estimated is $(K-1) \times (p(p+3)/2 + 1)$. The $K-1$ is required for comparison to the $K-1$ discriminant functions. The $p(p+3)/2$ is all the entries in the precision matrix plus the entries in the mean vector, and the plus 1, is the prior. It is apparent that QDA has significantly more parameters and is therefore much more flexible than LDA. In addition, when considering the number of parameters that need to be estimated for the QDA model, it is noted that when the number of variables (p),

and/or the number of classes (K) becomes large, a large number of parameters need to be estimated. This leads to a relatively complex model, which in turn could potentially yield high variance. This high variance can lead to poor generalisation performance of the model. As noted in Chapter 1, regularisation can be used to reduce variance.

The parameters for QDA can be estimated using maximum likelihood. If the training sample has the form (g_i, \mathbf{x}_i) , $i = 1, 2, \dots, N$, the likelihood of observing such a sample is given by,

$$L = \prod_{i=1}^N P(G = g_i, \mathbf{X} = \mathbf{x}_i). \quad (2.11)$$

Applying the Bayes rule, (2.11) can be written as

$$L = \prod_{i=1}^N P(G = g_i) P(\mathbf{X} = \mathbf{x}_i | G = g_i), \quad (2.12)$$

which is equivalent to

$$L = \prod_{k=1}^K \prod_{g_i = \mathcal{Y}_k} \pi_k f_k(\mathbf{x}_i). \quad (2.13)$$

When assuming Gaussian conditional densities, (2.13) becomes

$$L = \prod_{k=1}^K \prod_{g_i = \mathcal{Y}_k} \pi_k \phi(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad (2.14)$$

Taking the log of the likelihood in (2.14) we see that

$$l = \sum_{k=1}^K \sum_{g_i = \mathcal{Y}_k} \left\{ \log(\pi_k) + \log(\phi(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)) \right\}, \quad (2.15)$$

which can be written as

$$l = \sum_{k=1}^K N_k \log(\pi_k) + \sum_{k=1}^K \sum_{g_i = \mathcal{Y}_k} \left\{ \log(\phi(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)) \right\} \quad (2.16)$$

$$= \sum_{k=1}^K N_k \log(\pi_k) + \sum_{k=1}^K \left\{ l_k(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}. \quad (2.17)$$

Considering (2.17), it can be seen that the second term is the sum of the class conditional log-likelihoods, thus indicating that the parameters for the different classes can be estimated separately. From standard Gaussian maximum likelihood, the corresponding MLEs are as follows:

- $\hat{\pi}_k = \frac{N_k}{N}$
- $\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{g_i = \mathcal{Y}_k} \mathbf{x}_i$
- $\hat{\boldsymbol{\Sigma}}_k = \frac{1}{N_k} \sum_{g_i = \mathcal{Y}_k} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^T$

for $k = 1, \dots, K$. As mentioned in Section 2.2.1, for Gaussian graphical models the graph structure is captured by the inverse covariance matrix. If the estimated inverse covariance matrix for a given class is sparse (i.e. contains zero elements), we can associate a sparse graphical model, which captures the conditional independence structure, with the corresponding estimated conditional class density. The process of generating sparse inverse covariance matrices can be viewed as a form of regularisation and can lead to a potential improvement on the MLE.

2.2.3 Reduced-Rank LDA

Following Hastie et al. (2009), when performing LDA an observation is classified to the class with the closest centroid in terms of the Mahalanobis distance (modulo the effect of the class prior probabilities), that is

$$\hat{G}(\mathbf{x}) = \underset{k}{\operatorname{argmin}} \left\{ (\mathbf{x} - \bar{\mathbf{x}}_k)^T \hat{\Sigma}^{-1} (\mathbf{x} - \bar{\mathbf{x}}_k) - 2 \log(\hat{\pi}_k) \right\}. \quad (2.18)$$

By using the eigen-decomposition of Σ we can sphere the data, viz. $\mathbf{z} = \hat{\Sigma}^{-1/2} \mathbf{x}$. The classifier can then be rewritten as

$$\hat{G}(\mathbf{x}) = \underset{k}{\operatorname{argmin}} \left\{ (\mathbf{z} - \bar{\mathbf{z}}_k)^T (\mathbf{z} - \bar{\mathbf{z}}_k) - 2 \log(\hat{\pi}_k) \right\}, \quad (2.19)$$

where $\bar{\mathbf{z}}_k$ is the mean of class k of the sphered data. The class means of the sphered data lie in an affine subspace H_{K-1} , of at most $K-1$ dimensions. Since the distances between an observation and the K centroids are of interest the points can be represented in the subspace by projecting \mathbf{z} onto H_{K-1} (this is a large dimension reduction, from p to $K-1$). There thus exists a projection matrix $\mathbf{P} : p \times l$, where $l \leq K-1$, such that the LDA classifier can be written as

$$\hat{G}(\mathbf{x}) = \underset{k}{\operatorname{argmin}} \left\{ \left\| \mathbf{P}^T (\mathbf{x} - \bar{\mathbf{x}}_k) \right\|^2 - 2 \log(\hat{\pi}_k) \right\}. \quad (2.20)$$

The projection matrix can be found by calculating the eigen decomposition of $\mathbf{B} = \mathbf{V} \mathbf{D} \mathbf{V}^{-1}$, where $\mathbf{B} = \sum_{k=1}^K \frac{N_k}{N} (\bar{\mathbf{z}}_k - \bar{\mathbf{z}})(\bar{\mathbf{z}}_k - \bar{\mathbf{z}})^T$. The projection matrix is then given by $\mathbf{P} = \hat{\Sigma}^{-1/2} \mathbf{V}^*$, where \mathbf{V}^* is the first l columns of \mathbf{V} (assuming the columns are normalised and ordered according to decreasing eigenvalues). These components or variables are known as the canonical or discriminant variables. It is important to note that the dimension reduction from p to $K-1$ gives the same results as ordinary LDA. The key behind reduced-rank LDA is that we can further reduce the dimensions by selecting the first $m < l$ columns of \mathbf{V} as \mathbf{V}^* . There are two reasons for this; the

first is representing the data visually, and the second is that it can improve test misclassification error by performing LDA in this lower subspace (which is a form of regularisation). The test misclassification error rate could improve since a reduction in dimensions can possibly prevent overfitting.

2.2.4 Regularised discriminant analysis

Continuing the discussion in Hastie et al. (2009), regularised discriminant analysis can be viewed as a compromise between LDA and QDA. It entails shrinking the individual covariance matrices of QDA to the common covariance matrix used in LDA. The regularised covariance matrix has the form

$$\hat{\Sigma}_k(\alpha) = \alpha \hat{\Sigma}_k + (1 - \alpha) \hat{\Sigma}, \quad (2.21)$$

where $\hat{\Sigma}$ is the pooled covariance matrix used in LDA, $\hat{\Sigma}_k$ is the class specific covariance matrix used in QDA, and $\alpha \in [0, 1]$ is the regularisation parameter that needs to be tuned. Alternatively, in LDA, $\hat{\Sigma}$ can be shrunk to a scalar covariance using

$$\hat{\Sigma}_k(\gamma) = \gamma \hat{\Sigma}_k + (1 - \gamma) \hat{\sigma}^2 \mathbf{I}, \quad (2.22)$$

where $\gamma \in [0, 1]$ is the regularisation parameter that needs to be tuned and $\hat{\sigma}^2$ is the estimated pooled variance.

In addition, a more general family of covariance matrices is given by,

$$\hat{\Sigma}_k(\alpha, \gamma) = \alpha \hat{\Sigma}_k + (1 - \alpha) (\gamma \hat{\Sigma} + (1 - \gamma) \hat{\sigma}^2 \mathbf{I}). \quad (2.23)$$

Considering (2.23), it can be seen that α shrinks the class covariance matrices to a common pooled covariance matrix and γ shrinks the common covariance matrix to a scalar covariance matrix. This makes it possible to obtain an estimated covariance matrix anywhere between a scalar covariance matrix, a common covariance matrix and a class conditional covariance matrix. Hastie et al. (2009) performed regularised discriminant analysis on the *Vowel* data (in Chapter 5 we investigate this data in more detail). Their results indicate that the regularisation improved the misclassification rate on the test set significantly. In the following section non-Gaussian based discriminant analysis is discussed.

2.3 Non-Gaussian Based Discriminant Analysis

Gaussian-based discriminant analysis is often too restrictive or inflexible and when the true conditional density of \mathbf{X} is far from normal, Gaussian-based discriminant analysis will perform poorly. Unlike Gaussian-based discriminant analysis techniques that explicitly assume the class conditional density of \mathbf{X} to be Gaussian, non-Gaussian based methods do not make this assumption. Three popular more flexible and non-Gaussian based methods, are discussed in the following subsections.

2.3.1 Flexible discriminant analysis

Following Hastie et al. (2009) and Hastie et al. (1994), flexible discriminant analysis (FDA) is a generalisation of an alternative way of computing the canonical coordinates in LDA, using linear regression on derived responses. The generalisation makes it possible to obtain a flexible nonparametric discriminant function. As previously, assume a categorical response G , where Y_k refers to the value of class k , with each class having measured features \mathbf{X} . Let $\theta(\cdot)$ be a function that assigns scores to class labels, such that the scores are optimally predicted by a linear regression model. If the training sample has the form (g_i, \mathbf{x}_i) , $i=1,2,\dots,N$, then FDA considers the following optimization problem,

$$\min_{\beta, \theta} \sum_{i=1}^N (\theta(g_i) - \mathbf{x}_i^T \beta)^2, \quad (2.24)$$

with restrictions on θ to avoid trivial solutions. This produces a one-dimensional separation between the classes. More generally, up to $L \leq K-1$ sets of independent scorings $\theta_1, \dots, \theta_L$ for the class labels can be found, and L corresponding linear maps $\eta_l(\mathbf{X}) = \mathbf{X}^T \beta_l$, $l=1, \dots, L$. The scores $\theta_l(g)$ and the maps β_l are chosen to minimise the average squared residual (ASR) which is defined as

$$ASR = \frac{1}{N} \sum_{l=1}^L \left[\sum_{i=1}^N (\theta_l(g_i) - \mathbf{x}_i^T \beta_l)^2 \right] \quad (2.25)$$

The set of scores are assumed to be mutually orthogonal and normalised with respect to an appropriate inner product to prevent trivial zero solutions. It can be shown that the canonical variables of LDA discussed in Section 2.2.3 are identical to the β_l vectors up to a constant. In Section 2.2.3 it was found that the canonical distances are all that is needed for classification. It

can also be shown that the Mahalanobis distance of a test point \mathbf{x} to the centroid of class k , $\hat{\boldsymbol{\mu}}_k$, is given by

$$\delta_k(\mathbf{x}, \hat{\boldsymbol{\mu}}_k) = \sum_{l=1}^{K-1} \omega_l (\hat{\eta}_l(\mathbf{x}) - \bar{\eta}_l^k)^2 + D(\mathbf{x}), \quad (2.26)$$

where $\bar{\eta}_l^k$ is the mean of $\hat{\eta}_l(\mathbf{x})$ in the k -th class, $D(\mathbf{x})$ does not depend on k , and ω_l are coordinate weights given by

$$\omega_l = \frac{1}{r_l^2(1-r_l^2)}, \quad (2.27)$$

with r_l^2 equal to the mean squared residual of the l -th optimally scored fit. “Thus, LDA can be performed by a sequence of linear regressions, followed by classification to the closest class centroid in the space of fits” (Hastie et al., 2009). The regressions and the classification to the closest centroid can also be used to perform reduced-rank LDA classification.

With this generalisation of LDA, the linear regressions $\eta_l(\mathbf{X}) = \mathbf{X}^T \boldsymbol{\beta}_l$ can be replaced with more flexible nonparametric models, thus leading to more flexible classifiers. In the more general form, the ASR is given by,

$$ASR = \frac{1}{N} \sum_{l=1}^L \left[\sum_{i=1}^N (\theta_l(g_i) - \eta_l(\mathbf{x}_i))^2 + \lambda J(\eta_l) \right], \quad (2.28)$$

where J is a regulariser, appropriate for some forms of nonparametric regression. Hastie et al. (2009) suggest using generalised additive fits, spline functions, and multivariate adaptive regression splines (MARS) for the regression. After the optimal scores $\eta_l(\mathbf{X})$ have been calculated, (2.26) is used to perform the classification. By analogy to reduced-rank LDA, dimension reduction can also be performed. Lastly, the literature makes no mention of the prior probabilities π_k , but we believe the estimated prior probabilities should also be taken into consideration when performing FDA, perhaps in a similar fashion to (2.18).

2.3.2 Penalised discriminant analysis

Following Hastie et al. (2009) and Hastie et al. (2000), when FDA regression procedures amount to linear regression on a basis expansion of derived variables, followed by a quadratic penalty on the coefficients in the enlarged space, it can be shown that FDA performs penalised discriminant

analysis (PDA). Suppose a linear regression model with basis expansion $\mathbf{h}(\mathbf{X})$ and a quadratic penalty on the coefficients is fitted. The average squared residual is then given by,

$$ASR = \frac{1}{N} \sum_{l=1}^L \left[\sum_{i=1}^N (\theta_l(g_i) - \mathbf{h}^T(\mathbf{x}_i) \boldsymbol{\beta}_l)^2 + \lambda \boldsymbol{\beta}_l^T \boldsymbol{\Omega} \boldsymbol{\beta}_l \right]. \quad (2.29)$$

The choice of $\boldsymbol{\Omega}$ depends on the problem. If $\mathbf{h}^T(\mathbf{x}_i) \boldsymbol{\beta}_l$ is an expansion on spline functions, $\boldsymbol{\Omega}$ will constrain the function to be smooth; however, if the expansions are the original variables, and $\boldsymbol{\Omega}$ is taken to be the identity matrix, then a ridge regression model is fitted that penalises the size of the canonical variable coefficients and shrinks them to zero. From the PDA view, we enlarge the set of predictors with basis expansion $\mathbf{h}(\mathbf{X})$ and perform (penalised) LDA in the enlarged space. The penalised Mahalanobis distance for classification is given by

$$(\mathbf{h}(\mathbf{X}) - \mathbf{h}(\boldsymbol{\mu}_k))^T (\boldsymbol{\Sigma} + \boldsymbol{\Omega})^{-1} (\mathbf{h}(\mathbf{X}) - \mathbf{h}(\boldsymbol{\mu}_k)). \quad (2.30)$$

The penalised Mahalanobis distance gives less weight to rough coordinates and more weight to smooth ones, since the penalty is not diagonal.

2.3.3 Mixture discriminant analysis

We follow the discussion of Hastie et al. (2009) and Hastie and Tibshirani (1996). In order to discuss mixture discriminant analysis, an introduction to so called prototype classifiers is required. Prototype classifiers represent the training data by a set of points in the feature space. Each prototype has an associated class label. Classification of a point can then be made to the class corresponding to the closest prototype. Note that LDA can be viewed as a prototype classifier. Each class is represented by its centroid (prototype) and we classify an observation to the closest centroid (prototype) using an appropriate metric. In many cases a single prototype is not sufficient. Mixture models are more appropriate for representing inhomogeneous classes. The Gaussian mixture model for the k -th class has class conditional density function

$$f_k(\mathbf{x}) = \sum_{r=1}^{R_k} \pi_{kr} \phi(\mathbf{x}, \boldsymbol{\mu}_{kr}, \boldsymbol{\Sigma}), \quad (2.31)$$

where π_{kr} are the mixing proportions that sum to one, R_k is the number of prototypes for class k , ϕ denotes the Gaussian density function, and $\boldsymbol{\Sigma}$ is the common covariance matrix. Substituting (2.31) into the Bayes classifier given in (2.5) yields

$$P(G = Y_k | X = \mathbf{x}) = \frac{\sum_{r=1}^{R_k} \pi_{kr} \phi(\mathbf{x}, \boldsymbol{\mu}_{kr}, \boldsymbol{\Sigma}) \Pi_k}{\sum_{l=1}^K \sum_{r=1}^{R_l} \pi_{lr} \phi(\mathbf{x}, \boldsymbol{\mu}_{lr}, \boldsymbol{\Sigma}) \Pi_l}, \quad (2.32)$$

where Π_k , $k = 1, \dots, K$ now represent the class prior probabilities. The parameters of the model can be estimated by maximum likelihood, using the joint log-likelihood as in Section 2.2.2. The joint log-likelihood is,

$$l = \sum_{k=1}^K \sum_{g_i = Y_k} \log \left\{ \sum_{r=1}^{R_k} \pi_{kr} \left(\phi(\mathbf{x}_i | \boldsymbol{\mu}_{kr}, \boldsymbol{\Sigma}) \right) \right\}. \quad (2.33)$$

Obtaining the maximum likelihood estimates directly from (2.33) is rather difficult. For this reason, the maximum-likelihood estimates for MDA is usually calculated using the EM algorithm.

2.4 Sparse Graphical Model Based Discriminant Analysis

In this section the approach which we propose and investigate in this research is introduced. In essence, a sparse graphical model is associated with each class density. Under the Gaussian assumption, the graph structure is captured by the inverse covariance matrix (precision matrix). To make the graph sparse an L1 penalty is imposed on the precision matrix. This results in a *sparse graphical model*. For non-Gaussian distributions this is not as easy, and a semi-parametric approach is proposed to overcome this problem.

The algorithm to estimate the L1 penalised precision matrix is known as the graphical lasso and originates from Gaussian graphical models. Multiple authors have suggested using the L1 penalised maximum likelihood estimate for QDA. For example, Xu et al. (2011) used the graphical lasso with QDA for character recognition on both a digit, and Chinese character datasets. Their results appear promising. However, it is interesting to note that Xu et al. (2011) set the tuning parameter equal to 0.0001 without justification. Another application is given by Le and Hastie (2014), where they propose using the L1 penalty to enforce similar patterns of sparsity between the precision matrices used in QDA. Their proposed model possesses the property that when the tuning parameter is 0, it results in ordinary QDA, and when the tuning parameter is infinite, the resulting model is the naïve Bayes classifier. Le and Hastie (2014) also applied their proposed model on a digit classification dataset, using 5-fold cross validation to select the tuning parameter. Le et al. (2019) used the graphical lasso to estimate the precision matrix for LDA. In addition, they develop a variable selection procedure based on the graph associated with the estimated precision matrix. Their application was based on positron emission tomography (PET) images and they concluded that their results outperformed a similar study using QDA and sparse

estimates of class precision matrices. From these studies it appears that QDA with the graphical lasso can lead to promising results. Sparse Graphical Model based discriminant analysis is the focus of this thesis. This technique is revisited in Chapter 4.

2.5 Summary

The purpose of this chapter was to provide an overview of discriminant analysis. A review of the most popular Gaussian-based methods (viz. LDA and QDA) was given together with two methods that can be used for regularisation, viz. reduced-rank LDA and regularised discriminant analysis for QDA. Furthermore, a short introduction to non-Gaussian based discriminant analysis was provided, where FDA, PDA and mixture discriminant analysis were discussed. Flexible discriminant analysis exploits the relationship between the canonical or discriminant variables of LDA and linear regression. It then generalises the linear regression model to more flexible regression techniques. Penalised discriminant analysis is a special case of FDA, where the regression procedure of FDA are basis expansions of derived variables and penalized regression is performed in the new space. The penalised regressions are a generalisation of ridge regression. Mixture discriminant analysis looks at LDA from a prototype classifier viewpoint and introduces more prototypes per class by means of a Gaussian mixture model.

Finally, in this chapter sparse graphical model based discriminant analysis was introduced. For this technique, an L1 penalty is imposed on the inverse covariance matrix. The idea originates from estimating sparser graphs for Gaussian graphical models. As noted previously, multiple authors have investigated this approach and the results seem promising. In the next chapter we introduce all the concepts necessary to understand our proposal.

CHAPTER 3

BASIC METHODOLOGY

3.1 Introduction

Probabilistic graphical models use a graph structure to depict the relationships between the variables in a probabilistic model. When a probabilistic model comprises of hundreds or thousands of variables, the graph structure provides a convenient way to represent the relationships between the variables. For the purposes of this study, undirected graphs, also referred to as Markov graphs, are considered. Depending on a researcher's objective for using a probabilistic graphical model, *the relationship* between variables can mean different things.

The development of a probabilistic graphical model can broadly be divided into three primary components, as follows:

- Fitting the probabilistic model to the random variables.
- Determining which relationships to depict in the undirected graph.
- Performing inference on the distribution using the undirected graph.

One of the main objectives of this chapter is to provide sufficient information to establish a fundamental understanding of these three components. To achieve this, an examination of a probabilistic model for continuous random variables will be provided. It will be shown how this probabilistic model fits the conditions necessary for the inference algorithms. An illustration of how the conditional independence structure of this model can be found will be provided, followed by an investigation of how to estimate the conditional independence structure.

To determine which relationships to depict in the undirected graph, popular relationships types between variables are described in Section 3.2. The relationship types considered are *marginal correlations* (Section 3.2.1), *partial correlations* (Section 3.2.2) and *conditional independence* (Section 3.2.3). The latter relationship type, i.e., conditional independence, is the most common type used for undirected graphs. Therefore, a more detailed description of this relationship type is provided, as well as an introduction to the basic concepts which are used in the inference algorithms. Section 3.3 introduces Markov random fields for continuous random variables in which the relationship between Gaussian graphical models and pairwise Markov graphs are investigated together with the relation to partial correlation graphs. Section 3.4 investigates estimation of the precision matrix for the Gaussian graphical model as well methods to select the tuning parameter in the graphical lasso algorithm. Section 3.5 provides a comprehensive summary of this chapter.

3.2 The Relationships Between Variables Found in Graphs

As noted in Section 3.1, three popular relationships between variables that can be depicted in graphs are described in the follow subsections. The relationship types considered are marginal correlations, partial correlations and conditional independence. Before exploring the relationships between the variables, it is important to introduce key terminology which is pertinent to the sections to follow. Suppose we have a graph G with vertex set V and edge set E . Then the vertices represent random variables with joint probability distribution P , and the edges illustrate the relationships between the variables. The relationships illustrated by the edges determine the type of the graph.

3.2.1 Marginal correlation graphs

Following Wasserman (2019), in a marginal correlation graph the edge between two vertices would indicate a significant association to exist between the two variables. When working with marginal correlation graphs two questions have to be answered:

- i) “What measure of association between the two variables will be used?”; and
- ii) “What level would imply the association between the variables to be significant?”

Popular measures include the Pearson correlation, Kendall’s Tau and the distance correlation. Since the estimate of association between two variables is based on a sample, it is unlikely to obtain an estimate of no association (exactly zero values). For this reason, it is necessary to test whether the estimated association is significantly different from zero. Permutation tests can be applied to determine whether the association measure is significant (i.e., significantly different from zero) or not. To perform a permutation test, one can permute the values of the variables and recalculate the association measure. The p-value describes the proportion of measures from the permutations greater in absolute value than the original measure. If the p-value is small, it indicates a significant association.

Generally, marginal correlation graphs are not popular in the literature. This is largely because the measures do not account for the influence of the other variables, which in turn may show relationships between variables that do not exist. This can be illustrated through the following simplified example. Consider the variables ice-cream sales, shark attacks, and the weather. Fitting a marginal correlation graph would indicate that there is a relationship between all three variables (since they are all correlated), however, the true relationships are between weather and ice-cream sales, and weather and shark attacks, and not between ice-cream sales and shark attacks.

It is important to note that this technique should not be completely discarded since it can have useful applications in similar to the marginal correlations are used to screen variables in

regression and classification problems. For instance, optimisation algorithms for undirected graphs can be computationally very demanding, whereas using the screening structure of marginal correlation graphs could potentially reduce the computational burden. However, when using the marginal correlation graph as a screening method, the user should still be cautious, since marginal independence does not imply conditional independence.

3.2.2 Partial correlation graphs

Continuing the discussion of Wasserman (2019), the edges in a partial correlation graph represent partial correlations between variables. Partial correlation is defined as the correlation between two variables after removing the effect of all other random variables in the set. To develop this more formally, assume that X and Y are random variables, and that Z is a vector of random variables. The partial correlation between X and Y , $\rho(X, Y | Z)$, is then defined to be the ordinary correlation between the residuals of X regressed onto Z , and the residuals of Y regressed onto Z . If ρ_{ij} denotes the partial correlation between variables i and j , then the graph will have an absent edge between variables i and j if ρ_{ij} equals zero. For the same reason as correlation graphs, it is highly unlikely to observe partial correlations that are exactly equal to zero. Therefore, to generate sparser graphs, some partial correlations need to be set or truncated to zero. Since partial correlations are proportional to linear regression coefficients, regularisation or variable selection can be used for this purpose.

3.2.3 Conditional independence graphs

Conditional independence graphs are commonly used when working with undirected graphs. For two continuous random variables (or two sets of continuous random variables), e.g. X and Y , to be conditionally independent given another random variable (or set of random variables), e.g. Z , the distribution of X is independent of Y given Z . This implies having observed Z , observing Y adds no additional information about the state of X , and vice versa.

More formally, when $P(X)$ denotes the probability mass function of the random variable X , then for a random variable, or sets of random variables, to be conditionally independent given Z , we need to have

$$P(X, Y | Z) = P(X | Z) \times P(Y | Z) \quad (3.1)$$

With regard to Hastie et al. (2009), if G is an undirected graph, then the absence of an edge between two vertices would imply that the two random variables corresponding to these vertices are conditionally independent, given all other vertices or random variables. This is referred to as the pairwise Markov independencies of G and the probability distribution represented in the

graph must satisfy all the independencies illustrated by the graph. An example is considered below.

