



THE UNIVERSITY OF QUEENSLAND  
AUSTRALIA

The University of Queensland  
Faculty of Business, Economics & Law  
School of Economics

# **High-dimensional mixture-based discriminant analysis for binary data**

Estimated words:15000

An Honours Thesis submitted to the School of Economics,  
The University of Queensland,  
in partial fulfillment of the requirements for the degree of BEcon(Honours)

**By Mengfan Long**  
Bachelor of Economics

October 2023

**Supervisor:Dr Mohamad Khaled**

## **Acknowledgments**

Firstly, I would like to express my heartfelt appreciation to my supervisor, Dr. Mohamad Khaled, who has provided me with huge support throughout the entire project. I am deeply grateful for his dedication, the time he invested, and his patience with me. I feel privileged to have had the opportunity to collaborate with him. He is one of the most remarkable people I know, and I wish him all the best in his future pursuits.

Additionally, I would like to express my heartfelt gratitude to my family and friends for their invaluable support and encouragement throughout the entire process. Their unwavering belief in me and their willingness to provide assistance have been instrumental in my journey, and I am deeply appreciative of their contributions.

And it was also a delightful experience getting to know all the honors cohort. I wish everyone the best of luck.

## **Declaration Statement**

I declare that the work presented in this Honours thesis is, to the best of my knowledge and belief, original and my own work, except as acknowledged in the text, and that material has not been submitted, either in whole or in part, for a degree at this or any other university.

.....

Mengfan Long

27/10/2023

## Abstract

Binary choice models play a pivotal role in econometrics and applied economics due to their wide applicability and the ubiquitous presence of binary dependent variables in applied research more generally.

Discriminant analysis is a major approach for binary choice models, where the joint distribution of the covariates is used in order to compute posterior probabilities of the outcome variable through Bayes rule. However, the traditional approaches in discriminant analysis (Linear and Quadratic Discriminant Analysis, known respectively as LDA and QDA) suffer from major drawbacks: they both assume the covariate distribution to be normal, which may be too restrictive in real applications. They further suffer from poor performance in high-dimensional settings, or more generally in settings where the number of variables is too high compared to the sample size.

In light of these limitations, this thesis offers a novel model which overcomes both limitations, and also performs well in low-dimensional settings. The model the thesis introduces builds on very recent research on high-dimensional estimation of mixture models. Exploiting the CHIME (Clustering of high-dimensional Gaussian mixtures with EM) algorithm, the thesis introduces a novel and fast algorithm for fitting binary choice models in high dimensional settings.

Subsequently, the thesis applies this model to both high-dimensional data simulations and real data examples. We show that the model has very good performance in different scenarios and does outperform traditional approaches such as high dimensional logistic regression (with L1 penalty) and sparse linear discriminant analysis which is a sparse version of LDA using lasso penalty.

# Contents

<b>Contents</b>	<b>4</b>
<b>List of Figures</b>	<b>7</b>
<b>1 Introduction</b>	<b>8</b>
1.1 Motivation and Objectives . . . . .	8
1.2 Organization of the Thesis . . . . .	9
<b>2 Binary Choice model</b>	<b>10</b>
2.1 Definition . . . . .	10
2.2 Discriminant analysis . . . . .	10
2.2.1 Linear Discriminant Analysis . . . . .	10
2.2.2 Quadratic Discriminant Analysis (QDA) . . . . .	11
2.3 Limitations of previous discriminant analysis methods . . . . .	12
2.3.1 Assumption of Normality . . . . .	12
2.3.2 High-Dimensional Data . . . . .	12
2.4 Solutions . . . . .	12
2.4.1 Solutions for high-dimension data . . . . .	12
2.4.2 Solution of this thesis . . . . .	13
<b>3 Mixture model</b>	<b>14</b>
3.1 Overview . . . . .	14
3.1.1 Definition . . . . .	14
3.1.2 Advantages of Gaussian Mixture Models . . . . .	14
3.1.3 Probability Density Function of Mixture model . . . . .	14
3.1.4 Key Components in a Finite Mixture Model . . . . .	15
3.2 Issues of mixture model density estimation . . . . .	16
3.2.1 Uncertainty in component assignment . . . . .	16
3.2.2 Clustering . . . . .	16
3.2.3 Clustering method . . . . .	16

3.3	EM algorithm . . . . .	17
3.3.1	Overview . . . . .	17
3.3.2	Advantages of the EM algorithm . . . . .	17
3.3.3	Background of EM . . . . .	18
3.3.4	Whole Process . . . . .	20
3.4	Discriminant function based on mixture model . . . . .	22
3.4.1	Function . . . . .	22
<b>4</b>	<b>CHIME</b>	<b>24</b>
4.1	High-Dimensional Data Characteristics . . . . .	24
4.2	Clustering Challenge of High-dimension data . . . . .	24
4.2.1	Problems of Fisher's Linear Discriminant Rule . . . . .	24
4.2.2	Problems of EM in Clustering . . . . .	25
4.2.3	High-dimension clustering methods . . . . .	26
4.3	High-dimension classification . . . . .	27
4.3.1	Problems . . . . .	27
4.3.2	Common High-Dimension Classification Methods . . . . .	27
4.4	Overview of CHIME . . . . .	28
4.4.1	Definition . . . . .	28
4.4.2	Discriminant Vectors $\beta^*$ . . . . .	28
4.5	CHIME algorithm . . . . .	30
4.5.1	Overview . . . . .	30
4.5.2	Initiation . . . . .	30
4.5.3	Initial of $\beta^*$ . . . . .	30
4.5.4	EM steps of CHIME . . . . .	33
4.5.5	Output and Classification . . . . .	35
4.6	Advantage of CHIME . . . . .	36
4.6.1	Theoretical Optimality . . . . .	36
4.6.2	Dimensionality Reduction . . . . .	36
4.6.3	Sparsity Requirement and Better Generalization . . . . .	36
4.7	CHIME for high-dimension classification . . . . .	37

4.7.1	Overview . . . . .	37
4.7.2	Steps . . . . .	37
<b>5</b>	<b>Simulation on CHIME Clustering and Classification</b>	<b>39</b>
5.1	Overview . . . . .	39
5.2	Model . . . . .	39
5.2.1	Model 1 . . . . .	39
5.2.2	Model 2 . . . . .	40
5.2.3	Model 3 . . . . .	40
5.3	Comparison of Classifiers . . . . .	41
5.3.1	Results . . . . .	41
5.4	Conclusion . . . . .	45
<b>6</b>	<b>Real Data Analysis</b>	<b>46</b>
6.1	Overview . . . . .	46
6.2	Data . . . . .	46
6.2.1	Dataset Description . . . . .	46
6.3	Results . . . . .	47
6.4	Conclusion . . . . .	47
<b>7</b>	<b>Conclusion</b>	<b>48</b>
7.1	Summary . . . . .	48
7.2	Possible Future Directions for Research . . . . .	49
7.2.1	Automatic Component Selection in Mixture Models . . . . .	49
7.2.2	Extending Mixture Models to Discrete Variables with Copulas . . . . .	49
7.2.3	Handling Heavy-Tailed Distributions with Multivariate Mixtures . . . . .	49
	<b>References</b>	<b>50</b>
<b>8</b>	<b>Appendix</b>	<b>54</b>
8.1	Codes for Model1 . . . . .	54
8.1.1	Explanations of the Code . . . . .	54

## List of Figures

1	Boxplot Model1 . . . . .	42
2	Model1 high-dimension logistic . . . . .	42
3	Model1 Sparse LDA . . . . .	42
4	Model1 CHIME . . . . .	42
5	Model2 boxplot . . . . .	43
6	Model2 high-dimension logistic . . . . .	43
7	Model2 Sparse LDA . . . . .	43
8	Model2 CHIME . . . . .	43
9	Model2 boxplot . . . . .	44
10	Model3 high-dimension logistic . . . . .	44
11	Model3 Sparse LDA . . . . .	44
12	Model3 CHIME . . . . .	44



# 1 Introduction

## 1.1 Motivation and Objectives

Binary choice models occupy a significant position within the field of econometrics due to their adaptability in situations where the dependent variable adopts binary values, making them invaluable for comprehending and forecasting outcomes in a multitude of disciplines. Traditionally, discriminant analysis has been one of the approaches for binary choice models, with the primary goal of uncovering the most effective variable combinations for distinguishing among multiple groups. Two commonly employed methods in this regard are linear discriminant analysis (LDA) and quadratic discriminant analysis.

Nonetheless, both LDA and QDA make the implicit assumption of normality; however, in practical applications, this assumption frequently proves inadequate as real-world data distributions often deviate significantly from the normal. Moreover, the application of discriminant analysis grapples with numerous complexities in high-dimensional data scenarios, like the curse of dimensionality, which can lead to overfitting. They will also encounter difficulties in estimating covariance matrices, which are essential for discrimination, as they may become singular or unreliable.

A substantial body of literature exists on high-dimensional binary classification methods. Among the various high-dimensional classification methods, two are commonly employed. The first is known as sparse linear discriminant analysis, which involves penalizing the discriminant vectors in Fisher's discriminant problem. [Hastie et al., 2015] The second commonly used method is penalized generalized linear models, which incorporate convex penalties such as lasso, ridge regression, and combinations of the two (elastic net) in the estimation process. [Van de Geer, 2008]. Besides, according to [Fan and Fan, 2008], they propose to use features of the Annealed Independence Rules. Moreover [Li et al., 2022] uses probabilistic neural networks for high-dimension classification. [Roy, 2015] also develops a classification algorithm for high-dimensional data.

This thesis introduces an innovative discriminant analysis model designed to overcome these issues while exhibiting robust performance in low-dimensional scenarios. Within this model, we remove the constraints imposed by the normality assumptions by introducing a mixture model. To estimate mixture model parameters and fit high-dimensional binary models, we employ CHIME clustering of high-dimensional Gaussian mixtures with the EM algorithm[Cai et al., 2019]. an algorithm used for fitting high-dimensional normal mixture models, which are based on the EM algorithm and a direct estimation method for the sparse discriminant vector.

To evaluate the effectiveness of this innovative approach, we apply it to both artificially generated high-dimensional datasets and real-world examples. Furthermore, we carry out comprehensive comparative analyses with conventional high-dimensional classification methods, including high-dimensional logistic regression[Friedman et al., 2010] and sparse Linear Discriminant Analysis (LDA)[Clemmensen et al., 2011], in order to assess its performance and suitability across a diverse array of scenarios.

## 1.2 Organization of the Thesis

The outline of the thesis is as follows: Section 2 will review the binary choice model and discriminant analysis. Section 3 will introduce mixture models as a compelling solution to address the unrealistic assumption of normality in binary choice models. In Section 4, we will delve into the CHIME algorithm, which stands for "Clustering of high-dimensional Gaussian Mixtures with the Expectation-Maximization Algorithm." This method is specifically designed to address the challenges of traditional discriminant analysis when dealing with high-dimensional data. It will cover the issues related to high-dimensional data and introduce the concept of CHIME, emphasizing its role as an estimation methodology for high-dimensional data. Section 5 will introduce the simulation applied to high-dimension data classification. Section 6 provides a real-life example application before finally concluding in Section 7 with some possible future directions for research. Section 8 is an appendix where codes for the simulation model are provided.

## 2 Binary Choice model

### 2.1 Definition

Binary choice models, also known as binary classification models, are used to predict one of two possible outcomes or classes, typically denoted as 0 and 1. Many fields within empirical economics often involve the presence of binary-dependent data. For instance, this can be observed in studies related to decisions regarding transportation choices, unemployment analysis, labor supply, educational choices, fertility decisions, as well as the innovation behavior of companies.

### 2.2 Discriminant analysis

In a given scenario, objects can be categorized into one of  $g$  potential groups, which are denoted as  $C_1, C_2, \dots, C_g$ . Additionally, there is a feature vector,  $X$ , consisting of  $p$ -measurable attributes associated with these objects. The object's association with a particular category is represented by the categorical variable  $Y$ , where  $Y = i$  indicates that the object belongs to category  $C_i$ , and  $i$  can take on values from 1 to  $g$ .

In this context, the primary focus of discriminant analysis is to investigate and comprehend the relationship between the categorical variable  $Y$ , which denotes class membership, and the feature vector  $X$ . This analysis seeks to uncover how the variables within the feature vector  $X$  are connected to the assignment of objects to specific categories. [McLachlan, 2012]

It's worth noting that there are two main steps of discriminant analysis: linear discriminant analysis (LDA) and quadratic discriminant analysis (QDA).

#### 2.2.1 Linear Discriminant Analysis

Linear discriminant analysis is a widely used statistical learning method in multiple applied fields. Its primary purpose is to discover a linear combination of features that effectively distinguishes between two or more classes or groups of objects. [Clemmensen et al., 2011] In other words, linear discriminant analysis (LDA) relies on a linear combination of features as a classification criterion. [Cai and Liu, 2011]

Despite its simplicity, linear discriminant analysis (LDA) has proven to be a valuable classifier in many applications.

Let  $X = (x_1, \dots, x_p)$  denote the predictor vector, and  $Y \in \{0, 1\}$  be the class label. The LDA model states that:

$$X|Y \sim \mathcal{N}(\mu_Y, \Sigma)$$

This means that  $X$  conditional on  $Y$  follows a multivariate normal distribution with a mean vector  $\mu_Y$  and a covariance matrix  $\Sigma$ , yielding the Bayes rule:

$$\hat{Y}_{\text{Bayes}} = \text{sign} \left\{ X - \frac{(\mu_1 + \mu_2)}{2} \right\}^T \Sigma^{-1}(\mu_1 - \mu_2) + \log \left( \frac{\pi}{1 - \pi} \right) \quad (1)$$

Now, let's break down this equation step by step:

- $\hat{Y}_{\text{Bayes}}$ : This is the estimated class label based on a Bayesian decision rule.
- $\text{sign} \left\{ X - \frac{(\mu_1 + \mu_2)}{2} \right\}$ : This part decides which class the data point belongs to by comparing it to the means of the classes.
- $\Sigma^{-1}(\mu_1 - \mu_2)$ : This term quantifies the separation between the means of the classes, taking into account the spread (covariance matrix).
- $\log \left( \frac{\pi}{1 - \pi} \right)$ : This adjusts the decision based on the prior probabilities of the classes.

[Mai and Zou, 2015]

### 2.2.2 Quadratic Discriminant Analysis (QDA)

For LDA, it assumes that two classes share the same covariance matrix, which is challenging to test, especially in high dimensions. [Cai et al., 2013] For quadratic discriminant analysis, it relaxes the assumption made by LDA by allowing for different covariance matrices for each class. This provides greater flexibility when dealing with data that doesn't follow the equal covariance assumption.

## 2.3 Limitations of previous discriminant analysis methods

### 2.3.1 Assumption of Normality

One significant limitation of LDA and QDA is their assumption of normality, particularly the assumption that the predictor variables within each class follow a multivariate normal distribution. [Alayande and Adekunle, 2015] This assumption can be problematic in real-world applications since, in many cases, real-world data may not follow a multivariate normal distribution. Data can exhibit complex and non-Gaussian distributions, especially when dealing with categorical or count data or when outliers are present.

### 2.3.2 High-Dimensional Data

For linear discriminant analysis (LDA), it can perform poorly in high-dimensional space, primarily due to the accumulation of noise in the estimation of population centroids  $\mu_1$  and  $\mu_2$ . [Fan and Fan, 2008]. Similarly, QDA also involves estimating a substantially larger number of unknown parameters, which may pose a significantly greater challenge than LDA in high-dimensional settings. [Wu et al., 2019]

## 2.4 Solutions

### 2.4.1 Solutions for high-dimension data

To overcome these challenges, alternative methods need to be employed. For linear discriminant analysis, one approach is to select a subset of important features by performing two-sample t-tests before applying the independence rule. [Fan and Fan, 2008] Another popular approach in the literature is to impose sparse assumptions. For example, by assuming both  $\Sigma$  and the mean difference vector,  $\mu_1 - \mu_2$ , to be sparse, we estimated them by thresholding. [Shao et al., 2011] And for quadratic discriminant analysis, one solution is to create a series of quadratic discriminant rules by streamlining the complexity of the covariance matrices without requiring them to be sparse, imposing restrictions on their inverses, or on the standardized between-class distance. [Wu et al., 2019]

### 2.4.2 Solution of this thesis

In our thesis, we want to overcome the normality and high-dimension limitations of LDA and QDA. We will introduce mixture models as a solution to address the non-normality assumption. Additionally, we will utilize CHIME for the estimation of mixture coefficients in high-dimension settings. CHIME is an innovative approach that not only addresses sparsity concerns through the application of a lasso penalty in the estimation of discriminant vector  $\beta$  (please refer to Section 4.4.2 for a detailed description of  $\beta$ ). Additionally, CHIME integrates this discriminant vector into the estimation of mixture covariates. Furthermore, we incorporate posterior probabilities into the binary data classification process, enabling high-dimensional data classification.

### 3 Mixture model

#### 3.1 Overview

##### 3.1.1 Definition

A mixture model is a convex combination of probability distributions or densities  $\phi_1, \dots, \phi_k$  with mixing weights  $\pi_1, \dots, \pi_k$ , where  $k$  is the number of components.[McLachlan et al., 2019].

##### 3.1.2 Advantages of Gaussian Mixture Models

Gaussian mixture models are a popular choice when dealing with complex, real-world data that exhibits intricate patterns. It offers flexibility by combining multiple Gaussian components to capture diverse data structures, such as variations in customer behavior. This approach provides clear results, making the Gaussian Mixture Model effective for modeling complex, non-Gaussian data patterns. This flexibility is especially valuable when data deviates from the normality assumptions of LDA.

##### 3.1.3 Probability Density Function of Mixture model

From the law of total probability, we know that the marginal probability of  $X_i$  is:

$$\begin{aligned} p(X_i) &= \sum_{k=1}^K p(X_i|Z_i = k)\Pr(Z_i = k) \\ &= \pi_k \sum_{k=1}^K p(X_i|Z_i = k) \end{aligned} \tag{2}$$

The equation is derived from the law of total probability, which allows us to find the marginal probability of an event  $X_i$  by summing over all possible values of the intermediate event  $Z_i$ . The probabilities  $\pi_k$  represent the weights associated with each possible value of  $Z_i$ , and  $p(X_i|Z_i = k)$  gives the likelihood of  $X_i$  for each value of  $Z_i$ .

Where

- $p(X_i)$  represents the marginal density of observing  $X_i$ , which is equal to  $\sum_{k=1}^K p(X_i|Z_i = k)Pr(Z_i = k)$  by the law of total probability.
- $Pr(Z_i = k)$  represents the probability of  $Z_i$  taking on the value  $k$ . The sum is taken over all possible values of the cluster  $Z_i$ , from  $k = 1$  to  $K$ , and  $\pi_k$  represents the prior probability of component  $k$  being selected.

### 3.1.4 Key Components in a Finite Mixture Model

In a finite mixture model, there are four key components:

- **Component Distribution Type:** Select the type of distribution used in the mixture, such as Gaussian.
- **Number of Components (k):** Determine the appropriate number of component distributions in the mixture model, representing the underlying groups or classes in the data.
- **Parameters for Component Distributions:** Specify the parameters for each component distribution. For instance, in a one-dimensional Gaussian, this includes parameters like mean and standard deviation. In higher-dimensional Gaussian, it involves mean vectors and covariance matrices.
- **Mixing Weights ( $\pi_i$ ):** Assign weights to each component distribution to indicate their relative importance or prevalence in the overall mixture.

These components are essential for density estimation using finite mixture models, providing a framework for classification.



## 3.2 Issues of mixture model density estimation

### 3.2.1 Uncertainty in component assignment

One of the fundamental challenges in mixture model density estimation is the uncertainty about which component generated each observation or data point. And for a proper density estimation, it is significant that we are able to cluster the observations correctly based on it. And in the following section, we will introduce the clustering problem, including the common ways to cluster as well as the EM algorithm, one of the most common ways to cluster.

### 3.2.2 Clustering

Clustering, which falls under the category of unsupervised learning, involves categorizing a set of data points or observations into distinct groups based on their similarities. The objective is to create clusters where the data points within each group exhibit greater similarity to one another compared to those in different groups.

### 3.2.3 Clustering method

- **Common Ways to Cluster**

1. **K-means:** K-means is a clustering algorithm used for unsupervised clustering. It aims to partition data into K clusters, where each data point belongs to the cluster with the nearest mean value. [Bradley et al., 1999]
2. **K-median:** K-median is a clustering algorithm used for unsupervised clustering. It aims to partition data into K clusters, where each data point belongs to the cluster with the nearest median value.
3. **Mixture Models:** Mixture models are invaluable for clustering data, offering a flexible and probabilistic framework that excels at uncovering hidden structures within datasets. Mixture models assume that data points are generated from a combination of component distributions, often Gaussian, enabling them to adapt to a wide range of data patterns and complexities. One of their key advantages is their soft clustering capability, which allows

data points to have partial memberships in multiple clusters even when cluster boundaries are not well defined. The Expectation-Maximization (EM) algorithm, central to mixture models, iteratively refines cluster assignments and estimates model parameters, ensuring an accurate representation of the data distribution. [Scott and Symons, 1971]

### **3.3 EM algorithm**

#### **3.3.1 Overview**

The Expectation-Maximization (EM) algorithm is an iterative optimization technique that seeks to identify maximum posterior likelihood and maximum parameter estimates in statistical models, including latent variables that are not observed. It majorly involves two steps: the estimating phase (E-step) and the maximization step. [Dempster et al., 1977]

#### **3.3.2 Advantages of the EM algorithm**

- **Latent Variable**
- The Expectation-Maximization (EM) algorithm is valuable in practical scenarios involving latent variables. For instance, in economics, it helps uncover the underlying factors affecting complex measures like a country's GDP, which depends on various unobserved variables such as government spending, consumer consumption, investment, and international trade.
- **Parameter Estimation:**
- The EM algorithm offers computational efficiency compared to many alternative competing methods. This efficiency is particularly advantageous when working with extensive data, enabling quicker and more accurate parameter estimation.

