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

List of Figures

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 $C_1, C_2, \ldots, C_g$. 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 $X = (x_1, \ldots, x_p)$ 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 μ_Y and a covariance matrix Σ, yielding the Bayes rule:

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

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 µ1 and µ2. [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 Σ and the mean difference vector, μ_1-μ_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 β (please refer to Section 4.4.2 for a detailed description of β). 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 $\phi_1, \ldots, \phi_k$ with mixing weights $\pi_1, \ldots, \pi_k$, 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 π_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 π_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** (π_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 θ is the observed data. It quantifies the likelihood of the distribution with the parameter vector x, given the observed data θ.

- **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.).

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

In this equation:

  - $L(X|\pi, \theta)$ represents the likelihood of the entire dataset X given the model parameters π and θ.
  - X is the dataset consisting of N data points, often represented as $X = \{x_1, x_2, \ldots, x_N\}$.
  - $\pi = (\pi_1, \pi_2, \ldots, \pi_K)$ represents the vector of mixing coefficients, where π_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.
  - $\theta = (\theta_1, \theta_2, \ldots, \theta_K)$ represents a set of parameters, where θ_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 π_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 θ.
  - $P(x|\theta)$ is the likelihood of the data x given the parameter θ.
  - $\int P(x, z|\theta)\, dz$ represents the integral over the joint distribution of x and latent variables z given the parameter θ.

These equations are used in statistical estimation to find the parameter θ 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 μ_k and π_k and Σ_k

   - μ_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.
   - Σ_k denotes the covariance matrix associated with the k-th component. Each component is represented as a multivariate normal distribution, and Σ_k describes the covariance structure of that component.
   - π_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.

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

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

Probability Density Function of multivariate normal Distribution

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

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

Where:

- $\phi(\mathbf{x}, \mu, \Sigma)$ is the Probability density function.
- $\mathbf{x}$ is the Vector of random variables.
- μ is the Mean vector.
- Σ 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 θ by maximizing the expected log-likelihood with respect to the latent variable:

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

   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 θ.

the equation for updating parameters are as follows

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

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

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

In these equations:

- μ_k represents the updated mean for the k-th component.
- Σ_k represents the updated covariance matrix for the k-th component.
- π_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 ϵ, stop. Otherwise, return to the E-step (Step 2).

3.4 Discriminant function based on mixture model

3.4.1 Function

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

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

- $p(x|y=1)$ 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:

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

Where:

- $G_{\theta^*}(z)$ represents the classification result for the observation z.
- θ^* denotes a set of known parameters, including class means μ_1^* and μ_2^*, discriminant direction β^*, and class proportions π^* and $1 - \pi^*$.
- z is an unlabeled observation that we aim to classify.
- μ_1^* and μ_2^* are the class means for class 1 and class 2, respectively.

- β^* represents the discriminant direction, which can be calculated as $\beta^* = \Omega^*\delta^*$. Here, Ω^* is the inverse of the pooled sample covariance matrix, denoted as $\Omega^* = (\Sigma^*)^{-1}$, and $\delta^* = \mu_1^* - \mu_2^*$.

- π^* 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 β^* 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 θ^*. However, it's important to note that when dealing with high-dimensional data, the estimation of Ω^* (the inverse of the covariance matrix) can become a challenging task. This difficulty in estimating Ω^* 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 μ_k and Σ_k with the following equation:

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

In the case of high-dimensional data, the intricate nature of the symbol Σ_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 β^*. 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 β^*

β^* 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 β

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 β^* is sparse, The optimization problem to find $\hat{\beta}$ is given by:

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

In this equation:

- $\hat{\beta}$ represents the estimated variable or vector β.
- $\hat{\Sigma}$ represents the estimated covariance matrix Σ.
- $\hat{\mu}_1$ and $\hat{|}mu_2$ represent estimated means μ_1 and μ_2.
- λ_n is a regularization parameter.
- $||\beta||_1$ represents the L_1 norm of the vector β.

Importance of estimation of β

- High-Dimensional Settings: The optimization problem encourages sparsity in β 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 Initialiation

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 β^*

One of the significant distinctions in CHIME is the requirement to initialize the discriminant direction β^*, 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:**

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

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

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

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

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

Estimation of β^* as Convex Optimization

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

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

Where:

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

the term $\cdot \left(|\hat{\pi}| \vee ||\hat{\mu}_1^{(0)} - \hat{\mu}_2^{(0)}||_{2,s} \vee \|\Sigma^{(0)}\|_{2,s} \right)$ 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 ℓ^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_λ, 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_λ 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_λ 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:

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

Where

- $w(t)$ represents the mixing weight of the second cluster.
- μ_1 is the sample mean for cluster 1, and μ_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:

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

Advantages of CHIME in this E-steps

Compared with traditional calculation of probability

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

In this E-step, the biggest difference is that the calculation of the posterior probability will use β^*, 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:

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

(25)

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

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

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

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

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

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

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, T0, steps. And the termination condition can be set as the l1 norm between the previous μ_1 and the current μ_1 being smaller than a tolerance level, which can be expressed as:

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

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

Where:

- μ_1 is the final mean vector of component one.

- μ_2 is the final mean vector of component two.
- z is the observations.
- β^* 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 β^* 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 β^* with L1 regularization helps avoid this problem.

4.6.3 Sparsity Requirement and Better Generalization

- CHIME only requires sparsity for the discriminant vector β^* 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:

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

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:

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

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 μ_0 (mean vector) and Σ_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 μ_1 (mean vector) and Σ_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

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

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 μ_1. The first 10 elements of μ_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 μ_2. The first 10 elements of μ_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 μ_1. The first 10 elements of μ_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 μ_2. The first 10 elements of μ_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 μ_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 μ_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 μ_1. whose first 10 elements of μ_1 are $5, 6, 7, 8, 9, 10, 11, 12, 13, 14$, and the second mixture model has μ_2. whose first 10 elements of μ_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 μ_1 and μ_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