Suppose the interest is in the joint probability distribution of random variables $\{A, B$ and $C\}$ and Figure 3.1 depicts the joint distributions with different independence assumptions. Then Figure 3.1(a) shows that variables B and C are conditionally independent, given that variable A is observed. More formally, we have $P(B, C | A) = P(B | A) \times P(C | A)$. Figure 3.1(b) illustrates a scenario where none of the variables are conditionally independent given the others. As noted previously, conditional independence can exist between sets of random variables and is not limited to pairs of random variables. The conditional independence between sets of random variables can be inferred from the undirected graph.

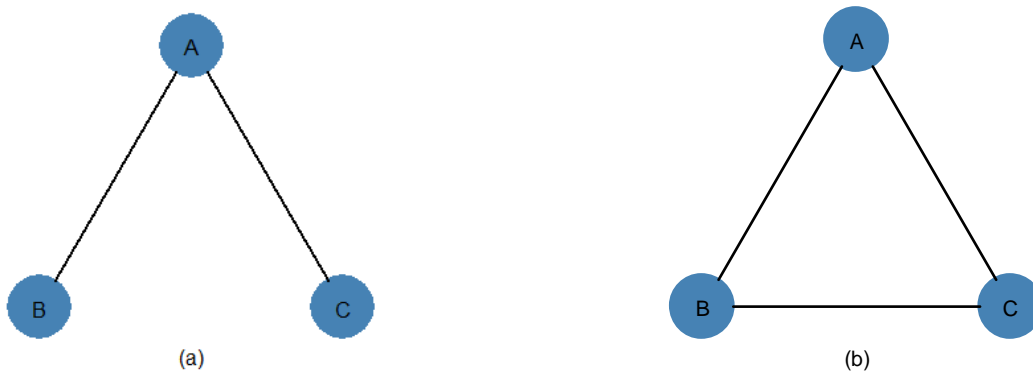


Figure 3.1: Undirected graphical model representation of a joint distribution with conditional independence between two variables vs a joint distribution with no independence.

In an undirected graph, a subset of the vertices, together with their edges are referred to as a subgraph. If G_1 , G_2 and G_3 are subgraphs, and G_1 separates G_2 and G_3 such that the only path between subgraphs G_1 and G_2 is through G_3 , then the set of variables in subgraphs G_1 and G_2 are conditionally independent, given the set of variables in subgraph G_3 . Graphs with this property are known to possess the *global Markov property*.

Figure 3.2 illustrates two examples, where A - J are random variables and the graph represents their joint distribution. If $\{A, B, C\}$; $\{D, E\}$; $\{F, G, H\}$ and $\{I, J\}$ are assumed subgraphs, then $\{D, E\}$ is observed to separate $\{A, B, C\}$ from $\{F, G, H\}$. Thus, it can be concluded that $\{A, B, C\}$ and $\{F, G, H\}$ are conditionally independent given $\{D, E\}$. In Figure 3.2, $\{A, B, C, D, E, F, G, H\}$ and $\{I, J\}$ can be seen

as two subgraphs, separated by an empty set. These two subgraphs and the corresponding subsets of random variables are independent of each other.

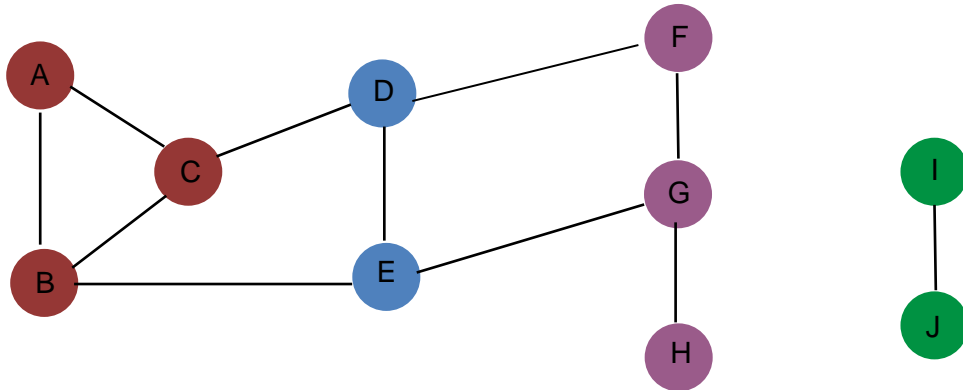


Figure 3.2: Illustrating the conditional independencies between subgraphs.

Hastie et al. (2009) state that the pairwise and global Markov properties of a graph are equivalent. This implies that the pairwise Markov property holds true, only if the global Markov property holds true, and vice versa. This is useful since the graph structure is often determined by specifying or estimating the pairwise Markov independencies. Thus, the global independence relationships can be inferred from these constructed graphs. Not only is the global Markov property appropriate for interpreting relationships, it also simplifies computation of inference algorithms.

As noted in Section 3.1, one reason a probability distribution is represented in terms of a graphical model is for inference calculations. These inference algorithms are dependent on factorisation of the probability distribution. One possibility is factorisation into the product of functions on the maximal cliques. A clique is a subgraph where all the vertices are connected and is referred to as a fully connected subgraph. A maximal clique is clique that is not a subset of a larger clique. In other words, including an additional variable in the subgraph will result in the subgraph not being a clique anymore. According to the Hammersley-Clifford theorem, if a distribution satisfies the conditional independency represented in the graph, the probability density can be represented as a product of factors or potential functions over each maximal clique. More formally, the density function can be represented as,

$$f(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \psi_c(\mathbf{x}_c), \quad (3.2)$$

where $\psi_c(\cdot)$ is the potential function associated with clique c . The arguments of the potential function are the variables associated with its corresponding cliques. Note that a potential function can be any positive function of its arguments.

In (3.2), Z is the normalising constant, also known as the partition function, and is defined for the discrete case as,

$$Z = \sum_{x \in X} \prod_{c \in C} \psi_c(x_c). \quad (3.3)$$

For the continuous case, the summation is simply replaced by integration. With reference to Figure 3.1 (b), the fully connected graph can represent the density function with the following structure:

$$f(a, b, c) = \frac{1}{Z} \psi(a, b, c). \quad (3.4)$$

Equivalently, the same graph can also describe the density function with the structure:

$$f(a, b, c) = \frac{1}{Z} \psi_1(a, b) \psi_2(a, c) \psi_3(b, c). \quad (3.5)$$

In practice, users of Markov graphs assume that the density function can be factorised into the potential function defined on the links between the variables, as in Equation (3.5). These types of graphs are referred to as *pairwise Markov networks* (Barber, 2012). This factorisation is the reason pairwise Markov graphs are easier and less complex to use.

By extending (3.2) to non-maximal cliques and assuming a pairwise network through the potential function, it can be seen that the general form of a density function represented in a pairwise Markov graph is then given by:

$$f(\mathbf{x}) = \frac{1}{Z} \prod_{(i,j) \in E} \psi_{ij}(x_i, x_j) \prod_{i \in V} \psi_i(x_i) \quad (3.6)$$

where $(i, j) \in E$ refers to the set of indices with an edge between the variables i and j , and where $i \in V$ refers to the nodes which are separated from all other nodes. More formally, $E = \{(i, j) : i < j, \text{ nodes } i \text{ and } j \text{ are linked}\}$. It is important to note that the factorisation cannot always be read off the undirected graph. Thus, it needs to be stated explicitly or be represented in a factor graph.

Henceforth, the discussion will focus on pairwise Markov graphs. Therefore, representing the factorisation in a factor graph is not of benefit. The following section presents a discussion of probabilistic models for continuous random variables.

3.3 Markov Random Fields for Continuous Random Variables

Section 3.2 discussed the structure of undirected graphs, how these graphs can be factorised, and what the graph vertices and nodes represent. In this section a common probabilistic model, often represented in undirected graphs, is described. Note that the discussion that follows is largely based upon Hastie et al. (2009).

When working with continuous random variables in probabilistic graphical models, it is common to assume that the joint distribution of the random variables is Gaussian with mean vector $\boldsymbol{\mu}$ and with covariance matrix $\boldsymbol{\Sigma}$. Graphical models that assume the random variables to follow a Gaussian distribution are referred to as *Gaussian graphical models*. Although this assumption may appear restrictive, it often yields dividends in high-dimensional datasets, since making structural assumptions is often the best way to manage the curse of dimensionality. Furthermore, the Gaussian assumption has two convenient analytical properties: firstly, the distribution can be represented by a pairwise Markov graph (discussed in Section 3.3.1 below), and secondly, the topologies of its pairwise Markov graph and partial correlation graph are equivalent (discussed in Section 3.3.2) which reduces the conceptual burden of the graphical distribution representation.

3.3.1 The Gaussian graphical model and its relation to a pairwise Markov graph

Assume that the random variables X_1, \dots, X_p follow a Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The graph structure is captured by the inverse covariance matrix denoted by $\boldsymbol{\Sigma}^{-1}$, often also referred to as the precision matrix $\boldsymbol{\Theta}$. If the ij -th element of $\boldsymbol{\Theta}$ is zero, then variables i and j are conditionally independent, given all other variables. The Gaussian distribution automatically satisfies the assumptions of a pairwise Markov graph. To illustrate this, consider the probability density function of a multivariate Gaussian distribution,

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right). \quad (3.7)$$

Centring the data around the origin, (3.7) becomes,

$$\begin{aligned} f(\mathbf{x}) &= \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2} \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x}\right) \\ &= \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2} \sum_{i=1}^p \sum_{j=1}^p x_i \theta_{ij} x_j\right) \end{aligned} \quad (3.8)$$

where θ_{ij} is the ij -th element of $\boldsymbol{\Sigma}^{-1}$ or $\boldsymbol{\Theta}$. By writing the exponent as a product, (3.8) can be written as

$$\begin{aligned}
& \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \prod_{i,j} \exp\left(-\frac{1}{2} x_i \theta_{ij} x_j\right) \\
&= \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \prod_{i \neq j} \exp\left(-\frac{1}{2} x_i \theta_{ij} x_j\right) \times \prod_i \exp\left(-\frac{1}{2} \theta_{ii} x_i^2\right) \\
&= \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \prod_{i < j} \exp\left(-x_i \theta_{ij} x_j\right) \times \prod_i \exp\left(-\frac{1}{2} \theta_{ii} x_i^2\right) \\
&= \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \prod_{(i,j) \in E} \exp\left(-x_i \theta_{ij} x_j\right) \times \prod_i \exp\left(-\frac{1}{2} \theta_{ii} x_i^2\right), \tag{3.9}
\end{aligned}$$

where $E = \{(i, j) : i < j, \theta_{ij} \neq 0\}$. When comparing (3.9) to (3.6) it is apparent that the normal density is equivalent to a pairwise Markov graph with potential functions given by

$$\psi_{ij}(x_i, x_j) = \exp\left(-x_i \theta_{ij} x_j\right) \tag{3.10}$$

and

$$\psi_i(x_i) = \exp\left(-\frac{1}{2} \theta_{ii} x_i^2\right). \tag{3.11}$$

In addition, the normalising constant is given by

$$Z = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}}. \tag{3.12}$$

Expression (3.9) also shows that if variables X_1, \dots, X_p are normally distributed and if an element in the inverse covariance matrix is zero, then the potential function over the two variables equals one. This indicates that the associated variables are conditionally independent. Thus, it can be concluded that the precision matrix captures the structure of the graph for normally distributed random variables.

3.3.2 The Gaussian graphical model and its relation to partial correlation graphs

Continuing the discussion of Hastie et al. (2009). As noted previously, the topologies of the Markov graph and partial correlation graph associated with a Gaussian distribution are equivalent. It is well-known that the multiple linear regression coefficients, $\boldsymbol{\beta}$, represent the influence of each individual variable on the response after considering the influence of all the other variables. If a coefficient equals zero, it implies that the variable has no influence on the response, given all other variables. If a coefficient equals zero, it also implies that the partial correlation between the variable and the response is zero. More formally, the population least squares coefficients are given by

$$\boldsymbol{\beta}_i = \left(\boldsymbol{\Sigma}_{\mathbf{x}_{-i}} \right)^{-1} \boldsymbol{\sigma}_{\mathbf{x}_{-i}X_i}, \quad i = 1, \dots, p, \quad (3.13)$$

where $\boldsymbol{\beta}_i$ denotes the coefficients of the i -th variable regressed on the remaining $p-1$ variables, $\boldsymbol{\Sigma}_{\mathbf{x}_{-i}}$ is the covariance matrix with the i -th variable removed, and $\boldsymbol{\sigma}_{\mathbf{x}_{-i}X_i}$ is the vector of covariances between the i -th variable and the other $p-1$ variables. Considering (3.13), the relationship between each pair of variables is only dependent on the covariance matrix between the variables. Alternatively, (3.13) can be rewritten in terms of the precision matrix. First, partition the precision matrix, $\boldsymbol{\Theta}$, and covariance matrix, $\boldsymbol{\Sigma}$, as:

$$\boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\Theta}_{\mathbf{x}_{-i}} & \boldsymbol{\theta}_{\mathbf{x}_{-i}X_i} \\ \boldsymbol{\theta}_{\mathbf{x}_{-i}X_i}^T & \theta_{X_iX_i} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{\mathbf{x}_{-i}} & \boldsymbol{\sigma}_{\mathbf{x}_{-i}X_i} \\ \boldsymbol{\sigma}_{\mathbf{x}_{-i}X_i}^T & \sigma_{X_iX_i} \end{bmatrix} \quad (3.14)$$

where $\boldsymbol{\Theta}_{\mathbf{x}_{-i}}$ is $\boldsymbol{\Theta}$ with the i -th row and i -th column removed, $\boldsymbol{\theta}_{\mathbf{x}_{-i}X_i}$ is the i -th column of $\boldsymbol{\Theta}$ with the i -th row removed, and $\theta_{X_iX_i}$ is the entry in i -th row and i -th column of $\boldsymbol{\Theta}$. The same notation holds for $\boldsymbol{\Sigma}$. When utilising the partitioned inverse of $\boldsymbol{\Sigma}$, $\boldsymbol{\theta}_{\mathbf{x}_{-i}X_i}$ can be written as,

$$\boldsymbol{\theta}_{\mathbf{x}_{-i}X_i} = -\theta_{X_iX_i} \cdot \left(\boldsymbol{\Sigma}_{\mathbf{x}_{-i}} \right)^{-1} \boldsymbol{\sigma}_{\mathbf{x}_{-i}X_i}. \quad (3.15)$$

Rewriting (3.15) in terms of $\boldsymbol{\sigma}_{\mathbf{x}_{-i}X_i}$ we have,

$$\boldsymbol{\sigma}_{\mathbf{x}_{-i}X_i} = \frac{-1}{\theta_{X_iX_i}} \cdot \left(\boldsymbol{\Sigma}_{\mathbf{x}_{-i}} \right) \cdot \boldsymbol{\theta}_{\mathbf{x}_{-i}X_i}. \quad (3.16)$$

Substituting (3.16) into (3.13) yields,

$$\boldsymbol{\beta}_i = \frac{-\boldsymbol{\theta}_{\mathbf{x}_{-i}X_i}}{\theta_{X_iX_i}}, \quad i = 1, \dots, p. \quad (3.17)$$

From (3.17) it is observed that the regression coefficients are proportional to the elements of the inverse covariance matrix. Furthermore, if an element in the inverse covariance matrix is zero, the regression coefficients between the two corresponding variables will be zero. Thus, if the data are assumed to follow a Gaussian distribution, the graph structure can be estimated by fitting p multiple linear regression models and letting a zero coefficient indicate the absence of an edge. When considering the linear regression model, it is also observed that the Gaussian assumption assumes that the dependence between the variables are linear. Even though structural assumptions often perform well in a high-dimensional setting, it can be too restrictive.

In summary, for Gaussian graphical models it is the precision matrix which determines the structure of the Markov graph. The precision matrix is a parameter in the multivariate Gaussian

distribution and needs to be estimated. In the next section, the different approaches for estimating this parameter are discussed.

3.4 Estimating the Precision Matrix for the Gaussian Graphical Model

In this section, two methods for estimating the precision matrix are reviewed. These methods are (1) the *maximum likelihood estimate*, and (2) the *L1 penalised estimate*. In addition, the algorithm to solve the L1 penalised estimate is discussed, together with improvements of this algorithm in terms of computational speed. The algorithms are also compared on a dataset in order to confirm that they provide the same result, and to illustrate the adopted algorithms. For the L1 penalised estimate, a regularisation parameter λ needs to be estimated. In the sections that follow, different techniques to select λ are also assessed.

3.4.1 The maximum likelihood estimate

Maximum likelihood estimation (MLE) is the most common method used to estimate the parameters of a distribution. For a Gaussian distribution two parameters need to be estimated, viz. the mean vector and the covariance matrix. Since the precision matrix is simply the inverse of the covariance matrix, the precision matrix can be viewed as a parameter of the distribution instead of the covariance matrix.

Assume n samples $\mathbf{X}_1, \dots, \mathbf{X}_n$ are drawn i.i.d from a p -dimensional multivariate Gaussian distribution with mean vector $\mathbf{0}$ and covariance matrix Σ . Note that the assumption of a $\mathbf{0}$ mean vector is simply for cosmetics and is not really necessary, since it only determines the location of the distribution and therefore does not influence the result. The likelihood function is then given by

$$L(\Theta) = \prod_{i=1}^n f(\mathbf{x}_i) \quad (3.18)$$

where $f(\mathbf{x}_i) = \frac{1}{(2\pi)^{p/2}} |\Theta|^{1/2} \exp\left(-\frac{1}{2} \mathbf{x}_i^T \Theta \mathbf{x}_i\right)$. Finding the precision matrix that maximises the likelihood, is equivalent to finding the precision matrix that maximises the log likelihood,

$$\begin{aligned} l(\Theta) &= \sum_{i=1}^n \log \left(\frac{1}{(2\pi)^{p/2}} |\Theta|^{1/2} \exp\left(-\frac{1}{2} \mathbf{x}_i^T \Theta \mathbf{x}_i\right) \right) \\ &= c + \frac{1}{2} \sum_{i=1}^n \log(|\Theta|) - \frac{1}{2} \sum_{i=1}^n \mathbf{x}_i^T \Theta \mathbf{x}_i \\ &= c + \frac{n}{2} \log(|\Theta|) - \frac{n}{2} \text{tr}(\mathbf{S}\Theta). \end{aligned} \quad (3.19)$$

Finding the precision matrix that maximises (3.19) is equivalent to finding the precision matrix that maximises

$$l(\Theta) = \log(\det(\Theta)) - \text{tr}(\mathbf{S}\Theta), \quad (3.20)$$

where $\mathbf{S} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T$ is the sample covariance matrix.

The negative of Equation (3.20) is a convex function of Θ (Uhler, 2017). To maximise (3.20), the derivative with respect to Θ is taken, that is,

$$\frac{\partial l(\Theta)}{\partial \Theta} = \Theta^{-1} - \mathbf{S}. \quad (3.21)$$

After setting (3.21) equal to zero, and solving for Θ , the MLE of Θ is obtained as $\hat{\Theta} = \mathbf{S}^{-1}$, which is the inverse of the sample covariance matrix. It is commonly known that the MLE converges to the population parameter as the sample size increases ($n \rightarrow \infty$). Although the MLE appears to solve the problem, it is not always the most desirable or feasible estimate. The reason for this is that when the number of variables is larger than the number of observations, the inverse \mathbf{S}^{-1} cannot be calculated since the estimated covariance matrix is not of full rank. Furthermore, as noted previously, a zero entry in the precision matrix indicates conditional independence. It is very unlikely that the MLE would contain zero entries. To improve the usefulness of the graph, some of the edges could be assumed missing, based on expert knowledge or alternatively, through using some feature selection method.

As mentioned in Chapter 1, the accuracy of an estimated squared error loss can be divided into three parts: (1) the irreducible error, (2) the bias (squared), and (3) the variance. MLEs can exhibit high variance and regularisation is often needed to combat this..

In the next section, a method for addressing the issues associated with the MLE is investigated. The resulting estimate is referred to as the L1 penalised estimate.

3.4.2 The L1 penalised estimate

Multiple authors, including Yuan and Lin (2007), Banerjee et al. (2008) and Dahl et al. (2008), suggest that the L1 penalised estimator should be used in order to solve the problems associated with the MLE of Θ . L1 penalised estimation finds Θ that maximises

$$l(\Theta) = \log(\det(\Theta)) - \text{tr}(\mathbf{S}\Theta) - \lambda \|\Theta\|_1, \quad (3.22)$$

where $\|\Theta\|_1$ is the L_1 norm of Θ and λ is the regularisation or tuning parameter. From (3.22) it is observed that if the tuning parameter is large, maximising (3.22) will shrink the elements of Θ

toward zero. When the tuning parameter is zero, we return to the MLE solution. Unlike using the L2 norm for regularisation, the L1 norm has the property that it allows the shrunken elements to automatically be truncated to zero. This property of the L1 norm is illustrated by the following example, which also provides insight as to how the L1 penalty influences the estimates. Assume a scenario with two random variables, where the covariance between the two variables are negative, and the tuning parameter is less than or equal to the absolute value of the covariance. Under these assumptions, the objective function to maximise as given in (3.22) can be written as

$$l(\Theta) = \log(\det(\Theta)) - \text{tr}(\mathbf{S}\Theta) - \lambda \mathbf{1}^T \Theta \mathbf{1}, \quad (3.23)$$

where $\mathbf{1}$ is a vector with all elements equal to one. Equation (3.23) can be maximised by taking its derivative and setting it equal to $\mathbf{0}$. The derivative of (3.23) is

$$\frac{\partial l(\Theta)}{\partial \Theta} = \Theta^{-1} - \mathbf{S} - \lambda \mathbf{1}\mathbf{1}^T. \quad (3.24)$$

Setting (3.24) equal to $\mathbf{0}$, yields

$$\hat{\Theta} = (\mathbf{S} + \lambda \mathbf{1}\mathbf{1}^T)^{-1}, \quad (3.25)$$

where $\mathbf{1}\mathbf{1}^T : p \times p$ is the matrix with all its elements equal to one. Let the sample covariance matrix of the two variable examples be denoted by

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix}, \quad (3.26)$$

where $s_{21} = s_{12} < 0$.

Substituting (3.26) into (3.25), the estimated precision matrix becomes:

$$\begin{aligned} (\mathbf{S} + \lambda \mathbf{1}\mathbf{1}^T)^{-1} &= \begin{bmatrix} s_{11} + \lambda & s_{12} + \lambda \\ s_{12} + \lambda & s_{22} + \lambda \end{bmatrix}^{-1} \\ &= \frac{1}{(s_{11} + \lambda)(s_{22} + \lambda) - (s_{12} + \lambda)^2} \begin{bmatrix} s_{22} + \lambda & -(s_{12} + \lambda) \\ -(s_{12} + \lambda) & s_{11} + \lambda \end{bmatrix}. \end{aligned} \quad (3.27)$$

Considering (3.27), it is observed that the L1 penalty influences the estimated precision matrix in two ways. The first is by a multiplication factor. As long as $s_{11} + s_{22} > 2s_{12}$, in other words the sum of the variances of the two variables exceed the covariance between the two variables multiplied by two, the L1 penalty will shrink the coefficients proportionally as λ increases. Using the Cauchy-Schwarz inequality, it is possible to prove that in fact $s_{11} + s_{22} \geq 2s_{12}$. The second influence is the subtraction of the cost parameter from the entries in the precision matrix. This illustrates that the

L1 penalty can induce zero inverse covariance elements when it is equal to the absolute value of the off-diagonal entries of the covariance matrix.

The most popular method in the literature for solving the maximisation problem of (3.22) exactly, is the *Graphical lasso* algorithm, first proposed by Friedman et al. (2008). In the next section, this algorithm, as well as its derivation, are discussed.