### 3.3.3 Background of EM

In order to understand exactly how it works and why it is a good way to solve both density estimation and clustering problems, we will have a review of the key terms in the EM algorithm.

#### Key Terms in EM Algorithm:

- **Latent Variable:** A latent variable is a variable that is not observed in the data. [Aigner et al., 1984] In the context of latent variables, we make inferences about these unobservable quantities using Bayes' rule.
- **Likelihood Function:** The likelihood function is represented as  $L(x|\theta)$ , where  $x$  is the parameter vector being estimated, and  $\theta$  is the observed data. It quantifies the likelihood of the distribution with the parameter vector  $x$ , given the observed data  $\theta$ .
- **Likelihood Function for Entire Dataset in Mixture Models:** This likelihood function is expressed as a product over all data points because we assume that the data points are independent and identically distributed (i.i.d.).

$$L(X|\pi, \theta) = \prod_{i=1}^N \left( \sum_{k=1}^K \pi_k \cdot p(x_i|\theta_k) \right) \quad (3)$$

In this equation:

- $L(X|\pi, \theta)$  represents the likelihood of the entire dataset  $X$  given the model parameters  $\pi$  and  $\theta$ .
- $X$  is the dataset consisting of  $N$  data points, often represented as  $X = \{x_1, x_2, \dots, x_N\}$ .
- $\pi = (\pi_1, \pi_2, \dots, \pi_K)$  represents the vector of mixing coefficients, where  $\pi_k$  is the weight associated with the  $k$ -th component in the mixture. These coefficients represent the prior probabilities of data points belonging to each component.
- $\theta = (\theta_1, \theta_2, \dots, \theta_K)$  represents a set of parameters, where  $\theta_k$  is the set of parameters associated with the  $k$ -th component in the mixture model. These parameters define the probability distribution for each component.
- For each data point  $x_i$ , the expression inside the product symbol represents the density of  $x_i$  being generated by any of the  $K$  components. The mixture density is

a weighted sum of the components' densities  $p(x_i|\theta_k)$  for all  $K$  components, where the weights are given by the mixing coefficients  $\pi_k$ .

The likelihood function captures how well the given model with mixing coefficients and component-specific parameters explains the entire dataset  $X$ . It's used in various statistical methods, such as the Expectation-Maximization (EM) algorithm, to estimate the model parameters that maximize the likelihood of the observed data.

- **Maximum Likelihood Estimation:** MLE is a mathematical approach used to find the parameter values for a probability distribution that maximize the likelihood of the observed data. It is achieved by identifying the highest point on the likelihood function. [Cole et al., 2014]
- **Maximum Likelihood Estimation with Latent Variables:** For the EM algorithm, the goal is to find the maximum likelihood with latent variables. This is achieved through an iterative process involving E-steps and M-steps. In the context of EM, maximum likelihood fitting involves computing and maximizing the marginal probability, which can be expressed as:

$$\begin{aligned}\hat{\theta} &= \arg \max_{\theta} P(x|\theta) \\ &= \arg \max_{\theta} \int P(x, z|\theta) dz\end{aligned}\tag{4}$$

In this equation:

- $\hat{\theta}$  represents the maximum likelihood estimate of the parameter  $\theta$ .
- $P(x|\theta)$  is the likelihood of the data  $x$  given the parameter  $\theta$ .
- $\int P(x, z|\theta) dz$  represents the integral over the joint distribution of  $x$  and latent variables  $z$  given the parameter  $\theta$ .

These equations are used in statistical estimation to find the parameter  $\theta$  that maximizes the likelihood of the observed data.

### 3.3.4 Whole Process

1. **Initialization:** The EM algorithm begins with the initialization of model parameters  $\mu_k$  and  $\pi_k$  and  $\Sigma_k$ 
  - $\mu_k$  represents the mean (average) of the  $k$ -th component of the mixture model. It serves as the central value around which the data within this component is concentrated.
  - $\Sigma_k$  denotes the covariance matrix associated with the  $k$ -th component. Each component is represented as a multivariate normal distribution, and  $\Sigma_k$  describes the covariance structure of that component.
  - $\pi_k$  is the mixture proportion or weight assigned to the  $k$ -th component. This parameter indicates the probability that a data point belongs to this specific component within the mixture.

The initialization process sets these parameters to initial values, and the log-likelihood is evaluated using these parameters as the starting point.

2. **E-step:** For each data point  $x^{(i)}$ , perform the following:

- (a) Calculate the posterior probability of the latent variable given the data:

$$\gamma^{(i)}(z) = Pr(z|x^{(i)}, \theta)$$

Here,  $\gamma^{(i)}(z)$  represents the probability that data point  $x^{(i)}$  is associated with latent variable  $z$ .

$$\gamma_{Z_i}(k) = \frac{\pi_k \cdot \phi(\mathbf{X}_i; \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \cdot \phi(\mathbf{X}_i; \mu_j, \Sigma_j)} \quad (5)$$

where  $\phi(\mathbf{X}; \mu, \Sigma)$  is the probability density function representing the normal distribution of a random vector  $\mathbf{X}$  with mean vector  $\mu$  and covariance matrix  $\Sigma$ .

### Probability Density Function of multivariate normal Distribution

The probability density function  $\phi(\mathbf{x}, \mu, \Sigma)$  for a multivariate normal distribution is given by:

$$\phi(\mathbf{x}, \mu, \Sigma) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right) \quad (6)$$

Where:

- $\phi(\mathbf{x}, \mu, \Sigma)$  is the Probability density function.
- $\mathbf{x}$  is the Vector of random variables.
- $\mu$  is the Mean vector.
- $\Sigma$  is the Covariance matrix.
- $p$  is the Number of dimensions.

This formula represents the probability density function of a multivariate normal distribution.

### 3. M-step:

Update the model parameters to maximize the expected log-likelihood:

- (a) Update the parameter estimates  $\theta$  by maximizing the expected log-likelihood with respect to the latent variable:

$$\theta^{(t+1)} = \arg \max_{\theta} \sum_i \sum_z \gamma^{(i)}(z) \log P(x^{(i)}, z | \theta) \quad (7)$$

In this equation:

- $\theta^{(t+1)}$  represents the updated parameter estimates.
- $\sum_i$  and  $\sum_z$  denote the summation over data points and latent variable values, respectively.
- $\gamma^{(i)}(z)$  typically represents the responsibility of latent variable  $z$  for data point  $x^{(i)}$ .
- $\log P(x^{(i)}, z | \theta)$  is the log-likelihood of the data point and latent variable given the parameter  $\theta$ .

the equation for updating parameters are as follows

$$\mu_k = \frac{1}{\sum_{i=1}^N \gamma_{Z_i}(k)} \sum_{i=1}^N \gamma_{Z_i}(k) \cdot X_i \quad (8)$$

$$\Sigma_k = \frac{1}{\sum_{i=1}^N \gamma_{Z_i}(k)} \sum_{i=1}^N \gamma_{Z_i}(k) (X_i - \mu_k)(X_i - \mu_k)^T \quad (9)$$

$$\pi_k = \frac{1}{N} \sum_{i=1}^N \gamma_{Z_i}(k) \quad (10)$$

In these equations:

- $\mu_k$  represents the updated mean for the  $k$ -th component.
- $\Sigma_k$  represents the updated covariance matrix for the  $k$ -th component.
- $\pi_k$  represents the updated mixing coefficient for the  $k$ -th component.
- $\gamma_{Z_i}(k)$  is typically the responsibility of component  $k$  for data point  $X_i$ .

These equations are used in various algorithms, such as the Expectation-Maximization (EM) algorithm, to update the parameters of a mixture model based on the responsibilities  $\gamma_{Z_i}(k)$ .

4. **Convergence Check:** Evaluate the log-likelihood with the new parameter estimates. If the log-likelihood has changed by less than a small  $\epsilon$ , stop. Otherwise, return to the E-step (Step 2).

### 3.4 Discriminant function based on mixture model

#### 3.4.1 Function

$$Pr(y = 1|x) = \frac{p(x|y = 1) \cdot Pr(y = 1)}{p(x|y = 0) \cdot Pr(y = 0) + p(x|y = 1) \cdot Pr(y = 1)} \quad (11)$$

$$p(x|y = 1) = \sum_{i=1}^K \pi_i \cdot \phi(x|\mu_i, \Sigma_i) \quad (12)$$

- $p(x|y = 1)$  represents the probability of  $x$  given  $y = 1$ , which is calculated as the sum of  $K$  Gaussian distributions.

Traditional mixture models can be useful when dealing with data that doesn't follow a normal distribution. However, they face challenges when working with high-dimensional data since, on high dimensions, it becomes difficult to accurately estimate the covariance matrix because you may have very few data points compared to the number of variables. Therefore, in the next section, we will introduce CHIME, a method that applies the EM algorithm to cluster high-dimensional data using a Gaussian mixture model. CHIME helps estimate the parameters in the mixture model and combines them with posterior probabilities for high-dimensional binary choice models.



## 4 CHIME

### 4.1 High-Dimensional Data Characteristics

High-dimensional data is defined as data in which the variables are close to or even higher than the observations, and it is becoming more and more common these days due to the advancement of technology. However, characterized by a large number of variables, often leads to issues of overfitting, computational complexity, and decreased classification performance. Addressing these challenges requires the development of specialized techniques. [Pappu and Pardalos, 2014]

### 4.2 Clustering Challenge of High-dimension data

#### 4.2.1 Problems of Fisher's Linear Discriminant Rule

##### Fisher's Linear Discriminant Rule

Fisher's Linear Discriminant Analysis (LDA) is a widely used and powerful data analysis technique employed in econometrics. Its primary purpose is to investigate the relationship between a set of predictor variables and a categorical response or outcome variable. [Hastie et al., 1995]

In the context of Fisher's Linear Discriminant Rule, the objective is to classify an observation  $z$  into one of two classes, typically referred to as class 1 and class 2. This classification is determined by the function  $G_{\theta^*}(z)$ , which is defined as follows:

$$G_{\theta^*}(z) = \begin{cases} 1, & \text{when } \left(z - \frac{\mu_1^* + \mu_2^*}{2}\right)^T \beta^* \geq \log\left(\frac{\pi^*}{1-\pi^*}\right) \\ 2, & \text{when } \left(z - \frac{\mu_1^* + \mu_2^*}{2}\right)^T \beta^* < \log\left(\frac{\pi^*}{1-\pi^*}\right) \end{cases} \quad (13)$$

Where:

- $G_{\theta^*}(z)$  represents the classification result for the observation  $z$ .
- $\theta^*$  denotes a set of known parameters, including class means  $\mu_1^*$  and  $\mu_2^*$ , discriminant direction  $\beta^*$ , and class proportions  $\pi^*$  and  $1 - \pi^*$ .
- $z$  is an unlabeled observation that we aim to classify.
- $\mu_1^*$  and  $\mu_2^*$  are the class means for class 1 and class 2, respectively.

- $\beta^*$  represents the discriminant direction, which can be calculated as  $\beta^* = \Omega^* \delta^*$ . Here,  $\Omega^*$  is the inverse of the pooled sample covariance matrix, denoted as  $\Omega^* = (\Sigma^*)^{-1}$ , and  $\delta^* = \mu_1^* - \mu_2^*$ .
- $\pi^*$  and  $1 - \pi^*$  indicate the class proportions, reflecting the proportion of observations in class 1 and class 2.
- $\log\left(\frac{\pi^*}{1-\pi^*}\right)$  computes a threshold value for classification. If the inner product between  $z$  and  $\beta^*$  is greater than or equal to this threshold, the observation is classified as class 1; otherwise, it's classified as class 2.

In essence, Fisher's Linear Discriminant Rule offers an effective methodology for classifying observations into one of two classes based on their similarity to the class means, the discriminant direction, and class proportions, all of which are part of the known parameters represented by  $\theta^*$ . However, it's important to note that when dealing with high-dimensional data, the estimation of  $\Omega^*$  (the inverse of the covariance matrix) can become a challenging task. This difficulty in estimating  $\Omega^*$  can introduce complexities and potential issues in the clustering process. [Cai et al., 2019]

#### 4.2.2 Problems of EM in Clustering

##### Problems in the Definition of the Maximum Likelihood Estimator

When variables are greater than observations, the maximum likelihood estimator can become ill-defined. This is because, with too few data points relative to the number of parameters to estimate, the likelihood function may become extremely flat or even discontinuous, making it difficult to find a unique maximum. [Wang et al., 2013]

##### Problems in estimating the parameter

**E-step:** Evaluate the posterior probabilities  $\gamma_{Z_i}(k)$  using the current values of  $\mu_k$  and  $\Sigma_k$  with the following equation:

$$\gamma_{Z_i}(k) = \frac{\pi_k \cdot \phi(\mathbf{X}_i | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \cdot \phi(\mathbf{X}_i | \mu_j, \Sigma_j)} \quad (14)$$

In the case of high-dimensional data, the intricate nature of the symbol  $\Sigma_k$  adds complexity, posing significant challenges for computational tasks. [Cai et al., 2019]

### 4.2.3 High-dimension clustering methods

To address the challenges posed by high-dimensional data classification, specialized algorithms and techniques have been developed to improve efficiency and accuracy. There exists a vast literature on classifying high-dimensional data.

#### High-dimensional EM

The authors have presented a framework that uses the expectation-maximization (EM) algorithm in complex situations with lots of variables. They make two important contributions. First, they create a new EM algorithm that can estimate parameters accurately even when there are very few variables involved. This algorithm works well when it starts with the right initial values and gives nearly optimal statistical results. Second, they use this estimated solution to come up with a new way to test hypotheses about aspects of these parameters that are simpler and have fewer variables. [Wang et al., 2013]

#### Subspace clustering

Subspace clustering is an advanced approach to clustering data that goes beyond traditional methods by identifying clusters in distinct subspaces within a dataset. In high-dimensional data, numerous dimensions might be uninformative and introduce noise that obscures underlying clusters. To address this issue, feature selection techniques are applied to eliminate irrelevant and redundant dimensions based on the entire dataset. Subspace clustering algorithms take a more focused approach by pinpointing relevant dimensions, enabling them to uncover clusters that may exist across multiple, potentially overlapping subspaces.[Parsons et al., 2004]

#### Projections

An alternative approach involves employing projections, which are traditional techniques for reducing the dimensionality of data for the purpose of visualizing information. This method is used to convert data from a high-dimensional space into a lower-dimensional representation. [Thrun and Ultsch, 2020]

## 4.3 High-dimension classification

### 4.3.1 Problems

#### Curse of Dimensionality

Classification algorithms become less accurate in high-dimensional data spaces due to a phenomenon known as the "curse of dimensionality" [14, 15]. The best test results are achieved with a limited number of features, and with an infinite number of features, the test results become as accurate as random guessing. [Thrun and Ultsch, 2020]

#### Poor Generalization Ability

In high-dimensional data classification, a significant challenge is preventing overfitting to the training data. It's crucial to create a classification model that not only works well on the training data but also performs effectively on a separate testing dataset, showing what we call good "generalization ability." However, in high-dimensional data scenarios, there are usually very few samples, and this can cause the classification model to become overly specialized to the training data, resulting in poor generalization to new data.

### 4.3.2 Common High-Dimension Classification Methods

#### Sparse LDA

In high-dimensional data, linear discriminant analysis faces two problems. Firstly, the maximum likelihood estimate of the within-class covariance matrix is approximately singular if the variable is almost the same as the observations, or singular if the variable is larger than the observations. Besides, when  $p$  is large, the resulting classifier is difficult to interpret since the classification involves a linear combination of all variables. citehastie2015statistical And according to [Bickel and Levina, 2004], LDA is asymptotically as bad as random guessing in high dimensions.

Sparse Linear Discriminant Analysis is a variant of Linear Discriminant Analysis (LDA) that incorporates sparsity constraints into the model. In sparse LDA, sparsity constraints are applied to the linear discriminant vectors. This sparsity can be beneficial in high-dimensional data settings, where it helps reduce overfitting and enhances interpretability by selecting a subset

of the most informative features.

### **Generalized Linear Models with L1 (Lasso) and L2 (Ridge) Regularization**

Traditional generalized linear models, historically used for classification, face numerous challenges with high-dimensional data. [Friedman et al., 2010]

Consequently, an approach involves fitting generalized linear models with L1 and L2 regularization to perform binary or multi-class classification by utilizing the logistic link function. These regularization methods offer several advantages. They promote sparsity in model coefficients, automatically discerning essential features. Additionally, they counteract overfitting through penalty terms, thus enhancing the model's ability to generalize. Moreover, these techniques demonstrate proficiency in addressing multicollinearity, a frequent problem in high-dimensional data scenarios.

## **4.4 Overview of CHIME**

### **4.4.1 Definition**

CHIME, which stands for Clustering High-Dimensional Data based on the EM (Expectation-Maximization) algorithm, is a clustering method specifically designed for high-dimensional Gaussian mixture models. While CHIME is built upon the foundation of EM, one of its most distinctive features is the direct estimation and continuous update of the discriminant direction, denoted as  $\beta^*$ . This characteristic sets CHIME apart and addresses significant issues related to computational complexity and the ill-defined nature of maximum likelihood in traditional EM algorithms.

### **4.4.2 Discriminant Vectors $\beta^*$**

$\beta^*$  is actually a discriminant direction with:

$$\beta^* = \Omega^* \delta^* \tag{15}$$

Where:

- $\hat{\Omega}$  is the inverse of the pooled sample covariance matrix, denoted as  $\Omega^* = (\Sigma^*)^{-1}$ .
- $\delta^* = \mu_1^* - \mu_2^*$ .

### Estimation of $\beta$

Let  $\hat{\mu}_k$  be the sample mean for class  $k$  ( $k = 1, 2$ ) and  $\hat{\Sigma}$  be the pooled sample covariance matrix. Assuming that  $\beta^*$  is sparse, The optimization problem to find  $\hat{\beta}$  is given by:

$$\hat{\beta} = \underset{\beta \in R^p}{\operatorname{argmin}} \left[ \frac{1}{2} \beta^T \hat{\Sigma} \beta - \beta^T (\hat{\mu}_1 - \hat{\mu}_2) + \lambda_n \|\beta\|_1 \right] \quad (16)$$

In this equation:

- $\hat{\beta}$  represents the estimated variable or vector  $\beta$ .
- $\hat{\Sigma}$  represents the estimated covariance matrix  $\Sigma$ .
- $\hat{\mu}_1$  and  $\hat{\mu}_2$  represent estimated means  $\mu_1$  and  $\mu_2$ .
- $\lambda_n$  is a regularization parameter.
- $\|\beta\|_1$  represents the  $L_1$  norm of the vector  $\beta$ .

### Importance of estimation of $\beta$

- **High-Dimensional Settings:** The optimization problem encourages sparsity in  $\beta$  through the lasso penalty, and this can help focus on important features. Besides, estimating beta with sparsity constraints enhances computational efficiency.
- **Discriminant Analysis:** Accurate beta construction leads to optimal separation between classes and also results in better classification.

## 4.5 CHIME algorithm

### 4.5.1 Overview

For CHIME in this thesis, it primarily utilizes a two-component Gaussian mixture model for its operations.

### 4.5.2 Initialization

In the first step of CHIME, it is similar to the Expectation-Maximization (EM) algorithm, in which it sets the initial parameters  $\hat{\pi}^{(0)}$ ,  $\hat{\mu}_1^{(0)}$ , and  $\hat{\mu}_2^{(0)}$ , and  $\hat{\Sigma}^{(0)}$ , and then proceeds with the initialization. We will attempt to implement this initialization through K-means.

#### Advantages of using K-means in initialization

K-Means Clustering allows all the data points in a cluster to be similar to each other and the data points from different clusters to be as different as possible. Both can contribute to a correct density estimation. [Zubair et al., 2022]

### 4.5.3 Initial of $\beta^*$

One of the significant distinctions in CHIME is the requirement to initialize the discriminant direction  $\beta^*$ , denoted as  $\hat{\beta}^{(0)}$ . This initialization process involves estimating  $\hat{\beta}^{(0)}$  using a convex optimization approach, while simultaneously utilizing cross-validation to select the parameters of this optimization function.

This approach allows for a precise determination of  $\hat{\beta}^{(0)}$ , which plays a crucial role in the CHIME algorithm. By integrating convex optimization and cross-validation, the method ensures that  $\hat{\beta}^{(0)}$  is initialized in a manner that optimally suits the problem and data at hand, contributing to the overall effectiveness of the CHIME model.