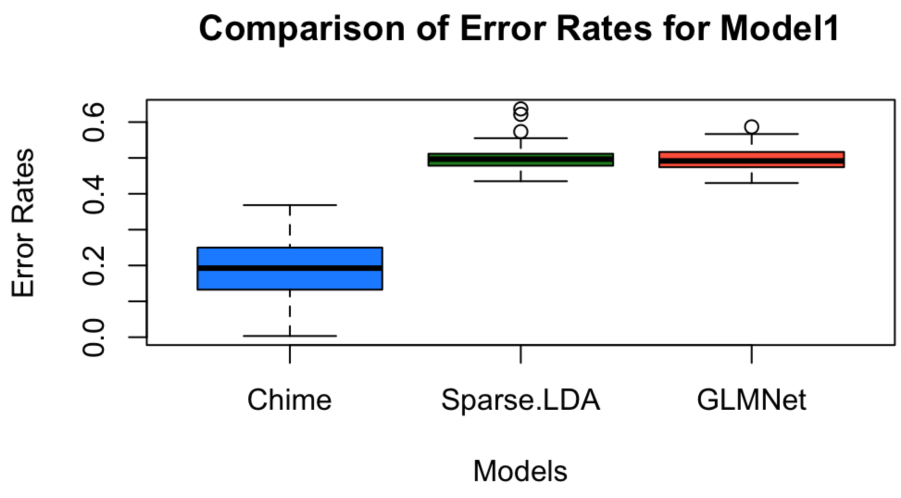

Figure 1: Boxplot Model1

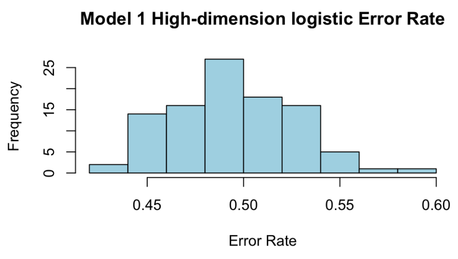

Figure 2: Model1 high-dimension logistic

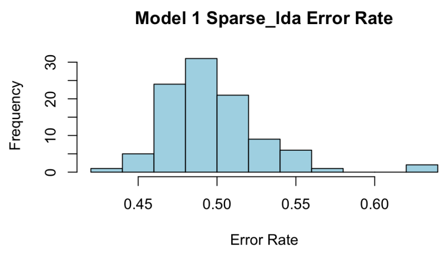

Figure 3: Model1 Sparse LDA

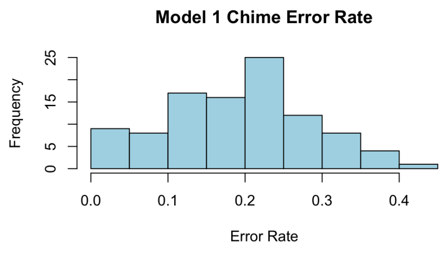

Figure 4: Model1 CHIME

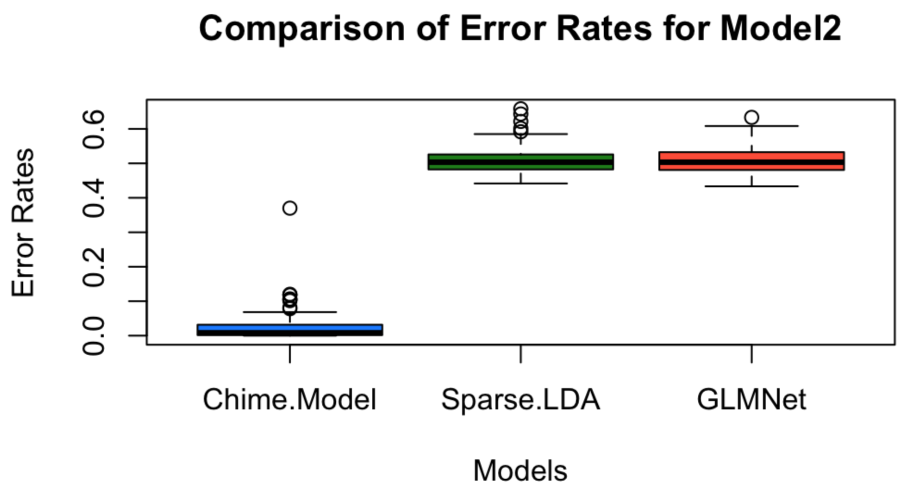

Figure 5: Model2 boxplot

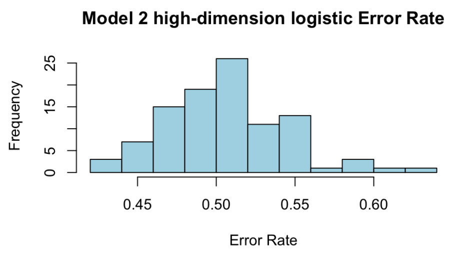

Figure 6: Model2 high-dimension logistic

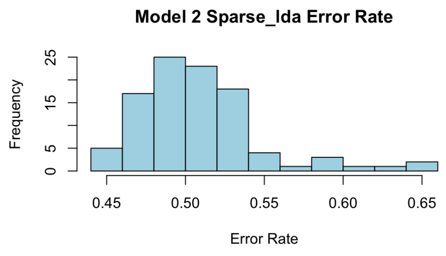

Figure 7: Model2 Sparse LDA

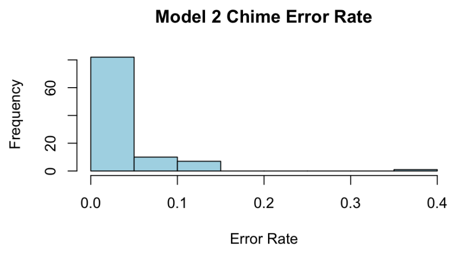

Figure 8: Model2 CHIME

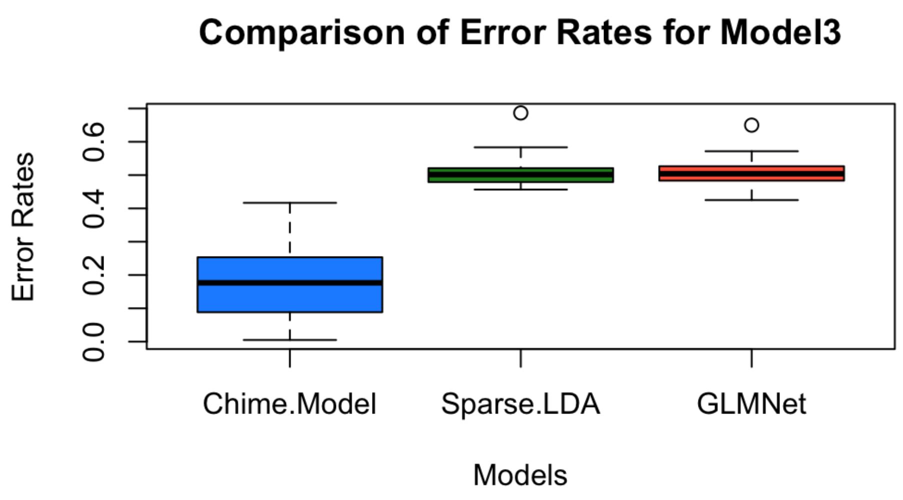

Figure 9: Model2 boxplot

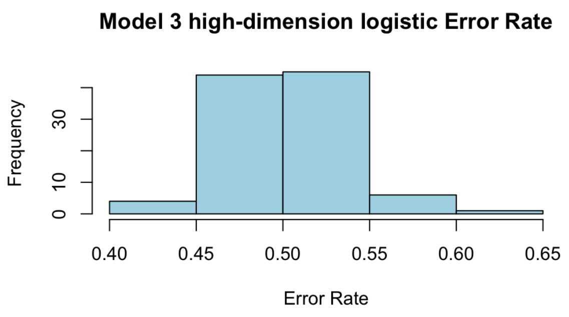

Figure 10: Model3 high-dimension logistic

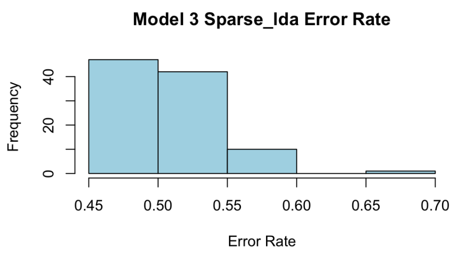

Figure 11: Model3 Sparse LDA

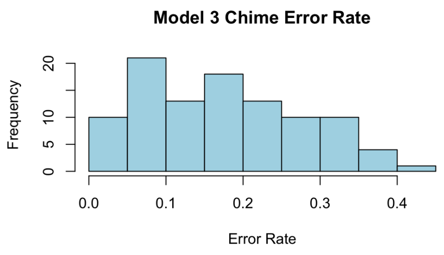

Figure 12: Model3 CHIME

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. [1]. 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. [2] ”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

[1]for more information, please visit https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic
[2]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 unflexible. 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 Explainations 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.

```r
library(mixtools)
library(MASS) # for generating multivariate normal data
library(CVXR)
library(glmnet)
library(mvtnorm)
library(ggplot2)
library(dplyr)
library(sparseLDA)

# Set the number of simulations
num_simulations <- 100

# Create vectors to store correctness rates for Chime, glm, and lda
Chime_error_rates <- numeric(num_simulations)
glmnet_error_rates <- numeric(num_simulations)
Sparse_lda_error_rates <- numeric(num_simulations)

# Repeat the simulation 100 times
for (sim in 1:num_simulations){


# Set the number of rows for your data frame
  n <- 400 # You can change this to your desired number of rows

  # Create 'Y' data frame
  Y= ifelse(runif(n) > 0.5, 1, 0)




  #
  # Generate some example data
  n <-sum(Y==0)


  #label0
  # Define means and covariance matrices for 'X' and 'X1'
  # Generate the dataset
  p<-100
  mean_1 <- rep(0, p)
  mean_1[1:10] <- seq(0.1, 1, by = 0.1)
  mean_2 <- rep(0, p)
  mean_2[1:10] <- seq(-0.2, -2, by = -0.2)

```

```
#
cov_matrix <- matrix(0.8, nrow = p, ncol = p)
for (i in 1:p) {
  for (j in 1:p) {
    cov_matrix[i, j] <- 0.8^abs(i - j)
  }
}




Sigma <- solve(cov_matrix)
# Covariance matrix for 'X1'
w<-100/n
# Generate 'X' and 'X1' using multivariate normal distribution
X <- mvrnorm(100, mu = mean_1, Sigma = Sigma)
X1 <- mvrnorm(n-100, mu = mean_2, Sigma = Sigma)
Y1 = rep(0,sum(Y==0) )
Data <- rbind(X,X1)
Data1 <- cbind(Y = Y1, Data)







#
# Generate some example data
n <- sum(Y==1)


#label0
# Define means and covariance matrices for 'X' and 'X1'
# Generate the dataset
p<-100
mean_1 <- rep(0, p)
mean_1[1:10] <- seq(10, 20, by = 1)
mean_2 <- rep(0, p)