3.4.3 The graphical lasso

The Graphical lasso is an algorithm used to optimise (3.22). Geometrically, the L1 penalty will be a rhomboid constraint on the convex function, making the optimisation problem of (3.22) still convex. Thus, taking the sub-gradient of (3.22) and setting it equal to zero, would give the optimal estimate for Θ . In other words, by the Karush-Kuhn-Tucker conditions, a necessary and sufficient condition for Θ to maximise (3.22), satisfies (3.28) (Witten et al., 2011). Using sub-gradient notation,

$$\frac{\partial l(\Theta)}{\partial \Theta} = \Theta^{-1} - \mathbf{S} - \lambda \cdot \text{Sign}(\Theta) = \mathbf{0}, \quad (3.28)$$

where $\text{Sign}(\Theta)$ is a matrix with elements $\text{Sign}(\theta_{ij}) = \text{sign}(\theta_{ij})$ if $\theta_{ij} \neq 0$, else if $\theta_{ij} = 0$, $\text{Sign}(\theta_{ij}) \in [-1, 1]$, where $\text{sign}(\theta_{ij})$ denotes the sign of θ_{ij} . Denoting Θ^{-1} as \mathbf{W} , Friedman et al. (2008) suggested solving Θ and \mathbf{W} one row and column at a time using regression. First, the matrices are partitioned into two parts; part 1: the first $p-1$ rows and $p-1$ columns, and part 2: the p -th row and column:

$$\therefore \begin{bmatrix} \mathbf{W}_{11} & \mathbf{w}_{12} \\ \mathbf{w}_{12}^T & w_{22} \end{bmatrix}, \begin{bmatrix} \Theta_{11} & \boldsymbol{\theta}_{12} \\ \boldsymbol{\theta}_{12}^T & \theta_{22} \end{bmatrix}, \begin{bmatrix} \mathbf{S}_{11} & \mathbf{s}_{12} \\ \mathbf{s}_{12}^T & s_{22} \end{bmatrix} \text{ and } \begin{bmatrix} \text{Sign}(\Theta_{11}) & \text{Sign}(\boldsymbol{\theta}_{12}) \\ \text{Sign}(\boldsymbol{\theta}_{12}^T) & \text{Sign}(\theta_{22}) \end{bmatrix}.$$

The upper right block of (3.28) can then be written as,

$$\mathbf{w}_{12} - \mathbf{s}_{12} - \lambda \cdot \text{Sign}(\boldsymbol{\theta}_{12}) = \mathbf{0} \quad (3.29)$$

Since $\mathbf{W} = \Theta^{-1}$, we have,

$$\begin{bmatrix} \mathbf{W}_{11} & \mathbf{w}_{12} \\ \mathbf{w}_{12}^T & w_{22} \end{bmatrix} \begin{bmatrix} \Theta_{11} & \boldsymbol{\theta}_{12} \\ \boldsymbol{\theta}_{12}^T & \theta_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \quad (3.30)$$

$$\therefore \mathbf{W}_{11} \boldsymbol{\theta}_{12} + \mathbf{w}_{12} \theta_{22} = \mathbf{0}. \quad (3.31)$$

This implies that

$$\mathbf{w}_{12} = -\mathbf{W}_{11} \boldsymbol{\theta}_{12} / \theta_{22}. \quad (3.32)$$

Recall from Equation (3.17) that the least squares regression coefficients are given by

$$\boldsymbol{\beta}_i = \frac{-\boldsymbol{\theta}_{\mathbf{x}_i X_i}}{\theta_{X_i X_i}} \quad i = 1, \dots, p.$$

From (3.32), $\frac{-\boldsymbol{\theta}_{12}}{\theta_{22}}$ can be considered the constrained regression coefficients $\boldsymbol{\beta}$, obtained by regressing the p -th variable on the remaining $p-1$ variables. Substituting (3.17) into Equation (3.32) leads to,

$$\mathbf{w}_{12} = \mathbf{W}_{11} \boldsymbol{\beta}, \quad (3.33)$$

while substituting (3.33) into (3.29) gives:

$$\mathbf{W}_{11} \boldsymbol{\beta} - \mathbf{s}_{12} - \lambda \cdot \text{Sign}(-\boldsymbol{\theta}_{22} \boldsymbol{\beta}) = \mathbf{0}. \quad (3.34)$$

Since $\theta_{22} > 0$, (3.34) becomes,

$$\mathbf{W}_{11} \boldsymbol{\beta} - \mathbf{s}_{12} + \lambda \cdot \text{Sign}(\boldsymbol{\beta}) = \mathbf{0}. \quad (3.35)$$

It is well known that the objective function of the lasso regression model is given by,

$$\frac{1}{2} (\mathbf{y} - \mathbf{Z}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{Z}\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}\|_1, \quad (3.36)$$

where \mathbf{y} is the response variable, and \mathbf{Z} is the set of predictors. Taking the sub-gradient of (3.36) yields

$$\mathbf{Z}^T \mathbf{Z}\boldsymbol{\beta} - \mathbf{Z}^T \mathbf{y} + \lambda \cdot \text{Sign}(\boldsymbol{\beta}). \quad (3.37)$$

From (3.37) it can be seen that if $\mathbf{Z}^T \mathbf{Z}$ is replaced with \mathbf{W}_{11} , and $\mathbf{Z}^T \mathbf{y}$ with \mathbf{s}_{12} , we arrive back at (3.35). Therefore solving $\boldsymbol{\beta}$ in (3.35) can be achieved by using the algorithm for solving $\boldsymbol{\beta}$ in the lasso. Friedman et al. (2008) suggest solving the lasso problem using coordinate descent. Letting $\mathbf{V} = \mathbf{W}_{11}$, the estimated $\boldsymbol{\beta}$ coefficients are obtained by cycling the following:

$$\hat{\beta}_j \leftarrow S \left(\mathbf{s}_{12j} - \sum_{k \neq j} V_{kj} \hat{\beta}_k; \lambda \right) / V_{jj} \quad (3.38)$$

$j = 1, 2, \dots, p-1$, where $S(x, t) = \text{sign}(x)(|x| - t)_+$, known as the soft-threshold operator, and x_+ denotes the positive part of x . Equation (3.38) is cycled until convergence.

More formally, the graphical lasso algorithm is given in Algorithm 1.

Algorithm 1: Graphical lasso

Step 1

- Set $\mathbf{W} = \mathbf{S} + \lambda \mathbf{I}$

Step 2

- Partition \mathbf{W} into two parts. Part 1 all but the j -th column and j -th row (\mathbf{W}_{-j-j}), and part two the j -th column and j -th row (\mathbf{W}_{jj}).
- Solve the equation $\mathbf{W}_{-j-j} \boldsymbol{\beta} - \mathbf{s}_{1j} + \lambda \cdot \text{Sign}(\boldsymbol{\beta}) = \mathbf{0}$ using coordinate descent, as in (3.38).
- Update $\mathbf{w}_{1j} = \mathbf{W}_{jj} \boldsymbol{\beta}$.
- Repeat Step 2 for $j = 1, 2, \dots, p$, until convergence.

Step 3

- In the final cycle, for each j , set $\hat{\boldsymbol{\theta}}_{1j} = -\boldsymbol{\beta} \hat{\boldsymbol{\theta}}_{jj}$.

It is important to note that the solution for the diagonal elements of \mathbf{W} is the diagonal elements of $\mathbf{S} + \lambda \mathbf{I}$. This can be observed from (3.31), since the diagonal elements of $\text{Sign}(\boldsymbol{\Theta})$ will always be positive. Thus, in the graphical lasso algorithm, the diagonal elements of \mathbf{W} remain unchanged. If the user does not want to penalise the diagonal elements of $\boldsymbol{\Theta}$, then the diagonal elements of the final solution for \mathbf{W} is set to the diagonal of \mathbf{S} . In any optimisation problem, time and/or computation power is always a consideration. Recent research, such as found in the paper by Witten et al. (2011), propose algorithms to speed up the Graphical lasso.

3.4.4 The faster graphical lasso

Witten et al. (2011) propose a screening procedure which can be applied before solving (3.22). This screening procedure is based on a necessary and sufficient condition for the solution to the graphical lasso to be block-diagonal (the meaning of block-diagonal will become apparent on the next page). It is claimed that this check will achieve “massive computational gains”. Following Witten et al. (2011):

Theorem 1. *A necessary and sufficient condition for the solution to the graphical lasso to be block-diagonal with blocks C_1, C_2, \dots, C_K is that $|s_{ij}| \leq \lambda$ for all $i \in C_k, j \in C_{k'}, i \neq j, k \neq k'$.*

To prove this statement, we consider the optimality conditions given in (3.28).

Necessary condition. We assume that the solution to the Graphical lasso is block-diagonal. Since Θ is block-diagonal, Θ^{-1} will be block-diagonal with the same diagonal blocks. By the optimality conditions we have

$$\Theta^{-1} = \mathbf{S} + \lambda \cdot \text{Sign}(\Theta). \quad (3.39)$$

Hence, $s_{ij} + \lambda \cdot \text{Sign}(\theta_{ij}) = 0$ whenever $i \in C_k, j \in C_{k'},$ with $i \neq j, k \neq k'$. Since $-1 \leq \text{Sign}(\theta_{ij}) \leq 1$, it can be seen that $|s_{ij}| \leq \lambda$.

Sufficient condition. We assume that $|s_{ij}| \leq \lambda$ for all $i \in C_k, j \in C_{k'},$ with $i \neq j, k \neq k'$. We also assume without loss of generality, that the covariance matrix \mathbf{S} has been ordered according to the clusters $C_k: k = 1, 2, \dots, K$. Hence, we need to show that the optimality conditions can be satisfied by a matrix of the form:

$$\tilde{\Theta} = \begin{pmatrix} \Theta_1 & & \\ & \ddots & \\ & & \Theta_K \end{pmatrix}.$$

First, consider $s_{ij} + \lambda \cdot \text{Sign}(\tilde{\Theta}_{ij})$ for all $i \in C_k, j \in C_{k'},$ with $i \neq j, k \neq k'$. Since $\tilde{\Theta}_{ij} = 0$ for this case, we can set $\text{Sign}(\tilde{\Theta}_{ij})$ anywhere in $[-1, 1]$. Furthermore, since $|s_{ij}| \leq \lambda$, we can set

$\text{Sign}(\tilde{\Theta}_{ij}) = \frac{-s_{ij}}{\lambda}$. Hence, $s_{ij} + \lambda \cdot \text{Sign}(\tilde{\Theta}_{ij}) = 0$. For this selection we see that $\mathbf{S} + \lambda \cdot \text{Sign}(\tilde{\Theta})$ is

also block-diagonal, and the optimality conditions reduce to:

$$\tilde{\Theta}_k^{-1} = \mathbf{S}_k + \lambda \cdot \text{Sign}(\tilde{\Theta}_k), \quad (3.40)$$

for $k = 1, 2, \dots, K$, where \mathbf{S}_k are the cluster specific covariance matrices. Each $\tilde{\Theta}_k$ can be solved by applying the standard graphical lasso algorithm. This implies that by screening the off-diagonal elements in a given column of \mathbf{S} , it can be determined whether or not the corresponding node in the solution to the graphical lasso problem is disconnected from all other nodes.

Based on the above findings, Witten et al. (2011) propose two new algorithms for faster computation. These algorithms are presented in Algorithm 2 and Algorithm 3. With reference to Algorithm 2, it can be observed that its implementation is simplistic, i.e., set all the off-diagonal elements in \mathbf{S} for which $|s_{ij}| \leq \lambda$, equal to zero. The ordering of the columns can be done by extracting all the columns and rows for which all but the diagonal elements in \mathbf{S} are zero. The

Algorithm 2: Faster graphical lasso 1

Step 1

Identify the fully unconnected nodes in the Graphical lasso solution. That is, all nodes i such that $|s_{ij}| \leq \lambda$ for all $i \neq j$.

Step 2

Order the features such that the q fully unconnected features precede the $p-q$ remaining features.

Step 3

The solution to the Graphical lasso problem has the form,

$$\Theta = \begin{pmatrix} \frac{1}{s_{11} + \lambda} & & & \\ & \ddots & & \\ & & \frac{1}{s_{qq} + \lambda} & \\ & & & \Theta_{q+1} \end{pmatrix},$$

where Θ_{q+1} solves the graphical lasso problem applied to the sub-matrix of \mathbf{S} containing the features that are not fully disconnected from the other features.

graphical lasso can then be applied on the remaining columns. On the extracted columns the entries in the precision matrix will be $1/s_{ii}$, as described in Step 3 of Algorithm 2.

Algorithm 3 is more complex than Algorithm 2. In Step 1, an adjacency matrix is created by indicating which off-diagonal elements in \mathbf{S} are less than $|\lambda|$. Steps 2 and 3 entail ordering the adjacency matrix in a manner where the order of the variables have sets of features that are connected. The method to obtain this ordering is not mentioned or provided by Witten et al. (2011). Our approach to ordering the variables proposed herein is the popular depth-first search algorithm.

Algorithm 3: Faster graphical lasso 2**Step 1**

Let \mathbf{A} denote a $p \times p$ matrix with off-diagonal elements \mathbf{A}_{ij} equal to 1 if $|s_{ij}| > \lambda$ else 0, and the diagonal elements of \mathbf{A} all equal to 1.

Step 2

Identify the $K \geq 1$ connected components of the graph for which \mathbf{A} is the adjacency matrix. For each $k = 1, \dots, K$, let C_k denote the set of indices of the features in the k th connected component.

Step 3

Assume that the features are ordered such that if $i \in C_k$, $j \in C_{k'}$, with $k < k'$, then $i < j$.

Step 4

The solution to the graphical lasso problem takes the form,

$$\Theta = \begin{pmatrix} \Theta_1 & & & \\ & \Theta_2 & & \\ & & \ddots & \\ & & & \Theta_K \end{pmatrix},$$

where Θ_k is the solution to the graphical lasso problem applied only to the (square) symmetric submatrix of \mathbf{S} consisting of the features whose indices are in C_k . If C_k consists

of one variable, then that variable is disconnected from all others and $\Theta_k = \frac{1}{s_{ii} + \lambda}$.

When comparing Algorithms 2 and 3, Witten et al. (2011) expect Algorithm 3 to be the fastest, since it exploits all block-diagonal structures in the graphical lasso, whereas Algorithm 2 only exploits the fully unconnected nodes. Hence, only if the solution to the optimisation problem is a fully unconnected graph will Algorithm 2 be faster than Algorithm 3.

To explore Algorithms 1, 2 and 3, the three algorithms were coded in *R* (available in Appendix A) and implemented on the flow-cytometry dataset, which Hastie et al. (2009) used to illustrate four different graphical lasso solutions. The flow-cytometry dataset consists of 11 variables which were measured on 7466 cells. This dataset is available from <https://web.stanford.edu/~hastie/ElemStatLearn/data.html>. For more information relating to this dataset, the reader is referred to Sachs et al. (2005). Hastie et al. (2009) applied the graphical lasso algorithm using four different values for the tuning parameter. These were 0, 7, 27 and 36. Here, the same values were adopted for active comparison purposes. In addition the estimated precision matrix for the graphical lasso algorithm is given when $\lambda = 52$. The choice of $\lambda = 52$ was selected since this value renders the solution block-diagonal for Algorithm 3. The results of the three algorithms on the flow-cytometry dataset are presented next.

With reference to Figure 3.3, the graphs produced by each algorithm are the same. It is also noted that the results agree with those of Hastie et al. (2009). Next, the estimated precision matrix is given in the case of $\lambda = 52$. With reference to Table 3.1 the results for the three algorithms are identical. Thus, it can be concluded that our understanding of these algorithms is sufficient. Up to this point, the approach to estimate the precision matrix has been investigated. In the next section, the complexity parameter λ is assessed. This section also includes a review of methods for estimating λ , which is required when implementing the graphical lasso algorithm.

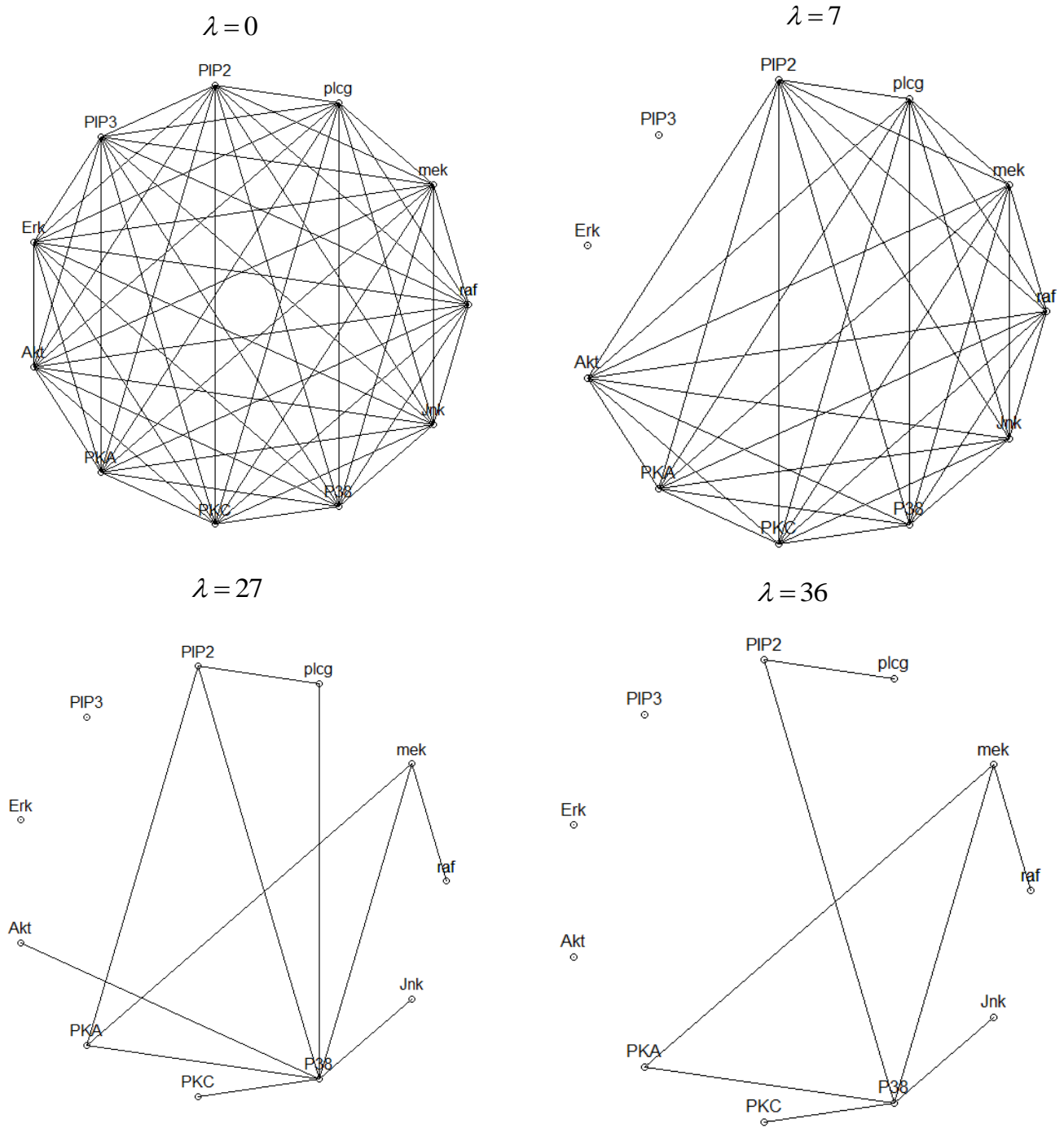


Figure 3.3: Algorithm 1, 2 and 3 applied to the flow-cytometry data.

Table 3.1: Precision matrix obtained by applying algorithm 1, 2 and 3 to the flow-cytometry data with $\lambda = 52$

	raf	PIP2	PKA	plcg	mek	PIP3	Erk	Akt	PKC	P38	Jnk
raf	0.0095				0.0020						
PIP2		0.0071									
PKA			0.0021								
plcg				0.0122							
mek	0.0020				0.0056						
PIP3						0.0186					
Erk							0.0185				
Akt								0.0141			
PKC									0.0165		
P38										0.0035	0.0012
Jnk										0.0012	0.0105

3.4.5 Selecting the tuning parameter for the graphical lasso

In this section, three methods for selecting the tuning parameter are assessed. There are broadly two method types for consideration when selecting the tuning parameter. These are *criterion-based* and *cross-validation-based*. In the following sections, two criterion-based methods, and one cross-validation-based method, are discussed.

3.4.5.1 Criterion based methods

The first criterion-based method is the *extended Bayesian information criterion (eBIC)*, first introduced by Chen and Chen (2008). To understand its origin, the ordinary *Bayesian information criterion (BIC)* needs to be considered first. Mathematically the BIC is defined as

$$BIC = -2 \cdot \log \text{lik} + d \cdot \log(n), \quad (3.41)$$

where d is the degrees of freedom (the effective number of free parameters) of the model, and $\log \text{lik}$ is the evaluated log-likelihood function on the fitted model, where model refers to the estimated precision matrix. The tuning parameter is selected as the value that minimises the BIC (Hastie et al., 2009). For the Gaussian graphical model, the degrees of freedom are calculated as $m/2 + p$, where m denotes the number of non-zero off-diagonal elements in the precision matrix, and p is the number of variables (Liu and Wasserman, 2010). Justification for the BIC criterion assumes that the dimensions are fixed as the sample size increases. When the number of dimensions is greater than the number of observations, this justification does not hold. It has

been observed that when the number of variables is large relative to the sample size, the BIC performs poorly (Chen and Chen, 2012). Chen and Chen (2008) suggest using the eBIC criterion when the model space is large. Its application to Gaussian graphical models is given in Foygel and Drton (2010). In the paper by Foygel and Drton (2010), the extended BIC criterion is given by:

$$eBIC = -2 \cdot \log \text{lik} + d \cdot \log(n) + 4 \cdot d \cdot \tau \log(p). \quad (3.42)$$

When comparing Equation (3.41) to Equation (3.42), it is noted that the only difference between the two criteria is the third term in (3.42). Here, τ is an additional parameter with $\tau \in [0, 1]$. If τ is set to zero, the original BIC is obtained. With reference to (3.42), positive values for τ penalises large graphs (i.e., graphs with many edges) more, which has been numerically shown to improve graph inference in high-dimensional settings. Chen and Chen (2008), and Foygel and Drton (2010), both found that the eBIC performs relatively well when $\tau = 0.5$.

The second criterion-based method uses the so-called *Stability Approach to Regularization Selection* (StARS). The StARS approach was first introduced by Liu et al. (2010). The fundamental concept of StARS is to select the least amount of regularisation that will reproduce the same graph under resampling. Let b denote the size of each sample ($1 < b < n$). Draw B samples of size b without replacement from the original data. Theoretically, it would be ideal to use all possible subsamples; however, adopting a large number is sufficient. For a specified grid of the regularisation parameter, the graphical lasso algorithm is fitted to each subsample over the grid of values. For a given value of the parameter, the fraction of times the graphs disagree on the presence of an edge is obtained. This fraction is used as a measure of instability of the edge across the subsamples. The total instability is calculated by averaging over all edges. The tuning parameter is then selected as the least amount of regularisation required for the total instability to be less than ε . The authors of StARS suggest setting $\varepsilon = 0.05$.

3.4.5.2 Cross-validation-based method

A cross-validation-based method was adopted by Friedman et al. (2008) to select λ . Their approach is referred to as the “*Regression*” approach. The method is as follows. First split the data into k folds. For each fold, fit the Graphical lasso algorithm on the remaining $k - 1$ folds. Recall the relationship between the precision matrix and the least squares coefficients from (3.20). The precision matrix can thus be used as a set of linear regression models to predict each variable. Using the left-out set, the squared prediction errors of the linear regression models for all the variables are calculated. The prediction errors are then averaged over all the variables, giving an average prediction error for the fold. Next, the average prediction error over the folds

are averaged, and the tuning parameter value that minimises this average error, is selected as the optimal value for the tuning parameter.

3.5 Summary

This chapter introduced the reader to the basic methodology needed when studying probabilistic graphical models, focussing on undirected graphs. Three popular statistical relationships between random variables that can be depicted in undirected graphs, were reviewed. These were marginal correlations, partial correlations and conditional independence. Conditional independence is the most popular relationship depicted in undirected graphs. For graphs representing conditional independence it is usually assumed that the graph of the probability model is a pairwise Markov graph, since this simplifies inference. For continuous random variables, it is commonly assumed that the probability model is a multivariate Gaussian distribution, when working with probabilistic graphical models. The multivariate Gaussian distribution has the property whereby its conditional independencies can be represented by a pairwise Markov graph, and the conditional independencies are captured in the precision matrix of the distribution. In addition, the off-diagonal entries in the precision matrix are proportional to the linear regression coefficients obtained by regressing each variable on the remaining variables in the distribution. In the literature, two popular methods for estimating the precision matrix can be found. These are the MLE and the L1 penalised estimate. The L1 penalised estimate is the preferred estimate, since it leads to sparser graphs, by shrinking the elements in the precision matrix to zero. The graphical lasso algorithm is used to obtain the L1 penalised estimate. By applying screening to the sample covariance matrix, improvements in the computational speed of the graphical lasso algorithm are obtained. When using the L1 penalised estimate, a tuning parameter needs to be selected. Three popular methods for selecting the tuning parameter are the eBIC criterion, the StARS criterion and cross-validation. In the next chapter we combine the concepts and methodology presented in this chapter in order to develop a novel statistical learning technique.

CHAPTER 4

DISCRIMINANT ANALYSIS USING SPARSE GRAPHICAL MODELS

4.1 Introduction

In the previous chapter an approach for estimating a sparse precision matrix for a normal distribution was discussed. This was achieved by altering maximum likelihood estimation to impose an L1 penalty on the likelihood function. The solution to penalised likelihood estimation can be obtained using the graphical lasso algorithm. After the precision matrix or covariance matrix has been estimated, the stochastic model can be used for inference by means of constructing graphs and calculating probabilities.

In addition, multiple traditional statistical learning techniques assume the data to follow a multivariate Gaussian distribution. For example, a popular statistical learning method that assumes class conditional densities are multivariate Gaussian densities is QDA. In QDA we need to estimate a precision matrix for each of the classes. We can therefore combine L1 regularisation with QDA, by penalising each of the precision matrices to be estimated. We label this method graphical lasso quadratic discriminant analysis (glQDA). This combination could potentially improve the prediction accuracy when using QDA as a classification model.

In statistical learning, the assumption of normality can often be too rigid. A nonparanormal model provides a means of relaxing the assumption of normality. We propose using the nonparanormal model to extend glQDA by relaxing the assumption of normality. This is done through the use of a transformation and application of the graphical lasso in the transformed space. We label this novel learning technique graphical lasso nonparanormal discriminant analysis (gINPDA).