- **Convex Optimization Problem:**

$$\text{minimize } f_0(x) \quad (17)$$

$$\text{subject to } f_i(x) \leq b_i, \quad i = 1, \dots, m \quad (18)$$

- In this context,  $f_i(x)$  and  $f_0(x)$  are convex functions, meaning they adhere to the following convexity property:

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \quad \text{for all } x, y \in R^n, \alpha, \beta \in R \text{ with } \alpha + \beta = 1, \alpha \geq 0, \beta \geq 0. \quad (19)$$

- This property holds in a high-dimensional space, which extends to vectors and matrices of higher dimensions.[Boyd and Vandenberghe, 2004]

### Estimation of $\beta^*$ as Convex Optimization

We are interested in estimating the value of  $\hat{\beta}^{(0)}$

$$\hat{\beta}^{(0)} = \underset{\beta \in R^p}{\operatorname{argmin}} \left\{ \frac{1}{2} \beta^T \cdot \hat{\Sigma}^{(0)} \cdot \beta - \beta^T \cdot (\hat{\mu}_1^{(0)} - \hat{\mu}_2^{(0)}) + \lambda_n^{(0)} \cdot \|\beta\|_1 \right\} \quad (20)$$

Where:

$$\lambda_n^{(0)} = \frac{1}{\sqrt{s}} \left( C_1 \cdot (|\hat{\pi}| \vee \|\hat{\mu}_1^{(0)} - \hat{\mu}_2^{(0)}\|_{2,s} \vee \|\Sigma^{(0)}\|_{2,s}) \right) + C_\lambda \sqrt{\frac{\log p}{n}} \quad (21)$$

the term  $\cdot (|\hat{\pi}| \vee \|\hat{\mu}_1^{(0)} - \hat{\mu}_2^{(0)}\|_{2,s} \vee \|\Sigma^{(0)}\|_{2,s})$  selects the maximum value among the absolute value of  $\hat{\pi}$ , the L2 norm of the difference between vectors  $\hat{\mu}_1^{(0)}$  and  $\hat{\mu}_2^{(0)}$  under the "s" norm, and the L2 norm of the matrix  $\Sigma^{(0)}$ . In this context:

-  $\|\hat{\mu}_1^{(0)} - \hat{\mu}_2^{(0)}\|_{2,s}$  calculates the L2 norm (Euclidean distance) between two vectors  $\hat{\mu}_1^{(0)}$  and  $\hat{\mu}_2^{(0)}$  with respect to the "s" norm. It quantifies the difference between these two vectors.

-  $|\Sigma^{(0)}|_{2,s}$  calculates the L2 norm of a matrix  $\Sigma^{(0)}$  with respect to the "s" norm. This operation provides a measure of the matrix's characteristics within the context of the "s" norm.

-  $\sqrt{\frac{\log p}{n}}$  is the square root of the ratio of the natural logarithm of  $p$  (the number of features or variables) to  $n$  (the sample size). It represents a trade-off between model complexity and



sample size.

This is a convex optimization problem that aims to find the optimal value of  $\hat{\beta}^{(0)}$  while considering the given data, covariance matrix  $\hat{\Sigma}$ , penalty term  $\lambda_n^{(0)}$  which aims at encouraging the sparsity of  $\hat{\beta}^{(0)}$  through the regularized  $\ell^1$  minimization:

### **Using Cross Validation to estimate $\hat{\beta}^{(0)}$**

One issue in the estimation of  $\hat{\beta}^{(0)}$  is that we need to choose the optimal of  $C_\lambda$ ,  $C_1$  and this issue will be solved using cross validation.

### **Cross Validation**

Cross-validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments: one used to learn or train a model, and the other used to validate the model. [Refaeilzadeh et al., 2018]

### **Training set and Test set**

**Training Set:** Use your training set to train multiple instances of your clustering model, each with different parameters or configurations. For each configuration, measure the clustering error rate and the sum of absolute differences, as you mentioned.

**Test Set:** Once the best parameters are chosen using the training set, apply this selected model to the test set. Evaluate the clustering error rate on the test set using the chosen parameters.

**Parameter Tuning:** Based on the results obtained from the training set, select the parameters that result in the best clustering performance (lowest error rate and sum of absolute differences).

### **Steps**

#### **Step 1: Training**

- Use the training dataset to find optimal parameters  $C_\lambda$  and  $C_1$  by minimizing clustering error rates and the sum of absolute differences.

#### **Step 2: Testing and Evaluation**

- Apply the clustering algorithm to the test dataset using the updated parameters  $C_\lambda$  and  $C_1$ .
- Evaluate the algorithm's performance on the test data by calculating clustering error rates.

These rates can help assess how well the clustering model generalizes to new, unseen data.

#### 4.5.4 EM steps of CHIME

E-steps

in the high-dimensional setting, the update of  $\hat{\gamma}_\theta^{(t)}(z^{(i)})$  which is the probability that the observations belongs to the second group in the E-step is proposed to be:

$$\begin{aligned} \gamma_{\hat{\theta}}^{(t)}(z^{(i)}) &:= \frac{\hat{P}_\theta^{(t)}(y^i = 2|z^{(i)})}{\hat{\pi}^{(t)}} \\ &= \frac{\hat{\pi}^{(t)}}{\hat{\pi}^{(t)} + (1 - \hat{\pi}^{(t)}) \exp \left\{ (\hat{\beta}^{(t)})^T \left( z^{(i)} - \frac{\hat{u}_1^{(t)} + \hat{u}_2^{(t)}}{2} \right) \right\}} \end{aligned} \quad (22)$$

Where

- $w(t)$  represents the mixing weight of the second cluster.
- $\mu_1$  is the sample mean for cluster 1, and  $\mu_2$  is the sample mean for cluster 2.
- $(z^{(i)})$  represents the observations.
- $\beta(t)$  represents the discriminant vector.

After this step is the calculation of maximum log likelihood The expected log-likelihood in the  $t$ -th iteration with respect to the estimated parameters  $\hat{\theta}^{(t)}$  is given by:

$$\begin{aligned} Q_n(\theta|\hat{\theta}^{(t)}) &= E_{\hat{\theta}^{(t)}} [\log L_C(\theta; y, z)|z] \\ &= -\frac{1}{2n} \sum_{i=1}^n \{ (1 - \gamma_{\hat{\theta}}^{(t)}(z^{(i)})) \cdot (z^{(i)} - \mu_1)^T \cdot \Omega \cdot (z^{(i)} - \mu_1) \\ &\quad + \gamma_{\hat{\theta}}^{(t)}(z^{(i)}) \cdot (z^{(i)} - \mu_2)^T \cdot \Omega \cdot (z^{(i)} - \mu_2) \} \\ &\quad + \frac{1}{n} \sum_{i=1}^n \{ (1 - \gamma_{\hat{\theta}}^{(t)}(z^{(i)})) \cdot \log(1 - \pi) + \gamma_{\hat{\theta}}^{(t)}(z^{(i)}) \log \pi \} \\ &\quad + \frac{1}{2} \log |\Omega| \end{aligned} \quad (23)$$

#### Advantages of CHIME in this E-steps

Compared with traditional calculation of probability

$$\begin{aligned}
\gamma_{\hat{\theta}}^{(t)}(z^{(i)}) &:= \frac{\hat{P}_{\theta}^{(t)}(y^i = 2|z^{(i)})}{\hat{\pi}^{(t)}} \\
&= \frac{\hat{\pi}^{(t)}}{\hat{\pi}^{(t)} + (1 - \hat{\pi}^{(t)}) \exp \left\{ (\hat{\Omega}^{(t)}) \left( \hat{u}_1^{(t)} - \hat{u}_2^{(t)} \right)^T \left( z^{(i)} - \frac{\hat{u}_1^{(t)} + \hat{u}_2^{(t)}}{2} \right) \right\}}
\end{aligned} \tag{24}$$

In this E-step, the biggest difference is that the calculation of the posterior probability will use  $\beta^*$ , and this is quite advantageous in high-dimensional data since it is much easier to calculate.

### M-steps

The M-step proceeds by maximizing  $Q_n(\theta|\hat{\theta}^{(t)})$  given  $\hat{\gamma}_{\theta}^{(t)}(z^{(i)})$ , and is interpreted as parameter estimation given the labels. The maximizer,

$$\hat{\theta}^{(t+1)} = \arg \max_{\theta} Q_n(\theta|\hat{\theta}^{(t)}),$$

can be calculated analytically.

### Parameter estimation

It is straightforward to define and calculate:

$$\hat{\pi}^{(t+1)} = \frac{1}{n} \sum_{i=1}^n \gamma_{\hat{\theta}}^{(t)}(z^{(i)}) \tag{25}$$

which is the sum of the probabilities that an observation belongs to cluster 2 for all the observations.

$$\hat{\mu}_1^{(t+1)} = \left( n - \sum_{i=1}^n \gamma_{\hat{\theta}}^{(t)}(z^{(i)}) \right)^{-1} \cdot \left( \sum_{i=1}^n (1 - \gamma_{\hat{\theta}}^{(t)}(z^{(i)})) z^{(i)} \right) \tag{26}$$

$$\hat{\mu}_2^{(t+1)} = \left( \sum_{i=1}^n (\gamma_{\hat{\theta}}^{(t)}(z^{(i)})) \right)^{-1} \cdot \left( \sum_{i=1}^n \gamma_{\hat{\theta}}^{(t)}(z^{(i)}) z^{(i)} \right) \quad (27)$$

$$\begin{aligned} \hat{\Sigma}^{(t+1)} &= \hat{\Sigma} \hat{\theta}(t) \\ &= \frac{1}{n} \sum_{i=1}^n (1 - \gamma_{\hat{\theta}}^{(t)}(z^{(i)})) (z^{(i)} - \hat{\mu}_1^{(t+1)}) (z^{(i)} - \hat{\mu}_1^{(t+1)})^T \\ &\quad + \gamma_{\hat{\theta}}^{(t)}(z^{(i)}) (z^{(i)} - \hat{\mu}_2^{(t+1)}) (z^{(i)} - \hat{\mu}_2^{(t+1)})^T. \end{aligned} \quad (28)$$

$$\hat{\beta}^{(t+1)} = \underset{\beta \in R^p}{\operatorname{argmin}} \left\{ \frac{1}{2} \beta^T \cdot \hat{\Sigma}^{(t+1)} \cdot \beta - \beta^T \cdot (\hat{\mu}_1^{(t+1)} - \hat{\mu}_2^{(t+1)}) + \lambda_n^{(t+1)} \cdot \|\beta\|_1 \right\} \quad (29)$$

$$\lambda_n^{(t+1)} = \kappa \lambda_n^{(t)} + C_\lambda \sqrt{\frac{\log(p)}{n}}. \quad (30)$$

And the turing parameters can also be solved by cross validation.

#### 4.5.5 Output and Classification

##### Convergence Criteria

Given a suitable initialization, the EM algorithm iterates between the E-step and M-Step, as described above, terminates in, say,  $T_0$ , steps. And the termination condition can be set as the l1 norm between the previous  $\mu_1$  and the current  $\mu_1$  being smaller than a tolerance level, which can be expressed as:

$$\|\mu_{1,\text{previous}} - \mu_{1,\text{current}}\|_1 < \varepsilon \quad (31)$$

$$G_{\theta^*}(z) = \begin{cases} 1, & \text{when } \left( z - \frac{\mu_1^* + \mu_2^*}{2} \right)^T \beta^* \geq \log \left( \frac{\pi^*}{1-\pi^*} \right) \\ 2, & \text{when } \left( z - \frac{\mu_1^* + \mu_2^*}{2} \right)^T \beta^* < \log \left( \frac{\pi^*}{1-\pi^*} \right) \end{cases} \quad (32)$$

Where:

- $\mu_1$  is the final mean vector of component one.

- $\mu_2$  is the final mean vector of component two.
- $z$  is the observations.
- $\beta^*$  is the final vector of discriminant analysis.
- $w$  represents the portion of component two when the convergence condition is met.

## 4.6 Advantage of CHIME

### 4.6.1 Theoretical Optimality

- Analysis establishes the rate of convergence for estimating  $\beta^*$  under the  $l_2$  norm loss and the convergence rate of the expected excess mis-clustering error.
- Mini max lower bounds are obtained, showing that the estimator  $\hat{\beta}$  and the CHIME procedure are rate-optimal. The first optimality result for clustering high-dimensional Gaussian mixtures and a rate-optimal clustering procedure are introduced.

### 4.6.2 Dimensionality Reduction

- High-dimensional datasets often contain many features, some of which may be irrelevant or redundant.
- Estimation of  $\beta^*$  with L1 regularization helps avoid this problem.

### 4.6.3 Sparsity Requirement and Better Generalization

- CHIME only requires sparsity for the discriminant vector  $\beta^*$  as opposed to both mean vectors and precision matrices.
- This selective sparsity requirement allows the use of all features, potentially improving fit to training data and predictive performance.

## 4.7 CHIME for high-dimension classification

### 4.7.1 Overview

Combining posterior probabilities with CHIME (Clustering of High-Dimensional Gaussian Mixtures with EM Algorithm ) is a powerful approach to addressing the high-dimensional issue in discriminant analysis. This combination can help improve the robustness, generalization, and interpretability of high-dimensional binary choice models. Here's how it can be done:

### 4.7.2 Steps

#### CHIME Parameter Estimation

To facilitate our probabilistic classification approach, we first employ CHIME to estimate the probability density functions (PDFs) for each label. Specifically, we determine the conditional PDFs of label 0 and label 1 based on our data.

##### PDF Estimation

We proceed to estimate the probability density functions which is given by:

$$\phi(\mathbf{x}) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) \right) \quad (33)$$

for each label within the combined dataset. This involves:

- Estimating the PDF for the first label using the train data results in a PDF that characterizes the distribution of the first label. Similarly, estimating the PDF for another label using the training data results in a PDF that represents the distribution of the second label.

#### Classification Based on PDFs

In the classification step, we utilize Bayesian classification to assign new data points to the most probable class based on the estimated PDFs. Bayesian classification involves computing the posterior probabilities for each class given the observed data. For the two classes, Label 0 and Label 1.

For a new data point  $x^*$ , we calculate the posterior probabilities for each class using Bayes' theorem:

$$p(\text{Label } 0|\mathbf{x}^*) = \frac{\phi(\mathbf{x}^*; \mu_0, \Sigma_0) \cdot Pr(\text{Label } 0)}{\phi(\mathbf{x}^*; \mu_0, \Sigma_0) \cdot Pr(\text{Label } 0) + \phi(\mathbf{x}^*; \mu_1, \Sigma_1) \cdot Pr(\text{Label } 1)} \quad (34)$$

Where

- $Pr(\text{Label } 0|\mathbf{x}^*)$ : The conditional probability that the input  $\mathbf{x}^*$  belongs to Label 0, given the features in  $\mathbf{x}^*$ .
- $\phi(\mathbf{x}^*; \mu_0, \Sigma_0)$ : The probability density function (PDF) of the feature vector  $\mathbf{x}^*$  under a multivariate Gaussian distribution with parameters  $\mu_0$  (mean vector) and  $\Sigma_0$  (covariance matrix) for Label 0. This describes how well the features in  $\mathbf{x}^*$  match the statistical properties of Label 0.
- $Pr(\text{Label } 0)$ : The prior probability of Label 0 represents the probability of an observation belonging to Label 0 without considering any specific features. This is a measure of the prior belief that an observation is labeled 0.
- $\phi(\mathbf{x}^*; \mu_1, \Sigma_1)$ : The probability density function (PDF) of the feature vector  $\mathbf{x}^*$  under a multivariate Gaussian distribution with parameters  $\mu_1$  (mean vector) and  $\Sigma_1$  (covariance matrix) for Label 1. Similar to term 2, it characterizes how well the features in  $\mathbf{x}^*$  match the statistical properties of label 1.
- $Pr(\text{Label } 1)$ : The prior probability of Label 1 represents the probability of an observation belonging to Label 1 without considering any specific features. This is the prior belief that an observation is label 1.

And it is the same for label 1

$$p(\text{Label } 1|\mathbf{x}^*) = \frac{\phi(\mathbf{x}^*; \mu_1, \Sigma_1) \cdot Pr(\text{Label } 1)}{\phi(\mathbf{x}^*; \mu_0, \Sigma_0) \cdot Pr(\text{Label } 0) + \phi(\mathbf{x}^*; \mu_1, \Sigma_1) \cdot Pr(\text{Label } 1)} \quad (35)$$

Once we have computed these posterior probabilities for both classes, we assign the data point  $x^*$  to the class with the higher posterior probability.

## 5 Simulation on CHIME Clustering and Classification

### 5.1 Overview

In this section, I will perform an extensive simulation study to thoroughly assess the capabilities of CHIME. The simulations will be carried out using high-dimensional datasets, which are known to introduce complexity into clustering tasks. The primary aim is to gauge CHIME's performance by closely examining its clustering error rate in comparison to other widely-used algorithms. This analysis will offer valuable insights into CHIME's effectiveness in handling the challenges posed by high-dimensional data, which is prevalent across numerous domains. Furthermore, this simulation will aim to uncover nuances in CHIME's specific steps and processes, identify potential challenges that may arise when applying it to complex data, and then come up with some improvements that can be made.

### 5.2 Model

#### 5.2.1 Model 1

##### Label 1:

- First Mixture Model: 100 observations with a 100-dimensional mean vector  $\mu_1$ . The first 10 elements of  $\mu_1$  are 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, and the rest are set to 0.

The covariance matrix

$$\Sigma_{ij} = 0.8^{|i-j|}$$

$$\Sigma = \begin{bmatrix} 0.8^0 & 0.8^1 & 0.8^2 & \dots & 0.8^{p-1} \\ 0.8^1 & 0.8^0 & 0.8^1 & \dots & 0.8^{p-2} \\ 0.8^2 & 0.8^1 & 0.8^0 & \dots & 0.8^{p-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.8^{p-1} & 0.8^{p-2} & 0.8^{p-3} & \dots & 0.8^0 \end{bmatrix}$$



- **Second Mixture Model:** 100 observations with a 100-dimensional mean vector  $\mu_2$ . The first 10 elements of  $\mu_2$  are  $-0.2, -0.4, -0.6, -0.8, -1, -1.2, -1.4, -1.6, -1.8, -2$ , and the rest are set to 0. The covariance matrix is also

$$\Sigma_{ij} = 0.8^{|i-j|}$$

### Label 2:

- **\*\*First Mixture Model:\*\*** 100 observations with a 100-dimensional mean vector  $\mu_1$ . The first 10 elements of  $\mu_1$  are 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, and the rest are set to 0. The covariance matrix is also

$$\Sigma_{ij} = 0.8^{|i-j|}$$

- **\*\*Second Mixture Model:\*\*** 100 observations with a 100-dimensional mean vector  $\mu_2$ . The first 10 elements of  $\mu_2$  are  $-10, -11, -12, -13, -14, -15, -16, -17, -18, -19$ , and the rest are set to 0. The covariance matrix is also

$$\Sigma_{ij} = 0.8^{|i-j|}$$

### 5.2.2 Model 2

This model is similar to Model 1 with the same covariance matrix but with different values for label 1 with the first mixture model  $\mu_1$ , whose first 10 elements are 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, and  $\mu_2$ , whose first 10 elements are  $-0.1, -0.2, -0.3, -0.4, -0.5, -0.6, -0.7, -0.8, -0.9, -1$ .

And for second labels, the first mixture model has  $\mu_1$ . whose first 10 elements of  $\mu_1$  are 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and the second mixture model has  $\mu_2$ . whose first 10 elements of  $\mu_1$  are  $-5, -6, -7, -8, -9, -10, -11, -12, -13, -14$ .

### 5.2.3 Model 3

In this model, the covariance matrix  $\Sigma(u)$  is a diagonal matrix, meaning that all off-diagonal elements are 0. It follows the same  $\mu_1$  and  $\mu_2$  in model 2.

### 5.3 Comparison of Classifiers

We use CHIME to denote our proposed semiparametric location model. For comparison, we also consider the following classifiers:

- Sparse LDA.
- high-dimension logistic model.
- CHIME.

We fixed the sample sizes for Label 1 and Label 2 to be  $n_1 = n_2 = 200$ . To compare these three methods with the four models described above, we conducted a 5-fold cross-validation (CV) with 100 simulations.