mean_2[1:10] <- seq(-10, -20, by = -1)

#
cov_matrix <- matrix(0.8, nrow = p, ncol = p)
for (i in 1:p) {
  for (j in 1:p) {
    cov_matrix[i, j] <- 0.8^abs(i - j)
  }
}




Sigma <- solve(cov_matrix)
# Covariance matrix for 'X1'
w<-100/n
# Generate 'X' and 'X1' using multivariate normal distribution
X <- mvrnorm(100, mu = mean_1, Sigma = Sigma)
X1 <- mvrnorm(n-100, mu = mean_2, Sigma = Sigma)
Y2 = rep(1, n)
Data <- rbind(X,X1)
Data2<- cbind(Y = Y2, Data)

Data<-rbind(Data1,Data2)
Data<- as.data.frame(Data)



#
# Create a vector to store correctness rates for each fold
# Number of folds (you can change this as needed)
k <- 5

# Create a list to store training and testing data for each fold
data_folds <- list()

# Perform 5-fold cross-validation
for (fold in 1:k) {
  # Split the data into training and testing folds
  split_index <- sample(1:nrow(Data), 0.2 * nrow(Data))
  train_data_fold <- Data[-split_index, ]
  test_data_fold <-Data[split_index, ]

  # Store the data folds in the list
  data_folds[[fold]] <- list(train_data = train_data_fold, test_data = test_data_fold)
}

## Perform 5-fold cross-validation
#first fold
train_data <- data_folds[[1]]$train_data
test_data <- data_folds[[1]]$test_data


split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
train_data <- Data[split_index, ]
test_data <- Data[-split_index, ]

# Create labels for training and testing data
training_labels <- train_data$Y
testing_labels <- test_data$Y



##
train_good_data <- train_data[train_data$Y== "1", ]
train_bad_data <- train_data[train_data$Y == "0", ]

train_good_data <- train_good_data%>%
  select(-Y)

train_bad_data <- train_bad_data%>%
  select(-Y)


test_data <- test_data%>%
  select(-Y)

train_data<- train_data%>%
  select(-Y)


#
data<- as.matrix(train_good_data)
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)







```

```
## Choose number of mixture components
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 210# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
```

```r
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)

n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2) / nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
}



# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}
```

```r





# Perform the iterative algorithm
convergence_threshold <- 10
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ] - (mu_1 + mu_2)
        / 2)))
  }

  # Update 'w'
  w <- sum(a) / n

  # M-step
  # Update 'mu' and 'cov' based on 'a'


  # M-step
  # Update 'mu' and 'cov' based on 'a'
  mu <- matrix(0, nrow = 2, ncol = p)
  mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
  mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
  mu_1 <- mu[1, ]
  mu_2 <- mu[2, ]

  cov <- matrix(0, nrow = p, ncol = p)
  for (i in 1:n) {
    cov <-cov+1/n * (
      (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
        a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
    )
  }