The aim of this chapter is to establish a theoretical understanding of gINPDA, as well as clarification of the motivation behind its development. To achieve this goal, Section 4.2 investigates combining QDA and the graphical lasso. In Section 4.3, the approach for relaxing the assumption of normality is then considered, along with a description of gINPDA.

4.2 QDA and The Graphical Lasso

Recall from Equation (2.9) that the discriminant function for QDA is given by

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \log |\boldsymbol{\Sigma}_k| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) + \log(\boldsymbol{\pi}_k), \quad k = 1, \dots, K. \quad (4.1)$$

This discriminant function can be rewritten in a form containing the precision matrix,

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \log |\boldsymbol{\Theta}_k^{-1}| - \frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Theta}_k (\mathbf{x} - \boldsymbol{\mu}_k) + \log(\boldsymbol{\pi}_k), \quad k = 1, \dots, K. \quad (4.2)$$

From (4.2) it can be seen that the mean vector and precision matrix for each class needs to be estimated. Traditionally, the precision matrix is estimated using MLE. In Chapter 3 we assessed an alternative for estimating the precision matrix, viz. the L1 penalised maximum likelihood estimate. In Equation (2.17) the likelihood function for QDA was presented. By adding a penalty on the precision matrix for each of the classes, the log likelihood function for glQDA in terms of the precision matrices becomes,

$$l = \sum_{k=1}^K N_k \log(\pi_k) + \sum_{k=1}^K \left\{ l_k(\boldsymbol{\mu}_k, \boldsymbol{\Theta}_k) - \lambda_k \|\boldsymbol{\Theta}_k\|_1 \right\}. \quad (4.3)$$

The second term in (4.3) is the sum of the class penalised likelihoods. The likelihood function can thus be optimised by maximising the penalised likelihood for each class.

When using the graphical lasso to estimate the precision matrix for the QDA model, the tuning parameter needs to be estimated for each class. In Chapter 3 we investigated three different methods for selecting the tuning parameter. We want to select the tuning parameters, $\lambda_1, \dots, \lambda_K$, such that the generalisation performance of glQDA is optimal in terms of 0-1 loss. The first approach considers using cross-validation over a search of m possible values for each of the parameters. This would entail evaluating m^K models over all of the folds which is infeasible for moderately sized K . To overcome this, one approach is to set $\lambda = \lambda_1 = \lambda_2 = \dots = \lambda_K$ in (4.3) and perform cross-validation over λ only. The second approach, which is a more sophisticated approach, is to view the glQDA as a density estimation problem. In this approach, we try to find the tuning parameter yielding the best density estimate for each class separately. This allows us to apply criteria such as eBIC and StARS separately to each class. The third approach can be seen as a hybrid approach of the above two methods. Note that the eBIC and StARS each have a single tuning parameter. In this hybrid approach, eBIC (say) is still applied to each class density estimate separately; however, the tuning parameter is chosen through cross-validation.

4.3 The Non-Normal Case

Thus far it was assumed that the class conditional densities are multivariate Gaussian densities. In this section, two methods which can be used to relax this assumption are assessed. The first method entails transforming the observations to follow a normal distribution, while the second makes use of a nonparanormal model.

4.3.1 Transforming multivariate observations

A popular method for transforming observations is by the use of power transformations. Power transformations are only defined for positive observations. However, by adding a positive constant to the observations, the data can be made positive. For our purposes, the power transformation

will be indexed by the parameter α . Different values for α lead to different transformations. The transformed observation will be denoted by x^α , where x is the observation. Some popular transformations include setting $\alpha = -1, 0, 0.5$ or 2 , resulting in the transformations, $1/x$, $\ln(x)$, \sqrt{x} and x^2 . Choosing $\alpha < 1$ will result in large values of x shrinking more than smaller values (which is useful if the distribution is skewed to the right), while choosing $\alpha > 1$ will result in large values of x increasing more than small ones. The problem with this approach is that it can be tedious to identify appropriate transformations. A convenient method for choosing power transformations is the well-known Box-Cox method. Let $\alpha_1, \dots, \alpha_p$ denote power transformations for variables X_1, \dots, X_p . Each α_k is then selected by maximising

$$l_k(\alpha) = -\frac{n}{2} \ln \left[\frac{1}{n} \sum_{j=1}^n \left(x_{jk}^{(\alpha_k)} - \bar{x}_k^{(\alpha_k)} \right)^2 \right] + (\alpha_k - 1) \sum_{j=1}^n \ln(x_{jk}), \quad (4.4)$$

where j refers to the observation index, and k denotes the variable index. That is,

$$\bar{x}_k^{(\alpha_k)} = \frac{1}{n} \sum_{j=1}^n x_{jk}^{(\alpha_k)} \quad \text{and} \quad x_{jk}^{(\alpha_k)} = \begin{cases} \frac{x_{jk}^{\alpha_k} - 1}{\alpha_k} & \text{if } \alpha_k \neq 0 \quad \text{and} \\ \ln(x_{jk}) & \text{if } \alpha_k = 0 \end{cases}.$$

The procedure is equivalent to adjusting each marginal distribution to be approximately normal. However, this does not guarantee that the joint distribution will be normal. An alternative method finds $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_p]$, that collectively maximises the function:

$$l(\alpha_1, \alpha_2, \dots, \alpha_p) = -\frac{n}{2} \ln |\mathbf{S}(\boldsymbol{\alpha})| + (\alpha_1 - 1) \sum_{j=1}^n \ln(x_{j1}) + \dots + (\alpha_p - 1) \sum_{j=1}^n \ln(x_{jp}), \quad (4.5)$$

where $\mathbf{S}(\boldsymbol{\alpha})$ is the sample covariance matrix of the transformed observations. The procedure usually offers a significant improvement over the previous method. It is interesting to note that apart from the constant, the first term in the univariate case is the normal log likelihood, after maximising with respect to the mean and variance parameters. The multivariate case is the multivariate log-likelihood after maximising with respect to the mean vector and covariance matrix (Johnson and Wichern, 2007). To use the transformed data in QDA the transformation should be the same over all classes. If the transformations are unique to each class, the density needs to be determined for each class on the original scale. The reason being, we do not know the class label for test data and thus we do not know which transformation to apply. This makes using these transformations rather difficult. The nonparanormal does not suffer from this problem since we can determine the class conditional densities on the original scale.

4.3.2 The nonparanormal model

The nonparanormal model is a semiparametric model. The basic concept is to estimate the marginal distributions of each variable in a nonparametric way, and then combine them into a parametric model. Our discussion of the theory underlying the nonparanormal model closely follows that of Lafferty et al. (2012).

Let $\mathbf{X} = (X_1, \dots, X_p)^T$ be a set of random variables. The random vector $\mathbf{X} = (X_1, \dots, X_p)^T$ has a nonparanormal distribution, written as $\mathbf{X} \sim NPN(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \mathbf{g})$, if there exist functions g_j , $j = 1, \dots, p$, such that:

$$\mathbf{g}(\mathbf{X}) \equiv (g_1(X_1), \dots, g_p(X_p)) \sim N_p(\boldsymbol{\mu}, \boldsymbol{\Sigma}). \quad (4.6)$$

When the functions g_j are monotone increasing and differentiable, the joint density of \mathbf{X} is given by:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{g}(\mathbf{x}) - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{g}(\mathbf{x}) - \boldsymbol{\mu})\right) \prod_{j=1}^p |g'_j(x_j)|. \quad (4.7)$$

With reference to the density in (4.7), it can be seen that $g_j(\cdot)$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}^{-1}$, are the parameters of the density that need to be estimated. If each of the functions $g_j(\cdot)$ is scaled by a constant, scaling the diagonals of $\boldsymbol{\Sigma}$, or $\boldsymbol{\mu}$ in a similar way, will not change the density. The density is thus not *identifiable*. To make the density identifiable, Lafferty et al. (2012), suggest that the functions g_j preserve the means and variance of the variables, i.e.

$$E(X_j) = \mu_j = E(g_j(X_j)) \text{ and } \text{var}(X_j) = \sigma_j^2 = \text{var}(g_j(X_j)).$$

Furthermore, considering the density in (4.7) it is apparent that the nonparanormal density is a pairwise Markov graph with potential functions given by,

$$\psi_{ij}(x_i; x_j) = \exp(-g_i(x_i) \theta_{ij} g_j(x_j)) \quad (4.8)$$

and

$$\psi_i(x_i) = \exp\left(-\frac{1}{2} \theta_{ii} g_i(x_i)^2\right) |g'_i(x_i)|. \quad (4.9)$$

In addition, the normalising constant is given by

$$Z = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}}. \quad (4.10)$$

This indicates that the conditional independencies between the variables are captured in the precision matrix, as in the case of the normal distribution.

Since the joint distribution of $\mathbf{g}(\mathbf{X})$ is multivariate normal, the marginal distributions of $g_j(X_j)$ will be normal, i.e. $N(\mu_j, \sigma_j^2)$. Letting $F_j(X_j)$ denote the marginal cumulative distribution function (CDF) of variable j , then:

$$F_j(x_j) = P(X_j \leq x_j) = P(g_j(X_j) \leq g_j(x_j)) = \Phi\left(\frac{g_j(x_j) - \mu_j}{\sigma_j}\right), \quad (4.11)$$

where Φ is the standard normal distribution function. From (4.11) it follows that

$$g_j(x_j) = \Phi^{-1}(F_j(x_j)) \cdot \sigma_j + \mu_j, \quad (4.12)$$

where Φ^{-1} is the inverse of the standard normal distribution function.

In probability theory, any joint distribution function can be separated into two parts; marginal distributions, and the dependence structure between the random variables. The function describing the dependence structure is known as a *copula* and was first introduced by Sklar (1959). It has been shown that the nonparanormal model has a strong connection with the Gaussian copula. As in Lafferty et al. (2012), assume that each X_j has a defined continuous CDF, viz. $F_j(x_j) = P(X_j \leq x_j)$. By applying the popular probability integral transformation, it is known that each $F_j(X_j)$ follows a uniform distribution on the interval $[0,1]$, denoted by U_j . The copula C of (X_1, \dots, X_p) is the joint CDF of (U_1, \dots, U_p) , i.e.

$$\begin{aligned} C(u_1, u_2, \dots, u_p) &= P(U_1 \leq u_1, U_2 \leq u_2, \dots, U_p \leq u_p) \\ &= P(U_1 \leq F_1(X_1), U_2 \leq F_2(X_2), \dots, U_p \leq F_p(X_p)) \\ &= P(X_1 \leq F_1^{-1}(u_1), X_2 \leq F_2^{-1}(u_2), \dots, X_p \leq F_p^{-1}(u_p)). \end{aligned} \quad (4.13)$$

The copula function thus contains the information pertaining to the dependency structure between the variables. According to Sklar's theorem, any joint distribution can be written as

$$F(x_1, \dots, x_p) = C(F_1(x_1), \dots, F_p(x_p)). \quad (4.14)$$

If c is the density function of copula C , then the joint density function of the random variables X_1, \dots, X_p , can be written as,

$$f(x_1, \dots, x_p) = c(F_1(x_1), \dots, F_p(x_p)) \prod_{j=1}^p f_j(x_j), \quad (4.15)$$

where $f_j(x_j)$ is the marginal density of X_j . The Gaussian copula density is given by:

$$\frac{1}{|\mathbf{R}|^{1/2}} \exp\left\{-\frac{1}{2} \Phi^{-1}(F(\mathbf{x}))^T (\mathbf{R}^{-1} - \mathbf{I}) \Phi^{-1}(F(\mathbf{x}))\right\}, \quad (4.16)$$

where $\Phi^{-1}(F(\mathbf{x})) = (\Phi^{-1}(F_1(x_1)), \Phi^{-1}(F_2(x_2)), \dots, \Phi^{-1}(F_p(x_p)))$, and \mathbf{R} is the correlation matrix. The Gaussian copula has played a significant role in the financial risk market and some might claim that its misuse played a significant role in the 2007 financial crisis (MacKenzie and Spears, 2014).

The density function of the random variables whose distribution function can be described by the Gaussian copula, is given by,

$$f(x_1, \dots, x_p) = \frac{1}{|\mathbf{R}|^{1/2}} \exp\left\{-\frac{1}{2} \Phi^{-1}(F(\mathbf{x}))^T (\mathbf{R}^{-1} - \mathbf{I}) \Phi^{-1}(F(\mathbf{x}))\right\} \prod_{j=1}^p f_j(x_j), \quad (4.17)$$

which is equivalent to,

$$f(x_1, \dots, x_p) = \frac{1}{|\mathbf{R}|^{1/2}} \exp\left\{-\frac{1}{2} \Phi^{-1}(F(\mathbf{x}))^T (\mathbf{R}^{-1} - \mathbf{I}) \Phi^{-1}(F(\mathbf{x}))\right\} \prod_{j=1}^p F'_j(x_j), \quad (4.18)$$

if it can be proved that (4.18) is equivalent to (4.7) with transformation functions as denoted in (4.11). Subsequently it is proven that the nonparanormal model is equivalent to the Gaussian copula with marginals given in (4.11).

Substituting (4.11) into (4.7), and rewriting this in terms of its correlation matrix, gives

$$f(x_1, \dots, x_p) = \frac{1}{(2\pi)^{p/2} |\mathbf{R}|^{1/2} \prod_{j=1}^p \sigma_j} \exp\left(-\frac{1}{2} \Phi^{-1}(F(\mathbf{x}))^T (\mathbf{R}^{-1} - \mathbf{I}) \Phi^{-1}(F(\mathbf{x}))\right) \times \prod_{j=1}^p \frac{\partial \Phi^{-1}(F_j(x_j)) \cdot \sigma_j}{\partial x_j}, \quad (4.19)$$

which can be written as

$$f(x_1, \dots, x_p) = \frac{1}{|\mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2} \mathbf{\Phi}^{-1}(F(\mathbf{x}))^T (\mathbf{R}^{-1} - \mathbf{I}) \mathbf{\Phi}^{-1}(F(\mathbf{x}))\right) \times \prod_{j=1}^p \frac{\phi(\mathbf{\Phi}^{-1}(F_j(x_j)))}{\sigma_j} \times \frac{\partial \mathbf{\Phi}^{-1}(F_j(x_j)) \cdot \sigma_j}{\partial x_j}. \quad (4.20)$$

Since $\frac{\partial \mathbf{\Phi}^{-1}(u)}{\partial u} = \frac{1}{\phi(\mathbf{\Phi}^{-1}(u))}$, (4.20) becomes,

$$f(x_1, \dots, x_p) = \frac{1}{|\mathbf{R}|^{1/2}} \exp\left\{-\frac{1}{2} \mathbf{\Phi}^{-1}(F(\mathbf{x}))^T (\mathbf{R}^{-1} - \mathbf{I}) \mathbf{\Phi}^{-1}(F(\mathbf{x}))\right\} \prod_{j=1}^p F'_j(x_j), \quad (4.21)$$

which is identical to the Gaussian copula. This indicates that the nonparanormal distribution, with the transformation function given in (4.7), is equivalent to the Gaussian copula.

Next, the proposed methods for estimation of the parameters of the nonparanormal model are considered.

4.3.3 Estimation

For the nonparanormal model, $g(\cdot)$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}^{-1}$, are the parameters of the density which need to be estimated. Lafferty et al. (2012) suggest estimating the functions $g_j(\cdot)$ by

$$\tilde{g}_j(x) = \mathbf{\Phi}^{-1}(\tilde{F}_j(x)) \cdot \hat{\sigma}_j + \hat{\mu}_j, \quad (4.22)$$

where $\tilde{F}_j(x_j)$ is the truncated estimator for the empirical distribution function, $\hat{\mu}_j$ is the sample mean, and $\hat{\sigma}_j$ is the sample standard deviation, $j = 1, \dots, p$. More formally,

$$\hat{\mu}_j = \frac{1}{n} \sum_{i=1}^n x_{ij}, \quad \hat{\sigma}_j = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \hat{\mu}_j)^2} \quad \text{and} \quad \tilde{F}_j(x) = \begin{cases} \delta_n, & \text{if } \hat{F}_j(x) < \delta_n \\ \hat{F}_j(x), & \text{if } \delta_n \leq \hat{F}_j(x) \leq 1 - \delta_n \\ 1 - \delta_n, & \text{if } \hat{F}_j(x) > 1 - \delta_n \end{cases}, \quad (4.23)$$

where $\hat{F}_j(x)$ is the empirical distribution function given by,

$$\hat{F}_j(t) \equiv \frac{1}{n} \sum_{i=1}^n 1_{\{x_{ij} \leq t\}}, \quad (4.24)$$

and δ_n is the truncation parameter. The truncated estimator has an advantage in settings where the number of observations is small relative to the number of variables. In a sense, it restricts the complexity of the empirical distribution function. It reduces the variance of the estimated distribution function, but at the cost of an increase in bias. For example, in the extreme case where $\delta_n = 1$, the truncated empirical distribution function will be one for all values of x . This will be the case for every sample, resulting in zero variance but high bias.

Lafferty et al. (2012) suggest setting the truncation parameter as

$$\delta_n = \frac{1}{4n^{1/4} \sqrt{\pi \log n}}. \quad (4.25)$$

For estimating the inverse covariance matrix Σ^{-1} , or precision matrix Θ , Lafferty et al. (2012) suggest using the graphical lasso algorithm. Let $S_n(\tilde{g})$ denote the sample covariance matrix of $\tilde{g}_1(X_1), \dots, \tilde{g}_p(X_p)$. The L1 regularised estimator is the precision matrix Θ that maximises $\log(\det(\Theta)) - \text{tr}(S_n(\tilde{g})\Theta) - \lambda \|\Theta\|_1$. The solution to Θ can then be found using the graphical lasso algorithm and the tuning parameter can be selected as discussed in Chapter 3. Note that we did not enforce the variances that can be derived from Θ to be equal to the value of $\hat{\sigma}_j$, although we expect this to be approximately true.

With reference to the above estimation procedure, it can be observed that the marginal CDFs are discrete. This prevents estimation of the nonparanormal density function explicitly, since it requires calculating the derivative of $g_j(x)$. In (4.22), $g_j(x)$ is estimated by $\tilde{g}_j(x)$, and taking the derivative of $\tilde{g}_j(x)$ gives,

$$\tilde{g}'_j(x) = \frac{1}{\phi(\Phi^{-1}(\tilde{F}_j(x)))} \cdot \hat{\sigma}_j \cdot \frac{\partial \tilde{F}_j(x)}{\partial x}. \quad (4.26)$$

Thus it can be seen that the derivative of $\tilde{g}_j(x)$ requires the derivative of the empirical distribution function, which is a discrete function. This could be problematic if the nonparanormal density is adopted into the Bayes classifier of (3.5) and is used as a discriminant function. For this reason, an alternative approach for estimating the empirical distribution function is evaluated. The suggested approach is Gaussian kernel density estimation. The reason for considering this approach will become clear in the next section.

4.3.4 Kernel density estimation

Following the discussion of Hastie et al. (2009), kernel density estimation entails estimating a density function in a nonparametric manner. Let x_1, \dots, x_n denote a sample of n observations, drawn from a density function $f(x)$. A natural estimate for the density function at x_0 , given x_1, \dots, x_n , is

$$\hat{f}(x_0) = \frac{\#x_i \in N(x_0)}{n \cdot \omega}, \quad (4.27)$$

where $N(x_0)$ is a neighbourhood around x_0 , and ω is the width of the neighbourhood. The natural estimate thus estimates the density at x_0 by calculating the number of observations within a window of x_0 , and dividing it by the average number of observations in a window of size ω . The issue with this method is that it results in a bumpy estimate. Figure 4.1 illustrates this.

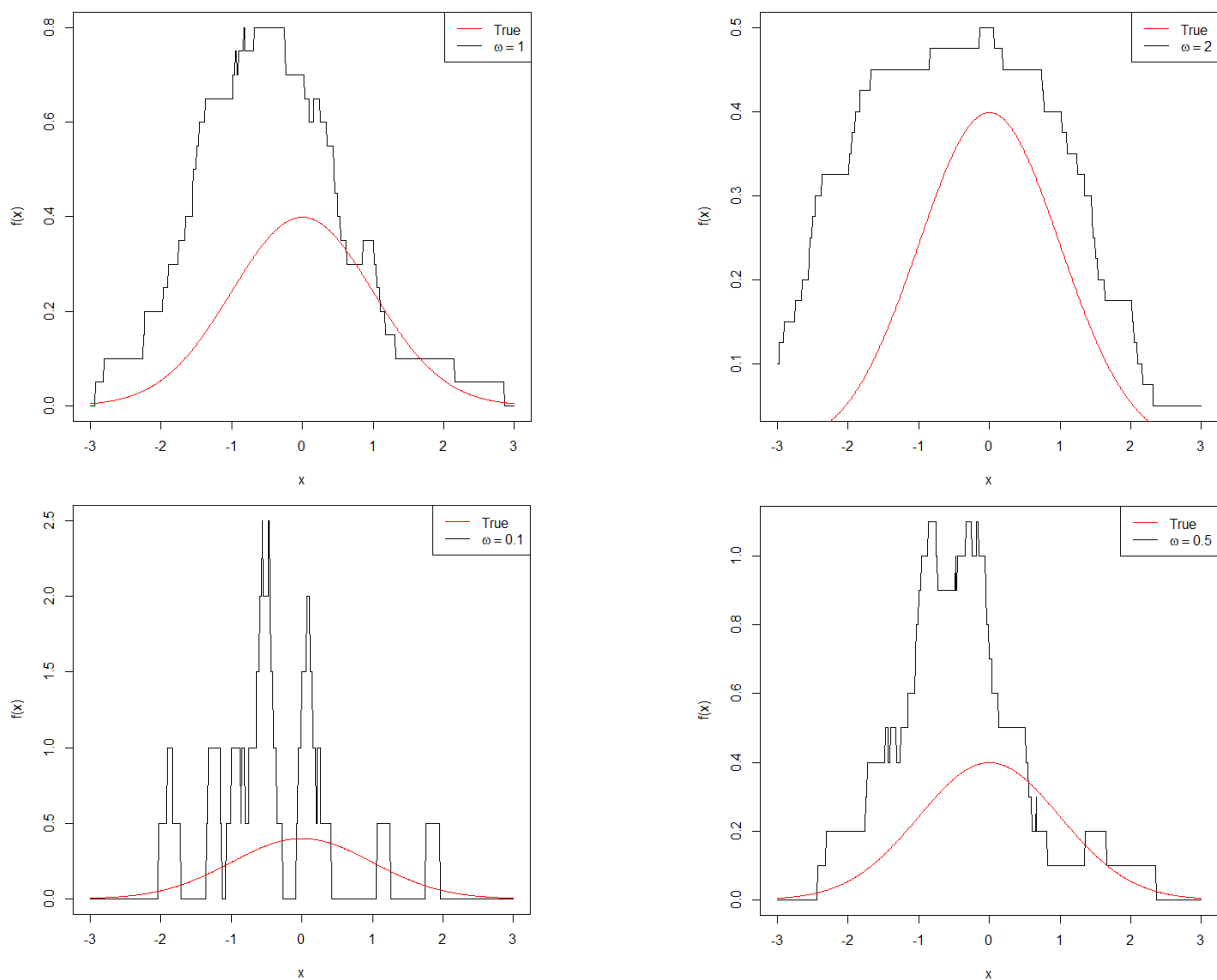


Figure 4.1: Natural density estimate example with bandwidth ω equal to 0.1, 0.5, 1 and 2.

Figure 4.1 was generated by selecting a random sample of size 20 from a standard normal distribution, and selecting the neighbourhood width equal to 0.1, 0.5, 1 and 2 respectively. From Figure 4.1, it can be noted that the estimated density confirms the suspicion that it is indeed bumpy. In addition, it is clear that the natural estimate performs poorly in estimating the true normal density.

A smoother estimate is given by

$$\hat{f}(x_0) = \frac{1}{n\omega} \sum_{i=1}^n K_{\omega}(x_0, x_i), \quad (4.28)$$

where K_{ω} is the kernel function that decreases the weight associated with the observations, the farther away the observations are from x_0 . The Gaussian kernel is given by

$$K_{\omega}(x_0, x) = \phi((x_0 - x) / \omega), \quad (4.29)$$

where ϕ is the standard normal density function. The Gaussian kernel density estimate is then given by

$$\hat{f}(x_0) = \frac{1}{n\omega} \sum_{i=1}^n \phi((x_0 - x_i) / \omega). \quad (4.30)$$

With reference to the density function in (4.30), it can be observed that the kernel weighs observations with a decreasing weight the farther the observation is from x_0 . When comparing (4.30) with Equation (6.23) in Hastie et al. (2009), the latter forgets to divide their representation of (4.30) by the bandwidth ω .

Applying the Gaussian kernel density to the same sample as used in Figure 4.1, Figure 4.2 was generated. The densities in Figure 4.2 are significantly smoother than those of Figure 4.1. For $\omega = 0.5$ and 1, we obtain superior density estimates than those of Figure 4.1. Figure 4.2 also shows that the choice of the bandwidth can have a significant impact on the quality of the estimated density function. Hence, the selection of ω also needs to be addressed. The rule of

thumb suggested by Silverman (1986) is to specify ω as $\left(\frac{4\hat{\sigma}^5}{3n}\right)^{\frac{1}{5}}$. The rule of thumb is the

optimal bandwidth when the true underlying density is Gaussian. Looking at the rule of thumb it can be seen that as the sample size increases the bandwidth gets smaller and when the sample standard deviation increases the bandwidth increases. Intuitively this makes sense.