#### 5.3.1 Results

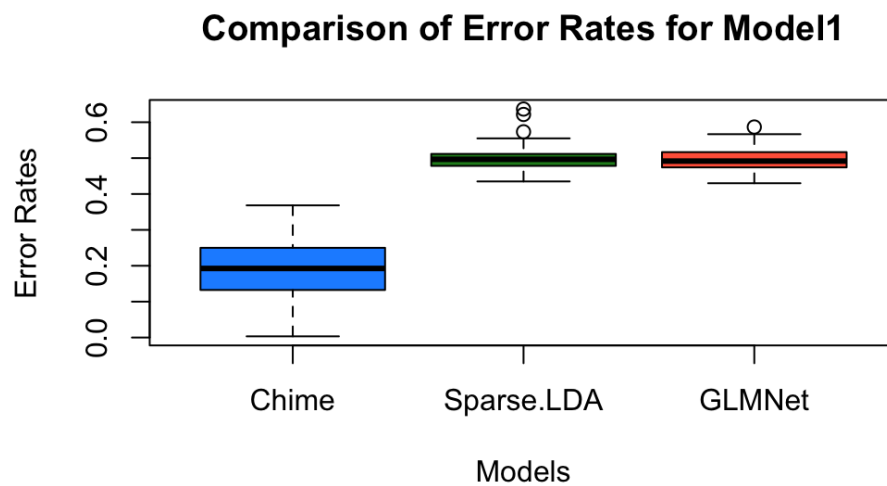


Figure 1: Boxplot Model1

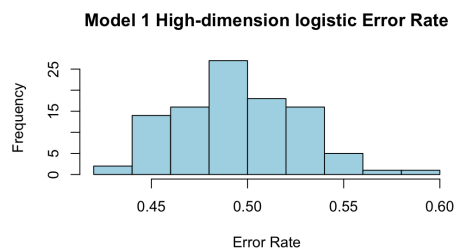


Figure 2: Model1 high-dimension logistic

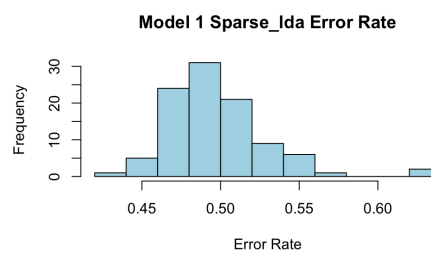


Figure 3: Model1 Sparse LDA

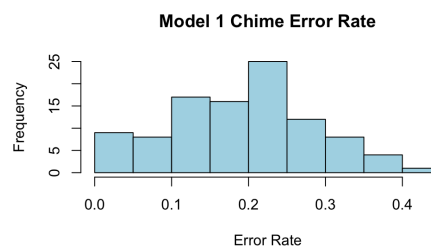


Figure 4: Model1 CHIME

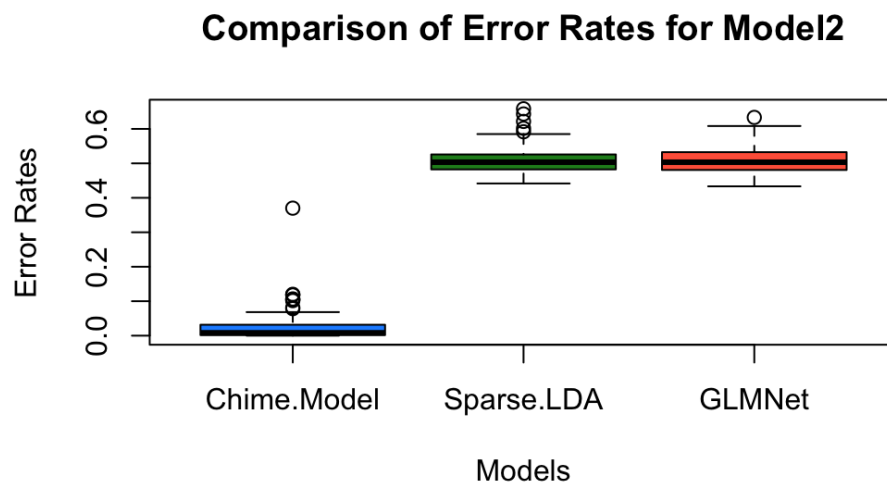


Figure 5: Model2 boxplot

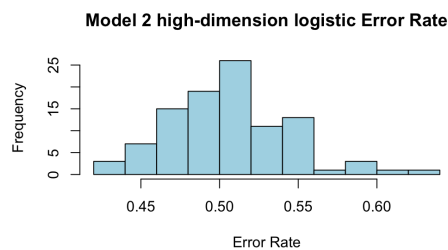


Figure 6: Model2 high-dimension logistic

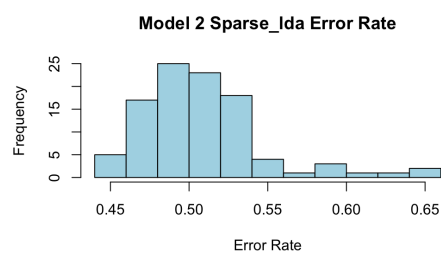


Figure 7: Model2 Sparse LDA

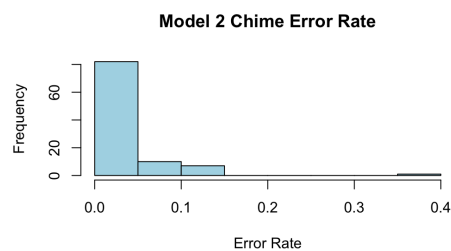


Figure 8: Model2 CHIME

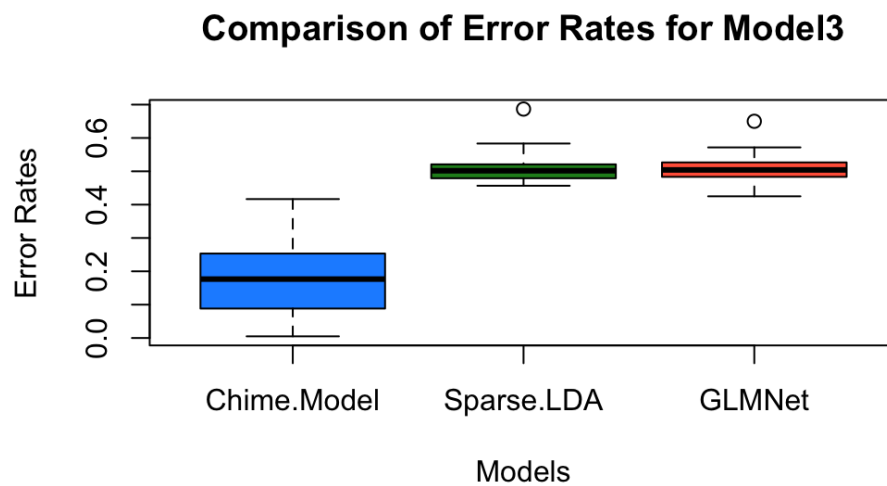


Figure 9: Model2 boxplot

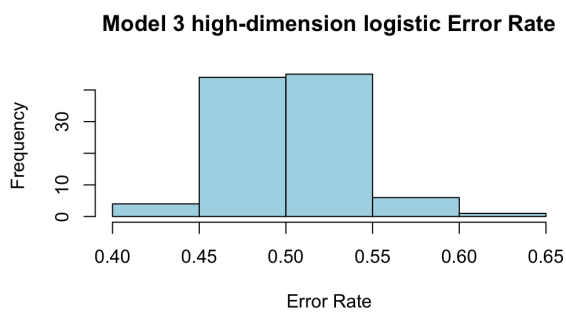


Figure 10: Model3 high-dimension logistic



Figure 11: Model3 Sparse LDA

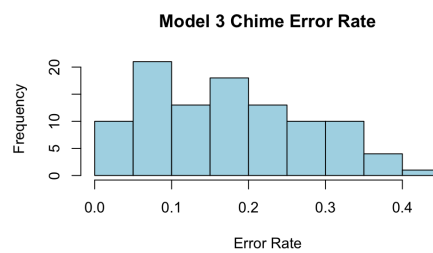


Figure 12: Model3 CHIME

Table 1: 5-fold cross validation error Rates (Model 1)

	CHIME	High-dimension logistic	Sparse LDA
mean	0.1911	0.49895	0.4949167
sd	0.08120522	0.03252699	0.03079964

Table 2: 5-fold cross validation error Rates (Model 2)

	CHIME	High-dimension logistic	Sparse LDA
mean	0.02608333	0.50620000	0.50915000
sd	0.04669694	0.03930192	0.03786241

Table 3: 5-fold cross validation error Rates (Model 3)

	CHIME	High-dimension logistic	Sparse LDA
mean	0.1747833	0.5055500	0.5057833
sd	0.10244148	0.03430687	0.03456484

## 5.4 Conclusion

From our simulations, it is evident that CHIME outperforms the other models across all three scenarios. The testing error rates (out of sample predictions) for CHIME are consistently lower, indicating its superior performance in high-dimensional classification tasks. These results are promising for real-life applications in economics, where precise classification is essential, such as economic forecasting, fraud detection in financial transactions, and image recognition for economic data analysis. CHIME's strong performance in our simulations suggests that it could be a valuable tool for solving similar problems in practical, real-world settings. Further research and testing in real-life scenarios will help confirm and refine these findings.

## 6 Real Data Analysis

### 6.1 Overview

In this section, we investigate the performance of the CHIME by analyzing several real data points. We compare CHIME to the two other methods used in simulation. In addition, we compute the error rates of each method.

### 6.2 Data

The real cases we studied include:

1. Breast Cancer Data
2. Spam Data

The two datasets were recovered from the well-known UCI( University of California, Irvine) repository and have been analyzed by multiple studies.

#### 6.2.1 Dataset Description

##### Breast Cancer

Breast cancer is one of the most serious and commonly seen cancers. This dataset was downloaded from the UCI repository. <sup>1</sup>. The dataset comprises 569 observations and 32 variables. The primary goal of this dataset is to differentiate between tumors that are classified as malignant and those that are benign. Among others, Bouveyron[Bouveyron and Brunet-Saumard, 2014] applied the high-dimension clustering method to this dataset. And [Xiao et al., 2015] also use least-angle regression to study this data.

##### Spam Data

The dataset spam contains 2,788 emails marked as "not spam" and 1,813 emails marked as "spam." There are 58 variables in this dataset. This dataset was downloaded from the UCI repository. <sup>2</sup> "Spam" here covers a wide range of unwanted emails, like product ads, money-making schemes, chain letters, and explicit content. These spam emails were collected from

---

<sup>1</sup>for more information, please visit <https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic>

<sup>2</sup>for more information, please visit <https://archive.ics.uci.edu/dataset/94/spambase>

various sources, including postmasters and individuals who reported them. The "not spam" emails are typically work or personal emails. They may contain words like "george" or the area code "650," which are signs of non-spam.

This dataset was studied from [Ferri et al., 2019] and [Gramacy et al., 2013].

### 6.3 Results

Table 4: 5-fold cross validation error Rates for Breast Cancer

	CHIME	High-dimension logistic	Sparse LDA
1	0.062	0.0602	0.3292

Table 5: 5-fold cross validation error Rates for Spam Data

	CHIME	High-dimension logistic	Sparse LDA
1	0.1897744	0.3888487	0.2910934

### 6.4 Conclusion

The analysis of actual data indicates that CHIME exhibits robust performance. In the context of breast cancer, it outperforms the sparse LDA method and matches the performance of high-dimensional logistic regression. Similarly, when applied to Spam data, which is a high-dimensional dataset, CHIME surpasses both high-dimensional logistic regression and sparse LDA. These results underscore CHIME's efficacy in handling both high-dimensional and low-dimensional binary classification, implying its potential as a valuable tool for binary data classification.



## 7 Conclusion

### 7.1 Summary

In conclusion, this thesis has introduced an innovative approach to address the challenges associated with binary choice models and discriminant analysis, particularly in the context of high-dimensional data. Traditionally, methods like linear discriminant analysis (LDA) and quadratic discriminant analysis have relied on assumptions of normality, which often do not hold in real-world applications. Moreover, these traditional methods face issues related to overfitting, the curse of dimensionality, and the estimation of covariance matrices.

To overcome these limitations, this thesis proposes a novel discriminant analysis model that liberates itself from normality assumptions by employing a mixture model. This approach allows for more flexibility in modeling binary outcomes in various disciplines. The use of the CHIME method, which stands for "Clustering of high-dimensional Gaussian Mixtures with the Expectation-Maximization Algorithm," was introduced to estimate mixture model coefficients and covariance matrices in high-dimensional scenarios. CHIME is based on the EM algorithm and offers a direct estimation method for sparse discriminant vectors.

However, it's important to acknowledge that the approach presented in this thesis, which relies on mixtures of multivariate normal distribution assumptions, has certain limitations and may not be suitable for all scenarios. For instance, in fields and applications where data is known to have heavy tails, the multivariate normal assumption becomes inflexible. Examples include financial data, count data, or data with binary outcomes that do not conform to normality.

## **7.2 Possible Future Directions for Research**

Obtaining these results, this launches a variety of avenues to pursue further research.

### **7.2.1 Automatic Component Selection in Mixture Models**

Despite the widespread use of mixture models in economics, the estimation of the number of mixture components ( $k$ ) is an unsolved problem. According to [Miloslavsky and van der Laan, 2003], available methods include bootstrapping the likelihood ratio test statistic and optimizing a variety of validity functionals to guide this selection.

### **7.2.2 Extending Mixture Models to Discrete Variables with Copulas**

Mixture models can be expanded to encompass a combination of continuous and discrete variables. This extension can be facilitated through the use of copulas, which are effective tools for understanding the relationships between variables. Copulas are powerful because they reveal the underlying patterns of dependence and exhibit an invariance to changes resulting from strictly increasing transformations of random variables. Additionally, they serve as the foundational components for constructing probability distributions. [Schweizer and Wolff, 1981]

### **7.2.3 Handling Heavy-Tailed Distributions with Multivariate Mixtures**

Heavy-tailed distributions, such as the Cauchy or Student's  $t$ -distributions, pose unique challenges. Extending mixture models to multivariate, heavy-tailed distributions, like multivariate  $t$ -distributions, is essential for effectively modeling data with heavy tails. This is particularly relevant in domains where extreme events are common.

## References

- [Aigner et al., 1984] Aigner, D. J., Hsiao, C., Kapteyn, A., and Wansbeek, T. (1984). Latent variable models in econometrics. *Handbook of econometrics*, 2:1321–1393.
- [Alayande and Adekunle, 2015] Alayande, S. and Adekunle, B. (2015). An overview and application of discriminant analysis in data analysis. *IOSR Journal of Mathematics*, 11(1):12–15.
- [Bickel and Levina, 2004] Bickel, P. J. and Levina, E. (2004). Some theory for fisher’s linear discriminant function, ‘naive bayes’, and some alternatives when there are many more variables than observations. *Bernoulli: Official Journal of the Bernoulli Society for Mathematical Statistics and Probability*, 10(6):989–1010.
- [Bouveyron and Brunet-Saumard, 2014] Bouveyron, C. and Brunet-Saumard, C. (2014). Model-based clustering of high-dimensional data: A review. *Computational Statistics & Data Analysis*, 71:52–78.
- [Boyd and Vandenberghe, 2004] Boyd, S. P. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.
- [Bradley et al., 1999] Bradley, P. S., Fayyad, U. M., and Mangasarian, O. L. (1999). Mathematical programming for data mining: Formulations and challenges. *INFORMS Journal on Computing*, 11(3):217–238.
- [Cai and Liu, 2011] Cai, T. and Liu, W. (2011). A direct estimation approach to sparse linear discriminant analysis. *Journal of the American statistical association*, 106(496):1566–1577.
- [Cai et al., 2013] Cai, T., Liu, W., and Xia, Y. (2013). Two-sample covariance matrix testing and support recovery in high-dimensional and sparse settings. *Journal of the American Statistical Association*, 108(501):265–277.
- [Cai et al., 2019] Cai, T. T., Ma, J., and Zhang, L. (2019). CHIME: Clustering of High-Dimensional Gaussian Mixtures with EM Algorithm and Its Optimality. *The Annals of Statistics*, 47(3):1234–1267.
- [Clemmensen et al., 2011] Clemmensen, L., Hastie, T., Witten, D., and Ersbøll, B. (2011). Sparse discriminant analysis. *Technometrics*, 53(4):406–413.

- [Cole et al., 2014] Cole, S. R., Chu, H., and Greenland, S. (2014). Maximum likelihood, profile likelihood, and penalized likelihood: a primer. *American journal of epidemiology*, 179(2):252–260.
- [Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B, Methodological*, 39(1):1–38.
- [Fan and Fan, 2008] Fan, J. and Fan, Y. (2008). High-dimensional classification using features annealed independence rules. *The Annals of Statistics*, 36(6):2605–2637.
- [Ferri et al., 2019] Ferri, C., Hernández-Orallo, J., and Flach, P. (2019). Setting decision thresholds when operating conditions are uncertain. *Data mining and knowledge discovery*, 33:805–847.
- [Friedman et al., 2010] Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1.
- [Gramacy et al., 2013] Gramacy, R. B., Taddy, M., and Wild, S. M. (2013). Variable selection and sensitivity analysis using dynamic trees, with an application to computer code performance tuning. *The Annals of Applied Statistics*, pages 51–80.
- [Hastie et al., 1995] Hastie, T., Buja, A., and Tibshirani, R. (1995). Penalized discriminant analysis. *The Annals of Statistics*, 23(1):73–102.
- [Hastie et al., 2015] Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical learning with sparsity: the lasso and generalizations*. CRC press.
- [Li et al., 2022] Li, J., Chao, X., and Xu, Q. (2022). Multi-classification for high-dimensional data using probabilistic neural networks. *Journal of Radiation Research and Applied Sciences*, 15(2):111–118.
- [Mai and Zou, 2015] Mai, Q. and Zou, H. (2015). Sparse semiparametric discriminant analysis. *Journal of Multivariate Analysis*, 135:175–188.
- [McLachlan, 2012] McLachlan, G. J. (2012). Discriminant analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(5):421–431.

- [McLachlan et al., 2019] McLachlan, G. J., Lee, S. X., and Rathnayake, S. I. (2019). Finite mixture models. *Annual Review of Statistics and Its Application*, 6(1):355–378.
- [Miloslavsky and van der Laan, 2003] Miloslavsky, M. and van der Laan, M. J. (2003). Fitting of mixtures with unspecified number of components using cross validation distance estimate. *Computational statistics & data analysis*, 41(3-4):413–428.
- [Pappu and Pardalos, 2014] Pappu, V. and Pardalos, P. M. (2014). High-dimensional data classification. In *Clusters, Orders, and Trees: Methods and Applications*, pages 119–150. Springer New York, New York, NY.
- [Parsons et al., 2004] Parsons, L., Haque, E., and Liu, H. (2004). Subspace clustering for high-dimensional data. *SIGKDD Explorations*, 6(1):90–105.
- [Refaeilzadeh et al., 2018] Refaeilzadeh, P., Tang, L., and Liu, H. (2018). Cross-validation. In *Encyclopedia of Database Systems*, pages 677–684. Springer New York.
- [Roy, 2015] Roy, A. (2015). A classification algorithm for high-dimensional data. *Procedia Computer Science*, 53(1):345–355.
- [Schweizer and Wolff, 1981] Schweizer, B. and Wolff, E. F. (1981). On nonparametric measures of dependence for random variables. *The annals of statistics*, 9(4):879–885.
- [Scott and Symons, 1971] Scott, A. J. and Symons, M. J. (1971). Clustering methods based on likelihood ratio criteria. *Biometrics*, pages 387–397.
- [Shao et al., 2011] Shao, J., Wang, Y., Deng, X., and Wang, S. (2011). Sparse linear discriminant analysis by thresholding for high dimensional data. *The Annals of Statistics*, 39(2).
- [Thrun and Ultsch, 2020] Thrun, M. C. and Ultsch, A. (2020). Uncovering high-dimensional structures of projections from dimensionality reduction methods. *MethodsX*, 7:101093.
- [Van de Geer, 2008] Van de Geer, S. A. (2008). High-dimensional generalized linear models and the lasso.
- [Wang et al., 2013] Wang, Z., Gu, Q., Ning, Y., and Liu, H. (2013). High dimensional expectation-maximization algorithm: Statistical optimization and asymptotic normality. *The Annals of Statistics*, 41(6):2791–2823.

- [Wu et al., 2019] Wu, Y., Qin, Y., and Zhu, M. (2019). Quadratic discriminant analysis for high-dimensional data. *Statistica Sinica*, 29(2):939–960.
- [Xiao et al., 2015] Xiao, W., Wu, Y., and Zhou, H. (2015). Convexlar: an extension of least angle regression. *Journal of Computational and Graphical Statistics*, 24(3):603–626.
- [Zubair et al., 2022] Zubair, M., Iqbal, M. A., Shil, A., Chowdhury, M. J. M., Moni, M. A., and Sarker, I. H. (2022). An improved k-means clustering algorithm towards an efficient data-driven modeling. *Annals of Data Science*, pages 1–20.