```

```r
    # Update 'beta_optimal' based on 'cov'
    Clambda <- 210# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

    # Get the updated 'beta_optimal'
    beta_optimal <- result$getValue(beta)


    # Check for convergence
    change_in_mu_1<- abs(mu_1 - prev_mu_1)
    if (sum(change_in_mu_1)<convergence_threshold) {
      break
    }

    # Increment the iteration counter
    iteration <- iteration + 1
  }



  #
  pdf<-numeric(120)
  # Calculate density estimates and PDF values for each data point
  for (i in 1:120) {
    data_point <- test_data[i,]
    density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
    density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

    pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
  }

  pdf
```

```r



#
data<- train_bad_data
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)







## Choose number of mixture components
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index
```

```

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 7# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)
# E-step
n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2)/nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
       2))))
}


# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
```

```r
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}





# Perform the iterative algorithm
convergence_threshold <- 1
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ] - (mu_1 + mu_2)
         / 2))))
  }

  # Update 'w'
  w <- sum(a) / n

  # M-step
  # Update 'mu' and 'cov' based on 'a'


  # M-step
```

```
    # Update 'mu' and 'cov' based on 'a'
    mu <- matrix(0, nrow = 2, ncol = p)
    mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
    mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
    mu_1 <- mu[1, ]
    mu_2 <- mu[2, ]

    cov <- matrix(0, nrow = p, ncol = p)
    for (i in 1:n) {
      cov <-cov+1/n * (
        (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
          a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
      )
    }

    # Update 'beta_optimal' based on 'cov'
    Clambda <- 7# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

    # Get the updated 'beta_optimal'
    beta_optimal <- result$getValue(beta)



    # Check for convergence
    change_in_mu_1<- abs(mu_1 - prev_mu_1)
    if (sum(change_in_mu_1)<convergence_threshold) {
      break
    }

    # Increment the iteration counter
    iteration <- iteration + 1
  }
```

```r



#
pdf1<-numeric(120)
# Calculate density estimates and PDF values for each data point
for (i in 1:120) {
  data_point <- test_data[i,]
  density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
  density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

  pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
}










# Create vectors to store probabilities
probabilities <- numeric(nrow(test_data))

# Iterate through each data point
for (i in 1:nrow(test_data)) {
  pdf_value <- pdf[i]  # Density from the first GMM
  pdf1_value <- pdf1[i]  # Density from the second GMM
  y <- sum(testing_labels == 1)
  z<- sum(testing_labels == 0)

  # Calculate probability using the formula
  probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)

  # Store the probability in the vector
  probabilities[i] <- probability
}

threshold<-0.5
# Classify instances based on the threshold
classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
classification<- na.omit(classification)
testing_labels
```

```r
classification

Chime_correct_rates_1<-sum(testing_labels==classification)/120



## Load the glmnet package if not already loaded
if (!require(glmnet)) {
  install.packages("glmnet")
  library(glmnet)
}

# Define the response variable (Y) and predictor variables (X) for training data
training_response <- as.numeric(training_labels)
training_predictors <- as.matrix(train_data)  # Exclude the first column (Y)

# Fit a generalized linear model with elastic net regularization
alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
lambda <- 0.5 # Adjust the regularization strength (tune this parameter)

# Create the glmnet model
model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)

# Print a summary of the model
print(model)

# Plot the coefficients along the regularization path
plot(model)

# To make predictions on test data, you can use the predict() function
# Define the predictor variables for test data
testing_predictors <- as.matrix(test_data)  # Exclude the first column (Y)

# Make predictions on the test data
predictions <- predict(model, newx = testing_predictors, s = lambda)

threshold<-0.5
# Classify instances based on the threshold
classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
classification1<- na.omit(classification1)
testing_labels
classification1

glmnet_correct_rates_1<-sum(testing_labels==classification1)/120
```

```
#
# Load data
X <- as.matrix(train_data)
Y <- as.matrix(training_labels)


new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1)  # Creating a new column
    with zeros
colnames(new_column) <- "NewColumn"  # Setting the column name (change "NewColumn" to your
    desired name)

# Combine Y and the new column using cbind
Y <- cbind(Y, new_column)

colnames(Y) <- c("0",
                 "1")
## training data
Xtr<-X
k<-2
n<-dim(Xtr)[1]
## Normalize data
Xc<-normalize(Xtr)
Xn<-Xc$Xc
p<-dim(Xn)[2]
## Perform SDA with one non-zero loading for each discriminative
## direction with Y as matrix input
out <- sda(Xn, Y,
           lambda = 1e-6,
           stop = -1,
           maxIte = 25,
           trace = TRUE)
## predict training samples
train <- predict(out, Xn)
## testing
Xtst<-as.matrix(test_data)
Xtst<-normalizetest(Xtst,Xc)

test <- predict(out, Xtst)
print(test$class)
```

```


Sparse_lda_correct_rates_1<-sum(testing_labels==test$class)/120


## Perform 5-fold cross-validation
#second fold
train_data <- data_folds[[2]]$train_data
test_data <- data_folds[[2]]$test_data


split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
train_data <- Data[split_index, ]
test_data <- Data[-split_index, ]

# Create labels for training and testing data
training_labels <- train_data$Y
testing_labels <- test_data$Y



##
train_good_data <- train_data[train_data$Y== "1", ]
train_bad_data <- train_data[train_data$Y == "0", ]

train_good_data <- train_good_data%>%
  select(-Y)

train_bad_data <- train_bad_data%>%
  select(-Y)


test_data <- test_data%>%
  select(-Y)

train_data<- train_data%>%
  select(-Y)


#
data<- as.matrix(train_good_data)
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)
```

```
## Choose number of mixture components
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2
```

```



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 210# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)

n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2) / nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
}



# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