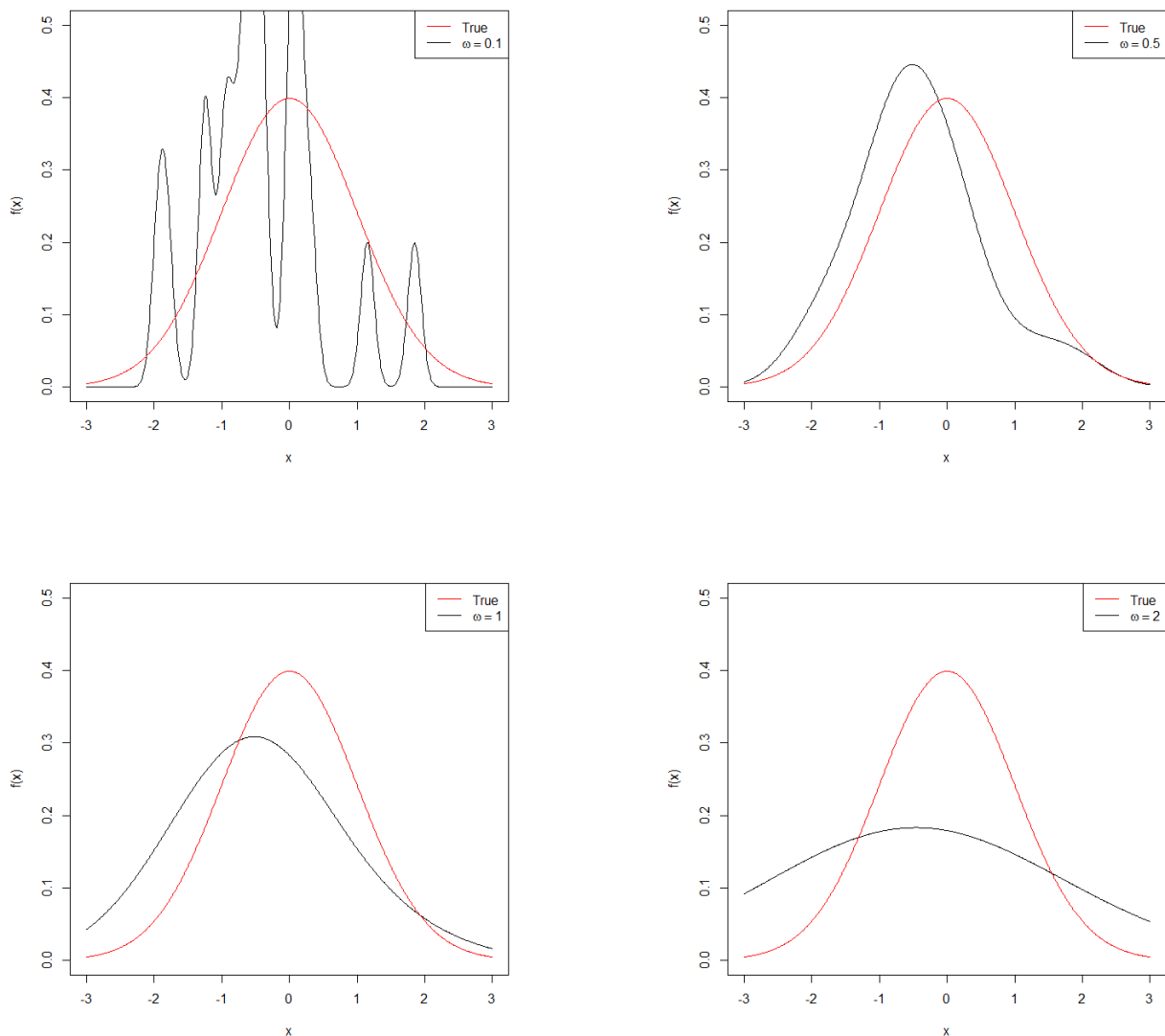


Figure 4.2: Gaussian kernel density estimate with ω equal to 0.1, 0.5, 1 and 2.

Adopting the same random sample used to create Figures 4.1 and 4.2, Silverman's (1986) rule of thumb was used to calculate the bandwidth. The rule of thumb bandwidth for this sample is 0.527. Figure 4.3 represents the estimated density function using the Gaussian kernel density estimate with the bandwidth set to 0.527. From Figure 4.3, it is apparent that the rule of thumb works well, although it is important to note this rule of thumb can often lead to poor results, since it is optimal for Gaussian densities. For this reason, cross-validation is often used to obtain the optimal bandwidth.

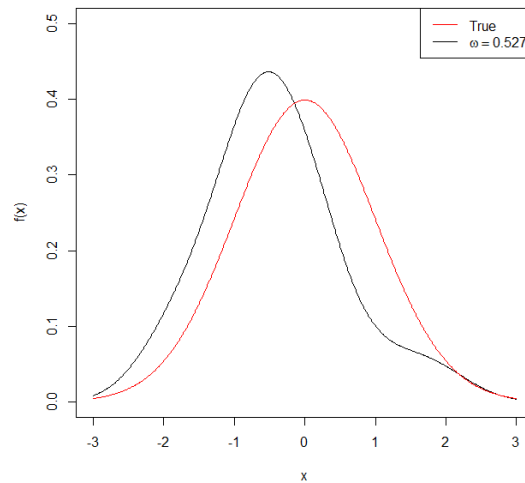


Figure 4.3: Gaussian kernel density estimate with bandwidth set to 0.527.

4.3.5 The graphical lasso nonparanormal model (gINPN)

Since the derivative of the empirical distribution function in (4.26) is estimated using the Gaussian kernel density estimator, it may be preferable to substitute the CDF estimates from (4.24), with the CDF obtained from the Gaussian kernel density estimator. Let $f(x)$ denote a density function. The CDF is given by,

$$F(x) = \int_{-\infty}^x f(t) dt. \quad (4.31)$$

Recall from (4.30) that the Gaussian kernel density estimate is given by

$$\hat{f}(x_0) = \frac{1}{n\omega} \sum_{i=1}^n \phi((x_0 - x_i) / \omega). \quad (4.32)$$

The estimated CDF is then

$$\begin{aligned} \hat{F}(x_0) &= \int_{-\infty}^{x_0} \frac{1}{n\omega} \sum_{i=1}^n \phi((t - x_i) / \omega) dt. \\ &= \frac{1}{n\omega} \sum_{i=1}^n \int_{-\infty}^{x_0} \phi((t - x_i) / \omega) dt \\ &= \frac{1}{n\omega} \sum_{i=1}^n \left[\omega \times \Phi\left(\frac{t - x_i}{\omega}\right) \right]_{-\infty}^{x_0} \end{aligned}$$

$$= \frac{1}{n} \sum_{i=1}^n \Phi \left(\frac{x_0 - x_i}{\omega} \right). \quad (4.33)$$

where $\Phi()$ is the standard normal CDF.

Using the Gaussian kernel density estimate, (4.26) becomes:

$$\tilde{g}_j(x) = \Phi^{-1} \left(\left(\frac{1}{n} \sum_{i=1}^n \Phi \left(\frac{x - x_i}{\omega} \right) \right) \right) \cdot \hat{\sigma}_j + \hat{\mu}_j, \quad (4.34)$$

and the derivative of (4.26) becomes:

$$\tilde{g}'_j(x) = \frac{1}{\phi \left(\Phi^{-1} \left(\frac{1}{n} \sum_{i=1}^n \Phi \left(\frac{x - x_i}{\omega} \right) \right) \right)} \cdot \hat{\sigma}_j \cdot \frac{1}{n\omega} \sum_{i=1}^n \phi \left((x - x_i) / \omega \right). \quad (4.35)$$

To use the nonparanormal model as a classifier, the density of (4.7) is substituted into the optimal Bayes classifier, as given in (2.5). The resulting discriminant function is,

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \log |\mathbf{\Theta}_k| - \frac{1}{2} (\mathbf{g}_k(\mathbf{x}) - \boldsymbol{\mu}_k)^T \mathbf{\Theta}_k (\mathbf{g}_k(\mathbf{x}) - \boldsymbol{\mu}_k) + \log(\pi_k) + \sum_{j=1}^p \log(g'_{kj}(\mathbf{x})), \quad (4.36)$$

where $k = 1, \dots, K$. The precision matrix of the nonparanormal model is then estimated using the graphical lasso algorithm, and the marginal density function and CDF is estimated using the Gaussian kernel density estimator. To the best of this author's knowledge, combining these two techniques into the Bayes classifier is a novel approach. In the next chapter, this method is illustrated empirically.

4.4 Summary

This chapter established a robust theoretical basis and understanding for a proposed novel statistical learning technique. The technique was developed by relaxing the assumption of normality imposed on QDA using a semiparametric approach. The assumption of normality was relaxed by assuming that the class conditional densities follow different nonparanormal distributions. The topology of the graph structure associated with a nonparanormal model is captured by the precision matrix of a transformed version of the associated random vector. The nonparanormal model requires estimating the marginal distribution functions and their densities. The authors of the nonparanormal model suggested using the truncated empirical functions. Here it is proposed that Gaussian kernel density estimation be used, since it has defined derivatives which are required for gINPDA. The gINPDA technique entails using the nonparanormal model in the Bayes classifier and estimating the precision matrices after the class densities have been transformed, using the graphical lasso algorithm. Note that gINPDA has more flexibility than QDA,

making it possible to fit a wider set of functions, but also has the ability to reduce the additional variance associated with such a classifier by means of regularisation of the parameters of the model.

CHAPTER 5

PRACTICAL APPLICATION

5.1 Introduction

In the previous chapter, the use of the graphical lasso to estimate the precision matrix for QDA was investigated. This method was referred to as glQDA. The chapter also included a discussion of a method for extending glQDA to a semiparametric model using the nonparanormal distribution and the proposed novel classifier. This method was referred to as the glNPDA model. The previous chapter also included different approaches for selecting the tuning parameter when implementing the graphical lasso algorithm.

The objective of this chapter is to assess the performance of the glNPDA and glQDA methods. This will be achieved by measuring the performance of each method on three datasets, and comparing the outcomes to a wide range of statistical learning techniques. In addition, the different tuning parameter selection techniques, discussed in Chapters 3 and 4, will also be evaluated using the three datasets.

For evaluation purposes, all methods were implemented using the *R* programming language version 3.5.2. The code was run on an Intel Core i5-2450M central processing unit (CPU) @ 2.50 gigahertz (GHz) processor with 8.00 gigabytes (GB) of installed random-access memory (RAM). Significant use of the *R* package *huge* (High-dimensional Undirected Graph Estimation), version 1.2.7, developed by Zhao et al. (2012), was made for the Graphical lasso application. The *huge* Package has both StARS and eBIC built-in for tuning parameter selection. The package also implements the two screening methods discussed in Chapter 3. The default method uses the screening technique in Chapter 3, Section 3.4.4 (named “Algorithm 2” or “Faster graphical lasso 1”). This method was also used in our empirical work. One concern when using *huge* is that it automatically scales the columns of the input data to unit variance and zero mean, before applying the graphical lasso. The package does not provide the user an option or warning regarding this scaling. For our purpose, the code of *huge* was adjusted so as not to scale the data. All code written and used in this thesis are provided in Appendix A.

This chapter comprises a discussion of the methodology adopted for the practical applications on the three datasets (Section 5.2). Sections 5.3 through 5.5 focus on the application of the different learning techniques on the three datasets considered. The chapter concludes in Section 5.6 with a summary of the results and findings from the empirical studies.

5.2 Methodology

For each practical application, the glQDA and glNPDA models were fitted on a training set, and evaluated on a test (or holdout) set. This is common practice in statistical learning applications, since a model can potentially fit the noise in the sample, which can lead to poor generalisation of the model. For the glQDA model, the precision matrix was estimated for each class using the graphical lasso algorithm, the mean vectors for each class were estimated using the sample mean vectors, and the priors were estimated using the sample class proportions.

For selection of the tuning parameter, eight different methods were considered. The first method sets the tuning parameter equal to zero, thus resulting in the ordinary QDA model. The second, third and fourth methods use the eBIC criterion. The tuning parameter for the eBIC was selected by first setting it equal to 0.5 (its suggested value), followed by selecting the optimal value using cross-validation. Note that the one-standard-error rule was also applied when selecting the optimal tuning parameter value. For the grid of values for the eBIC tuning parameter, we used 100 values, evenly spaced between 0 and 1.

The fifth and sixth methods select the tuning parameter using the StARS criterion. The StARS criterion is computationally expensive, since the graphical lasso is fitted on a large number of samples, drawn with replacement, over a grid of tuning parameter values. The default number of samples used in the *huge* package is 20. Thus, selecting the StARS threshold ε using cross-validation can easily become computationally very expensive. For this reason, the StARS threshold was set equal to the proposed value of 0.05, as well as another default value of 0.1, as suggested by the authors of the *huge* package.

The seventh and eighth methods select the tuning parameter for the graphical lasso using cross-validation. A grid of 100 equally spaced values between 0 and 2 were used, and one tuning parameter is selected for all classes. In addition to this cross-validation approach, again the one-standard-error rule was also applied.

Recall that the estimated discriminant functions for the glQDA model are given by

$$\hat{\delta}_k(\mathbf{x}) = -\frac{1}{2} \log |\hat{\Theta}_k^{-1}| - \frac{1}{2} (\mathbf{x} - \bar{\mathbf{x}}_k)^T \hat{\Theta}_k (\mathbf{x} - \bar{\mathbf{x}}_k) + \log(\hat{\pi}_k), \quad k = 1, \dots, K. \quad (5.1)$$

To use these discriminant functions for classifying a new observation to one of the K classes, the user simply calculates $\hat{\delta}_k(\mathbf{x})$ for each class, and classifies the observation to the class for which $\hat{\delta}_k(\mathbf{x})$ is the largest.

For the glNPDA model, the precision matrices Θ_k , $k = 1, \dots, K$ were estimated using the graphical lasso algorithm in a similar manner to the glQDA model. The mean vectors for each class were

estimated using the appropriate sample mean vector, and the priors were estimated using the sample class proportions. For the gINPDA model, the same methods which were applied in the case of the glQDA model, were used to select the tuning parameter. In addition, the gINPDA has a second tuning parameter, the bandwidth of the kernel density. Searching over a two-dimensional grid for the optimal bandwidth and graphical lasso tuning parameter becomes computationally too expensive. For this reason, when selecting the tuning parameter for the graphical lasso, the bandwidth was set to Silverman's rule of thumb. In addition, different values for the bandwidth were considered by selecting the graphical lasso tuning parameter using the eBIC and cross-validation to search for the optimal bandwidth over a grid of 100 values equally spaced between 0 and 3. The one-standard-error rule was also applied in this search.

The estimated discriminant functions for the gINPDA model are given by

$$\hat{\delta}_k(\mathbf{x}) = -\frac{1}{2} \log |\hat{\Theta}_k^{-1}| - \frac{1}{2} (\tilde{\mathbf{g}}_k(\mathbf{x}) - \bar{\mathbf{x}}_k)^T \hat{\Theta}_k (\tilde{\mathbf{g}}_k(\mathbf{x}) - \bar{\mathbf{x}}_k) + \log(\hat{\pi}_k) + \sum_{j=1}^p \log(\tilde{\mathbf{g}}'_{kj}(\mathbf{x})), \quad (5.2)$$

where $k = 1, \dots, K$. The use of these discriminant functions differs from that of the glQDA in terms of the transformations of the observations. To classify a new observation to one of the K classes, the user needs to apply the transformation associated with the discriminant function to the observation, where the transformation is given by $\tilde{\mathbf{g}}_k(\mathbf{x})$. Once $\hat{\delta}_k(\mathbf{x})$ is calculated for each class, the observation is then classified to the class for which $\hat{\delta}_k(\mathbf{x})$ is the largest. When the number of classes is large, and the user wants to classify many observations, gINPDA can be undesirable, since it can take significantly longer than glQDA to classify observations. For this reason, when fitting the gINPDA model in the subsequent sections, the training error is not calculated for this model since it will require "scoring-out" all the training observations.

Due to the assumption of normality for the glQDA model, each dataset is tested for multivariate normality. The normality assessment is done on the training set. The measure of normality is the Henze-Zirkler test of normality (Henze and Zirkler, 1990). For the Henze-Zirkler test of normality the R package *MVN*, developed by Korkmaz et al. (2014), was used. The application of the above methods is described in the next section.

Lastly it is important to note that great caution was taken when performing cross-validation to avoid biases in the cross-validation approaches. The cross-validation folds were created before estimating any parameters of the models for every technique or method.

5.3 The Vowel Dataset

The first dataset considered in our empirical work is known as the *Vowel* dataset. The Vowel dataset is used throughout Hastie et al. (2009) and can also be found in Dua and Graff (2019). The dataset stems from a speech recognition problem with 11 classes and 10 predictors. The classes represent 11 vowel sounds, each contained in 11 different words.

The words are:

Heed	Hod	Hid	Hoard
Head	Hood	Had	Who'd
Hard	Heard	Hud	

Eight people spoke each word six times, resulting in a training set consisting of 528 observations. To obtain an independent test set, seven people spoke each word six times, thereby yielding 462 test observations. The 10 predictors are derived from digitised speech. Hastie et al. (2009) compared the performance of 17 different models on the vowel dataset. Their evaluation was based upon prediction accuracy as performance metric. Their results are summarised in Table 5.1 below.

Table 5.1: Vowel dataset results by Hastie et al. (2009).

Technique	Training error	Test Error
LDA	0.32	0.56
LDA (softmax)	0.48	0.67
QDA	0.01	0.53
CART	0.05	0.56
CART(Linear combination split)	0.05	0.54
Single-layer perceptron	-	0.67
Multi-layer perceptron	-	0.49
Gaussian node network	-	0.45
Nearest neighbour	-	0.44
FDA/BRUTO	0.06	0.44
FDA/BRUTO (Softmax)	0.11	0.50
FDA/MARS (degree=1)	0.09	0.45
(Best reduced dimension =2)	0.18	0.42
FDA/MARS(Softmax)	0.14	0.48
FDA/MARS (degree=2)	0.02	0.42
(Best reduced dimension =6)	0.13	0.39
FDA/MARS(Softmax)	0.1	0.50

From Table 5.1 it can be seen that the prediction accuracy of the fitted models ranges from 0.39 to 0.67, with the best model being the MARS model with degree 2, and using dimension reduction to the 6 best dimensions. For some benchmarks, note that the average prediction accuracy of all the methods is approximately 0.501, and that guessing will result in an expected error of approximately 0.909. Considering the methods above, it can be observed that the flexible models improved prediction accuracy.

Prior to comparing the results in Table 5.1 to the results generated in this thesis, the multivariate normality of the Vowel dataset is assessed with the aim of obtaining more insight into the results.

The results for the Henze-Zirkler test is given in Table 5.2.

Table 5.2: Multivariate normality test for the Vowel data.

Vowel	Henze-Zirkler statistic	P-value
Heed	1.7410	0.000
Head	1.9202	0.000
Hard	1.4694	0.000
Hod	1.6117	0.000
Hood	1.5946	0.000
Heard	1.3592	0.000
Hid	1.4494	0.000
Had	1.2630	0.000
Hud	1.4287	0.000
Hoard	1.5455	0.000
Who'd	1.5185	0.000

Considering Table 5.2, it is noted that the Vowel dataset is not normally distributed. Next, the results obtained by Hastie et al. (2009) in Table 5.1 are compared to our results, as presented in Tables 5.3 and 5.4 for the glQDA and glNPDA models respectively. The first column in both tables represents the fitted model. The second column indicates which criterion was used to select the regularisation parameter, while the selected tuning parameter value is given in the third column. The fourth column shows the regularisation parameter selected for methods where all classes received the same value. The proportion of misclassifications on the training data is shown in the fifth column, and the sixth column conveys the performance of the model on the test dataset. The columns annotated by “lam1” up to “lam11” are used to display the regularisation parameter selected for each class in the case of methods that selected different regularisation parameters for different classes.

Table 5.3: Results of application to the Vowel data using the glQDA model.

Model	Criterion	Criterion Parameter	Lambda	Train error	Test error
glQDA	-	-	0.0000	0.0114	0.5281
glQDA	ebic	0.5000	-	0.1326	0.4069
glQDA	ebic.CV	0.1600	-	0.1686	0.4091
glQDA	ebic.CV(1sd)	1.0000	-	0.1458	0.4091
glQDA	stars	0.1000	-	0.4470	0.5325
glQDA	stars	0.0500	-	0.4659	0.6104
glQDA	CV	-	0.0301	0.0700	0.4026
glQDA	CV(sd)	-	0.3517	0.2936	0.4827

lam1	lam2	lam3	lam4	lam5	lam6	lam7	lam8	lam9	lam10	lam11
-	-	-	-	-	-	-	-	-	-	-
0.1392	0.0761	0.0943	0.0937	0.0898	0.0948	0.0889	0.0924	0.0942	0.0875	0.0892
0.1392	0.0761	0.0943	0.0937	0.0898	0.0948	0.0889	0.0924	0.0899	0.0875	0.0892
0.1392	0.0761	0.0943	0.1005	0.0898	0.1017	0.0889	0.0924	0.0942	0.1361	0.0892
0.3292	0.3451	0.1895	0.9371	0.1606	0.1196	0.3271	0.5289	0.1685	0.2734	0.2014
0.7783	0.6319	0.3469	0.1840	0.2557	0.1818	0.3939	0.5541	0.5027	0.2931	0.2159
-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-

First consider Table 5.3. From this table it can be seen that the glQDA model, with the regularisation parameter chosen to be equal to zero over all classes, yields the same result as the QDA model in Table 5.1 (viz. a test error of approximately 0.53). This is expected, since setting $\lambda = 0$ over all classes fits the normal QDA model with no regularisation on the precision matrix. The corresponding test error may thus be viewed as a baseline performance. Considering the glQDA models fitted using the eBIC criterion, viz. ebic.CV and ebic.CV(1sd), it can be seen that the models improve on the error rate of 0.5281. In addition, considering the λ values for every class corresponding to the ebic.CV and ebic.CV(1sd) models, it is apparent that they do not vary much for the different values of the eBIC tuning parameter (γ). However, the training errors (viz. 0.1686 and 0.1458) differ a lot. From this we concluded that the value of γ did not have a significant impact on the prediction accuracy of the model. With reference to the remaining criteria, the StARS criterion performed the worst, yielding test error rates greater than those of the original QDA model. Compared to the other tuning parameter selection techniques, the optimal tuning parameter values selected by StARS are larger than those associated with the other techniques. Thus, it appears that the StARS technique leads to too much penalisation.

The approach of selecting a fixed value for λ over all the classes using cross-validation, performed well when a small value for λ was selected. Implementing the one-standard-error rule led to the selection of a very large λ value, which in turn resulted in a deterioration of the model's performance. For the glQDA model applied to the Vowel dataset, it appears that estimating the precision matrix using the graphical lasso algorithm leads to an improvement in prediction accuracy as long as λ is not too large. Lastly, comparing the best performance over all methods considered by Hastie et al. (2009) in Table 5.1, it is noted that the glQDA model with the eBIC criterion and a single λ over all classes performed slightly worse than the smallest test error rate of 0.39.

Next consider the results of the glNPDA model on the Vowel dataset in Table 5.4 below.

Table 5.4: Results of application to the Vowel data using the glNPDA model

Model	Criterion	Criterion Parameter	Lambda	Train error	Test error
glNPDA	-	-	0.0000	-	0.4848
glNPDA	ebic	0.5000	-	-	0.5238
glNPDA	ebic.CV	0.0000	-	-	0.5216
glNPDA	ebic.CV(1sd)	1.0000	-	-	0.5260
glNPDA	stars	0.1000	-	-	0.6623
glNPDA	stars	0.0500	-	-	0.6623
glNPDA	CV (sd)	-	0.0502	-	0.4610
glNPDA	CV	-	0.1105	-	0.5476
glNPDA	cv-BW	0.3511	-	-	0.4502
glNPDA	cv-BW(sd)	1.0133	-	-	0.5325

lam1	lam2	lam3	lam4	lam5	lam6	lam7	lam8	lam9	lam10	lam11
-	-	-	-	-	-	-	-	-	-	-
0.0963	0.0843	0.0956	0.0952	0.0903	0.0943	0.0937	0.0939	0.0921	0.0905	0.0944
0.0963	0.0824	0.0956	0.0952	0.0903	0.0943	0.0937	0.0939	0.0921	0.0905	0.0922
0.1009	0.0843	0.0956	0.0952	0.0903	0.0943	0.0937	0.0984	0.0921	0.0905	0.1060
0.2557	0.2239	0.1235	0.9522	0.1166	0.9435	0.2215	0.3792	0.1218	0.2090	0.1503
0.5509	0.4396	0.3435	0.1121	0.1991	0.1307	0.2488	0.4066	0.3235	0.2699	0.1538
-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-
0.2259	0.1519	0.0818	0.0324	0.0454	0.0367	0.0863	0.1782	0.1128	0.1384	0.0414
0.1528	0.0918	0.0407	0.0140	0.0209	0.0146	0.0407	0.1109	0.0611	0.0821	0.0131

From Table 5.4 it can be noted that the nonparanormal model with no regularisation was an improvement on the original QDA model. Adopting the eBIC method to select the tuning parameter did not result in a significant reduction in the test error rate when compared to the original QDA model. On the contrary, regularisation, together with the nonparanormal model, seems to slightly dampen the model's performance. It is also noted that the use of different values for the eBIC γ parameter does not make a big difference in the test error. With reference to the StARS selection technique, this method performed the worst. Selecting a fixed λ using cross-validation over all the classes led to a slight improvement in prediction accuracy. The selected λ value was also small relative to the selected λ values via eBIC. As before, application of the one-standard-error rule had a negative impact on the results. Lastly, when the bandwidth is selected using cross-validation, the optimal bandwidth was 0.3511. Comparing the corresponding test error to the test errors in Table 5.4, this method yields the best results. Once again, adopting the one-standard-error rule had a negative impact on the results.