## **8 Appendix**

### **8.1 Codes for Model1**

#### **8.1.1 Explanations of the Code**

The code includes my own implementation of the CHIME algorithm, and the code for discriminant analysis model based on mixtures that I derived. The code also includes the Monte-Carlo simulations for model 1 as well as the 5-fold cross validation calculations. The code calls on the R packages "glmnet" for high-dimension logistic regression and "sparseLDA" for sparse linear discriminant analysis. Model 2 and 3 are not included because the codes are very similar.

```
1 library(mixtools)
2 library(MASS) # for generating multivariate normal data
3 library(CVXR)
4 library(glmnet)
5 library(mvtnorm)
6 library(ggplot2)
7 library(dplyr)
8 library(sparseLDA)
9
10 # Set the number of simulations
11 num_simulations <- 100
12
13 # Create vectors to store correctness rates for Chime, glm, and lda
14 Chime_error_rates <- numeric(num_simulations)
15 glmnet_error_rates <- numeric(num_simulations)
16 Sparse_lda_error_rates <- numeric(num_simulations)
17
18 # Repeat the simulation 100 times
19 for (sim in 1:num_simulations){
20
21
22 # Set the number of rows for your data frame
23 n <- 400 # You can change this to your desired number of rows
24
25 # Create 'Y' data frame
26 Y= ifelse(runif(n) > 0.5, 1, 0)
27
28
29
30
31 #
32 # Generate some example data
33 n <-sum(Y==0)
34
35
36 #label0
37 # Define means and covariance matrices for 'X' and 'X1'
38 # Generate the dataset
39 p<-100
40 mean_1 <- rep(0, p)
41 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
42 mean_2 <- rep(0, p)
43 mean_2[1:10] <- seq(-0.2, -2, by = -0.2)
44
```



```

45 #
46 cov_matrix <- matrix(0.8, nrow = p, ncol = p)
47 for (i in 1:p) {
48   for (j in 1:p) {
49     cov_matrix[i, j] <- 0.8^abs(i - j)
50   }
51 }
52
53
54
55
56 Sigma <- solve(cov_matrix)
57 # Covariance matrix for 'X1'
58 w<-100/n
59 # Generate 'X' and 'X1' using multivariate normal distribution
60 X <- mvrnorm(100, mu = mean_1, Sigma = Sigma)
61 X1 <- mvrnorm(n-100, mu = mean_2, Sigma = Sigma)
62 Y1 = rep(0, sum(Y==0) )
63 Data <- rbind(X,X1)
64 Data1 <- cbind(Y = Y1, Data)
65
66
67
68
69
70
71
72 #
73 # Generate some example data
74 n <- sum(Y==1)
75
76
77 #label0
78 # Define means and covariance matrices for 'X' and 'X1'
79 # Generate the dataset
80 p<-100
81 mean_1 <- rep(0, p)
82 mean_1[1:10] <- seq(10, 20, by = 1)
83 mean_2 <- rep(0, p)
84 mean_2[1:10] <- seq(-10, -20, by = -1)
85
86 #
87 cov_matrix <- matrix(0.8, nrow = p, ncol = p)
88 for (i in 1:p) {

```

```

89   for (j in 1:p) {
90     cov_matrix[i, j] <- 0.8^abs(i - j)
91   }
92 }
93
94
95
96
97 Sigma <- solve(cov_matrix)
98 # Covariance matrix for 'X1'
99 w<-100/n
100 # Generate 'X' and 'X1' using multivariate normal distribution
101 X <- mvrnorm(100, mu = mean_1, Sigma = Sigma)
102 X1 <- mvrnorm(n-100, mu = mean_2, Sigma = Sigma)
103 Y2 = rep(1, n)
104 Data <- rbind(X,X1)
105 Data2<- cbind(Y = Y2, Data)
106
107 Data<-rbind(Data1,Data2)
108 Data<- as.data.frame(Data)
109
110
111
112 #
113 # Create a vector to store correctness rates for each fold
114 # Number of folds (you can change this as needed)
115 k <- 5
116
117 # Create a list to store training and testing data for each fold
118 data_folds <- list()
119
120 # Perform 5-fold cross-validation
121 for (fold in 1:k) {
122   # Split the data into training and testing folds
123   split_index <- sample(1:nrow(Data), 0.2 * nrow(Data))
124   train_data_fold <- Data[-split_index, ]
125   test_data_fold <-Data[split_index, ]
126
127   # Store the data folds in the list
128   data_folds[[fold]] <- list(train_data = train_data_fold, test_data = test_data_fold)
129 }
130
131 ## Perform 5-fold cross-validation
132 #first fold

```

```
133 train_data <- data_folds[[1]]$train_data
134 test_data <- data_folds[[1]]$test_data
135
136
137 split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
138 train_data <- Data[split_index, ]
139 test_data <- Data[-split_index, ]
140
141 # Create labels for training and testing data
142 training_labels <- train_data$Y
143 testing_labels <- test_data$Y
144
145
146
147 ##
148 train_good_data <- train_data[train_data$Y== "1", ]
149 train_bad_data <- train_data[train_data$Y == "0", ]
150
151 train_good_data <- train_good_data%>%
152   select(-Y)
153
154 train_bad_data <- train_bad_data%>%
155   select(-Y)
156
157
158 test_data <- test_data%>%
159   select(-Y)
160
161 train_data <- train_data%>%
162   select(-Y)
163
164
165 #
166 data <- as.matrix(train_good_data)
167 n <- nrow(data)
168 p <- ncol(data)
169 s <- sqrt(n)
170
171
172
173
174
175
176
```

```

177  ## Choose number of mixture components
178  k <- 2
179
180  # Use k-means to initialize the parameters
181  mean_1 <- rep(0,p)
182  mean_1[1:10] <- seq(0.1, 1, by = 0.1)
183  kmeans_fit <- kmeans(data, centers = k)
184  means_init <- kmeans_fit$centers
185  distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
186  # Find the index of the center that is closer to m1
187  closest_center_index <- which.min(distances)
188
189  # Assign mu_1 to the closer center and mu_2 to the other center
190  mu_1 <- means_init[closest_center_index, ]
191  mu_2 <- means_init[-closest_center_index, ]
192
193
194  # Calculate the covariance matrices of the two clusters
195  cluster_indices <- kmeans_fit$cluster == closest_center_index
196
197  # Subset the data points that belong to the cluster associated with mu_1
198  cluster_data <- data[cluster_indices, ]
199
200  # Calculate the covariance matrix for this cluster
201  cov_matrix_1 <- cov(cluster_data)
202
203  #
204  # Get the indices of data points belonging to the cluster associated with mu_2
205  cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
206
207  # Subset the data points that belong to the cluster associated with mu_2
208  cluster_data_mu2 <- data[cluster_indices_mu2, ]
209
210  # Calculate the covariance matrix for this cluster
211  cov_matrix_2 <- cov(cluster_data_mu2)
212
213  cov<-(cov_matrix_1+cov_matrix_2)/2
214
215
216
217  # Initial Step
218  # Update 'beta_optimal' based on 'cov'
219  Clambda <- 210# Adjust as needed
220  lambda_0 <- Clambda * sqrt(log(p) / n)

```

```

221 beta <- Variable(p)
222
223 # Define the objective function with L1 regularization
224 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))
225
226 # Define the problem
227 problem <- Problem(objective)
228
229 # Solve the problem
230 result <- solve(problem)
231
232 # Get the updated 'beta_optimal'
233 beta_optimal <- result$getValue(beta)
234
235 n <- nrow(data)
236 a <- numeric(n)
237 w <- nrow(cluster_data_mu2) / nrow(train_good_data)
238 for (i in 1:n) {
239   a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
240 }
241
242
243
244 # Update 'w'
245 w <- sum(a) / n
246
247 # M-step
248 # Update 'mu' and 'cov' based on 'a'
249 # M-step
250 # Update 'mu' and 'cov' based on 'a'
251 mu <- matrix(0, nrow = 2, ncol = p)
252 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %*% as.matrix(data))
253 mu[2, ] <- (sum(a))(-1) * (a %*% as.matrix(data))
254 mu_1 <- mu[1, ]
255 mu_2 <- mu[2, ]
256
257 cov <- matrix(0, nrow = p, ncol = p)
258 for (i in 1:n) {
259   cov <- cov + 1/n * (
260     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
261     a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
262   )
263 }

```

```

264
265
266
267
268
269 # Perform the iterative algorithm
270 convergence_threshold <- 10
271 max_iterations <- Inf
272 iteration <- 1
273
274 # Create a vector to store mu for this iteration
275 while (iteration <= max_iterations) {
276   prev_mu_1 <- mu_1
277   # E-step
278   n <- nrow(data)
279   a <- numeric(n)
280   for (i in 1:n) {
281     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ] - (mu_1 + mu_2)
282       / 2)))
283   }
284   # Update 'w'
285   w <- sum(a) / n
286
287   # M-step
288   # Update 'mu' and 'cov' based on 'a'
289
290
291   # M-step
292   # Update 'mu' and 'cov' based on 'a'
293   mu <- matrix(0, nrow = 2, ncol = p)
294   mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %*% as.matrix(data))
295   mu[2, ] <- (sum(a))(-1) * (a %*% as.matrix(data))
296   mu_1 <- mu[1, ]
297   mu_2 <- mu[2, ]
298
299   cov <- matrix(0, nrow = p, ncol = p)
300   for (i in 1:n) {
301     cov <- cov + 1/n * (
302       (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
303       a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
304     )
305   }
306

```

```

307 # Update 'beta_optimal' based on 'cov'
308 Clambda <- 210# Adjust as needed
309 k<-0.1
310 lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
311 beta <- Variable(p)
312
313 # Define the objective function with L1 regularization
314 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
    0 * sum(p_norm(beta, 1))))
315
316 # Define the problem
317 problem <- Problem(objective)
318
319 # Solve the problem
320 result <- solve(problem)
321
322 # Get the updated 'beta_optimal'
323 beta_optimal <- result$getValue(beta)
324
325
326 # Check for convergence
327 change_in_mu_1<- abs(mu_1 - prev_mu_1)
328 if (sum(change_in_mu_1)<convergence_threshold) {
329     break
330 }
331
332 # Increment the iteration counter
333 iteration <- iteration + 1
334 }
335
336
337
338 #
339 pdf<-numeric(120)
340 # Calculate density estimates and PDF values for each data point
341 for (i in 1:120) {
342     data_point <- test_data[i,]
343     density_estimations_cluster_1 <- dmnorm(data_point, mean = mu_1, sigma = cov)
344     density_estimations_cluster_2 <- dmnorm(data_point, mean = mu_2, sigma=cov)
345
346     pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
347 }
348
349 pdf

```

```
350
351
352
353 #
354 data<- train_bad_data
355 n <- nrow(data)
356 p <- ncol(data)
357 s <- sqrt(n)
358
359
360
361
362
363
364
365 ## Choose number of mixture components
366 k <- 2
367
368 # Use k-means to initialize the parameters
369 mean_1 <- rep(0,p)
370 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
371 kmeans_fit <- kmeans(data, centers = k)
372 means_init <- kmeans_fit$centers
373 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
374 # Find the index of the center that is closer to m1
375 closest_center_index <- which.min(distances)
376
377 # Assign mu_1 to the closer center and mu_2 to the other center
378 mu_1 <- means_init[closest_center_index, ]
379 mu_2 <- means_init[-closest_center_index, ]
380
381
382 # Calculate the covariance matrices of the two clusters
383 cluster_indices <- kmeans_fit$cluster == closest_center_index
384
385 # Subset the data points that belong to the cluster associated with mu_1
386 cluster_data <- data[cluster_indices, ]
387
388 # Calculate the covariance matrix for this cluster
389 cov_matrix_1 <- cov(cluster_data)
390
391 #
392 # Get the indices of data points belonging to the cluster associated with mu_2
393 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
```



```

394
395 # Subset the data points that belong to the cluster associated with mu_2
396 cluster_data_mu2 <- data[cluster_indices_mu2, ]
397
398 # Calculate the covariance matrix for this cluster
399 cov_matrix_2 <- cov(cluster_data_mu2)
400
401 cov<-(cov_matrix_1+cov_matrix_2)/2
402
403
404
405 # Initial Step
406 # Update 'beta_optimal' based on 'cov'
407 Clambda <- 7# Adjust as needed
408 lambda_0 <- Clambda * sqrt(log(p) / n)
409 beta <- Variable(p)
410
411 # Define the objective function with L1 regularization
412 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))
413
414 # Define the problem
415 problem <- Problem(objective)
416
417 # Solve the problem
418 result <- solve(problem)
419
420 # Get the updated 'beta_optimal'
421 beta_optimal <- result$getValue(beta)
422 # E-step
423 n <- nrow(data)
424 a <- numeric(n)
425 w <- nrow(cluster_data_mu2)/nrow(train_good_data)
426 for (i in 1:n) {
427   a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
    2))))
428 }
429
430
431 # Update 'w'
432 w <- sum(a) / n
433
434 # M-step
435 # Update 'mu' and 'cov' based on 'a'

```

```

436 # M-step
437 # Update 'mu' and 'cov' based on 'a'
438 mu <- matrix(0, nrow = 2, ncol = p)
439 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %%% as.matrix(data))
440 mu[2, ] <- (sum(a))(-1) * (a %%% as.matrix(data))
441 mu_1 <- mu[1, ]
442 mu_2 <- mu[2, ]
443
444 cov <- matrix(0, nrow = p, ncol = p)
445 for (i in 1:n) {
446   cov <- cov + 1/n * (
447     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %%% t(as.numeric(data[i, ]) - mu_1) +
448     a[i] * (as.numeric(data[i, ]) - mu_2) %%% t(as.numeric(data[i, ]) - mu_2)
449   )
450 }
451
452
453
454
455
456 # Perform the iterative algorithm
457 convergence_threshold <- 1
458 max_iterations <- Inf
459 iteration <- 1
460
461 # Create a vector to store mu for this iteration
462 while (iteration <= max_iterations) {
463   prev_mu_1 <- mu_1
464   # E-step
465   n <- nrow(data)
466   a <- numeric(n)
467   for (i in 1:n) {
468     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %%% (as.numeric(data[i, ]) - (mu_1 + mu_2)
469       / 2))))
470   }
471
472   # Update 'w'
473   w <- sum(a) / n
474
475   # M-step
476   # Update 'mu' and 'cov' based on 'a'
477
478   # M-step

```

```

479 # Update 'mu' and 'cov' based on 'a'
480 mu <- matrix(0, nrow = 2, ncol = p)
481 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %>% as.matrix(data))
482 mu[2, ] <- (sum(a))(-1) * (a %>% as.matrix(data))
483 mu_1 <- mu[1, ]
484 mu_2 <- mu[2, ]
485
486 cov <- matrix(0, nrow = p, ncol = p)
487 for (i in 1:n) {
488   cov <- cov + 1/n * (
489     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %>% t(as.numeric(data[i, ]) - mu_1) +
490     a[i] * (as.numeric(data[i, ]) - mu_2) %>% t(as.numeric(data[i, ]) - mu_2)
491   )
492 }
493
494 # Update 'beta_optimal' based on 'cov'
495 Clambda <- 7# Adjust as needed
496 k <- 0.1
497 lambda_0 <- k*lambda_0 + Clambda * sqrt(log(p) / n)
498 beta <- Variable(p)
499
500 # Define the objective function with L1 regularization
501 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %>% (mu_1 - mu_2) + lambda_
502   0 * sum(p_norm(beta, 1))))
503
504 # Define the problem
505 problem <- Problem(objective)
506
507 # Solve the problem
508 result <- solve(problem)
509
510 # Get the updated 'beta_optimal'
511 beta_optimal <- result$getValue(beta)
512
513 # Check for convergence
514 change_in_mu_1 <- abs(mu_1 - prev_mu_1)
515 if (sum(change_in_mu_1) < convergence_threshold) {
516   break
517 }
518
519 # Increment the iteration counter
520 iteration <- iteration + 1
521 }

```

```

522
523
524
525 #
526 pdf1<-numeric(120)
527 # Calculate density estimates and PDF values for each data point
528 for (i in 1:120) {
529   data_point <- test_data[i,]
530   density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
531   density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)
532
533   pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
534 }
535
536
537
538
539
540
541
542
543
544 # Create vectors to store probabilities
545 probabilities <- numeric(nrow(test_data))
546
547 # Iterate through each data point
548 for (i in 1:nrow(test_data)) {
549   pdf_value <- pdf[i] # Density from the first GMM
550   pdf1_value <- pdf1[i] # Density from the second GMM
551   y <- sum(testing_labels == 1)
552   z<- sum(testing_labels == 0)
553
554   # Calculate probability using the formula
555   probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)
556
557   # Store the probability in the vector
558   probabilities[i] <- probability
559 }
560
561 threshold<-0.5
562 # Classify instances based on the threshold
563 classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
564 classification<- na.omit(classification)
565 testing_labels

```

```
566 classification
567
568 Chime_correct_rates_1<-sum(testing_labels==classification)/120
569
570
571
572 ## Load the glmnet package if not already loaded
573 if (!require(glmnet)) {
574   install.packages("glmnet")
575   library(glmnet)
576 }
577
578 # Define the response variable (Y) and predictor variables (X) for training data
579 training_response <- as.numeric(training_labels)
580 training_predictors <- as.matrix(train_data) # Exclude the first column (Y)
581
582 # Fit a generalized linear model with elastic net regularization
583 alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
584 lambda <- 0.5 # Adjust the regularization strength (tune this parameter)
585
586 # Create the glmnet model
587 model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)
588
589 # Print a summary of the model
590 print(model)
591
592 # Plot the coefficients along the regularization path
593 plot(model)
594
595 # To make predictions on test data, you can use the predict() function
596 # Define the predictor variables for test data
597 testing_predictors <- as.matrix(test_data) # Exclude the first column (Y)
598
599 # Make predictions on the test data
600 predictions <- predict(model, newx = testing_predictors, s = lambda)
601
602 threshold<-0.5
603 # Classify instances based on the threshold
604 classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
605 classification1<- na.omit(classification1)
606 testing_labels
607 classification1
608
609 glmnet_correct_rates_1<-sum(testing_labels==classification1)/120
```

```

610
611
612
613
614
615 #
616 # Load data
617 X <- as.matrix(train_data)
618 Y <- as.matrix(training_labels)
619
620
621 new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1) # Creating a new column
        with zeros
622 colnames(new_column) <- "NewColumn" # Setting the column name (change "NewColumn" to your
        desired name)
623
624 # Combine Y and the new column using cbind
625 Y <- cbind(Y, new_column)
626
627 colnames(Y) <- c("0",
628                 "1")
629 ## training data
630 Xtr<-X
631 k<-2
632 n<-dim(Xtr)[1]
633 ## Normalize data
634 Xc<-normalize(Xtr)
635 Xn<-Xc$Xc
636 p<-dim(Xn)[2]
637 ## Perform SDA with one non-zero loading for each discriminative
638 ## direction with Y as matrix input
639 out <- sda(Xn, Y,
640           lambda = 1e-6,
641           stop = -1,
642           maxIte = 25,
643           trace = TRUE)
644 ## predict training samples
645 train <- predict(out, Xn)
646 ## testing
647 Xtst<-as.matrix(test_data)
648 Xtst<-normalizetest(Xtst,Xc)
649
650 test <- predict(out, Xtst)
651 print(test$class)

```

```

652
653
654 Sparse_lda_correct_rates_1<-sum(testing_labels==test$class)/120
655
656
657 ## Perform 5-fold cross-validation
658 #second fold
659 train_data <- data_folds[[2]]$train_data
660 test_data <- data_folds[[2]]$test_data
661
662
663 split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
664 train_data <- Data[split_index, ]
665 test_data <- Data[-split_index, ]
666
667 # Create labels for training and testing data
668 training_labels <- train_data$Y
669 testing_labels <- test_data$Y
670
671
672
673 ##
674 train_good_data <- train_data[train_data$Y== "1", ]
675 train_bad_data <- train_data[train_data$Y == "0", ]
676
677 train_good_data <- train_good_data%>%
678   select(-Y)
679
680 train_bad_data <- train_bad_data%>%
681   select(-Y)
682
683
684 test_data <- test_data%>%
685   select(-Y)
686
687 train_data<- train_data%>%
688   select(-Y)
689
690
691 #
692 data<- as.matrix(train_good_data)
693 n <- nrow(data)
694 p <- ncol(data)
695 s <- sqrt(n)

```

```

696
697
698
699
700
701
702
703 ## Choose number of mixture components
704 k <- 2
705
706 # Use k-means to initialize the parameters
707 mean_1 <- rep(0,p)
708 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
709 kmeans_fit <- kmeans(data, centers = k)
710 means_init <- kmeans_fit$centers
711 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
712 # Find the index of the center that is closer to m1
713 closest_center_index <- which.min(distances)
714
715 # Assign mu_1 to the closer center and mu_2 to the other center
716 mu_1 <- means_init[closest_center_index, ]
717 mu_2 <- means_init[-closest_center_index, ]
718
719
720 # Calculate the covariance matrices of the two clusters
721 cluster_indices <- kmeans_fit$cluster == closest_center_index
722
723 # Subset the data points that belong to the cluster associated with mu_1
724 cluster_data <- data[cluster_indices, ]
725
726 # Calculate the covariance matrix for this cluster
727 cov_matrix_1 <- cov(cluster_data)
728
729 #
730 # Get the indices of data points belonging to the cluster associated with mu_2
731 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
732
733 # Subset the data points that belong to the cluster associated with mu_2
734 cluster_data_mu2 <- data[cluster_indices_mu2, ]
735
736 # Calculate the covariance matrix for this cluster
737 cov_matrix_2 <- cov(cluster_data_mu2)
738
739 cov<-(cov_matrix_1+cov_matrix_2)/2

```



```

740
741
742
743 # Initial Step
744 # Update 'beta_optimal' based on 'cov'
745 Clambda <- 210# Adjust as needed
746 lambda_0 <- Clambda * sqrt(log(p) / n)
747 beta <- Variable(p)
748
749 # Define the objective function with L1 regularization
750 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
751   * sum(p_norm(beta, 1))))
752
753 # Define the problem
754 problem <- Problem(objective)
755
756 # Solve the problem
757 result <- solve(problem)
758
759 # Get the updated 'beta_optimal'
760 beta_optimal <- result$getValue(beta)
761
762 n <- nrow(data)
763 a <- numeric(n)
764 w <- nrow(cluster_data_mu2) / nrow(train_good_data)
765 for (i in 1:n) {
766   a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
767 }
768
769
770 # Update 'w'
771 w <- sum(a) / n
772
773 # M-step
774 # Update 'mu' and 'cov' based on 'a'
775 # M-step
776 # Update 'mu' and 'cov' based on 'a'
777 mu <- matrix(0, nrow = 2, ncol = p)
778 mu[1, ] <- (n - sum(a))^( -1) * ((1 - a) %*% as.matrix(data))
779 mu[2, ] <- (sum(a))^( -1) * (a %*% as.matrix(data))
780 mu_1 <- mu[1, ]
781 mu_2 <- mu[2, ]
782