```

```
cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}





# Perform the iterative algorithm
convergence_threshold <- 10
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ] - (mu_1 + mu_2)
        / 2)))
  }

  # Update 'w'
  w <- sum(a) / n

  # M-step
  # Update 'mu' and 'cov' based on 'a'


  # M-step
  # Update 'mu' and 'cov' based on 'a'
  mu <- matrix(0, nrow = 2, ncol = p)
  mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
  mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
  mu_1 <- mu[1, ]
  mu_2 <- mu[2, ]

  cov <- matrix(0, nrow = p, ncol = p)
```

```
    for (i in 1:n) {
      cov <-cov+1/n * (
        (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
          a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
      )
    }

    # Update 'beta_optimal' based on 'cov'
    Clambda <- 210# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

    # Get the updated 'beta_optimal'
    beta_optimal <- result$getValue(beta)


    # Check for convergence
    change_in_mu_1<- abs(mu_1 - prev_mu_1)
    if (sum(change_in_mu_1)<convergence_threshold) {
      break
    }

    # Increment the iteration counter
    iteration <- iteration + 1
  }



  #
  pdf<-numeric(120)
  # Calculate density estimates and PDF values for each data point
  for (i in 1:120) {
    data_point <- test_data[i,]
    density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
    density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

    pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
  }

  pdf



  #
  data<- train_bad_data
  n <- nrow(data)
  p <- ncol(data)
  s <- sqrt(n)








  ## Choose number of mixture components
  k <- 2

  # Use k-means to initialize the parameters
  mean_1 <- rep(0,p)
  mean_1[1:10] <- seq(0.1, 1, by = 0.1)
  kmeans_fit <- kmeans(data, centers = k)
  means_init <- kmeans_fit$centers
  distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
  # Find the index of the center that is closer to m1
  closest_center_index <- which.min(distances)

  # Assign mu_1 to the closer center and mu_2 to the other center
  mu_1 <- means_init[closest_center_index, ]
  mu_2 <- means_init[-closest_center_index, ]


  # Calculate the covariance matrices of the two clusters
  cluster_indices <- kmeans_fit$cluster == closest_center_index

  # Subset the data points that belong to the cluster associated with mu_1
  cluster_data <- data[cluster_indices, ]
```

```

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 7# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)
# E-step
n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2)/nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
       2))))
}
```

```


# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}





# Perform the iterative algorithm
convergence_threshold <- 1
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ] - (mu_1 + mu_2)
         / 2))))
  }

  # Update 'w'
```

```
    w <- sum(a) / n

    # M-step
    # Update 'mu' and 'cov' based on 'a'


    # M-step
    # Update 'mu' and 'cov' based on 'a'
    mu <- matrix(0, nrow = 2, ncol = p)
    mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
    mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
    mu_1 <- mu[1, ]
    mu_2 <- mu[2, ]

    cov <- matrix(0, nrow = p, ncol = p)
    for (i in 1:n) {
      cov <-cov+1/n * (
        (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
          a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
      )
    }

    # Update 'beta_optimal' based on 'cov'
    Clambda <- 7# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

    # Get the updated 'beta_optimal'
    beta_optimal <- result$getValue(beta)


    # Check for convergence
    change_in_mu_1<- abs(mu_1 - prev_mu_1)
```

```
  if (sum(change_in_mu_1)<convergence_threshold) {
    break
  }

  # Increment the iteration counter
  iteration <- iteration + 1
}



#
pdf1<-numeric(120)
# Calculate density estimates and PDF values for each data point
for (i in 1:120) {
  data_point <- test_data[i,]
  density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
  density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

  pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
}









# Create vectors to store probabilities
probabilities <- numeric(nrow(test_data))

# Iterate through each data point
for (i in 1:nrow(test_data)) {
  pdf_value <- pdf[i]  # Density from the first GMM
  pdf1_value <- pdf1[i]  # Density from the second GMM
  y <- sum(testing_labels == 1)
  z<- sum(testing_labels == 0)

  # Calculate probability using the formula
  probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)

  # Store the probability in the vector
  probabilities[i] <- probability
```

```
}

threshold<-0.5
# Classify instances based on the threshold
classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
classification<- na.omit(classification)
testing_labels
classification

Chime_correct_rates_2<-sum(testing_labels==classification)/120



## Load the glmnet package if not already loaded
if (!require(glmnet)) {
  install.packages("glmnet")
  library(glmnet)
}

# Define the response variable (Y) and predictor variables (X) for training data
training_response <- as.numeric(training_labels)
training_predictors <- as.matrix(train_data)  # Exclude the first column (Y)

# Fit a generalized linear model with elastic net regularization
alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
lambda <- 0.5 # Adjust the regularization strength (tune this parameter)

# Create the glmnet model
model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)

# Print a summary of the model
print(model)

# Plot the coefficients along the regularization path
plot(model)

# To make predictions on test data, you can use the predict() function
# Define the predictor variables for test data
testing_predictors <- as.matrix(test_data)  # Exclude the first column (Y)

# Make predictions on the test data
predictions <- predict(model, newx = testing_predictors, s = lambda)

threshold<-0.5
```

```
# Classify instances based on the threshold
classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
classification1<- na.omit(classification1)
testing_labels
classification1

glmnet_correct_rates_2<-sum(testing_labels==classification1)/120





#
# Load data
X <- as.matrix(train_data)
Y <- as.matrix(training_labels)


new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1)  # Creating a new column
    with zeros
colnames(new_column) <- "NewColumn"  # Setting the column name (change "NewColumn" to your
    desired name)

# Combine Y and the new column using cbind
Y <- cbind(Y, new_column)

colnames(Y) <- c("0",
                 "1")
## training data
Xtr<-X
k<-2
n<-dim(Xtr)[1]
## Normalize data
Xc<-normalize(Xtr)
Xn<-Xc$Xc
p<-dim(Xn)[2]
## Perform SDA with one non-zero loading for each discriminative
## direction with Y as matrix input
out <- sda(Xn, Y,
           lambda = 1e-6,
           stop = -1,
           maxIte = 25,
           trace = TRUE)
## predict training samples
```

```
train <- predict(out, Xn)
## testing
Xtst<-as.matrix(test_data)
Xtst<-normalizetest(Xtst,Xc)

test <- predict(out, Xtst)
print(test$class)


Sparse_lda_correct_rates_2<-sum(testing_labels==test$class)/120


## Perform 5-fold cross-validation
##third fold
train_data <- data_folds[[3]]$train_data
test_data <- data_folds[[3]]$test_data


split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
train_data <- Data[split_index, ]
test_data <- Data[-split_index, ]

# Create labels for training and testing data
training_labels <- train_data$Y
testing_labels <- test_data$Y



##
train_good_data <- train_data[train_data$Y== "1", ]
train_bad_data <- train_data[train_data$Y == "0", ]

train_good_data <- train_good_data%>%
  select(-Y)

train_bad_data <- train_bad_data%>%
  select(-Y)


test_data <- test_data%>%
  select(-Y)

train_data<- train_data%>%
  select(-Y)
```

```


#
data<- as.matrix(train_good_data)
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)







## Choose number of mixture components
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