When comparing the results of the gINPDA model to those of gIQDA on the Vowel dataset, it is noted that the gIQDA model generally performed better than the gINPDA model. It is also noted that large amounts of regularisation for both the gINPDA and gIQDA models led to worse results than in the case of the original QDA model. From the results presented in Tables 5.3 and 5.4, it is observed that both the gINPDA and gIQDA models performed well in terms of prediction accuracy, when compared to the statistical learning techniques presented in Table 5.1. In the next section, the two methods are assessed in terms of their performance on a digit recognition dataset.

5.4 The Zip Code Dataset

The zip code dataset was also used in Hastie et al. (2009). The dataset comprises handwritten zip codes on envelopes from postal mail. The dataset consists of images which are 16×16 eight-bit grayscale maps. Each pixel's intensity ranges from 0 to 255. The goal is to predict the digits 0, 1, ..., 9 using the pixel intensities. The dataset was made available by AT&T Labs. The training set consist of 7,291 rows (digits) with 256 columns corresponding to each pixel, while the test set consists of 2,007 rows (digits). Figure 5.1 shows an example of the handwritten digits.

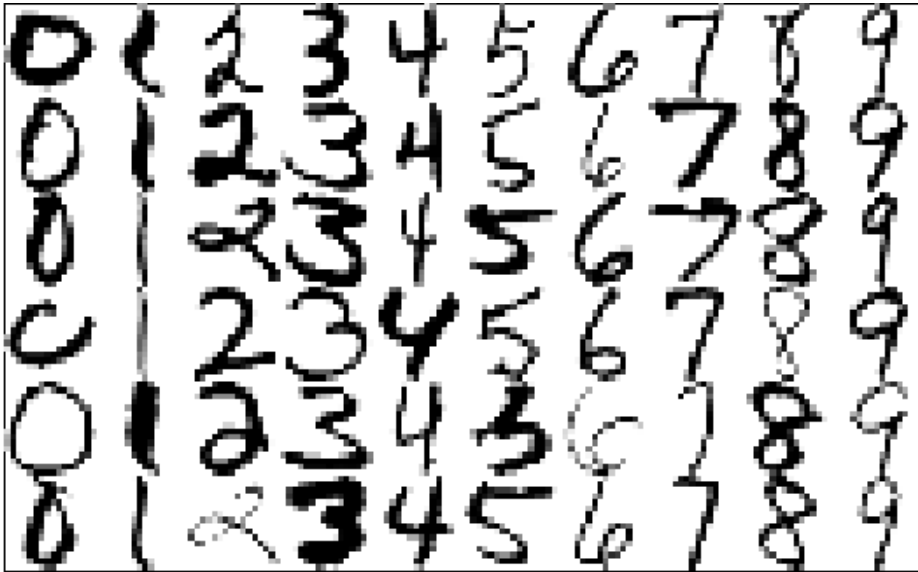


Figure 5.1: Handwritten digits example.

Initially, the MNIST (Modified National Institute of Standards and Technology) dataset was considered. The MNIST dataset also comprises handwritten digits, and consists of 60,000 training samples, and 10,000 test samples. The images in this dataset are 28×28 eight-bit grayscale maps, which results in 784 columns. This dataset is significantly wider and longer than the zip code dataset. Due to limited available computational resources, the *zip code* dataset was adopted instead. In the literature, applications of the full zip code dataset were not found.

Hastie et al. (2009) trained five neural networks, using a training and test set consisting of 320 and 160 digits respectively. Additional images were generated by horizontal shifts. The results of the Hastie et al. (2009) study are shown in Table 5.5. In addition, (limited) results obtained on the full MNIST dataset are presented in Table 5.6. The purpose of including results on the MNIST dataset relates to evaluating test errors obtained on handwritten digits recognition problems. Including the test errors on the MNIST dataset provides additional information for estimating test errors associated with the recognition problems. The models selected for comparison capture different degrees of complexity.

Table 5.5: Zip code dataset results by Hastie et al. (2009).

Technique	Description	Test Error
Single layer network	No hidden layer	0.200
Two layer network	One hidden layer, 12 hidden units fully connected.	0.130
Locally connected	Two hidden layers locally connected.	0.115
Constrained network 1	Two hidden layers, locally connected with weight sharing.	0.060
Constrained network 2	Two hidden layers, locally connected, two levels of weight sharing	0.016

Table 5.6: Results of applications to the MNIST dataset.

Technique	Test Error
Linear classifier (1-layer NN)	0.120
K-nearest-neighbours K=3 Euclidian distance	0.050
Boosted stumps	0.070
Boosted trees (17 leaves)	0.015
40 PCA with quadratic classifier	0.033
SVM Gaussian kernel	0.014
2-layer Neural network 1000 hidden units	0.045
6-layer neural network with units, 784, 2500, 2000, 1500, 1000, 500, 10	0.0035

With reference to Tables 5.5 and 5.6, it is noted that a wide range of models have been fitted to the digit recognition datasets. On the MNIST dataset, the test error ranges from less than 0.01 up to 0.12. On the dataset used in Hastie et al. (2009), the test error ranges from less than 0.02 up to 0.20.

As for the Vowel dataset, multivariate normality of the zip code dataset was evaluated. The normality assessment is carried out on the full training set. The results for the Henze-Zirkler test is given in Table 5.7.

Table 5.7: Multivariate normality test for zip code dataset.

Digit	Henze-Zirkler statistic	P-value
0	1.0000	0.000
1	1.0000	0.000
2	1.0000	0.000
3	1.0000	0.000
4	1.0000	0.000
5	1.0000	0.000
6	1.0000	0.000
7	1.00004	0.000
8	1.0000	0.000
9	1.0000	0.000

From Table 5.7 it is apparent that the zip code dataset is not normally distributed.

The results of the gIQDA model on the full zip code dataset are presented in Table 5.8.

Table 5.8: Results of application to the zip code dataset using the glQDA model.

Model	Criterion	Criterion. Par	Lambda	Train error	Test error
glQDA	-	-	0.0000	0.1439	0.2367
glQDA	ebic	0.5000	-	0.1004	0.1465
glQDA	ebic.CV	0.1600	-	0.1004	0.1465
glQDA	ebic.CV(1sd)	1.0000	-	0.1004	0.1465
glQDA	stars	0.1000	-	0.2046	0.2431
glQDA	stars	0.0500	-	0.2518	0.2820
glQDA	CV	-	0.3500	0.0826	0.1345
glQDA	CV(sd)	-	0.5800	0.1038	0.1505

lam1	lam2	lam3	lam4	lam5	lam6	lam7	lam8	lam9	lam10
-	-	-	-	-	-	-	-	-	-
0.1709	0.1657	0.1548	0.1661	0.1640	0.1451	0.1672	0.1591	0.1538	0.1682
0.1709	0.1657	0.1548	0.1661	0.1640	0.1451	0.1672	0.1591	0.1538	0.1682
0.1709	0.1657	0.1548	0.1661	0.1640	0.1451	0.1672	0.1591	0.1538	0.1682
0.1452	0.1475	0.1660	0.1623	0.1460	0.5214	0.1180	0.1628	0.1435	0.1722
0.2785	0.2349	0.2900	0.3416	0.2435	0.8302	0.2160	0.2716	0.2180	0.4167
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-

From Table 5.8 we see that the glQDA model with λ chosen to be equal to zero over all classes, has a test error rate of 0.2367. This is equivalent to the original QDA model and is the adopted benchmark for this application. The fitted glQDA models, with the exception of those using the StARS parameter selection criterion, are seen to improve upon the test error rate for the original QDA model. Once again, it can be seen that the StARS selection method led to excessive regularisation. All eBIC selection methods yielded the same results. In addition, selecting one value for the regularisation parameters over all classes, seemed to perform best with a test error of 0.1345. When compared to the test error rates presented in Tables 5.5 and 5.6, this value may seem unremarkable. However, it does indicate the model to be relatively competitive. The use of the one-standard-error rule once again had a negative impact on the reported test errors. Next, the results of the glNPDA model are considered. The results are presented in Table 5.9.

Table 5.9: Results of application to the zip code data using the gINPDA model.

Model	Criterion	Criterion Parameter	Lambda	Train error	Test error
gINPDA	-	-	0.0000	-	0.2790
gINPDA	ebic	0.5000	-	-	0.4534
gINPDA	ebic.CV	0.0000	-	-	0.4534
gINPDA	ebic.CV(1sd)	1.0000	-	-	0.4534
gINPDA	Stars	0.1000	-	-	0.5087
gINPDA	Stars	0.0500	-	-	0.5924
gINPDA	CV (sd)	-	0.0100	-	0.3572
gINPDA	CV	-	0.0100	-	0.3572
gINPDA	cv-BW	1.6756	-	-	0.1091
gINPDA	cv-BW(sd)	2.6689	-	-	0.1166

lam1	lam2	lam3	lam4	lam5	lam6	lam7	lam8	lam9	lam10
-	-	-	-	-	-	-	-	-	-
0.1100	0.1060	0.0972	0.1097	0.1102	0.1203	0.1158	0.1098	0.0955	0.1193
0.1100	0.1060	0.0972	0.1097	0.1102	0.1203	0.1158	0.1098	0.0955	0.1193
0.1100	0.1060	0.0972	0.1097	0.1102	0.1203	0.1158	0.1098	0.0955	0.1193
1.1004	1.0601	0.9717	1.0966	1.1017	1.2026	1.1578	1.0982	0.9555	1.1932
0.1325	0.1191	0.1285	0.1321	0.1181	0.3121	1.1578	0.1418	0.1099	0.1691
-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-
0.0431	0.0419	0.0392	0.0421	0.0415	0.0373	0.0423	0.0404	0.0389	0.0420
0.0209	0.0203	0.0189	0.0203	0.0200	0.0178	0.0204	0.0195	0.0188	0.0205
-	-	-	-	-	-	-	-	-	-

From Table 5.9, it is noted that the nonparanormal model with no regularisation performed worse than the original QDA model. Using the eBIC as a method to select the tuning parameter, led to even poorer results. It is also noted that using different values for the eBIC, the γ parameter did not make a big difference to the test error. For the StARS selection technique, it is noted that this method performs the worst of all methods considered. Selecting a fixed λ using cross-validation over all the classes also performed poorly, and the one-standard-error rule did not make any difference. Considering the test error when the bandwidth is selected using cross-validation, a mentionable improvement in the performance of the model is observed. When the bandwidth is selected using cross-validation, the gINPDA model outperforms the gIQDA model. In the following

section the performance of the gINPDA and gIQDA models are investigated on a binary classification task.

5.5 The Spam Dataset

The spam dataset is the final dataset on which the different models and parameter selection methods will be tested. The spam dataset was also used throughout Hastie et al. (2009). The dataset consists of 4,601 emails which were marked “spam” or “email”. The columns of the dataset are the relative frequencies of the 57 most commonly occurring words and punctuation marks in the “emails” and “spams”. The results obtained by Hastie et al. (2009) on the spam dataset are summarised in Table 5.10. It is important to note that Hastie et al. (2009) did not specify a test and training set explicitly. Therefore, a direct comparison between their results and the results generated in the current assessment, is not possible.

Table 5.10: Spam dataset results by Hastie et al. (2009).

Technique	Test Error
Additive logistic regression	0.0550
17-node tree	0.0930
Gradient boosting	0.0450
CART tree fully grown and 10.9 Boosting Trees 353 pruned by cross-validation	0.0870
MARS	0.0550
Random forests	0.0488

Considering Table 5.10, the test errors reported by Hastie et al. (2009) varied between 0.045 and 0.093.

As before, multivariate normality of the dataset was assessed. The normality assessment was carried out on the full training set. Results from the Henze-Zirkler test is given in Table 5.11.

Table 5.11: Multivariate normality test for the spam dataset.

Digit	Henze-Zirkler statistic	P-value
“spam”	4.6686	0.000
“email”	19.9232	0.000

From the results in Table 5.11, it is apparent that the spam dataset is not normally distributed. The gIQDA and gINPDA results are provided in Tables 5.12 and 5.13, respectively. To facilitate a comparison between the two set of results, 5-fold cross-validation was performed on the 4,601 emails to determine an average test error, based upon the average error rate on the five left-out

folders. The obtained parameter values presented are also averaged over the folds. In addition, to allow a fairer comparison the cross-validation also allowed standard errors of the results to be calculated. The standard errors are reported in the columns that contain “sd” in the description, or in the third column (in brackets). Table 5.12 presents the results of the glQDA model.

Table 5.12: Results of application to the spam data using the glQDA model.

Model	Criterion	Criterion. Par	Lambda	Train error	Train error sd	Test error	Test error sd
glQDA	-	-	0.0000	0.1106	0.0053	0.1150	0.0114
glQDA	ebic	0.5000	-	0.0985	0.0045	0.1004	0.0093
glQDA	ebic.CV	0.36 (0.3532)	-	0.0985	0.0044	0.1006	0.0092
glQDA	ebic.CV(1 sd)	1 (0)	-	0.1014	0.0047	0.1022	0.0086
glQDA	stars	0.1000	-	0.1032	0.0054	0.1039	0.0086
glQDA	stars	0.0500	-	0.1024	0.0047	0.1048	0.0085
glQDA	CV	-	0.2260	0.0982	0.0040	0.1002	0.0089
glQDA	CV(sd)	-	1.0000	0.0978	0.0049	0.0976	0.0083

lam1	lam1 sd	lam2	lam2 sd
-	-	-	-
0.0837	0.0122	0.0986	0.0028
0.0898	0.0162	0.0951	0.0107
0.1031	0.0086	0.1019	0.0046
0.1325	0.0161	0.1179	0.0160
0.2067	0.0223	0.1843	0.0139
-	0.0737	-	0.0737
-	-	-	-

From the test errors in Table 5.12, it is noted that the original QDA model had a test error of 0.1150, with a standard error of 0.0114. Comparing this to the glQDA model results shows that the glQDA model improved on the test error (and standard error) in the case of all parameter selection criteria. The StARS parameter selection technique performed the worst. Selecting one value for λ over both classes using cross-validation performed the best. This is an unexpected result, since it led to the most regularisation of the precision matrices. Since all the test errors are close to each other, and since the amount of regularisation varies a lot, it appears that the value of λ did not have a large influence on the results. Provided λ was greater than 0, it outperformed the original QDA model. Comparing the results with that of Table 5.10, it is noted that the best result, based on the test error of the glQDA model, is close to the 17-node tree. The results of the glNPDA model are displayed in Table 5.13 below.

Table 5.13: Results of application to the spam data using the gINPDA model.

Model	Criterion	Criterion. Par	Lambda	Train error	Train error Sd	Test error	Test error Sd
gINPDA	-	-	0.0000	-	-	0.1128	0.0126
gINPDA	ebic	0.5000	-	-	-	0.1191	0.0126
gINPDA	ebic.CV	0(0)	-	-	-	0.1182	0.0135
gINPDA	ebic.CV(1 sd)	1(0)	-	-	-	0.1187	0.0130
gINPDA	stars	0.1000	-	-	-	0.1480	0.0297
gINPDA	stars	0.0500	-	-	-	0.1221	0.0155
gINPDA	CV	-	0.0221	-	-	0.1139	0.0133
gINPDA	CV(Sd)	-	0.2271	-	-	0.1328	0.0161
gINPDA	cv-BW	0.7545 (0.1684)	-	-	-	0.0963	0.0098
gINPDA	cv-BW(sd)	1.2722 (0.1724)	-	-	-	0.1074	0.0059

lam1	lam1 Sd	lam2	lam2 Sd
-	-	-	-
0.0645	0.0434	0.0581	0.0433
0.0628	0.0457	0.0579	0.0435
0.0649	0.0429	0.0611	0.0392
0.6109	0.4807	0.5633	0.4568
0.1411	0.0219	0.2727	0.3629
-	0.0180	-	0.0180
-	0.1105	-	0.1105
0.0538	0.0401	0.0511	0.0421
0.0509	0.0429	0.0494	0.0441

From Table 5.13 it is clear that the gINPDA model did not perform well, except when the bandwidth was selected using cross-validation. The StARS parameter selection technique performed the worst of the parameter selection techniques considered. In terms of the regularisation parameters selected when using StARS, it is noted that the regularisation parameters are significantly larger than those selected using the other methods. This indicates that the StARS method led to excessive penalisation. For the default value of the bandwidth, it is apparent that selecting a single value for the regularisation parameter over all classes, produced the best results. Over all gINPDA configurations, selecting the bandwidth using cross-validation performed the best, even outperforming the gIQDA model. When comparing the standard errors between the gINPDA model and the gIQDA model, the gINPDA model has slightly larger standard errors. This result is to be expected from the greater complexity of the gINPDA model.

5.6 Summary

In this chapter, the proposed new statistical learning technique (gINPDA), as well as the QDA model with its precision matrix estimated using the graphical lasso algorithm (glQDA), were empirically compared to a wide range of statistical learning methods, on three datasets.

The glQDA model assumes the data to be normally distributed. In order to test the multivariate assumption on each of the three datasets considered, the Henze-Zirkler test was used. In the case of all three datasets, the null hypothesis of normal data was rejected. It was found that the glQDA model still performed very well, which indicates that the performance of the glQDA model is not solely dictated by the assumption of normality. For the regularisation parameter selection techniques, it was found that the StARS selection technique consistently performed poorly. Too much regularisation was a persistent issue when the StARS criterion was considered. The eBIC parameter selection technique appeared to perform well with the glQDA model. However, its tuning parameter did not appear to have much influence on the test error. Selection of a fixed value for the regularisation parameter for all classes, by means of cross-validation, often provided the best result over all parameter selection techniques. When comparing the glQDA model results to those of the original QDA, for all datasets considered, the glQDA model showed an improvement in the test error. Hence it appears that limited or minor regularisation on the precision matrix is consistently advantageous. The glQDA model did not outperform the other models considered, although it was found to be relatively competitive.

The gINPDA model does not assume the data considered to be normally distributed. It was often found that the gINPDA model did not perform significantly better than the glQDA model. However, when the bandwidth of the gINPDA model was selected using cross-validation, a significant improvement in the test error was observed, and the model outperformed the glQDA model on two of the three datasets. The StARS parameter selection technique also performed poorly in the case of the gINPDA model. In cases where the kernel density bandwidth was fixed, selecting one value for the Graphical lasso tuning parameter using cross-validation yielded the best results.

In summary, from the results presented in this chapter it was found that both the glQDA and gINPDA models generate competitive results when compared to other more popular statistical learning techniques. It was also determined that the new gINPDA method has the potential to perform very well given that the bandwidth of the kernel density estimator is tuned to maximise prediction accuracy.

CHAPTER 6

CONCLUSION

6.1 Summary

In this thesis a novel statistical learning technique, referred to as the *gINPDA* model, was proposed. The new method involves the combination of probabilistic graphical models for continuous random variables with discriminant analysis. Considering the numerous and diverse real-world applications of statistical learning models, a method which could improve the prediction accuracy of these models could have significant practical advantages. The objective of this study was to propose such a method.

When the joint distribution of the probabilistic graphical model is assumed to be a multivariate Gaussian distribution, the associated graph structure of the model is captured in the inverse covariance matrix, also known as the precision matrix of the Gaussian distribution. The associated graph represents the conditional independencies between the random variables. The most common estimator for the precision matrix is the MLE. The MLE is the smallest variance unbiased estimator, and converges to the population parameter as the sample size increases. The disadvantage of the MLE is that it often suffers from high variance, and rarely leads to zero entries in the estimated precision matrix. Zero entries in the precision matrix indicate conditional independence and are required to obtain sparse graphs. A solution to this problem is the L1 penalised estimator. The L1 penalised estimator imposes an L1 penalty on the MLE, which leads to sparser graphs and lower variance for the estimated precision matrix. However, this solution is not without a downside: it results in an increase in the bias of the estimator. To find the L1 penalised estimator, the most common and recommended method in the literature, viz. the graphical lasso algorithm, was considered.

In statistical learning theory, multiple models use the covariance matrix or precision matrix of the input variables as parameters in the model. One popular model where this is the case, is the QDA model. The QDA model also assumes the input variables in the model to follow a multivariate Gaussian distribution. In the literature, the graphical lasso algorithm has been used to estimate the precision matrix of the QDA model. The results were found to be promising. Since the normality assumption is often found to be too restrictive, an extension to a non-normal approach was evaluated. A popular method, the nonparanormal model, was identified. The nonparanormal model is a semiparametric model in which the marginal distributions are estimated empirically, and then combined into a parametric model. It was noted that a strong connection exists between the nonparanormal model and the Gaussian copula. The conditional dependence structure of the nonparanormal model is also captured in the precision matrix. The graphical lasso algorithm can thus also be used to estimate the associated precision matrix.

By using the nonparanormal model in the optimal Bayes classifier, as well as the L1 penalised estimator for the precision matrix of the nonparanormal model, a new statistical learning model, named *gINPDA*, was proposed. In order to evaluate the performance of this model it was compared to a wide range of popular classifiers using three different datasets. In addition, several methods for selecting the regularisation parameter for the L1 penalised estimator, were evaluated. The results for the proposed *gINPDA* model were found to be promising. It was observed that the bandwidth adopted in the kernel density estimator, which is used to estimate the marginal distributions for the nonparanormal model, is a key factor in the performance of the model. Furthermore, it was noted that cross-validation was the best method for selecting the regularisation parameter when combining the graphical lasso algorithm with the QDA model, as well as with the *gINPDA* model. It was also observed that some degree of regularisation on the precision matrix consistently yielded better results on the three datasets in the case of both the QDA model and the *gINPDA* model.

6.2 Future Research

While the proposed *gINPDA* model yielded promising results, computational constraints limited the evaluation of model performance in multiple ways. The first constraint was encountered when selecting the tuning parameter for the L1 penalised estimator for each individual class using cross-validation. It is believed that this approach could have improved the results obtained. For the proposed model, when selecting the regularisation parameter together with the bandwidth for the kernel density estimator, the same constraint occurs. We speculate that this could have yielded significant improvements. In Chapter 3, the computational cost associated with grid searches was noted. However, in practice there are ways to circumvent this, for example by means of parallel computing. Since grid searches include looping through a grid of values, it is not difficult to implement cross-validation using parallel computation. Additionally, other innovative alternatives to traditional grid searches do exist. A simple example entails starting at random points between two extreme values and searching small neighbourhoods around these values.

In addition to the computational constraints with regards to cross-validation, the *huge* package struggled with memory management and computational efficiency when applied to a very high-dimensional dataset. The authors of the *huge* package indicate that the package is for “High-Dimensional Undirected Graph Estimation”, but when attempting to apply the functions in the package to a microarray classification problem with 16,000 gene expression measurements (variables) for 114 patients (observations), considerable memory issues were encountered. For this reason, the application of *gINPDA* and *gIQDA* on the microarray classification problem was discontinued, and thus not presented as part of this thesis.

Other areas of potential future research include finding other approaches for estimating the regularisation parameter. In the early stages of this thesis, some investigative work was undertaken to evaluate the relationship between the condition number of the precision matrix and the optimal value for the regularisation parameter. Several configurations were attempted; however, all came to an impasse. A focused study in this direction may yield different results.

Furthermore, other approaches towards relaxing the Gaussian assumption in a probabilistic graphical model (other than the nonparanormal model), may be investigated. A popular alternative is forest density estimation. Forest density estimation relaxes the Gaussian assumption, while restricting the graph structure to a forest (Lafferty et al., 2012).

Finally, in statistical learning, multiple methods make use of some version of a covariance matrix of the predictors to estimate the parameters of the statistical learning model. Some examples include least squares linear regression, smoothing splines and Gaussian mixture models. Combining these statistical learning models with the L1 penalised estimator for the precision matrix, could lead to interesting results.

6.3 Conclusion

Multiple concepts were combined in this thesis in order to develop the proposed new gINPDA model. The results reported of an empirical study illustrated the potential of this model to be competitive in terms of prediction accuracy when compared to other popular statistical learning models. However, currently the model has some limitations, which include the requirement for a significant amount of computation power to be suitably tuned. Using the gINPDA model to make predictions is also computationally relatively expensive. Furthermore, to obtain more support for this model, a wider range of applications is needed.