```

```

783 cov <- matrix(0, nrow = p, ncol = p)
784 for (i in 1:n) {
785   cov <- cov + 1/n * (
786     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
787     a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
788   )
789 }
790
791
792
793
794
795 # Perform the iterative algorithm
796 convergence_threshold <- 10
797 max_iterations <- Inf
798 iteration <- 1
799
800 # Create a vector to store mu for this iteration
801 while (iteration <= max_iterations) {
802   prev_mu_1 <- mu_1
803   # E-step
804   n <- nrow(data)
805   a <- numeric(n)
806   for (i in 1:n) {
807     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ]) - (mu_1 + mu_2)
808       / 2)))
809   }
810
811   # Update 'w'
812   w <- sum(a) / n
813
814   # M-step
815   # Update 'mu' and 'cov' based on 'a'
816
817   # M-step
818   # Update 'mu' and 'cov' based on 'a'
819   mu <- matrix(0, nrow = 2, ncol = p)
820   mu[1, ] <- (n - sum(a))^( -1) * ((1 - a) %*% as.matrix(data))
821   mu[2, ] <- (sum(a))^( -1) * (a %*% as.matrix(data))
822   mu_1 <- mu[1, ]
823   mu_2 <- mu[2, ]
824
825   cov <- matrix(0, nrow = p, ncol = p)

```

```

826   for (i in 1:n) {
827     cov <- cov + 1/n * (
828       (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
829       a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
830     )
831   }
832
833   # Update 'beta_optimal' based on 'cov'
834   Clambda <- 210 # Adjust as needed
835   k <- 0.1
836   lambda_0 <- k * lambda_0 + Clambda * sqrt(log(p) / n)
837   beta <- Variable(p)
838
839   # Define the objective function with L1 regularization
840   objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
841     0 * sum(p_norm(beta, 1))))
842
843   # Define the problem
844   problem <- Problem(objective)
845
846   # Solve the problem
847   result <- solve(problem)
848
849   # Get the updated 'beta_optimal'
850   beta_optimal <- result$getValue(beta)
851
852   # Check for convergence
853   change_in_mu_1 <- abs(mu_1 - prev_mu_1)
854   if (sum(change_in_mu_1) < convergence_threshold) {
855     break
856   }
857
858   # Increment the iteration counter
859   iteration <- iteration + 1
860 }
861
862 #
863 pdf <- numeric(120)
864 # Calculate density estimates and PDF values for each data point
865 for (i in 1:120) {
866   data_point <- test_data[i,]

```

```

869 density_estimations_cluster_1 <- dmnorm(data_point, mean = mu_1, sigma = cov)
870 density_estimations_cluster_2 <- dmnorm(data_point, mean = mu_2, sigma=cov)
871
872 pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
873 }
874
875 pdf
876
877
878
879 #
880 data<- train_bad_data
881 n <- nrow(data)
882 p <- ncol(data)
883 s <- sqrt(n)
884
885
886
887
888
889
890
891 ## Choose number of mixture components
892 k <- 2
893
894 # Use k-means to initialize the parameters
895 mean_1 <- rep(0,p)
896 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
897 kmeans_fit <- kmeans(data, centers = k)
898 means_init <- kmeans_fit$centers
899 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
900 # Find the index of the center that is closer to m1
901 closest_center_index <- which.min(distances)
902
903 # Assign mu_1 to the closer center and mu_2 to the other center
904 mu_1 <- means_init[closest_center_index, ]
905 mu_2 <- means_init[-closest_center_index, ]
906
907
908 # Calculate the covariance matrices of the two clusters
909 cluster_indices <- kmeans_fit$cluster == closest_center_index
910
911 # Subset the data points that belong to the cluster associated with mu_1
912 cluster_data <- data[cluster_indices, ]

```

```

913
914 # Calculate the covariance matrix for this cluster
915 cov_matrix_1 <- cov(cluster_data)
916
917 #
918 # Get the indices of data points belonging to the cluster associated with mu_2
919 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
920
921 # Subset the data points that belong to the cluster associated with mu_2
922 cluster_data_mu2 <- data[cluster_indices_mu2, ]
923
924 # Calculate the covariance matrix for this cluster
925 cov_matrix_2 <- cov(cluster_data_mu2)
926
927 cov<-(cov_matrix_1+cov_matrix_2)/2
928
929
930
931 # Initial Step
932 # Update 'beta_optimal' based on 'cov'
933 Clambda <- 7# Adjust as needed
934 lambda_0 <- Clambda * sqrt(log(p) / n)
935 beta <- Variable(p)
936
937 # Define the objective function with L1 regularization
938 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
939   * sum(p_norm(beta, 1))))
940
941 # Define the problem
942 problem <- Problem(objective)
943
944 # Solve the problem
945 result <- solve(problem)
946
947 # Get the updated 'beta_optimal'
948 beta_optimal <- result$getValue(beta)
949
950 # E-step
951 n <- nrow(data)
952 a <- numeric(n)
953 w <- nrow(cluster_data_mu2)/nrow(train_good_data)
954 for (i in 1:n) {
955   a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
956     2))))
957 }

```

```

955
956
957 # Update 'w'
958 w <- sum(a) / n
959
960 # M-step
961 # Update 'mu' and 'cov' based on 'a'
962 # M-step
963 # Update 'mu' and 'cov' based on 'a'
964 mu <- matrix(0, nrow = 2, ncol = p)
965 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %%% as.matrix(data))
966 mu[2, ] <- (sum(a))(-1) * (a %%% as.matrix(data))
967 mu_1 <- mu[1, ]
968 mu_2 <- mu[2, ]
969
970 cov <- matrix(0, nrow = p, ncol = p)
971 for (i in 1:n) {
972   cov <- cov + 1/n * (
973     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %%% t(as.numeric(data[i, ]) - mu_1) +
974     a[i] * (as.numeric(data[i, ]) - mu_2) %%% t(as.numeric(data[i, ]) - mu_2)
975   )
976 }
977
978
979
980
981
982 # Perform the iterative algorithm
983 convergence_threshold <- 1
984 max_iterations <- Inf
985 iteration <- 1
986
987 # Create a vector to store mu for this iteration
988 while (iteration <= max_iterations) {
989   prev_mu_1 <- mu_1
990   # E-step
991   n <- nrow(data)
992   a <- numeric(n)
993   for (i in 1:n) {
994     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %%% (as.numeric(data[i, ]) - (mu_1 + mu_2)
995       / 2))))
996   }
997   # Update 'w'

```

```

998 w <- sum(a) / n
999
1000 # M-step
1001 # Update 'mu' and 'cov' based on 'a'
1002
1003
1004 # M-step
1005 # Update 'mu' and 'cov' based on 'a'
1006 mu <- matrix(0, nrow = 2, ncol = p)
1007 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %>% as.matrix(data))
1008 mu[2, ] <- (sum(a))(-1) * (a %>% as.matrix(data))
1009 mu_1 <- mu[1, ]
1010 mu_2 <- mu[2, ]
1011
1012 cov <- matrix(0, nrow = p, ncol = p)
1013 for (i in 1:n) {
1014   cov <- cov + 1/n * (
1015     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %>% t(as.numeric(data[i, ]) - mu_1) +
1016     a[i] * (as.numeric(data[i, ]) - mu_2) %>% t(as.numeric(data[i, ]) - mu_2)
1017   )
1018 }
1019
1020 # Update 'beta_optimal' based on 'cov'
1021 Clambda <- 7# Adjust as needed
1022 k <- 0.1
1023 lambda_0 <- k*lambda_0 + Clambda * sqrt(log(p) / n)
1024 beta <- Variable(p)
1025
1026 # Define the objective function with L1 regularization
1027 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %>% (mu_1 - mu_2) + lambda_
1028   0 * sum(p_norm(beta, 1))))
1029
1030 # Define the problem
1031 problem <- Problem(objective)
1032
1033 # Solve the problem
1034 result <- solve(problem)
1035
1036 # Get the updated 'beta_optimal'
1037 beta_optimal <- result$getValue(beta)
1038
1039 # Check for convergence
1040 change_in_mu_1 <- abs(mu_1 - prev_mu_1)

```

```

1041     if (sum(change_in_mu_1)<convergence_threshold) {
1042         break
1043     }
1044
1045     # Increment the iteration counter
1046     iteration <- iteration + 1
1047 }
1048
1049
1050
1051 #
1052 pdf1<-numeric(120)
1053 # Calculate density estimates and PDF values for each data point
1054 for (i in 1:120) {
1055     data_point <- test_data[i,]
1056     density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
1057     density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)
1058
1059     pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
1060 }
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070 # Create vectors to store probabilities
1071 probabilities <- numeric(nrow(test_data))
1072
1073 # Iterate through each data point
1074 for (i in 1:nrow(test_data)) {
1075     pdf_value <- pdf[i] # Density from the first GMM
1076     pdf1_value <- pdf1[i] # Density from the second GMM
1077     y <- sum(testing_labels == 1)
1078     z<- sum(testing_labels == 0)
1079
1080     # Calculate probability using the formula
1081     probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)
1082
1083     # Store the probability in the vector
1084     probabilities[i] <- probability

```



```

1085 }
1086
1087 threshold<-0.5
1088 # Classify instances based on the threshold
1089 classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
1090 classification<- na.omit(classification)
1091 testing_labels
1092 classification
1093
1094 Chime_correct_rates_2<-sum(testing_labels==classification)/120
1095
1096
1097
1098 ## Load the glmnet package if not already loaded
1099 if (!require(glmnet)) {
1100   install.packages("glmnet")
1101   library(glmnet)
1102 }
1103
1104 # Define the response variable (Y) and predictor variables (X) for training data
1105 training_response <- as.numeric(training_labels)
1106 training_predictors <- as.matrix(train_data) # Exclude the first column (Y)
1107
1108 # Fit a generalized linear model with elastic net regularization
1109 alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
1110 lambda <- 0.5 # Adjust the regularization strength (tune this parameter)
1111
1112 # Create the glmnet model
1113 model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)
1114
1115 # Print a summary of the model
1116 print(model)
1117
1118 # Plot the coefficients along the regularization path
1119 plot(model)
1120
1121 # To make predictions on test data, you can use the predict() function
1122 # Define the predictor variables for test data
1123 testing_predictors <- as.matrix(test_data) # Exclude the first column (Y)
1124
1125 # Make predictions on the test data
1126 predictions <- predict(model, newx = testing_predictors, s = lambda)
1127
1128 threshold<-0.5

```

```

1129 # Classify instances based on the threshold
1130 classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
1131 classification1<- na.omit(classification1)
1132 testing_labels
1133 classification1
1134
1135 glmnet_correct_rates_2<-sum(testing_labels==classification1)/120
1136
1137
1138
1139
1140
1141 #
1142 # Load data
1143 X <- as.matrix(train_data)
1144 Y <- as.matrix(training_labels)
1145
1146
1147 new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1) # Creating a new column
    with zeros
1148 colnames(new_column) <- "NewColumn" # Setting the column name (change "NewColumn" to your
    desired name)
1149
1150 # Combine Y and the new column using cbind
1151 Y <- cbind(Y, new_column)
1152
1153 colnames(Y) <- c("0",
1154                 "1")
1155 ## training data
1156 Xtr<-X
1157 k<-2
1158 n<-dim(Xtr)[1]
1159 ## Normalize data
1160 Xc<-normalize(Xtr)
1161 Xn<-Xc$Xc
1162 p<-dim(Xn)[2]
1163 ## Perform SDA with one non-zero loading for each discriminative
1164 ## direction with Y as matrix input
1165 out <- sda(Xn, Y,
1166           lambda = 1e-6,
1167           stop = -1,
1168           maxIte = 25,
1169           trace = TRUE)
1170 ## predict training samples

```

```

1171 train <- predict(out, Xn)
1172 ## testing
1173 Xtst<-as.matrix(test_data)
1174 Xtst<-normalizetest(Xtst,Xc)
1175
1176 test <- predict(out, Xtst)
1177 print(test$class)
1178
1179
1180 Sparse_lda_correct_rates_2<-sum(testing_labels==test$class)/120
1181
1182
1183 ## Perform 5-fold cross-validation
1184 ##third fold
1185 train_data <- data_folds[[3]]$train_data
1186 test_data <- data_folds[[3]]$test_data
1187
1188
1189 split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
1190 train_data <- Data[split_index, ]
1191 test_data <- Data[-split_index, ]
1192
1193 # Create labels for training and testing data
1194 training_labels <- train_data$Y
1195 testing_labels <- test_data$Y
1196
1197
1198
1199 ##
1200 train_good_data <- train_data[train_data$Y== "1", ]
1201 train_bad_data <- train_data[train_data$Y == "0", ]
1202
1203 train_good_data <- train_good_data%>%
1204   select(-Y)
1205
1206 train_bad_data <- train_bad_data%>%
1207   select(-Y)
1208
1209
1210 test_data <- test_data%>%
1211   select(-Y)
1212
1213 train_data<- train_data%>%
1214   select(-Y)

```

```

1215
1216
1217 #
1218 data<- as.matrix(train_good_data)
1219 n <- nrow(data)
1220 p <- ncol(data)
1221 s <- sqrt(n)
1222
1223
1224
1225
1226
1227
1228
1229 ## Choose number of mixture components
1230 k <- 2
1231
1232 # Use k-means to initialize the parameters
1233 mean_1 <- rep(0,p)
1234 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
1235 kmeans_fit <- kmeans(data, centers = k)
1236 means_init <- kmeans_fit$centers
1237 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
1238 # Find the index of the center that is closer to m1
1239 closest_center_index <- which.min(distances)
1240
1241 # Assign mu_1 to the closer center and mu_2 to the other center
1242 mu_1 <- means_init[closest_center_index, ]
1243 mu_2 <- means_init[-closest_center_index, ]
1244
1245
1246 # Calculate the covariance matrices of the two clusters
1247 cluster_indices <- kmeans_fit$cluster == closest_center_index
1248
1249 # Subset the data points that belong to the cluster associated with mu_1
1250 cluster_data <- data[cluster_indices, ]
1251
1252 # Calculate the covariance matrix for this cluster
1253 cov_matrix_1 <- cov(cluster_data)
1254
1255 #
1256 # Get the indices of data points belonging to the cluster associated with mu_2
1257 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
1258

```

```

1259 # Subset the data points that belong to the cluster associated with mu_2
1260 cluster_data_mu2 <- data[cluster_indices_mu2, ]
1261
1262 # Calculate the covariance matrix for this cluster
1263 cov_matrix_2 <- cov(cluster_data_mu2)
1264
1265 cov<-(cov_matrix_1+cov_matrix_2)/2
1266
1267
1268
1269 # Initial Step
1270 # Update 'beta_optimal' based on 'cov'
1271 Clambda <- 210# Adjust as needed
1272 lambda_0 <- Clambda * sqrt(log(p) / n)
1273 beta <- Variable(p)
1274
1275 # Define the objective function with L1 regularization
1276 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))
1277
1278 # Define the problem
1279 problem <- Problem(objective)
1280
1281 # Solve the problem
1282 result <- solve(problem)
1283
1284 # Get the updated 'beta_optimal'
1285 beta_optimal <- result$getValue(beta)
1286
1287 n <- nrow(data)
1288 a <- numeric(n)
1289 w <- nrow(cluster_data_mu2) / nrow(train_good_data)
1290 for (i in 1:n) {
1291   a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
1292 }
1293
1294
1295
1296 # Update 'w'
1297 w <- sum(a) / n
1298
1299 # M-step
1300 # Update 'mu' and 'cov' based on 'a'
1301 # M-step

```

```

1302 # Update 'mu' and 'cov' based on 'a'
1303 mu <- matrix(0, nrow = 2, ncol = p)
1304 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %%% as.matrix(data))
1305 mu[2, ] <- (sum(a))(-1) * (a %%% as.matrix(data))
1306 mu_1 <- mu[1, ]
1307 mu_2 <- mu[2, ]
1308
1309 cov <- matrix(0, nrow = p, ncol = p)
1310 for (i in 1:n) {
1311   cov <- cov + 1/n * (
1312     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %%% t(as.numeric(data[i, ]) - mu_1) +
1313     a[i] * (as.numeric(data[i, ]) - mu_2) %%% t(as.numeric(data[i, ]) - mu_2)
1314   )
1315 }
1316
1317
1318
1319
1320
1321 # Perform the iterative algorithm
1322 convergence_threshold <- 10
1323 max_iterations <- Inf
1324 iteration <- 1
1325
1326 # Create a vector to store mu for this iteration
1327 while (iteration <= max_iterations) {
1328   prev_mu_1 <- mu_1
1329   # E-step
1330   n <- nrow(data)
1331   a <- numeric(n)
1332   for (i in 1:n) {
1333     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %%% as.numeric(data[i, ]) - (mu_1 + mu_2)
1334       / 2)))
1335   }
1336
1337   # Update 'w'
1338   w <- sum(a) / n
1339
1340   # M-step
1341   # Update 'mu' and 'cov' based on 'a'
1342
1343   # M-step
1344   # Update 'mu' and 'cov' based on 'a'

```

```

1345 mu <- matrix(0, nrow = 2, ncol = p)
1346 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %>% as.matrix(data))
1347 mu[2, ] <- (sum(a))(-1) * (a %>% as.matrix(data))
1348 mu_1 <- mu[1, ]
1349 mu_2 <- mu[2, ]
1350
1351 cov <- matrix(0, nrow = p, ncol = p)
1352 for (i in 1:n) {
1353   cov <- cov + 1/n * (
1354     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %>% t(as.numeric(data[i, ]) - mu_1) +
1355     a[i] * (as.numeric(data[i, ]) - mu_2) %>% t(as.numeric(data[i, ]) - mu_2)
1356   )
1357 }
1358
1359 # Update 'beta_optimal' based on 'cov'
1360 Clambda <- 210 # Adjust as needed
1361 k <- 0.1
1362 lambda_0 <- k * lambda_0 + Clambda * sqrt(log(p) / n)
1363 beta <- Variable(p)
1364
1365 # Define the objective function with L1 regularization
1366 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %>% (mu_1 - mu_2) + lambda_
1367   0 * sum(p_norm(beta, 1))))
1368
1369 # Define the problem
1370 problem <- Problem(objective)
1371
1372 # Solve the problem
1373 result <- solve(problem)
1374
1375 # Get the updated 'beta_optimal'
1376 beta_optimal <- result$getValue(beta)
1377
1378 # Check for convergence
1379 change_in_mu_1 <- abs(mu_1 - prev_mu_1)
1380 if (sum(change_in_mu_1) < convergence_threshold) {
1381   break
1382 }
1383
1384 # Increment the iteration counter
1385 iteration <- iteration + 1
1386 }
1387

```

```

1388
1389
1390 #
1391 pdf<-numeric(120)
1392 # Calculate density estimates and PDF values for each data point
1393 for (i in 1:120) {
1394   data_point <- test_data[i,]
1395   density_estimations_cluster_1 <- dmnorm(data_point, mean = mu_1, sigma = cov)
1396   density_estimations_cluster_2 <- dmnorm(data_point, mean = mu_2, sigma=cov)
1397
1398   pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
1399 }
1400
1401 pdf
1402
1403
1404
1405 #
1406 data<- train_bad_data
1407 n <- nrow(data)
1408 p <- ncol(data)
1409 s <- sqrt(n)
1410
1411
1412
1413
1414
1415
1416
1417 ## Choose number of mixture components
1418 k <- 2
1419
1420 # Use k-means to initialize the parameters
1421 mean_1 <- rep(0,p)
1422 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
1423 kmeans_fit <- kmeans(data, centers = k)
1424 means_init <- kmeans_fit$centers
1425 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
1426 # Find the index of the center that is closer to m1
1427 closest_center_index <- which.min(distances)
1428
1429 # Assign mu_1 to the closer center and mu_2 to the other center
1430 mu_1 <- means_init[closest_center_index, ]
1431 mu_2 <- means_init[-closest_center_index, ]

```



```

1432
1433
1434 # Calculate the covariance matrices of the two clusters
1435 cluster_indices <- kmeans_fit$cluster == closest_center_index
1436
1437 # Subset the data points that belong to the cluster associated with mu_1
1438 cluster_data <- data[cluster_indices, ]
1439
1440 # Calculate the covariance matrix for this cluster
1441 cov_matrix_1 <- cov(cluster_data)
1442
1443 #
1444 # Get the indices of data points belonging to the cluster associated with mu_2
1445 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
1446
1447 # Subset the data points that belong to the cluster associated with mu_2
1448 cluster_data_mu2 <- data[cluster_indices_mu2, ]
1449
1450 # Calculate the covariance matrix for this cluster
1451 cov_matrix_2 <- cov(cluster_data_mu2)
1452
1453 cov <- (cov_matrix_1 + cov_matrix_2) / 2
1454
1455
1456
1457 # Initial Step
1458 # Update 'beta_optimal' based on 'cov'
1459 Clambda <- 7# Adjust as needed
1460 lambda_0 <- Clambda * sqrt(log(p) / n)
1461 beta <- Variable(p)
1462
1463 # Define the objective function with L1 regularization
1464 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
1465   * sum(p_norm(beta, 1))))
1466
1467 # Define the problem
1468 problem <- Problem(objective)
1469
1470 # Solve the problem
1471 result <- solve(problem)
1472
1473 # Get the updated 'beta_optimal'
1474 beta_optimal <- result$getValue(beta)
1475 # E-step