```

```
# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 210# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)

n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2) / nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
}



# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
```

```r
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}





# Perform the iterative algorithm
convergence_threshold <- 10
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ] - (mu_1 + mu_2)
        / 2)))
  }

  # Update 'w'
  w <- sum(a) / n

  # M-step
  # Update 'mu' and 'cov' based on 'a'


  # M-step
  # Update 'mu' and 'cov' based on 'a'
```

```
    mu <- matrix(0, nrow = 2, ncol = p)
    mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
    mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
    mu_1 <- mu[1, ]
    mu_2 <- mu[2, ]

    cov <- matrix(0, nrow = p, ncol = p)
    for (i in 1:n) {
      cov <-cov+1/n * (
        (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
          a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
      )
    }

    # Update 'beta_optimal' based on 'cov'
    Clambda <- 210# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

    # Get the updated 'beta_optimal'
    beta_optimal <- result$getValue(beta)


    # Check for convergence
    change_in_mu_1<- abs(mu_1 - prev_mu_1)
    if (sum(change_in_mu_1)<convergence_threshold) {
      break
    }

    # Increment the iteration counter
    iteration <- iteration + 1
  }

```

```


#
pdf<-numeric(120)
# Calculate density estimates and PDF values for each data point
for (i in 1:120) {
  data_point <- test_data[i,]
  density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
  density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

  pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
}

pdf



#
data<- train_bad_data
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)







## Choose number of mixture components
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]
```

```r


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 7# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)
# E-step
```

```
n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2)/nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
       2))))
}


# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}





# Perform the iterative algorithm
convergence_threshold <- 1
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
```

```
    a <- numeric(n)
    for (i in 1:n) {
      a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ] - (mu_1 + mu_2)
            / 2))))
    }

    # Update 'w'
    w <- sum(a) / n

    # M-step
    # Update 'mu' and 'cov' based on 'a'


    # M-step
    # Update 'mu' and 'cov' based on 'a'
    mu <- matrix(0, nrow = 2, ncol = p)
    mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
    mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
    mu_1 <- mu[1, ]
    mu_2 <- mu[2, ]

    cov <- matrix(0, nrow = p, ncol = p)
    for (i in 1:n) {
      cov <-cov+1/n * (
        (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
          a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
      )
    }

    # Update 'beta_optimal' based on 'cov'
    Clambda <- 7# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

    # Get the updated 'beta_optimal'
    beta_optimal <- result$getValue(beta)


    # Check for convergence
    change_in_mu_1<- abs(mu_1 - prev_mu_1)
    if (sum(change_in_mu_1)<convergence_threshold) {
      break
    }

    # Increment the iteration counter
    iteration <- iteration + 1
  }



  #
  pdf1<-numeric(120)
  # Calculate density estimates and PDF values for each data point
  for (i in 1:120) {
    data_point <- test_data[i,]
    density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
    density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

    pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
  }









  # Create vectors to store probabilities
  probabilities <- numeric(nrow(test_data))

  # Iterate through each data point
  for (i in 1:nrow(test_data)) {
    pdf_value <- pdf[i]  # Density from the first GMM
    pdf1_value <- pdf1[i]  # Density from the second GMM
    y <- sum(testing_labels == 1)
    z<- sum(testing_labels == 0)

    # Calculate probability using the formula
    probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)

    # Store the probability in the vector
    probabilities[i] <- probability
  }

  threshold<-0.5
  # Classify instances based on the threshold
  classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
  classification<- na.omit(classification)
  testing_labels
  classification

  Chime_correct_rates_3<-sum(testing_labels==classification)/120



  ## Load the glmnet package if not already loaded
  if (!require(glmnet)) {
    install.packages("glmnet")
    library(glmnet)
  }

  # Define the response variable (Y) and predictor variables (X) for training data
  training_response <- as.numeric(training_labels)
  training_predictors <- as.matrix(train_data)  # Exclude the first column (Y)

  # Fit a generalized linear model with elastic net regularization
  alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
  lambda <- 0.5 # Adjust the regularization strength (tune this parameter)

  # Create the glmnet model
  model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)

  # Print a summary of the model
  print(model)

  # Plot the coefficients along the regularization path
  plot(model)

  # To make predictions on test data, you can use the predict() function
```

```r
# Define the predictor variables for test data
testing_predictors <- as.matrix(test_data)  # Exclude the first column (Y)

# Make predictions on the test data
predictions <- predict(model, newx = testing_predictors, s = lambda)

threshold<-0.5
# Classify instances based on the threshold
classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
classification1<- na.omit(classification1)
testing_labels
classification1

glmnet_correct_rates_3<-sum(testing_labels==classification1)/120





#
# Load data
X <- as.matrix(train_data)
Y <- as.matrix(training_labels)


new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1)  # Creating a new column
    with zeros
colnames(new_column) <- "NewColumn"  # Setting the column name (change "NewColumn" to your
    desired name)

# Combine Y and the new column using cbind
Y <- cbind(Y, new_column)

colnames(Y) <- c("0",
                 "1")
## training data
Xtr<-X
k<-2
n<-dim(Xtr)[1]
## Normalize data
Xc<-normalize(Xtr)
Xn<-Xc$Xc
p<-dim(Xn)[2]
## Perform SDA with one non-zero loading for each discriminative
## direction with Y as matrix input
out <- sda(Xn, Y,
           lambda = 1e-6,
           stop = -1,
           maxIte = 25,
           trace = TRUE)
## predict training samples
train <- predict(out, Xn)
## testing
Xtst<-as.matrix(test_data)
Xtst<-normalizetest(Xtst,Xc)

test <- predict(out, Xtst)
print(test$class)


Sparse_lda_correct_rates_3<-sum(testing_labels==test$class)/120


## Perform 5-fold cross-validation

train_data <- data_folds[[4]]$train_data
test_data <- data_folds[[4]]$test_data


split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
train_data <- Data[split_index, ]
test_data <- Data[-split_index, ]

# Create labels for training and testing data
training_labels <- train_data$Y
testing_labels <- test_data$Y



##
train_good_data <- train_data[train_data$Y== "1", ]
train_bad_data <- train_data[train_data$Y == "0", ]

train_good_data <- train_good_data%>%
  select(-Y)

train_bad_data <- train_bad_data%>%
  select(-Y)
```

```


test_data <- test_data%>%
  select(-Y)

train_data<- train_data%>%
  select(-Y)


#
data<- as.matrix(train_good_data)
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)







## Choose number of mixture components
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

```

```
# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 210# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)

n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2) / nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
}