REFERENCES

- Banerjee, O., Ghaoui, L.E. and d'Aspremont, A. 2008. Model selection through sparse maximum likelihood estimation for multivariate Gaussian or binary data. *Journal of Machine Learning Research*, 9, March:485-516.
- Barber, D. 2012. *Bayesian Reasoning and Machine Learning*. Cambridge: University Cambridge Press, pp.50.
- Chen, J. and Chen, Z. 2008. Extended Bayesian information criteria for model selection with large model spaces. *Biometrika*, 95(3):759-771.
- Chen, J. and Chen, Z., 2012. Extended BIC for small-n-large-P sparse GLM. *Statistica Sinica*, April:555-574.
- Dahl, J., Vandenberghe, L. and Roychowdhury, V. 2008. Covariance selection for nonchordal graphs via chordal embedding. *Optimization Methods & Software*, 23(4):501-520.
- Dua, D. and Graff, C. 2019. UCI Machine Learning Repository [Online]. Available: <http://archive.ics.uci.edu/ml> [2019, May 27].
- Friedman, J., Hastie, T. and Tibshirani, R. 2008. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432-441.
- Friedman, J., Hastie, T. and Tibshirani, R. 2010. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1-22.
- Foygel, R. and Drton, M. 2010. Extended Bayesian information criteria for Gaussian graphical models. *Advances in Neural Information Processing Systems*, 604-612.
- Guo, Y., Hastie, T. and Tibshirani, R. 2006. Regularized linear discriminant analysis and its application in microarrays. *Biostatistics*, 8(1):86-100.
- Hastie, T., Friedman, J. and Tibshirani, R. 2009. *The Elements of Statistical Learning*. 2nd ed. New York: Springer.
- Hastie, T., Tibshirani, R. and Buja, A. 1994. Flexible discriminant analysis by optimal scoring. *Journal of the American Statistical Association*, 89(428):1255-1270.
- Hastie, T., Tibshirani, R. and Buja, A. 2000. Flexible discriminant and mixture models, in J. Kay and M. Titterton (eds). *Statistics and Artificial Neural Networks*, Oxford University Press.
- Hastie, T. and Tibshirani, R. 1996. Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):155-176.
- Henze, N. and Zirkler, B. 1990. A class of invariant consistent tests for multivariate normality. *Communications in Statistics-Theory and Methods*, 19(10):3595-3617.

- Johnson, R.A. and Wichern, D.W. 2007. *Applied Multivariate Statistical Analysis*. 6th ed. New Jersey, US: Pearson Prentice Hall.
- Koller, D. and Friedman, N. 2012. *Probabilistic Graphical Models*. Cambridge, Mass: MIT Press.
- Korkmaz, S., Goksuluk, D. and Zararsiz, G. 2014. MVN: An R package for assessing multivariate normality. *The R Journal*, 6(2):151-162.
- Lafferty, J., Liu, H. and Wasserman, L. 2012. Sparse nonparametric graphical models. *Statistical Science*, 27(4):519-537.
- Le, K., Chaux, C., Richard, F. and Guedj, E. 2019. An adapted linear discriminant analysis for the classification in high-dimension, and an application to medical data. *HAL* [Electronic]. Available: <https://hal.archives-ouvertes.fr/hal-02009519> [2019, September 7].
- Le, Y. and Hastie, T. 2014. Sparse quadratic discriminant analysis and community Bayes. *arXiv preprint arXiv:1407.4543*.
- Liu, H., Roeder, K. and Wasserman, L. 2010. Stability Approach to Regularization Selection (StARS) for High Dimensional Graphical Models. *Advances in Neural Information Processing Systems*, June:1432-1440.
- MacKenzie, D. and Spears, T. 2014. 'The formula that killed Wall Street': The Gaussian copula and modelling practices in investment banking. *Social Studies of Science*, 44(3):393-417.
- Pretorius, A., Bierman, S. and Steel, S.J. 2016, A bias-variance analysis of ensemble learning for classification, in *Annual Proceedings of the South African Statistical Association Conference*. Cape Town: 57-64
- Sachs, K., Perez, O., Pe'er, D., Lauffenburger, D.A. and Nolan, G.P. 2005. Causal protein-signaling networks derived from multiparameter single-cell data. *Science*, 308(5721):523-529.
- Silverman, B.W. 1986. *Density estimation for statistics and data analysis*. London: Chapman and Hall.
- Sklar, M. 1959. Fonctions de repartition an dimensions et leurs marges. *Publ. Inst. Statist. Univ. Paris*, 8,:229-231.
- Uhler, C. 2017. Gaussian graphical models: An algebraic and geometric perspective. *arXiv preprint arXiv:1707.04345*.
- Wasserman, L. 2019. Statistical Methods for Machine Learning Class notes. Carnegie Mellon University [Online]. Available: <http://www.stat.cmu.edu/~larry/=sml/GraphicalModels.pdf> [2019, May 27].

- Witten, D.M., Friedman, J.H. and Simon, N. 2011. New insights and faster computations for the graphical lasso. *Journal of Computational and Graphical Statistics*, 20(4):892-900.
- Xu, B., Huang, K., King, I., Liu, C.L., Sun, J. and Satoshi, N. 2011. Graphical lasso quadratic discriminant function for character recognition, in *International Conference on Neural Information Processing*. Springer, Berlin, Heidelberg: 747-755.
- Yuan, M. and Lin, Y. 2007. Model selection and estimation in the Gaussian graphical model. *Biometrika*, 94(1):19-35.
- Zhao, T., Liu, H., Roeder, K., Lafferty, J. and Wasserman, L. 2012. The huge Package for High-dimensional Undirected Graph Estimation in R. *Journal of Machine Learning Research*, 13, April:1059-1062.

APPENDIX A: R CODE

A.1) Algorithm 1: Graphical lasso.

```
Graph.lasso <- function(S.mat, lam)
{
  #Number of dimensions of data
  p <- ncol(S.mat)

  #Initialize W
  W <- S.mat + lam * diag(1, p)
  #Starting value for measure of convergence
  ep <- 1
  if (lam == 0)
  {
    return(S.mat)
  }

  if (p == 2)
  {
    if (abs(W[1, 2]) - lam < 0)
    {
      diff = 0
    }
    else{
      diff <- abs(W[1, 2]) - lam
    }
    W[2, 1] <- sign(W[2, 1]) * diff
    W[1, 2] <- sign(W[1, 2]) * diff
  }

  else{
    while (ep > .Machine$double.eps ^ 0.5)
    {
      W.old <- W
      #Partitioning over each variable
      for (j in 1:p)
      {
        THETA <- solve(W)
        B.vec <- rep(1, p - 1)
        W1 <- W[-j, -j]
        w12 <- W[j, -j]
        V1 <- W1

        ep2 <- 1

        while (ep2 > .Machine$double.eps ^ 0.5)
        {
          B.vec.old <- B.vec
```

```

#Applying the coordinate descent algorithm until convergence
for (i in 1:(p - 1))
{
  Inner <- S.mat[j, -j][i] - V1[-i, i] %*% as.matrix(B.vec[-i])
  B.vec[i] <- sign(Inner)

  if ((abs(Inner) - lam) > 0)
  {
    B.vec[i] <- B.vec[i] * (abs(Inner) - lam) / V1[i, i]
  }
  else
  {
    B.vec[i] <- 0
  }
}
#Measure of convergence
ep2 <- sum((B.vec - B.vec.old) ^ 2)

}
#Updating
W[j, -j] <- W1 %*% as.matrix(B.vec)
diag(W) <- diag(S.mat + lam * diag(1, p))
}
ep <- sum((W - W.old) ^ 2)
}
}
list((W))
}

```

A.2) Algorithm 2: Faster graphical lasso 1.

```

Algorithm2<-function(cov.mat,lam)
{
  p <- ncol(cov.mat)
  pad_mat <- matrix(0, ncol = p, nrow = p)
  lam.mat <- diag(lam, p)
  test.mat <- cov.mat + lam.mat
  uncon <- apply(test.mat, 2, function(x)
    sum(abs(x) <= lam) == (p - 1))

  uncon_cov <- (cov.mat[uncon, uncon])
  con_cov <- cov.mat[!uncon, !uncon]

  unc_prec <- diag(1 / (diag(uncon_cov) + lam))
  colnames(unc_prec) <- colnames(uncon_cov)
  rownames(unc_prec) <- rownames(uncon_cov)

  con_prec <- solve(Graph.lasso(con_cov, lam)[[1]])
  pad_mat[uncon, uncon] <- unc_prec

```

```

pad_mat[!uncon, !uncon] <- con_prec
return(pad_mat)

}

```

A.3) Algorithm 3: Faster graphical lasso 2 and Algorithm 4: Ordering variables.

```

algorithm2 <- function(s, lam)
{
  p <- ncol(s)
  mata <- 1 * (abs(s) > lam)
  diag(mata) <- 1
  matb <- mata

  for (i in 1:(p - 2))
  {
    matb <- matb[, -1]
    mata <-
      mata[c(1:i, (rev(order(matb[1, ])) + i)),
c(1:i, (rev(order(matb[1, ])) + i))]

  }

  matc <- mata
  matd <- mata
  count <- 1

  while (length(matc) > 0)
  {
    selet_row <- matc[, 1] == 1
    select.col <- matc[selet_row, , drop = F]
    select.col <- apply(select.col, 2, function(x)
      sum(x) > 0)
    x <- matc[select.col, select.col, drop = F]
    matc <- matc[!select.col, !select.col, drop = F]

    if (dim(x)[1] == 1)
    {
      matd[colnames(x), colnames(x)] = 1 / (s[colnames(x), colnames(x)] +
lam)
    }

    else{
      matd[colnames(x), colnames(x)] = solve(Graph.lasso(s[colnames(x),
colnames(x)], lam)[[1]])
    }
  }
  return(mata)
}

```

A.4) Code to draw graphs.

```

draw.mod<-function(cov.mat=MRF.alg.1(S.mat,Indp.ed)[[1]],var.name,lambda)
{
  Theta1<-round((cov.mat),digits=6)
  p<-ncol(cov.mat)

  if(missing(var.name))
  {var.name<-paste("x",c(1:p),sep="")}

  dev.new()
  par(pty="s")
  p1<-p+1
  plot(x=0,y=0,xlim=c(-p1,p1),ylim=c(-
p1,p1),"n",xlab="",ylab="",xaxt="n",yaxt="n",bty="n")
  theta<-seq(from=0,to=2*pi,length=p1)
  points(x=p1*cos(theta),y=p1*sin(theta))
  text(x=p1*cos(theta),y=p1*sin(theta),labels=var.name,pos=3)
  col.num<-
c(matrix(c(1:p),ncol=p,nrow=p,byrow=T)[abs(Theta1)>(.Machine$double.eps^0.5)]
)
  row.num<-
c(matrix(c(1:p),ncol=p,nrow=p,byrow=F)[abs(Theta1)>(.Machine$double.eps^0.5)]
)

  for(i in 1:length(col.num))
  {
    lines(x=c(p1*cos(theta)[col.num[i]],p1*cos(theta)[row.num[i]]),
          y=c(p1*sin(theta)[col.num[i]],p1*sin(theta)[row.num[i]]))
  }
}

```

A.5) Natural and Gaussian kernel densities.

```

n=20
set.seed(1993)
val<-rnorm(n)
val=val[order(val)]
pos.vec<-seq(-3,3,by=0.01)
#####
#Natural local estimate
#####
for(j in 1:4)
{

```

```

    lam<-c(0.1,0.5,1,2)[j]
empt_vec<- numeric()
x0.vec<-pos.vec
for(i in 1:length(x0.vec))
{
  empt_vec[i]<-sum((x0.vec[i]+lam)>val & val>(x0.vec[i]-lam))/(lam*n)
}
dev.new()
plot(x=x0.vec,y=empt_vec,type="l",lwd=1.8,xlab="x",ylab="f(x)")
lines(x=seq(-3,3,by=0.01),y=dnorm(seq(-
3,3,by=0.01)),type='l',col="red",lwd=1.8)
legend("topright",c("True",paste("lamda=",lam,sep='')),col=c("red","black"),l
ty=1)
}

#####
#Gaussian kernel density
#####

for (j in 1:4)
{lam=c(0.1,0.5,1,2)[j]
empt_vec2<- numeric()
for(i in 1:length(pos.vec))
{
  empt_vec2[i]<-sum(dnorm(abs(pos.vec[i]-val)/lam))/(length(val)*lam)
}
dev.new()
plot(x=pos.vec,y=empt_vec2,type="l",lwd=1.8,xlab="x",ylab="f(x)",ylim=c(0,0.5
))
lines(x=seq(-3,3,by=0.01),y=dnorm(seq(-
3,3,by=0.01)),type='l',col="red",lwd=1.8)
legend("topright",c("True",paste("lamda=",lam,sep='')),col=c("red","black"),l
ty=1)
}

dev.new()
plot(density(x=val,bw=0.1)$x,
      CDF(density(x=val,bw=0.1)$y),type='l')
lines(x=seq(-3,3,by=0.01),y=dnorm(seq(-
3,3,by=0.01)),type='l',col="red",lwd=1.8)

```

A.6) Modified huge package functions.

```

huge.glasso.2.0<-function (x, lambda = NULL, lambda.min.ratio = NULL, nlambdas
= NULL,
                          scr = NULL, cov.output = FALSE, verbose = TRUE)
{
  gcinfo(FALSE)
  n = nrow(x)

```

```

d = ncol(x)
fit = list()
fit$cov.input = isSymmetric(x)
if (fit$cov.input) {
  if (verbose)
    cat("The input is identified as the covariance matrix.\n")
  S = x
}
if (!fit$cov.input) {
  x = x
  S = cov(x)
}
rm(x)
gc()
if (is.null(scr))
  scr = FALSE
fit$scr = scr
if (!is.null(lambda))
  nlambda = length(lambda)
if (is.null(lambda)) {
  if (is.null(nlambda))
    nlambda = 10
  if (is.null(lambda.min.ratio))
    lambda.min.ratio = 0.1
  lambda.max = max(max(S - diag(S)), -min(S - diag(S)))
  lambda.min = lambda.min.ratio * lambda.max
  lambda = exp(seq(log(lambda.max), log(lambda.min), length = nlambda))
}
fit$lambda = lambda
fit$loglik = rep(-d, nlambda)
fit$sparsity = rep(0, nlambda)
fit$df = rep(0, nlambda)
fit$path = list()
fit$icov = list()
fit$cov.output = cov.output
if (cov.output)
  fit$cov = list()
out.glasso = NULL
for (i in nlambda:1) {
  z = which(rowSums(abs(S) > lambda[i]) > 1)
  q = length(z)
  if (q > 0) {
    if (verbose) {
      if (scr) {
        cat(paste(c("Conducting the graphical lasso (glasso) with lossy
screening....in progress:",
                    floor(100 * (1 - i/nlambda)), "%"), collapse = ""),
            "\r")
      }
    }
    if (!scr) {
      cat(paste(c("Conducting the graphical lasso (glasso) with lossless
screening....in progress:",
                  floor(100 * (1 - i/nlambda)), "%"), collapse = ""),
          "\r")
    }
  }
}

```

```

        "\r")
    }
    flush.console()
}
if (scr) {
  if (!is.null(out.glasso))
    out.glasso = .C("hugeglassoscr", S = as.double(S[z,
                                                    z]), W =
as.double(tmp.cov[z, z]), T = as.double(tmp.icov[z,
z]), dd = as.integer(q), lambda = as.double(lambda[i]),
df = as.integer(0), PACKAGE = "huge")
  if (is.null(out.glasso))
    out.glasso = .C("hugeglassoscr", S = as.double(S[z,
                                                    z]), W =
as.double(S[z, z]), T = as.double(diag(q)),
df = as.integer(q), lambda = as.double(lambda[i]),
df = as.integer(0), PACKAGE = "huge")
}
else {
  if (!is.null(out.glasso))
    out.glasso = .C("hugeglasso", S = as.double(S[z,
                                                    z]), W =
as.double(tmp.cov[z, z]), T = as.double(tmp.icov[z,
z]), dd = as.integer(q), lambda = as.double(lambda[i]),
df = as.integer(0), PACKAGE = "huge")
  if (is.null(out.glasso))
    out.glasso = .C("hugeglasso", S = as.double(S[z,
                                                    z]), W =
as.double(S[z, z]), T = as.double(diag(q)),
df = as.integer(q), lambda = as.double(lambda[i]),
df = as.integer(0), PACKAGE = "huge")
}
  out.glasso$T = matrix(out.glasso$T, ncol = q)
  out.glasso$W = matrix(out.glasso$W, ncol = q)
}
if (q == 0)
  out.glasso = NULL
tmp.icov = matrix(0, d, d)
diag(tmp.icov) = 1/(diag(S) + lambda[i])
tmp.cov = matrix(0, d, d)
diag(tmp.cov) = diag(S) + lambda[i]
fit$path[[i]] = Matrix(0, d, d)
if (!is.null(out.glasso)) {
  tmp.icov[z, z] = out.glasso$T
  tmp.cov[z, z] = out.glasso$W
  fit$path[[i]][z, z] = abs(sign(out.glasso$T))
  diag(fit$path[[i]]) = 0
  fit$sparsity[i] = as.double(out.glasso$df)/d/(d -
1)

  fit$df[i] = out.glasso$df/2
  fit$loglik[i] = (log(det(out.glasso$T)) - sum(diag(out.glasso$T %*%

```

```

q)
  }
  fit$icov[[i]] = Matrix(tmp.icov)
  if (cov.output)
    fit$cov[[i]] = Matrix(tmp.cov)
  }
  rm(S, out.glasso, tmp.cov, tmp.icov)
  gc()
  if (verbose) {
    cat("Conducting the graphical lasso (glasso)...done.
\r")
    cat("\n")
    flush.console()
  }
  return(fit)
}
#####
#####
#####

huge2.0<-function (x, lambda = NULL, nlambda = NULL, lambda.min.ratio = NULL,
  method = "mb", scr = NULL, scr.num = NULL, cov.output = FALSE,
  sym = "or", verbose = TRUE)
{
  gcinfo(FALSE)
  est = list()
  est$method = method
  if (method == "ct") {
    fit = huge.ct(x, nlambda = nlambda, lambda.min.ratio = lambda.min.ratio,
      lambda = lambda, verbose = verbose)
    est$path = fit$path
    est$lambda = fit$lambda
    est$sparsity = fit$sparsity
    est$cov.input = fit$cov.input
    rm(fit)
    gc()
  }
  if (method == "mb") {
    fit = huge.mb(x, lambda = lambda, nlambda = nlambda,
      lambda.min.ratio = lambda.min.ratio, scr = scr,
      scr.num = scr.num, sym = sym, verbose = verbose)
    est$path = fit$path
    est$lambda = fit$lambda
    est$sparsity = fit$sparsity
    est$df = fit$df
    est$idx_mat = fit$idx_mat
    est$sym = sym
    est$scr = fit$scr
    est$cov.input = fit$cov.input
    rm(fit, sym)
    gc()
  }
}

```



```

if (method == "glasso") {
  fit = huge.glasso.2.0(x, nlambda = nlambda, lambda.min.ratio =
lambda.min.ratio,
                      lambda = lambda, scr = scr, cov.output =
cov.output,
                      verbose = verbose)

  est$path = fit$path
  est$lambda = fit$lambda
  est$icov = fit$icov
  est$df = fit$df
  est$sparsity = fit$sparsity
  est$loglik = fit$loglik
  if (cov.output)
    est$cov = fit$cov
  est$cov.input = fit$cov.input
  est$cov.output = fit$cov.output
  est$scr = fit$scr
  rm(fit)
  gc()
}
est$data = x
rm(x, scr, lambda, lambda.min.ratio, nlambda, cov.output,
  verbose)
gc()
class(est) = "huge"
return(est)
}
#####
#####
#####

huge.select2.0<-function (est, criterion = NULL, ebic.gamma = 0.5,
stars.thresh = 0.1,
                        stars.subsample.ratio = NULL, rep.num = 20, verbose
= TRUE)
{
  gcinfo(FALSE)
  if (est$cov.input) {
    cat("Model selection is not available when using the covariance matrix as
input.")
    class(est) = "select"
    return(est)
  }
  if (!est$cov.input) {
    if (est$method == "mb" && is.null(criterion))
      criterion = "ric"
    if (est$method == "ct" && is.null(criterion))
      criterion = "stars"
    if (est$method == "glasso" && is.null(criterion))
      criterion = "ebic"
    n = nrow(est$data)
    d = ncol(est$data)
    nlambda = length(est$lambda)

```



```

est$opt.sparsity = sum(est$refit)/d/(d - 1)
if (verbose) {
  cat("done\n")
  flush.console()
}
}
if (criterion == "ebic" && est$method == "glasso") {
  if (verbose) {
    cat("Conducting extended Bayesian information criterion (ebic)
selection....")
    flush.console()
  }
  est$ebic.score = -n * est$loglik + log(n) * est$df +
    4 * ebic.gamma * log(d) * est$df
  est$opt.index = which.min(est$ebic.score)
  est$refit = est$path[[est$opt.index]]
  est$opt.icov = est$icov[[est$opt.index]]
  if (est$cov.output)
    est$opt.cov = est$cov[[est$opt.index]]
  est$opt.lambda = est$lambda[est$opt.index]
  est$opt.sparsity = est$sparsity[est$opt.index]
  if (verbose) {
    cat("done\n")
    flush.console()
  }
}
}
if (criterion == "stars") {
  if (is.null(stars.subsample.ratio)) {
    if (n > 144)
      stars.subsample.ratio = 10 * sqrt(n)/n
    if (n <= 144)
      stars.subsample.ratio = 0.8
  }
  est$merge = list()
  for (i in 1:nlambda) est$merge[[i]] = Matrix(0,
                                                d, d)
  for (i in 1:rep.num) {
    if (verbose) {
      mes <- paste(c("Conducting Subsampling....in progress:",
                    floor(100 * i/rep.num), "%"), collapse = "")
      cat(mes, "\r")
      flush.console()
    }
    ind.sample = sample(c(1:n), floor(n * stars.subsample.ratio),
                      replace = FALSE)
    if (est$method == "mb")
      tmp = huge.mb(est$data[ind.sample, ], lambda = est$lambda,
                   scr = est$scr, idx.mat = est$idx.mat, sym = est$sym,
                   verbose = FALSE)$path
    if (est$method == "ct")
      tmp = huge.ct(est$data[ind.sample, ], lambda = est$lambda,
                   verbose = FALSE)$path
    if (est$method == "glasso")

```

```

        tmp = huge.glasso.2.0(est$data[ind.sample, ],
                             lambda = est$lambda, scr = est$scr, verbose =
FALSE)$path
        for (i in 1:nlambda) est$merge[[i]] = est$merge[[i]] +
            tmp[[i]]
        rm(ind.sample, tmp)
        gc()
    }
    if (verbose) {
        mes = "Conducting Subsampling....done."
        cat(mes, "\r")
        cat("\n")
        flush.console()
    }
    est$variability = rep(0, nlambda)
    for (i in 1:nlambda) {
        est$merge[[i]] = est$merge[[i]]/rep.num
        est$variability[i] = 4 * sum(est$merge[[i]] *
            (1 - est$merge[[i]]))/(d * (d - 1))
    }
    est$opt.index = max(which.max(est$variability >=
        stars.thresh)[1] - 1, 1)
    est$refit = est$path[[est$opt.index]]
    est$opt.lambda = est$lambda[est$opt.index]
    est$opt.sparsity = est$sparsity[est$opt.index]
    if (est$method == "glasso") {
        est$opt.icov = est$icov[[est$opt.index]]
        if (!is.null(est$cov))
            est$opt.cov = est$cov[[est$opt.index]]
    }
}
est$criterion = criterion
class(est) = "select"
return(est)
}
}

```

```

#####
#####

```

```

huge.npn.2.0<-function (x, npn.func = "shrinkage", npn.thresh = NULL, verbose
= TRUE)
{
    gcinfo(FALSE)
    n = nrow(x)
    d = ncol(x)
    x.col = colnames(x)
    x.row = rownames(x)
    if (npn.func == "shrinkage") {
        if (verbose)
            cat("Conducting the nonparanormal (npn) transformation via shrunkun
ECDF....")
        x = qnorm(apply(x, 2, rank)/(n + 1))
    }
}

```

```

x = x/sd(x[, 1])
if (verbose)
  cat("done.\n")
rm(n, d, verbose)
gc()
colnames(x) = x.col
rownames(x) = x.row
}
if (nfn.func == "truncation") {
  if (verbose)
    cat("Conducting nonparanormal (nfn) transformation via truncated
ECDF....")
  if (is.null(nfn.thresh))
    nfn.thresh = 1/(4 * (n^0.25) * sqrt(pi * log(n)))
  x = qnorm(pmin(pmax(apply(x, 2, rank)/n, nfn.thresh),
                1 - nfn.thresh))
  x = x/sd(x[, 1])
  if (verbose)
    cat("done.\n")
  rm(n, d, nfn.thresh, verbose)
  gc()
  colnames(x) = x.col
  rownames(x) = x.row
}
if (nfn.func == "skeptical") {
  if (verbose)
    cat("Conducting nonparanormal (nfn) transformation via skeptical....")
  x = 2 * sin(pi/6 * cor(x, method = "spearman"))
  if (verbose)
    cat("done.\n")
  rm(n, d, verbose)
  gc()
  colnames(x) = x.col
  rownames(x) = x.col
}
}
return(x)
}
#####

```

A.7) glQDA function code.

```

my.QDA<-function
(data.train,data.test,gl=TRUE,lam=0,crit1="ebic",eb.gam=0.5,stars.th=0.1)
{
  # k is the number of classes
  # data needs to be in form with classes as first variable
  library(huge)
  n<-nrow(data.train)
  p<-ncol(data.train)-1
  nt<-nrow(data.test)