```

```

1475 n <- nrow(data)
1476 a <- numeric(n)
1477 w <- nrow(cluster_data_mu2)/nrow(train_good_data)
1478 for (i in 1:n) {
1479   a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
1480     2))))
1481 }
1482
1483 # Update 'w'
1484 w <- sum(a) / n
1485
1486 # M-step
1487 # Update 'mu' and 'cov' based on 'a'
1488 # M-step
1489 # Update 'mu' and 'cov' based on 'a'
1490 mu <- matrix(0, nrow = 2, ncol = p)
1491 mu[1, ] <- (n - sum(a))^( -1) * ((1 - a) %*% as.matrix(data))
1492 mu[2, ] <- (sum(a))^( -1) * (a %*% as.matrix(data))
1493 mu_1 <- mu[1, ]
1494 mu_2 <- mu[2, ]
1495
1496 cov <- matrix(0, nrow = p, ncol = p)
1497 for (i in 1:n) {
1498   cov <- cov + 1/n * (
1499     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
1500     a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
1501   )
1502 }
1503
1504
1505
1506
1507
1508 # Perform the iterative algorithm
1509 convergence_threshold <- 1
1510 max_iterations <- Inf
1511 iteration <- 1
1512
1513 # Create a vector to store mu for this iteration
1514 while (iteration <= max_iterations) {
1515   prev_mu_1 <- mu_1
1516   # E-step
1517   n <- nrow(data)

```

```

1518 a <- numeric(n)
1519 for (i in 1:n) {
1520   a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ] - (mu_1 + mu_2)
1521     / 2))))
1522 }
1523 # Update 'w'
1524 w <- sum(a) / n
1525
1526 # M-step
1527 # Update 'mu' and 'cov' based on 'a'
1528
1529
1530 # M-step
1531 # Update 'mu' and 'cov' based on 'a'
1532 mu <- matrix(0, nrow = 2, ncol = p)
1533 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %*% as.matrix(data))
1534 mu[2, ] <- (sum(a))(-1) * (a %*% as.matrix(data))
1535 mu_1 <- mu[1, ]
1536 mu_2 <- mu[2, ]
1537
1538 cov <- matrix(0, nrow = p, ncol = p)
1539 for (i in 1:n) {
1540   cov <- cov + 1/n * (
1541     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
1542     a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
1543   )
1544 }
1545
1546 # Update 'beta_optimal' based on 'cov'
1547 Clambda <- 7# Adjust as needed
1548 k <- 0.1
1549 lambda_0 <- k * lambda_0 + Clambda * sqrt(log(p) / n)
1550 beta <- Variable(p)
1551
1552 # Define the objective function with L1 regularization
1553 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
1554   0 * sum(p_norm(beta, 1))))
1555
1556 # Define the problem
1557 problem <- Problem(objective)
1558
1559 # Solve the problem
1559 result <- solve(problem)

```

```

1560
1561     # Get the updated 'beta_optimal'
1562     beta_optimal <- result$getValue(beta)
1563
1564
1565     # Check for convergence
1566     change_in_mu_1<- abs(mu_1 - prev_mu_1)
1567     if (sum(change_in_mu_1)<convergence_threshold) {
1568         break
1569     }
1570
1571     # Increment the iteration counter
1572     iteration <- iteration + 1
1573 }
1574
1575
1576
1577 #
1578 pdf1<-numeric(120)
1579 # Calculate density estimates and PDF values for each data point
1580 for (i in 1:120) {
1581     data_point <- test_data[i,]
1582     density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
1583     density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)
1584
1585     pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
1586 }
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596 # Create vectors to store probabilities
1597 probabilities <- numeric(nrow(test_data))
1598
1599 # Iterate through each data point
1600 for (i in 1:nrow(test_data)) {
1601     pdf_value <- pdf[i] # Density from the first GMM
1602     pdf1_value <- pdf1[i] # Density from the second GMM
1603     y <- sum(testing_labels == 1)

```

```

1604     z<- sum(testing_labels == 0)
1605
1606     # Calculate probability using the formula
1607     probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)
1608
1609     # Store the probability in the vector
1610     probabilities[i] <- probability
1611 }
1612
1613 threshold<-0.5
1614 # Classify instances based on the threshold
1615 classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
1616 classification<- na.omit(classification)
1617 testing_labels
1618 classification
1619
1620 Chime_correct_rates_3<-sum(testing_labels==classification)/120
1621
1622
1623
1624 ## Load the glmnet package if not already loaded
1625 if (!require(glmnet)) {
1626     install.packages("glmnet")
1627     library(glmnet)
1628 }
1629
1630 # Define the response variable (Y) and predictor variables (X) for training data
1631 training_response <- as.numeric(training_labels)
1632 training_predictors <- as.matrix(train_data) # Exclude the first column (Y)
1633
1634 # Fit a generalized linear model with elastic net regularization
1635 alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
1636 lambda <- 0.5 # Adjust the regularization strength (tune this parameter)
1637
1638 # Create the glmnet model
1639 model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)
1640
1641 # Print a summary of the model
1642 print(model)
1643
1644 # Plot the coefficients along the regularization path
1645 plot(model)
1646
1647 # To make predictions on test data, you can use the predict() function

```

```

1648 # Define the predictor variables for test data
1649 testing_predictors <- as.matrix(test_data) # Exclude the first column (Y)
1650
1651 # Make predictions on the test data
1652 predictions <- predict(model, newx = testing_predictors, s = lambda)
1653
1654 threshold<-0.5
1655 # Classify instances based on the threshold
1656 classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
1657 classification1<- na.omit(classification1)
1658 testing_labels
1659 classification1
1660
1661 glmnet_correct_rates_3<-sum(testing_labels==classification1)/120
1662
1663
1664
1665
1666
1667 #
1668 # Load data
1669 X <- as.matrix(train_data)
1670 Y <- as.matrix(training_labels)
1671
1672
1673 new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1) # Creating a new column
    with zeros
1674 colnames(new_column) <- "NewColumn" # Setting the column name (change "NewColumn" to your
    desired name)
1675
1676 # Combine Y and the new column using cbind
1677 Y <- cbind(Y, new_column)
1678
1679 colnames(Y) <- c("0",
    "1")
1680
1681 ## training data
1682 Xtr<-X
1683 k<-2
1684 n<-dim(Xtr)[1]
1685 ## Normalize data
1686 Xc<-normalize(Xtr)
1687 Xn<-Xc$Xc
1688 p<-dim(Xn)[2]
1689 ## Perform SDA with one non-zero loading for each discriminative

```

```

1690  ## direction with Y as matrix input
1691  out <- sda(Xn, Y,
1692            lambda = 1e-6,
1693            stop = -1,
1694            maxIte = 25,
1695            trace = TRUE)
1696  ## predict training samples
1697  train <- predict(out, Xn)
1698  ## testing
1699  Xtst<-as.matrix(test_data)
1700  Xtst<-normalizetest(Xtst,Xc)
1701
1702  test <- predict(out, Xtst)
1703  print(test$class)
1704
1705
1706  Sparse_lda_correct_rates_3<-sum(testing_labels==test$class)/120
1707
1708
1709  ## Perform 5-fold cross-validation
1710
1711  train_data <- data_folds[[4]]$train_data
1712  test_data <- data_folds[[4]]$test_data
1713
1714
1715  split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
1716  train_data <- Data[split_index, ]
1717  test_data <- Data[-split_index, ]
1718
1719  # Create labels for training and testing data
1720  training_labels <- train_data$Y
1721  testing_labels <- test_data$Y
1722
1723
1724
1725  ##
1726  train_good_data <- train_data[train_data$Y== "1", ]
1727  train_bad_data <- train_data[train_data$Y == "0", ]
1728
1729  train_good_data <- train_good_data%>%
1730    select(-Y)
1731
1732  train_bad_data <- train_bad_data%>%
1733    select(-Y)

```

```
1734
1735
1736 test_data <- test_data%>%
1737   select(-Y)
1738
1739 train_data<- train_data%>%
1740   select(-Y)
1741
1742
1743 #
1744 data<- as.matrix(train_good_data)
1745 n <- nrow(data)
1746 p <- ncol(data)
1747 s <- sqrt(n)
1748
1749
1750
1751
1752
1753
1754
1755 ## Choose number of mixture components
1756 k <- 2
1757
1758 # Use k-means to initialize the parameters
1759 mean_1 <- rep(0,p)
1760 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
1761 kmeans_fit <- kmeans(data, centers = k)
1762 means_init <- kmeans_fit$centers
1763 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
1764 # Find the index of the center that is closer to m1
1765 closest_center_index <- which.min(distances)
1766
1767 # Assign mu_1 to the closer center and mu_2 to the other center
1768 mu_1 <- means_init[closest_center_index, ]
1769 mu_2 <- means_init[-closest_center_index, ]
1770
1771
1772 # Calculate the covariance matrices of the two clusters
1773 cluster_indices <- kmeans_fit$cluster == closest_center_index
1774
1775 # Subset the data points that belong to the cluster associated with mu_1
1776 cluster_data <- data[cluster_indices, ]
1777
```



```

1778 # Calculate the covariance matrix for this cluster
1779 cov_matrix_1 <- cov(cluster_data)
1780
1781 #
1782 # Get the indices of data points belonging to the cluster associated with mu_2
1783 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
1784
1785 # Subset the data points that belong to the cluster associated with mu_2
1786 cluster_data_mu2 <- data[cluster_indices_mu2, ]
1787
1788 # Calculate the covariance matrix for this cluster
1789 cov_matrix_2 <- cov(cluster_data_mu2)
1790
1791 cov<-(cov_matrix_1+cov_matrix_2)/2
1792
1793
1794
1795 # Initial Step
1796 # Update 'beta_optimal' based on 'cov'
1797 Clambda <- 210# Adjust as needed
1798 lambda_0 <- Clambda * sqrt(log(p) / n)
1799 beta <- Variable(p)
1800
1801 # Define the objective function with L1 regularization
1802 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
1803   * sum(p_norm(beta, 1))))
1804
1805 # Define the problem
1806 problem <- Problem(objective)
1807
1808 # Solve the problem
1809 result <- solve(problem)
1810
1811 # Get the updated 'beta_optimal'
1812 beta_optimal <- result$getValue(beta)
1813
1814 n <- nrow(data)
1815 a <- numeric(n)
1816 w <- nrow(cluster_data_mu2) / nrow(train_good_data)
1817 for (i in 1:n) {
1818   a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
1819 }
1820

```

```

1821
1822 # Update 'w'
1823 w <- sum(a) / n
1824
1825 # M-step
1826 # Update 'mu' and 'cov' based on 'a'
1827 # M-step
1828 # Update 'mu' and 'cov' based on 'a'
1829 mu <- matrix(0, nrow = 2, ncol = p)
1830 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %%% as.matrix(data))
1831 mu[2, ] <- (sum(a))(-1) * (a %%% as.matrix(data))
1832 mu_1 <- mu[1, ]
1833 mu_2 <- mu[2, ]
1834
1835 cov <- matrix(0, nrow = p, ncol = p)
1836 for (i in 1:n) {
1837   cov <- cov + 1/n * (
1838     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %%% t(as.numeric(data[i, ]) - mu_1) +
1839     a[i] * (as.numeric(data[i, ]) - mu_2) %%% t(as.numeric(data[i, ]) - mu_2)
1840   )
1841 }
1842
1843
1844
1845
1846
1847 # Perform the iterative algorithm
1848 convergence_threshold <- 10
1849 max_iterations <- Inf
1850 iteration <- 1
1851
1852 # Create a vector to store mu for this iteration
1853 while (iteration <= max_iterations) {
1854   prev_mu_1 <- mu_1
1855   # E-step
1856   n <- nrow(data)
1857   a <- numeric(n)
1858   for (i in 1:n) {
1859     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %%% as.numeric(data[i, ] - (mu_1 + mu_2)
1860       / 2)))
1861   }
1862
1863   # Update 'w'
1864   w <- sum(a) / n

```

```

1864
1865 # M-step
1866 # Update 'mu' and 'cov' based on 'a'
1867
1868
1869 # M-step
1870 # Update 'mu' and 'cov' based on 'a'
1871 mu <- matrix(0, nrow = 2, ncol = p)
1872 mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %%% as.matrix(data))
1873 mu[2, ] <- (sum(a))(-1) * (a %%% as.matrix(data))
1874 mu_1 <- mu[1, ]
1875 mu_2 <- mu[2, ]
1876
1877 cov <- matrix(0, nrow = p, ncol = p)
1878 for (i in 1:n) {
1879   cov <- cov + 1/n * (
1880     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %%% t(as.numeric(data[i, ]) - mu_1) +
1881     a[i] * (as.numeric(data[i, ]) - mu_2) %%% t(as.numeric(data[i, ]) - mu_2)
1882   )
1883 }
1884
1885 # Update 'beta_optimal' based on 'cov'
1886 Clambda <- 210 # Adjust as needed
1887 k <- 0.1
1888 lambda_0 <- k * lambda_0 + Clambda * sqrt(log(p) / n)
1889 beta <- Variable(p)
1890
1891 # Define the objective function with L1 regularization
1892 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %%% (mu_1 - mu_2) + lambda_
1893   0 * sum(p_norm(beta, 1))))
1894
1895 # Define the problem
1896 problem <- Problem(objective)
1897
1898 # Solve the problem
1899 result <- solve(problem)
1900
1901 # Get the updated 'beta_optimal'
1902 beta_optimal <- result$getValue(beta)
1903
1904 # Check for convergence
1905 change_in_mu_1 <- abs(mu_1 - prev_mu_1)
1906 if (sum(change_in_mu_1) < convergence_threshold) {

```

```

1907     break
1908   }
1909
1910   # Increment the iteration counter
1911   iteration <- iteration + 1
1912 }
1913
1914
1915
1916 #
1917 pdf<-numeric(120)
1918 # Calculate density estimates and PDF values for each data point
1919 for (i in 1:120) {
1920   data_point <- test_data[i,]
1921   density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
1922   density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)
1923
1924   pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
1925 }
1926
1927 pdf
1928
1929
1930
1931 #
1932 data<- train_bad_data
1933 n <- nrow(data)
1934 p <- ncol(data)
1935 s <- sqrt(n)
1936
1937
1938
1939
1940
1941
1942
1943 ## Choose number of mixture components
1944 k <- 2
1945
1946 # Use k-means to initialize the parameters
1947 mean_1 <- rep(0,p)
1948 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
1949 kmeans_fit <- kmeans(data, centers = k)
1950 means_init <- kmeans_fit$centers

```

```

1951 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
1952 # Find the index of the center that is closer to m1
1953 closest_center_index <- which.min(distances)
1954
1955 # Assign mu_1 to the closer center and mu_2 to the other center
1956 mu_1 <- means_init[closest_center_index, ]
1957 mu_2 <- means_init[-closest_center_index, ]
1958
1959
1960 # Calculate the covariance matrices of the two clusters
1961 cluster_indices <- kmeans_fit$cluster == closest_center_index
1962
1963 # Subset the data points that belong to the cluster associated with mu_1
1964 cluster_data <- data[cluster_indices, ]
1965
1966 # Calculate the covariance matrix for this cluster
1967 cov_matrix_1 <- cov(cluster_data)
1968
1969 #
1970 # Get the indices of data points belonging to the cluster associated with mu_2
1971 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
1972
1973 # Subset the data points that belong to the cluster associated with mu_2
1974 cluster_data_mu2 <- data[cluster_indices_mu2, ]
1975
1976 # Calculate the covariance matrix for this cluster
1977 cov_matrix_2 <- cov(cluster_data_mu2)
1978
1979 cov <- (cov_matrix_1 + cov_matrix_2) / 2
1980
1981
1982
1983 # Initial Step
1984 # Update 'beta_optimal' based on 'cov'
1985 Clambda <- 7 # Adjust as needed
1986 lambda_0 <- Clambda * sqrt(log(p) / n)
1987 beta <- Variable(p)
1988
1989 # Define the objective function with L1 regularization
1990 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))
1991
1992 # Define the problem
1993 problem <- Problem(objective)

```

```

1994
1995 # Solve the problem
1996 result <- solve(problem)
1997
1998 # Get the updated 'beta_optimal'
1999 beta_optimal <- result$getValue(beta)
2000 # E-step
2001 n <- nrow(data)
2002 a <- numeric(n)
2003 w <- nrow(cluster_data_mu2)/nrow(train_good_data)
2004 for (i in 1:n) {
2005   a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
2006     2))))
2007 }
2008
2009 # Update 'w'
2010 w <- sum(a) / n
2011
2012 # M-step
2013 # Update 'mu' and 'cov' based on 'a'
2014 # M-step
2015 # Update 'mu' and 'cov' based on 'a'
2016 mu <- matrix(0, nrow = 2, ncol = p)
2017 mu[1, ] <- (n - sum(a))^-1 * ((1 - a) %*% as.matrix(data))
2018 mu[2, ] <- (sum(a))^-1 * (a %*% as.matrix(data))
2019 mu_1 <- mu[1, ]
2020 mu_2 <- mu[2, ]
2021
2022 cov <- matrix(0, nrow = p, ncol = p)
2023 for (i in 1:n) {
2024   cov <- cov + 1/n * (
2025     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
2026     a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
2027   )
2028 }
2029
2030
2031
2032
2033
2034 # Perform the iterative algorithm
2035 convergence_threshold <- 1
2036 max_iterations <- Inf

```

```

2037 iteration <- 1
2038
2039 # Create a vector to store mu for this iteration
2040 while (iteration <= max_iterations) {
2041   prev_mu_1 <- mu_1
2042   # E-step
2043   n <- nrow(data)
2044   a <- numeric(n)
2045   for (i in 1:n) {
2046     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ] - (mu_1 + mu_2)
2047       / 2))))
2048   }
2049   # Update 'w'
2050   w <- sum(a) / n
2051
2052   # M-step
2053   # Update 'mu' and 'cov' based on 'a'
2054
2055   # M-step
2056   # Update 'mu' and 'cov' based on 'a'
2057   mu <- matrix(0, nrow = 2, ncol = p)
2058   mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %*% as.matrix(data))
2059   mu[2, ] <- (sum(a))(-1) * (a %*% as.matrix(data))
2060   mu_1 <- mu[1, ]
2061   mu_2 <- mu[2, ]
2062
2063   cov <- matrix(0, nrow = p, ncol = p)
2064   for (i in 1:n) {
2065     cov <- cov + 1/n * (
2066       (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
2067       a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
2068     )
2069   }
2070
2071   # Update 'beta_optimal' based on 'cov'
2072   Clambda <- 7# Adjust as needed
2073   k <- 0.1
2074   lambda_0 <- k*lambda_0 + Clambda * sqrt(log(p) / n)
2075   beta <- Variable(p)
2076
2077   # Define the objective function with L1 regularization
2078   objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_

```

```

0 * sum(p_norm(beta, 1)))
2080
2081 # Define the problem
2082 problem <- Problem(objective)
2083
2084 # Solve the problem
2085 result <- solve(problem)
2086
2087 # Get the updated 'beta_optimal'
2088 beta_optimal <- result$getValue(beta)
2089
2090
2091 # Check for convergence
2092 change_in_mu_1 <- abs(mu_1 - prev_mu_1)
2093 if (sum(change_in_mu_1) < convergence_threshold) {
2094   break
2095 }
2096
2097 # Increment the iteration counter
2098 iteration <- iteration + 1
2099 }
2100
2101
2102
2103 #
2104 pdf1 <- numeric(120)
2105 # Calculate density estimates and PDF values for each data point
2106 for (i in 1:120) {
2107   data_point <- test_data[i,]
2108   density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
2109   density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)
2110
2111   pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
2112 }
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122 # Create vectors to store probabilities

```



```

2123 probabilities <- numeric(nrow(test_data))
2124
2125 # Iterate through each data point
2126 for (i in 1:nrow(test_data)) {
2127   pdf_value <- pdf[i] # Density from the first GMM
2128   pdf1_value <- pdf1[i] # Density from the second GMM
2129   y <- sum(testing_labels == 1)
2130   z <- sum(testing_labels == 0)
2131
2132   # Calculate probability using the formula
2133   probability <- y * pdf_value / (z * pdf_value + y * pdf1_value)
2134
2135   # Store the probability in the vector
2136   probabilities[i] <- probability
2137 }
2138
2139 threshold <- 0.5
2140 # Classify instances based on the threshold
2141 classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
2142 classification <- na.omit(classification)
2143 testing_labels
2144 classification
2145
2146 Chime_correct_rates_4 <- sum(testing_labels == classification) / 120
2147
2148
2149
2150 ## Load the glmnet package if not already loaded
2151 if (!require(glmnet)) {
2152   install.packages("glmnet")
2153   library(glmnet)
2154 }
2155
2156 # Define the response variable (Y) and predictor variables (X) for training data
2157 training_response <- as.numeric(training_labels)
2158 training_predictors <- as.matrix(train_data) # Exclude the first column (Y)
2159
2160 # Fit a generalized linear model with elastic net regularization
2161 alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
2162 lambda <- 0.5 # Adjust the regularization strength (tune this parameter)
2163
2164 # Create the glmnet model
2165 model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)
2166