```

```

# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}





# Perform the iterative algorithm
convergence_threshold <- 10
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ] - (mu_1 + mu_2)
        / 2)))
  }

  # Update 'w'
  w <- sum(a) / n
```

```

    # M-step
    # Update 'mu' and 'cov' based on 'a'


    # M-step
    # Update 'mu' and 'cov' based on 'a'
    mu <- matrix(0, nrow = 2, ncol = p)
    mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
    mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
    mu_1 <- mu[1, ]
    mu_2 <- mu[2, ]

    cov <- matrix(0, nrow = p, ncol = p)
    for (i in 1:n) {
      cov <-cov+1/n * (
        (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
          a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
      )
    }

    # Update 'beta_optimal' based on 'cov'
    Clambda <- 210# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

    # Get the updated 'beta_optimal'
    beta_optimal <- result$getValue(beta)



    # Check for convergence
    change_in_mu_1<- abs(mu_1 - prev_mu_1)
    if (sum(change_in_mu_1)<convergence_threshold) {
```

```r
      break
    }

    # Increment the iteration counter
    iteration <- iteration + 1
  }



  #
  pdf<-numeric(120)
  # Calculate density estimates and PDF values for each data point
  for (i in 1:120) {
    data_point <- test_data[i,]
    density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
    density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

    pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
  }

  pdf



  #
  data<- train_bad_data
  n <- nrow(data)
  p <- ncol(data)
  s <- sqrt(n)







  ## Choose number of mixture components
  k <- 2

  # Use k-means to initialize the parameters
  mean_1 <- rep(0,p)
  mean_1[1:10] <- seq(0.1, 1, by = 0.1)
  kmeans_fit <- kmeans(data, centers = k)
  means_init <- kmeans_fit$centers
```

```
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 7# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)
```

```r

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)
# E-step
n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2)/nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
       2))))
}


# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}





# Perform the iterative algorithm
convergence_threshold <- 1
max_iterations <- Inf
```

```
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ] - (mu_1 + mu_2)
         / 2))))
  }

  # Update 'w'
  w <- sum(a) / n

  # M-step
  # Update 'mu' and 'cov' based on 'a'


  # M-step
  # Update 'mu' and 'cov' based on 'a'
  mu <- matrix(0, nrow = 2, ncol = p)
  mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
  mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
  mu_1 <- mu[1, ]
  mu_2 <- mu[2, ]

  cov <- matrix(0, nrow = p, ncol = p)
  for (i in 1:n) {
    cov <-cov+1/n * (
      (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
        a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
    )
  }

  # Update 'beta_optimal' based on 'cov'
  Clambda <- 7# Adjust as needed
  k<-0.1
  lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
  beta <- Variable(p)

  # Define the objective function with L1 regularization
  objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
      0 * sum(p_norm(beta, 1))))

  # Define the problem
  problem <- Problem(objective)

  # Solve the problem
  result <- solve(problem)

  # Get the updated 'beta_optimal'
  beta_optimal <- result$getValue(beta)


  # Check for convergence
  change_in_mu_1<- abs(mu_1 - prev_mu_1)
  if (sum(change_in_mu_1)<convergence_threshold) {
    break
  }

  # Increment the iteration counter
  iteration <- iteration + 1
}



#
pdf1<-numeric(120)
# Calculate density estimates and PDF values for each data point
for (i in 1:120) {
  data_point <- test_data[i,]
  density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
  density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

  pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
}










# Create vectors to store probabilities
```

```r
probabilities <- numeric(nrow(test_data))

# Iterate through each data point
for (i in 1:nrow(test_data)) {
  pdf_value <- pdf[i]  # Density from the first GMM
  pdf1_value <- pdf1[i]  # Density from the second GMM
  y <- sum(testing_labels == 1)
  z<- sum(testing_labels == 0)

  # Calculate probability using the formula
  probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)

  # Store the probability in the vector
  probabilities[i] <- probability
}

threshold<-0.5
# Classify instances based on the threshold
classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
classification<- na.omit(classification)
testing_labels
classification

Chime_correct_rates_4<-sum(testing_labels==classification)/120



## Load the glmnet package if not already loaded
if (!require(glmnet)) {
  install.packages("glmnet")
  library(glmnet)
}

# Define the response variable (Y) and predictor variables (X) for training data
training_response <- as.numeric(training_labels)
training_predictors <- as.matrix(train_data)  # Exclude the first column (Y)

# Fit a generalized linear model with elastic net regularization
alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
lambda <- 0.5 # Adjust the regularization strength (tune this parameter)

# Create the glmnet model
model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)

```

```r
# Print a summary of the model
print(model)

# Plot the coefficients along the regularization path
plot(model)

# To make predictions on test data, you can use the predict() function
# Define the predictor variables for test data
testing_predictors <- as.matrix(test_data)  # Exclude the first column (Y)

# Make predictions on the test data
predictions <- predict(model, newx = testing_predictors, s = lambda)

threshold<-0.5
# Classify instances based on the threshold
classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
classification1<- na.omit(classification1)
testing_labels
classification1

glmnet_correct_rates_4<-sum(testing_labels==classification1)/120





#
# Load data
X <- as.matrix(train_data)
Y <- as.matrix(training_labels)


new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1)  # Creating a new column
    with zeros
colnames(new_column) <- "NewColumn"  # Setting the column name (change "NewColumn" to your
    desired name)

# Combine Y and the new column using cbind
Y <- cbind(Y, new_column)

colnames(Y) <- c("0",
                 "1")
## training data
Xtr<-X
```

```r
k<-2
n<-dim(Xtr)[1]
## Normalize data
Xc<-normalize(Xtr)
Xn<-Xc$Xc
p<-dim(Xn)[2]
## Perform SDA with one non-zero loading for each discriminative
## direction with Y as matrix input
out <- sda(Xn, Y,
           lambda = 1e-6,
           stop = -1,
           maxIte = 25,
           trace = TRUE)
## predict training samples
train <- predict(out, Xn)
## testing
Xtst<-as.matrix(test_data)
Xtst<-normalizetest(Xtst,Xc)

test <- predict(out, Xtst)
print(test$class)


Sparse_lda_correct_rates_4<-sum(testing_labels==test$class)/120


## Perform 5-fold cross-validation

train_data <- data_folds[[5]]$train_data
test_data <- data_folds[[5]]$test_data


split_index <- sample(1:nrow(Data), 0.7 * nrow(Data))
train_data <- Data[split_index, ]
test_data <- Data[-split_index, ]

# Create labels for training and testing data
training_labels <- train_data$Y
testing_labels <- test_data$Y



##
train_good_data <- train_data[train_data$Y== "1", ]
```

```
train_bad_data <- train_data[train_data$Y == "0", ]

train_good_data <- train_good_data%>%
  select(-Y)

train_bad_data <- train_bad_data%>%
  select(-Y)


test_data <- test_data%>%
  select(-Y)

train_data<- train_data%>%
  select(-Y)


#
data<- as.matrix(train_good_data)
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)







## Choose number of mixture components
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]