```

```

Groups<-unique(data.train[,1])
k<-length(Groups)
pro.list<-vector("list",k)
Group.list<-vector("list",k)
#train
disc.fun.train<-numeric()
#test
disc.fun.test<-numeric()

test.resp<-data.test[,1]
test.in<-data.test[,-1]

for(i in 1:k)
{
  pro.list[[i]] <-sum(Groups[i]==data.train[,1])/n
  Group.list[[i]]<-data.train[data.train[,1]==Groups[i],-1]
}

mean.list<-lapply(Group.list,function(x) (apply(x,2,mean)))
var.list<-lapply(Group.list,function(x) (apply(x,2,sd)))
Group.list<-lapply(Group.list,function(x) scale(x))

if(gl==TRUE)
{
  if(lam==0)
    cov.list<-lapply(Group.list,function(x) {

      mod1<-huge2.0(as.matrix(x),nlambda=100,method="glasso",cov.output =
TRUE)
      out.select <- huge.select2.0(mod1,criterion = crit1,ebic.gamma
=eb.gam,stars.thresh=stars.th)
      return(list(out.select$opt.cov,out.select$opt.lambda))

    })

  if(lam>0)
    cov.list<-lapply(Group.list,function(x) {
      mod1<-huge2.0(as.matrix(x),lambda=lam,method="glasso",cov.output =
TRUE)

      return(list(mod1$cov[[1]],lam))
    })
}

if(gl==FALSE)
{
  cov.list<-lapply(Group.list,function(x) {

```

```

    lam<-0
    return(list(cov(x),lam))
  })

}
lam.vec<-numeric()

#Train
predict.train<-numeric()
for(j in 1:n)
{
  print("Train")
  print(j)
  for(i in 1:k)

  {

    #Train
    disc.fun.train[i]<- -0.5*log(det(cov.list[[i]][[1]]))-
0.5*(matrix(data.train[j,-1]-
mean.list[[i],nrow=1])%*%solve(cov.list[[i]][[1]])%*%t(matrix(data.train[j,-
1]-mean.list[[i],nrow=1)))+log(pro.list[[i]])

    if(j==1)
    {
      lam.vec[i]<- cov.list[[i]][[2]]
    }

  }

  predict.train[j]<-Groups[which.max(disc.fun.train)]

}

#Test
predict.test<-numeric()

for(j in 1:nt)
{
  print("Test")
  print(j)

  for(i in 1:k)

  {

    disc.fun.test[i]<- -0.5*log(det(cov.list[[i]][[1]]))-
0.5*(matrix(test.in[j,-
mean.list[[i],nrow=1])%*%solve(cov.list[[i]][[1]])%*%t(matrix(test.in[j,-
mean.list[[i],nrow=1)))+log(pro.list[[i]])
  }
}

```

```

    predict.test[j]<-Groups[which.max(disc.fun.test)]
  }

  list("train.error"=1-sum(predict.train==data.train[,1])/n,"test.error"=1-
sum(predict.test==test.resp)/length(test.resp),lam.vec)
}

```

A.8) gINPDA function code.

```

my.QDA.NPN<-function
(bw="none",data.train,data.test,gl=TRUE,lam=0,crit1="ebic",trun.par=NULL,eb.g
am=0.5,stars.th=0.1)
{
  # k is the number of classes
  # data needs to be in form with classes as first variable
  library(huge)
  n<-nrow(data.train)
  p<-ncol(data.train)-1
  nt<-nrow(data.test)

  Groups<-unique(data.train[,1])
  k<-length(Groups)
  pro.list<-vector("list",k)
  Group.list<-vector("list",k)
  #train
  disc.fun.train<-numeric()
  #test
  disc.fun.test<-numeric()

  #Split test in inputs and response
  test.resp<-data.test[,1]
  test.in<-data.test[,-1]

  for(i in 1:k)
  {
    pro.list[[i]] <-sum(Groups[i]==data.train[,1])/n
    Group.list[[i]]<-data.train[data.train[,1]==Groups[i],-1]
  }

  data.trans<-lapply(Group.list,function(x)
  {
    out1<-apply(x,2,function(j){

      sd1<-sqrt(var(j))
      mu1<-mean(j)
      n<-length(j)
      cum_d<-numeric()

```



```

if(bw=="none")
{bw=((4*sd1^5)/(3*n))^(1/5)}

for(i in 1:n)

{
  cum_d[i]<-sum(pnorm((j[i]-j),sd=bw))/n
}

tr.dat<-qnorm(cum_d)*sd1+mu1
return((tr.dat))})

return(out1)
})

test.data.trans<-lapply(Group.list,function(x)
{
  mat.test<-matrix(0,ncol=p,nrow=nrow(test.in))
  trans.form.der<-matrix(0,ncol=p,nrow=nrow(test.in))

  for(j in 1:ncol(x))
  {
    sd1<-sqrt(var(x[,j]))
    mu1<-mean(x[,j])
    test.varb<-test.in[,j]
    n<-length(x[,j])
    cum_d<-numeric()
    dens_d<-numeric()
    if(bw=="none")
    {bw=((4*sd1^5)/(3*n))^(1/5)}

    for(i in 1:nt)

    {
      dens_d[i]<-sum(dnorm((test.varb[i]-x[,j]),sd=bw))/n
      cum_d[i]<-sum(pnorm((test.varb[i]-x[,j]),sd=bw))/n
    }

    cum_d[cum_d>0.999]=0.999
    cum_d[cum_d<0.001]=0.001

    tr.dat<-qnorm(cum_d)*sd1+mu1

    trans.form.der[,j]<-sd1*dens_d/dnorm(qnorm(cum_d))
    mat.test[,j]<-tr.dat
  }
}

```

```

    }
    return(list(mat.test,trans.form.der))
})

mean.list<-lapply(Group.list,function(x) (apply(x,2,mean)))

if(gl==TRUE)
{
  if(lam==0)
    cov.list<-lapply(data.trans,function(x) {
      mod1<-huge2.0(as.matrix(x),nlambda = 100,method="glasso",cov.output =
TRUE)
      out.select <- huge.select2.0(mod1,criterion = crit1,ebic.gamma
=eb.gam,stars.thresh=stars.th)
      return(list(out.select$opt.cov,out.select$opt.lambda))
    })

  if(lam>0)
    cov.list<-lapply(data.trans,function(x) {
      mod1<-huge2.0(as.matrix(x),lambda=lam,method="glasso",cov.output =
TRUE,scr=T)

      return(list(mod1$cov[[1]],lam))
    })
}

if(gl==FALSE)
{
  cov.list<-lapply(data.trans,function(x) {

    lam<-0
    return(list(cov(x),lam))
  })
}

lam.vec<-numeric()

#Test
predict.test<-numeric()

for(j in 1:nt)
{

  for(i in 1:k)
  {

```

```

    disc.fun.test[i]<- -0.5*log(det(cov.list[[i]][[1]])) -
0.5*(matrix(test.data.trans[[i]][[1]][j,]-
mean.list[[i]],nrow=1)%*%solve(cov.list[[i]][[1]])%*%t(matrix(test.data.trans
[[i]][[1]][j,]-mean.list[[i]],nrow=1)))+
    log(pro.list[[i]])+sum(log(abs(test.data.trans[[i]][[2]][j,])))
    lam.vec[i]<-cov.list[[i]][[2]]

  }

  predict.test[j]<-Groups[which.max(disc.fun.test)]
}

list("train.error"=0,"test.error"=1-
sum(predict.test==test.resp)/length(test.resp),lam.vec)
}

```

A.9) gIQDA application to vowel data with different methods for selecting the tuning parameter.

```

library(ElemStatLearn)

train.dat<-as.matrix(vowel.train)
test.dat<-as.matrix(vowel.test)

out1<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=F,lam=0,crit1="ebic",eb.gam=0.5)
out2<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=0.5)

#QDA glasso ebic cv and 1sd cv
#####
gam.vec<-seq(from=0,to=1,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-cv.vec<-c(110,110,110,110,88)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {

```

```

    print(i)
    error.mat[i,j]<-
my.QDA(traincv,testcv,gl=T,crit1="ebic",eb.gam=gam.vec[i],lam=0)$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])
opt.gam2<-gam.vec[opt.pos]

opt.gam<-gam.vec[which.min(cv.error)]
#min cv errr
out3<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=opt.gam)

#1sd
out4<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=opt.gam2)
#####
#####

#QDA glasso Stars=0.1
out5<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="stars",stars.th=0.1)

#QDA glasso Stars=0.05
out6<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="stars",stars.th=0.05)

#Cross validation lambda
#####
###
lam.vec<-rev(seq(from=0.01,to=2,length=100))
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-cv.vec<-c(110,110,110,110,88)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)

```

```

{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    print(i)
    error.mat[i,j]<-my.QDA(traincv,testcv,gl=T,lam=lam.vec[i])$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-min(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])

#CV min
opt.lam<-lam.vec[which.min(cv.error)]
out7<-my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.lam)

#CV 1sd
opt.lam2<-lam.vec[opt.pos]
out8<-my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.lam2)

```

A.10) gINPDA application to vowel data with different methods for selecting the tuning parameter.

```

library(ElemStatLearn)

train.dat<-as.matrix(vowel.train)
test.dat<-as.matrix(vowel.test)

out1.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=F,lam=0,crit
1="ebic",eb.gam=0.5)
out2.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="ebic",eb.gam=0.5)

#QDA glasso ebic cv and 1sd cv
#####
gam.vec<-seq(from=0,to=1,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-cv.vec<-c(110,110,110,110,88)

```

```

folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    print(i)
    error.mat[i,j]<-
my.QDA.NPN(bw="none",traincv,testcv,gl=T,crit1="ebic",eb.gam=gam.vec[i],lam=0
)$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])
opt.gam2<-gam.vec[opt.pos]

opt.gam<-gam.vec[which.min(cv.error)]
#min cv errr
out3.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="ebic",eb.gam=opt.gam)

#1sd
out4.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="ebic",eb.gam=opt.gam2)
#####
#####

#QDA glasso Stars=0.1
out5.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="stars",stars.th=0.1)

#QDA glasso Stars=0.05
out6.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="stars",stars.th=0.05)

```

```

#Cross validation lambda
#####
###
lam.vec<-rev(seq(from=0.01,to=2,length=100))
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-cv.vec<-c(110,110,110,110,88)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    print(i)
    error.mat[i,j]<-
my.QDA.NPN(bw="none",traincv,testcv,gl=T,lam=lam.vec[i])$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-min(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])

#CV min
opt.lam<-lam.vec[which.min(cv.error)]
out7.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.la
m)

#CV 1sd
opt.lam2<-lam.vec[opt.pos]
out8.NPN<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.la
m2)

#####

#Cross validation lambda and Band Width
#####
###
BW.vec<-seq(0.02,3,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

```

```

cv.vec<-c(110,110,110,110,88)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  print('opt bw')
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {print(j)
    print(i)
    error.mat[i,j]<-my.QDA.NPN(bw=BW.vec[i],traincv,testcv,gl=T,lam=0)$test
  }
}
cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])
opt.bw<-BW.vec[which.min(cv.error)]
opt.bw2 =BW.vec[opt.pos]

out9.NPN<-
my.QDA.NPN(bw=opt.bw,as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0)

out10.NPN<-
my.QDA.NPN(bw=opt.bw2,as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0)

```

A.11) gIQDA application to zip code data with different methods for selecting the tuning parameter.

```

library(ElemStatLearn)

train.dat <- as.matrix(zip.train)
test.dat <- as.matrix(zip.test)

#QDA glasso lambda=0
out1<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=F,lam=0,crit1="ebic")

#QDA glasso ebic=0.5

```



```

out2<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=0.5)

#QDA glasso ebic cv and lsd cv
#####
gam.vec<-seq(from=0,to=1,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(1458,1458,1458,1458,1459)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    print(i)
    error.mat[i,j]<-
my.QDA(traincv,testcv,gl=T,crit1="ebic",eb.gam=gam.vec[i],lam=0)$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])
opt.gam2<-gam.vec[opt.pos]

opt.gam<-gam.vec[which.min(cv.error)]
#min cv errr
out3<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=opt.gam)

#lsd
out4<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=opt.gam2)

#####
#####

#QDA glasso Stars=0.1

```

```

out5<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="stars",star
s.th=0.1)

#QDA glasso Stars=0.05
out6<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="stars",star
s.th=0.05)

#Cross validation lambda

#####
###
lam.vec<-rev(seq(from=0.01,to=1,length=100))
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(1458,1458,1458,1458,1459)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    print(i)
    error.mat[i,j]<-my.QDA(traincv,testcv,gl=T,lam=lam.vec[i])$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-min(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])

#CV min
opt.lam<-lam.vec[which.min(cv.error)]
out7<-my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.lam)

#CV 1sd
opt.lam2<-lam.vec[opt.pos]
out8<-my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.lam2)

```

A.12) gINPDA application to zip code data with different methods for selecting the tuning parameter.

```

library(ElemStatLearn)

train.dat <- as.matrix(zip.train)
test.dat <- as.matrix(zip.test)

data_list = my.data_transf(bw='none',train.dat,test.dat)

out1.NPN<-
My.QDA.NPN(as.matrix(zip.train),as.matrix(zip.test),data.trans=data_list[[1]]
,test.data.trans=data_list[[2]],gl=F,lam=0,crit1="ebic",eb.gam=0.5)
out2.NPN<-
My.QDA.NPN(as.matrix(zip.train),as.matrix(zip.test),data.trans=data_list[[1]]
,test.data.trans=data_list[[2]],gl=T,lam=0,crit1="ebic",eb.gam=0.5)

#QDA glasso ebic cv and lsd cv
#####
gam.vec<-seq(from=0,to=1,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(1458,1458,1458,1458,1459)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)
data_list_cv <- vector('list',5)

for(j in 1:5)
{ print(j)
  print("optbic")
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]
  data_list_cv[[j]] = my.data_transf(bw='none',traincv,testcv)

  for(i in 1:100)
  {

    error.mat[i,j]<-
My.QDA.NPN(traincv,testcv,data.trans=data_list_cv[[j]][[1]],test.data.trans=d
ata_list_cv[[j]][[2]],
          gl=T,lam=0,crit1="ebic",eb.gam=gam.vec[i])$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

```

```

opt.pos<-max(c(1:length(cv.error)) [cv.error<int [2] & cv.error>int [1]])
opt.gam2<-gam.vec[opt.pos]

opt.gam<-gam.vec[which.min(cv.error)]
#min cv errr

out3.NPN<-
My.QDA.NPN(as.matrix(zip.train), as.matrix(zip.test), data.trans=data_list[[1]]
, test.data.trans=data_list[[2]], gl=TRUE, lam=0, crit1="ebic", eb.gam=opt.gam, stars.th=0.1)

#1sd
out4.NPN<-
My.QDA.NPN(as.matrix(zip.train), as.matrix(zip.test), data.trans=data_list[[1]]
, test.data.trans=data_list[[2]], gl=TRUE, lam=0, crit1="ebic", eb.gam=opt.gam2, stars.th=0.1)

#####
#####

#QDA glasso Stars=0.1
out5.NPN<-
My.QDA.NPN(as.matrix(zip.train), as.matrix(zip.test), data.trans=data_list[[1]]
, test.data.trans=data_list[[2]], gl=TRUE, lam=0, crit1="stars", stars.th=0.1)

#QDA glasso Stars=0.05
out6.NPN<-
My.QDA.NPN(as.matrix(zip.train), as.matrix(zip.test), data.trans=data_list[[1]]
, test.data.trans=data_list[[2]], gl=TRUE, lam=0, crit1="stars", stars.th=0.05)

#Cross validation lambda

#####
###
lam.vec<-rev(seq(from=0.01, to=2, length=100))
error.mat<-matrix(0, ncol=5, nrow=100)

cv.vec<-c(1458, 1458, 1458, 1458, 1459)
folds.up<-cumsum(cv.vec)
folds.low<-c(1, cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  print('opt lam cv')
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {

```

```

    error.mat[i,j]<-
My.QDA.NPN(traincv,testcv,data.trans=data_list_cv[[j]][[1]],test.data.trans=d
ata_list_cv[[j]][[2]],
                                gl=T,lam=lam.vec[i])$test
}
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)
min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-min(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])

#CV min
opt.lam<-lam.vec[which.min(cv.error)]
out7.NPN<-
My.QDA.NPN(as.matrix(zip.train),as.matrix(zip.test),data.trans=data_list[[1]]
,test.data.trans=data_list[[2]],gl=TRUE,lam=opt.lam)

#CV 1sd
opt.lam2<-lam.vec[opt.pos]
out8.NPN<-
My.QDA.NPN(as.matrix(zip.train),as.matrix(zip.test),data.trans=data_list[[1]]
,test.data.trans=data_list[[2]],gl=TRUE,lam=opt.lam2)

#Cross validation lambda and Band Width
#####
###
BW.vec<-seq(0.02,3,length=10)
error.mat<-matrix(0,ncol=5,nrow=10)

cv.vec<-c(1458,1458,1458,1458,1459)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  print('opt bw')
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:10)
  {
    error.mat[i,j]<-my.QDA.NPN(bw=BW.vec[i],traincv,testcv,gl=T,lam=0)$test
  }
}
cv.error<-apply(error.mat,1,mean)

```

```

cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])
opt.bw<-BW.vec[which.min(cv.error)]
opt.bw2 =BW.vec[opt.pos]

out9.NPN<-
my.QDA.NPN(bw=opt.bw,as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0)

out10.NPN<-
my.QDA.NPN(bw=opt.bw2,as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0)

```

A.13) gIQDA application to spam data with different methods for selecting the tuning parameter and 5 fold cross-validation.

```

library(ElemStatLearn)

all_data = as.matrix(spam)

#Changing predictor column to 1's and 0's
# spam=1 email=0

all_data[all_data[,"spam"]=="spam","spam"]=1
all_data[all_data[,"spam"]=="email","spam"]=0

all_data = cbind(as.numeric(all_data[,"spam"]),as.matrix(spam[, -
ncol(all_data)]))

#Splitting data into test and train
cv.gen<-c(920,920,920,920,921)
folds_gen_up<-cumsum(cv.gen)
folds_gen_low<-c(1,cumsum(cv.gen)[-5]+1)
set.seed(1993)
random.shuffel = sample(c(1:4601), size=4601, replace=FALSE)
all_data = all_data[random.shuffel, ]

out1 <- vector("list",5)
out2 <- vector("list",5)
out3 <- vector("list",5)
out4 <- vector("list",5)
out5 <- vector("list",5)
out6 <- vector("list",5)
out7 <- vector("list",5)
out8 <- vector("list",5)
opt.gam <- vector("list",5)

```

```

opt.gam2 <- vector("list",5)

for (fold in 1:5)
{
train.dat<-all_data[-c(folds_gen_low[fold]:folds_gen_up[fold]), ]
test.dat<-all_data[c(folds_gen_low[fold]:folds_gen_up[fold]), ]

#QDA glasso lambda=0
out1[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=F,lam=0,crit1="ebic")

#QDA glasso ebic=0.5
out2[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=0.5)

#QDA glasso ebic cv and lsd cv
#####
gam.vec<-seq(from=0,to=1,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(184,184,184,184,184)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    print(i)
    error.mat[i,j]<-
my.QDA(traincv,testcv,gl=T,crit1="ebic",eb.gam=gam.vec[i],lam=0)$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])
opt.gam2[[fold]]<-gam.vec[opt.pos]

opt.gam[[fold]]<-gam.vec[which.min(cv.error)]
#min cv errr

```

```

out3[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=opt.gam[[fold]])

#1sd
out4[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="ebic",eb.gam=opt.gam2[[fold]])
#####
#####

#QDA glasso Stars=0.1
out5[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="stars",stars.th=0.1)

#QDA glasso Stars=0.05
out6[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit1="stars",stars.th=0.05)

#Cross validation lambda
#####
###
lam.vec<-rev(seq(from=0.01,to=1,length=100))
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(184,184,184,184,184)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    print(i)
    error.mat[i,j]<-my.QDA(traincv,testcv,gl=T,lam=lam.vec[i])$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

```



```

opt.pos<-min(c(1:length(cv.error)) [cv.error<int [2] &cv.error>int [1]])

#CV min
opt.lam<-lam.vec[which.min(cv.error)]
out7[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.lam)

#CV 1sd
opt.lam2<-lam.vec[opt.pos]
out8[[fold]]<-
my.QDA(as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.lam2)
}

```

A.14) gINPDA application to spam data with different methods for selecting the tuning parameter and 5 fold cross-validation.

```

library(ElemStatLearn)

all_data = as.matrix(spam)

#Changing predictor column to 1's and 0's
# spam=1 email=0

all_data[all_data[,"spam"]=="spam","spam"]=1
all_data[all_data[,"spam"]=="email","spam"]=0

all_data = cbind(as.numeric(all_data[,"spam"]),as.matrix(spam[, -
ncol(all_data)]))

#Splitting data into test and train
cv.gen<-c(920,920,920,920,921)
folds_gen_up<-cumsum(cv.gen)
folds_gen_low<-c(1,cumsum(cv.gen)[-5]+1)
set.seed(1993)
random.shuffel = sample(c(1:4601), size=4601, replace=FALSE)
all_data = all_data[random.shuffel, ]

out1.NPN <- vector("list",5)
out2.NPN <- vector("list",5)
out3.NPN <- vector("list",5)
out4.NPN <- vector("list",5)
out5.NPN <- vector("list",5)
out6.NPN <- vector("list",5)
out7.NPN <- vector("list",5)
out8.NPN <- vector("list",5)
out9.NPN <- vector("list",5)
out10.NPN <- vector("list",5)
opt.bw<- vector("list",5)
opt.bw2<- vector("list",5)
opt.gam <- vector("list",5)

```

```

opt.gam2 <- vector("list",5)

for (fold in 1:5)
{
  train.dat<-all_data[-c(folds_gen_low[fold]:folds_gen_up[fold]), ]
  test.dat<-all_data[c(folds_gen_low[fold]:folds_gen_up[fold]), ]

  out1.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=F,lam=0,crit
1="ebic",eb.gam=0.5)
  out2.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="ebic",eb.gam=0.5)

#QDA glasso ebic cv and lsd cv
#####
gam.vec<-seq(from=0,to=1,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(184,184,184,184,184)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  print("optbic")
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {

    error.mat[i,j]<-
my.QDA.NPN(bw="none",traincv,testcv,gl=T,crit1="ebic",eb.gam=gam.vec[i],lam=0
)$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error))[cv.error<int[2]&cv.error>int[1]])
opt.gam2[[fold]]<-gam.vec[opt.pos]

opt.gam[[fold]]<-gam.vec[which.min(cv.error)]
#min cv errr

```

```

out3.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="ebic",eb.gam=opt.gam[[fold]])

#1sd
out4.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="ebic",eb.gam=opt.gam2[[fold]])
#####
#####

#QDA glasso Stars=0.1
out5.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="stars",stars.th=0.1)

#QDA glasso Stars=0.05
out6.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=0,crit
1="stars",stars.th=0.05)

#Cross validation lambda
#####
###
lam.vec<-rev(seq(from=0.01,to=2,length=100))
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(184,184,184,184,184)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  print('opt lam cv')
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    error.mat[i,j]<-
my.QDA.NPN(bw="none",traincv,testcv,gl=T,lam=lam.vec[i])$test
  }
}

cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)
min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

```

```

opt.pos<-min(c(1:length(cv.error)) [cv.error<int[2]&cv.error>int[1]])

#CV min
opt.lam<-lam.vec[which.min(cv.error)]
out7.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.la
m)

#CV 1sd
opt.lam2<-lam.vec[opt.pos]
out8.NPN[[fold]]<-
my.QDA.NPN(bw="none",as.matrix(train.dat),as.matrix(test.dat),gl=T,lam=opt.la
m2)

#Cross validation lambda and Band Width
#####
###
BW.vec<-seq(0.02,3,length=100)
error.mat<-matrix(0,ncol=5,nrow=100)

cv.vec<-c(184,184,184,184,184)
folds.up<-cumsum(cv.vec)
folds.low<-c(1,cumsum(cv.vec)[-5]+1)

for(j in 1:5)
{ print(j)
  print('opt bw')
  traincv<-train.dat[-c(folds.low[j]:folds.up[j]),]
  testcv<-train.dat[c(folds.low[j]:folds.up[j]),]

  for(i in 1:100)
  {
    error.mat[i,j]<-my.QDA.NPN(bw=BW.vec[i],traincv,testcv,gl=T,lam=0)$test
  }
}
cv.error<-apply(error.mat,1,mean)
cv.se<-apply(error.mat,1,sd)

min.error<-cv.error[which.min(cv.error)]
min.se<-cv.se[which.min(cv.error)]

int<-c(min.error-min.se,min.error+min.se)

opt.pos<-max(c(1:length(cv.error)) [cv.error<int[2]&cv.error>int[1]])
opt.bw[[fold]]<-BW.vec[which.min(cv.error)]
opt.bw2[[fold]] <- BW.vec[opt.pos]

out9.NPN[[fold]]<-
my.QDA.NPN(bw=opt.bw[[fold]],as.matrix(train.dat),as.matrix(test.dat),gl=T,la
m=0)

```

```
out10.NPN[[fold]]<-  
my.QDA.NPN(bw=opt.bw2[[fold]],as.matrix(train.dat),as.matrix(test.dat),gl=T,l  
am=0)  
}
```