```

```

2167 # Print a summary of the model
2168 print(model)
2169
2170 # Plot the coefficients along the regularization path
2171 plot(model)
2172
2173 # To make predictions on test data, you can use the predict() function
2174 # Define the predictor variables for test data
2175 testing_predictors <- as.matrix(test_data) # Exclude the first column (Y)
2176
2177 # Make predictions on the test data
2178 predictions <- predict(model, newx = testing_predictors, s = lambda)
2179
2180 threshold<-0.5
2181 # Classify instances based on the threshold
2182 classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
2183 classification1<- na.omit(classification1)
2184 testing_labels
2185 classification1
2186
2187 glmnet_correct_rates_4<-sum(testing_labels==classification1)/120
2188
2189
2190
2191
2192
2193 #
2194 # Load data
2195 X <- as.matrix(train_data)
2196 Y <- as.matrix(training_labels)
2197
2198
2199 new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1) # Creating a new column
    with zeros
2200 colnames(new_column) <- "NewColumn" # Setting the column name (change "NewColumn" to your
    desired name)
2201
2202 # Combine Y and the new column using cbind
2203 Y <- cbind(Y, new_column)
2204
2205 colnames(Y) <- c("0",
2206                 "1")
2207 ## training data
2208 Xtr<-X

```

```

2209 k<-2
2210 n<-dim(Xtr)[1]
2211 ## Normalize data
2212 Xc<-normalize(Xtr)
2213 Xn<-Xc$Xc
2214 p<-dim(Xn)[2]
2215 ## Perform SDA with one non-zero loading for each discriminative
2216 ## direction with Y as matrix input
2217 out <- sda(Xn, Y,
2218           lambda = 1e-6,
2219           stop = -1,
2220           maxIte = 25,
2221           trace = TRUE)
2222 ## predict training samples
2223 train <- predict(out, Xn)
2224 ## testing
2225 Xtst<-as.matrix(test_data)
2226 Xtst<-normalizetest(Xtst,Xc)
2227
2228 test <- predict(out, Xtst)
2229 print(test$class)
2230
2231
2232 Sparse_lda_correct_rates_4<-sum(testing_labels==test$class)/120
2233
2234
2235 ## Perform 5-fold cross-validation
2236
2237 train_data <- data_folds[[5]]$train_data
2238 test_data <- data_folds[[5]]$test_data
2239
2240
2241 split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
2242 train_data <- Data[split_index, ]
2243 test_data <- Data[-split_index, ]
2244
2245 # Create labels for training and testing data
2246 training_labels <- train_data$Y
2247 testing_labels <- test_data$Y
2248
2249
2250
2251 ##
2252 train_good_data <- train_data[train_data$Y== "1", ]

```

```
2253 train_bad_data <- train_data[train_data$Y == "0", ]
2254
2255 train_good_data <- train_good_data%>%
2256   select(-Y)
2257
2258 train_bad_data <- train_bad_data%>%
2259   select(-Y)
2260
2261
2262 test_data <- test_data%>%
2263   select(-Y)
2264
2265 train_data<- train_data%>%
2266   select(-Y)
2267
2268
2269 #
2270 data<- as.matrix(train_good_data)
2271 n <- nrow(data)
2272 p <- ncol(data)
2273 s <- sqrt(n)
2274
2275
2276
2277
2278
2279
2280
2281 ## Choose number of mixture components
2282 k <- 2
2283
2284 # Use k-means to initialize the parameters
2285 mean_1 <- rep(0,p)
2286 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
2287 kmeans_fit <- kmeans(data, centers = k)
2288 means_init <- kmeans_fit$centers
2289 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
2290 # Find the index of the center that is closer to m1
2291 closest_center_index <- which.min(distances)
2292
2293 # Assign mu_1 to the closer center and mu_2 to the other center
2294 mu_1 <- means_init[closest_center_index, ]
2295 mu_2 <- means_init[-closest_center_index, ]
2296
```

```

2297
2298 # Calculate the covariance matrices of the two clusters
2299 cluster_indices <- kmeans_fit$cluster == closest_center_index
2300
2301 # Subset the data points that belong to the cluster associated with mu_1
2302 cluster_data <- data[cluster_indices, ]
2303
2304 # Calculate the covariance matrix for this cluster
2305 cov_matrix_1 <- cov(cluster_data)
2306
2307 #
2308 # Get the indices of data points belonging to the cluster associated with mu_2
2309 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
2310
2311 # Subset the data points that belong to the cluster associated with mu_2
2312 cluster_data_mu2 <- data[cluster_indices_mu2, ]
2313
2314 # Calculate the covariance matrix for this cluster
2315 cov_matrix_2 <- cov(cluster_data_mu2)
2316
2317 cov<-(cov_matrix_1+cov_matrix_2)/2
2318
2319
2320
2321 # Initial Step
2322 # Update 'beta_optimal' based on 'cov'
2323 Clambda <- 210# Adjust as needed
2324 lambda_0 <- Clambda * sqrt(log(p) / n)
2325 beta <- Variable(p)
2326
2327 # Define the objective function with L1 regularization
2328 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
2329   * sum(p_norm(beta, 1))))
2330
2331 # Define the problem
2332 problem <- Problem(objective)
2333
2334 # Solve the problem
2335 result <- solve(problem)
2336
2337 # Get the updated 'beta_optimal'
2338 beta_optimal <- result$getValue(beta)
2339
2340 n <- nrow(data)

```

```

2340 a <- numeric(n)
2341 w <- nrow(cluster_data_mu2) / nrow(train_good_data)
2342 for (i in 1:n) {
2343   a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
2344 }
2345
2346
2347
2348 # Update 'w'
2349 w <- sum(a) / n
2350
2351 # M-step
2352 # Update 'mu' and 'cov' based on 'a'
2353 # M-step
2354 # Update 'mu' and 'cov' based on 'a'
2355 mu <- matrix(0, nrow = 2, ncol = p)
2356 mu[1, ] <- (n - sum(a))^-1 * ((1 - a) %*% as.matrix(data))
2357 mu[2, ] <- (sum(a))^-1 * (a %*% as.matrix(data))
2358 mu_1 <- mu[1, ]
2359 mu_2 <- mu[2, ]
2360
2361 cov <- matrix(0, nrow = p, ncol = p)
2362 for (i in 1:n) {
2363   cov <- cov + 1/n * (
2364     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
2365     a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
2366   )
2367 }
2368
2369
2370
2371
2372
2373 # Perform the iterative algorithm
2374 convergence_threshold <- 10
2375 max_iterations <- Inf
2376 iteration <- 1
2377
2378 # Create a vector to store mu for this iteration
2379 while (iteration <= max_iterations) {
2380   prev_mu_1 <- mu_1
2381   # E-step
2382   n <- nrow(data)
2383   a <- numeric(n)

```

```

2384   for (i in 1:n) {
2385     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ] - (mu_1 + mu_2)
2386       / 2)))
2387   }
2388
2389   # Update 'w'
2390   w <- sum(a) / n
2391
2392   # M-step
2393   # Update 'mu' and 'cov' based on 'a'
2394
2395   # M-step
2396   # Update 'mu' and 'cov' based on 'a'
2397   mu <- matrix(0, nrow = 2, ncol = p)
2398   mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %*% as.matrix(data))
2399   mu[2, ] <- (sum(a))(-1) * (a %*% as.matrix(data))
2400   mu_1 <- mu[1, ]
2401   mu_2 <- mu[2, ]
2402
2403   cov <- matrix(0, nrow = p, ncol = p)
2404   for (i in 1:n) {
2405     cov <- cov + 1/n * (
2406       (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
2407       a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
2408     )
2409   }
2410
2411   # Update 'beta_optimal' based on 'cov'
2412   Clambda <- 210 # Adjust as needed
2413   k <- 0.1
2414   lambda_0 <- k * lambda_0 + Clambda * sqrt(log(p) / n)
2415   beta <- Variable(p)
2416
2417   # Define the objective function with L1 regularization
2418   objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
2419     0 * sum(p_norm(beta, 1))))
2420
2421   # Define the problem
2422   problem <- Problem(objective)
2423
2424   # Solve the problem
2425   result <- solve(problem)

```

```

2426   # Get the updated 'beta_optimal'
2427   beta_optimal <- result$getValue(beta)
2428
2429
2430   # Check for convergence
2431   change_in_mu_1<- abs(mu_1 - prev_mu_1)
2432   if (sum(change_in_mu_1)<convergence_threshold) {
2433     break
2434   }
2435
2436   # Increment the iteration counter
2437   iteration <- iteration + 1
2438 }
2439
2440
2441
2442 #
2443 pdf<-numeric(120)
2444 # Calculate density estimates and PDF values for each data point
2445 for (i in 1:120) {
2446   data_point <- test_data[i,]
2447   density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
2448   density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)
2449
2450   pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
2451 }
2452
2453 pdf
2454
2455
2456
2457 #
2458 data<- train_bad_data
2459 n <- nrow(data)
2460 p <- ncol(data)
2461 s <- sqrt(n)
2462
2463
2464
2465
2466
2467
2468
2469 ## Choose number of mixture components

```



```

2470 k <- 2
2471
2472 # Use k-means to initialize the parameters
2473 mean_1 <- rep(0,p)
2474 mean_1[1:10] <- seq(0.1, 1, by = 0.1)
2475 kmeans_fit <- kmeans(data, centers = k)
2476 means_init <- kmeans_fit$centers
2477 distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
2478 # Find the index of the center that is closer to m1
2479 closest_center_index <- which.min(distances)
2480
2481 # Assign mu_1 to the closer center and mu_2 to the other center
2482 mu_1 <- means_init[closest_center_index, ]
2483 mu_2 <- means_init[-closest_center_index, ]
2484
2485
2486 # Calculate the covariance matrices of the two clusters
2487 cluster_indices <- kmeans_fit$cluster == closest_center_index
2488
2489 # Subset the data points that belong to the cluster associated with mu_1
2490 cluster_data <- data[cluster_indices, ]
2491
2492 # Calculate the covariance matrix for this cluster
2493 cov_matrix_1 <- cov(cluster_data)
2494
2495 #
2496 # Get the indices of data points belonging to the cluster associated with mu_2
2497 cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
2498
2499 # Subset the data points that belong to the cluster associated with mu_2
2500 cluster_data_mu2 <- data[cluster_indices_mu2, ]
2501
2502 # Calculate the covariance matrix for this cluster
2503 cov_matrix_2 <- cov(cluster_data_mu2)
2504
2505 cov<-(cov_matrix_1+cov_matrix_2)/2
2506
2507
2508
2509 # Initial Step
2510 # Update 'beta_optimal' based on 'cov'
2511 Clambda <- 7# Adjust as needed
2512 lambda_0 <- Clambda * sqrt(log(p) / n)
2513 beta <- Variable(p)

```

```

2514
2515 # Define the objective function with L1 regularization
2516 objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %%% (mu_1 - mu_2) + lambda_0
      * sum(p_norm(beta, 1))))
2517
2518 # Define the problem
2519 problem <- Problem(objective)
2520
2521 # Solve the problem
2522 result <- solve(problem)
2523
2524 # Get the updated 'beta_optimal'
2525 beta_optimal <- result$getValue(beta)
2526 # E-step
2527 n <- nrow(data)
2528 a <- numeric(n)
2529 w <- nrow(cluster_data_mu2)/nrow(train_good_data)
2530 for (i in 1:n) {
2531   a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %%% as.numeric((data[i, ] - (mu_1 + mu_2) /
      2))))
2532 }
2533
2534
2535 # Update 'w'
2536 w <- sum(a) / n
2537
2538 # M-step
2539 # Update 'mu' and 'cov' based on 'a'
2540 # M-step
2541 # Update 'mu' and 'cov' based on 'a'
2542 mu <- matrix(0, nrow = 2, ncol = p)
2543 mu[1, ] <- (n - sum(a))^( -1) * ((1 - a) %%% as.matrix(data))
2544 mu[2, ] <- (sum(a))^( -1) * (a %%% as.matrix(data))
2545 mu_1 <- mu[1, ]
2546 mu_2 <- mu[2, ]
2547
2548 cov <- matrix(0, nrow = p, ncol = p)
2549 for (i in 1:n) {
2550   cov <- cov + 1/n * (
2551     (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %%% t(as.numeric(data[i, ]) - mu_1) +
2552     a[i] * (as.numeric(data[i, ]) - mu_2) %%% t(as.numeric(data[i, ]) - mu_2)
2553   )
2554 }
2555

```

```

2556
2557
2558
2559
2560 # Perform the iterative algorithm
2561 convergence_threshold <- 1
2562 max_iterations <- Inf
2563 iteration <- 1
2564
2565 # Create a vector to store mu for this iteration
2566 while (iteration <= max_iterations) {
2567   prev_mu_1 <- mu_1
2568   # E-step
2569   n <- nrow(data)
2570   a <- numeric(n)
2571   for (i in 1:n) {
2572     a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ]) - (mu_1 + mu_2)
2573       / 2))))
2574   }
2575
2576   # Update 'w'
2577   w <- sum(a) / n
2578
2579   # M-step
2580   # Update 'mu' and 'cov' based on 'a'
2581
2582   # M-step
2583   # Update 'mu' and 'cov' based on 'a'
2584   mu <- matrix(0, nrow = 2, ncol = p)
2585   mu[1, ] <- (n - sum(a))(-1) * ((1 - a) %*% as.matrix(data))
2586   mu[2, ] <- (sum(a))(-1) * (a %*% as.matrix(data))
2587   mu_1 <- mu[1, ]
2588   mu_2 <- mu[2, ]
2589
2590   cov <- matrix(0, nrow = p, ncol = p)
2591   for (i in 1:n) {
2592     cov <- cov + 1/n * (
2593       (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
2594       a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
2595     )
2596   }
2597
2598   # Update 'beta_optimal' based on 'cov'

```

```

2599   Clambda <- 7# Adjust as needed
2600   k<-0.1
2601   lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
2602   beta <- Variable(p)
2603
2604   # Define the objective function with L1 regularization
2605   objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
      0 * sum(p_norm(beta, 1))))
2606
2607   # Define the problem
2608   problem <- Problem(objective)
2609
2610   # Solve the problem
2611   result <- solve(problem)
2612
2613   # Get the updated 'beta_optimal'
2614   beta_optimal <- result$getValue(beta)
2615
2616
2617   # Check for convergence
2618   change_in_mu_1<- abs(mu_1 - prev_mu_1)
2619   if (sum(change_in_mu_1)<convergence_threshold) {
2620     break
2621   }
2622
2623   # Increment the iteration counter
2624   iteration <- iteration + 1
2625 }
2626
2627
2628
2629 #
2630 pdf1<-numeric(120)
2631 # Calculate density estimates and PDF values for each data point
2632 for (i in 1:120) {
2633   data_point <- test_data[i,]
2634   density_estimations_cluster_1 <- dmnorm(data_point, mean = mu_1, sigma = cov)
2635   density_estimations_cluster_2 <- dmnorm(data_point, mean = mu_2, sigma=cov)
2636
2637   pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
2638 }
2639
2640
2641

```

```

2642
2643
2644
2645
2646
2647
2648 # Create vectors to store probabilities
2649 probabilities <- numeric(nrow(test_data))
2650
2651 # Iterate through each data point
2652 for (i in 1:nrow(test_data)) {
2653   pdf_value <- pdf[i] # Density from the first GMM
2654   pdf1_value <- pdf1[i] # Density from the second GMM
2655   y <- sum(testing_labels == 1)
2656   z <- sum(testing_labels == 0)
2657
2658   # Calculate probability using the formula
2659   probability <- y * pdf_value / (z * pdf_value + y * pdf1_value)
2660
2661   # Store the probability in the vector
2662   probabilities[i] <- probability
2663 }
2664
2665 threshold <- 0.5
2666 # Classify instances based on the threshold
2667 classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
2668 classification <- na.omit(classification)
2669 testing_labels
2670 classification
2671
2672 Chime_correct_rates_5 <- sum(testing_labels == classification) / 120
2673
2674
2675
2676 ## Load the glmnet package if not already loaded
2677 if (!require(glmnet)) {
2678   install.packages("glmnet")
2679   library(glmnet)
2680 }
2681
2682 # Define the response variable (Y) and predictor variables (X) for training data
2683 training_response <- as.numeric(training_labels)
2684 training_predictors <- as.matrix(train_data) # Exclude the first column (Y)
2685

```

```

2686 # Fit a generalized linear model with elastic net regularization
2687 alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
2688 lambda <- 0.5 # Adjust the regularization strength (tune this parameter)
2689
2690 # Create the glmnet model
2691 model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)
2692
2693 # Print a summary of the model
2694 print(model)
2695
2696 # Plot the coefficients along the regularization path
2697 plot(model)
2698
2699 # To make predictions on test data, you can use the predict() function
2700 # Define the predictor variables for test data
2701 testing_predictors <- as.matrix(test_data) # Exclude the first column (Y)
2702
2703 # Make predictions on the test data
2704 predictions <- predict(model, newx = testing_predictors, s = lambda)
2705
2706 threshold<-0.5
2707 # Classify instances based on the threshold
2708 classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
2709 classification1<- na.omit(classification1)
2710 testing_labels
2711 classification1
2712
2713 glmnet_correct_rates_5<-sum(testing_labels==classification1)/120
2714
2715
2716
2717
2718
2719 #
2720 # Load data
2721 X <- as.matrix(train_data)
2722 Y <- as.matrix(training_labels)
2723
2724
2725 new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1) # Creating a new column
    with zeros
2726 colnames(new_column) <- "NewColumn" # Setting the column name (change "NewColumn" to your
    desired name)
2727

```

```

2728 # Combine Y and the new column using cbind
2729 Y <- cbind(Y, new_column)
2730
2731 colnames(Y) <- c("0",
2732                  "1")
2733 ## training data
2734 Xtr<-X
2735 k<-2
2736 n<-dim(Xtr)[1]
2737 ## Normalize data
2738 Xc<-normalize(Xtr)
2739 Xn<-Xc$Xc
2740 p<-dim(Xn)[2]
2741 ## Perform SDA with one non-zero loading for each discriminative
2742 ## direction with Y as matrix input
2743 out <- sda(Xn, Y,
2744            lambda = 1e-6,
2745            stop = -1,
2746            maxIte = 25,
2747            trace = TRUE)
2748 ## predict training samples
2749 train <- predict(out, Xn)
2750 ## testing
2751 Xtst<-as.matrix(test_data)
2752 Xtst<-normalizetest(Xtst,Xc)
2753
2754 test <- predict(out, Xtst)
2755 print(test$class)
2756
2757
2758 Sparse_lda_correct_rates_5<-sum(testing_labels==test$class)/120
2759
2760
2761 Sparse_lda_correct_rates<-(Sparse_lda_correct_rates_1+Sparse_lda_correct_rates_2+Sparse_lda_
  correct_rates_3+Sparse_lda_correct_rates_4+Sparse_lda_correct_rates_5)/5
2762
2763 glmnet_correct_rates<-(glmnet_correct_rates_1+glmnet_correct_rates_2+glmnet_correct_rates_3+
  glmnet_correct_rates_4+glmnet_correct_rates_5)/5
2764
2765
2766 Chime_correct_rates<-(Chime_correct_rates_1+Chime_correct_rates_2+Chime_correct_rates_3+Chime_
  correct_rates_4+Chime_correct_rates_5)/5
2767
2768 # Store the correctness rates for this simulation

```

```
2769 Chime_error_rates[sim] <- 1-Chime_correct_rates # Replace with the correct variable
2770 glmnet_error_rates[sim] <- 1-glmnet_correct_rates # Replace with the correct variable
2771 Sparse_lda_error_rates[sim] <- 1-Sparse_lda_correct_rates#Replace with the correct variable
2772 }
2773
2774 # Calculate summary statistics for correctness rates (e.g., mean, standard deviation) for
      Chime, glm, and lda
2775 chime_summary_stats <- summary(Chime_error_rates)
2776 glm_summary_stats <- summary(glmnet_error_rates)
2777 lda_summary_stats <- summary(Sparse_lda_error_rates)
2778
2779
2780 # Create a data frame to combine the summary statistics
2781 summary_table <- data.frame(
2782   Model = c("Chime", "glmnet", "Sparse_lda"),
2783   Mean = c(mean(Chime_error_rates), mean(glmnet_error_rates), mean(Sparse_lda_error_rates)),
2784   SD = c(sd(Chime_error_rates), sd(glmnet_error_rates), sd(Sparse_lda_error_rates))
2785 )
2786
2787 # Print the summary table
2788 print(summary_table)
2789
2790
2791
2792 Chime_error_rates_model1<-Chime_error_rates
2793 Sparse_lda_error_rates_model1<-Sparse_lda_error_rates
2794 High_dimension_logistic_error_rates_model1<-glmnet_error_rates
```