```

```

# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 210# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)

n <- nrow(data)
```

```
  a <- numeric(n)
  w <- nrow(cluster_data_mu2) / nrow(train_good_data)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t((beta_optimal)) %*% (data[i, ] - (mu_1 + mu_2) / 2)))
  }



  # Update 'w'
  w <- sum(a) / n

  # M-step
  # Update 'mu' and 'cov' based on 'a'
  # M-step
  # Update 'mu' and 'cov' based on 'a'
  mu <- matrix(0, nrow = 2, ncol = p)
  mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
  mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
  mu_1 <- mu[1, ]
  mu_2 <- mu[2, ]

  cov <- matrix(0, nrow = p, ncol = p)
  for (i in 1:n) {
    cov <-cov+1/n * (
      (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
        a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
    )
  }





  # Perform the iterative algorithm
  convergence_threshold <- 10
  max_iterations <- Inf
  iteration <- 1

  # Create a vector to store mu for this iteration
  while (iteration <= max_iterations) {
    prev_mu_1<-mu_1
    # E-step
    n <- nrow(data)
    a <- numeric(n)
    for (i in 1:n) {
      a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric(data[i, ] - (mu_1 + mu_2)
          / 2)))
    }

    # Update 'w'
    w <- sum(a) / n

    # M-step
    # Update 'mu' and 'cov' based on 'a'


    # M-step
    # Update 'mu' and 'cov' based on 'a'
    mu <- matrix(0, nrow = 2, ncol = p)
    mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
    mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
    mu_1 <- mu[1, ]
    mu_2 <- mu[2, ]

    cov <- matrix(0, nrow = p, ncol = p)
    for (i in 1:n) {
      cov <-cov+1/n * (
        (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
          a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
      )
    }

    # Update 'beta_optimal' based on 'cov'
    Clambda <- 210# Adjust as needed
    k<-0.1
    lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
    beta <- Variable(p)

    # Define the objective function with L1 regularization
    objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
        0 * sum(p_norm(beta, 1))))

    # Define the problem
    problem <- Problem(objective)

    # Solve the problem
    result <- solve(problem)

```

```
  # Get the updated 'beta_optimal'
  beta_optimal <- result$getValue(beta)


  # Check for convergence
  change_in_mu_1<- abs(mu_1 - prev_mu_1)
  if (sum(change_in_mu_1)<convergence_threshold) {
    break
  }

  # Increment the iteration counter
  iteration <- iteration + 1
}



#
pdf<-numeric(120)
# Calculate density estimates and PDF values for each data point
for (i in 1:120) {
  data_point <- test_data[i,]
  density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
  density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

  pdf[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
}

pdf



#
data<- train_bad_data
n <- nrow(data)
p <- ncol(data)
s <- sqrt(n)







## Choose number of mixture components
```

```
k <- 2

# Use k-means to initialize the parameters
mean_1 <- rep(0,p)
mean_1[1:10] <- seq(0.1, 1, by = 0.1)
kmeans_fit <- kmeans(data, centers = k)
means_init <- kmeans_fit$centers
distances <- apply(means_init, 1, function(center) sum((center - mean_1)^2))
# Find the index of the center that is closer to m1
closest_center_index <- which.min(distances)

# Assign mu_1 to the closer center and mu_2 to the other center
mu_1 <- means_init[closest_center_index, ]
mu_2 <- means_init[-closest_center_index, ]


# Calculate the covariance matrices of the two clusters
cluster_indices <- kmeans_fit$cluster == closest_center_index

# Subset the data points that belong to the cluster associated with mu_1
cluster_data <- data[cluster_indices, ]

# Calculate the covariance matrix for this cluster
cov_matrix_1 <- cov(cluster_data)

#
# Get the indices of data points belonging to the cluster associated with mu_2
cluster_indices_mu2 <- kmeans_fit$cluster != closest_center_index

# Subset the data points that belong to the cluster associated with mu_2
cluster_data_mu2 <- data[cluster_indices_mu2, ]

# Calculate the covariance matrix for this cluster
cov_matrix_2 <- cov(cluster_data_mu2)

cov<-(cov_matrix_1+cov_matrix_2)/2



# Initial Step
# Update 'beta_optimal' based on 'cov'
Clambda <- 7# Adjust as needed
lambda_0 <- Clambda * sqrt(log(p) / n)
beta <- Variable(p)
```

```

# Define the objective function with L1 regularization
objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_0
    * sum(p_norm(beta, 1))))

# Define the problem
problem <- Problem(objective)

# Solve the problem
result <- solve(problem)

# Get the updated 'beta_optimal'
beta_optimal <- result$getValue(beta)
# E-step
n <- nrow(data)
a <- numeric(n)
w <- nrow(cluster_data_mu2)/nrow(train_good_data)
for (i in 1:n) {
  a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% as.numeric((data[i, ] - (mu_1 + mu_2) /
       2))))
}


# Update 'w'
w <- sum(a) / n

# M-step
# Update 'mu' and 'cov' based on 'a'
# M-step
# Update 'mu' and 'cov' based on 'a'
mu <- matrix(0, nrow = 2, ncol = p)
mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
mu_1 <- mu[1, ]
mu_2 <- mu[2, ]

cov <- matrix(0, nrow = p, ncol = p)
for (i in 1:n) {
  cov <-cov+1/n * (
    (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
      a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
  )
}

```

```




# Perform the iterative algorithm
convergence_threshold <- 1
max_iterations <- Inf
iteration <- 1

# Create a vector to store mu for this iteration
while (iteration <= max_iterations) {
  prev_mu_1<-mu_1
  # E-step
  n <- nrow(data)
  a <- numeric(n)
  for (i in 1:n) {
    a[i] <- w / (w + (1 - w) * exp(t(beta_optimal) %*% (as.numeric(data[i, ] - (mu_1 + mu_2)
         / 2))))
  }

  # Update 'w'
  w <- sum(a) / n

  # M-step
  # Update 'mu' and 'cov' based on 'a'


  # M-step
  # Update 'mu' and 'cov' based on 'a'
  mu <- matrix(0, nrow = 2, ncol = p)
  mu[1, ] <- (n - sum(a))^(-1) * ((1 - a) %*% as.matrix(data))
  mu[2, ] <- (sum(a))^(-1) * (a %*% as.matrix(data))
  mu_1 <- mu[1, ]
  mu_2 <- mu[2, ]

  cov <- matrix(0, nrow = p, ncol = p)
  for (i in 1:n) {
    cov <-cov+1/n * (
      (1 - a[i]) * (as.numeric(data[i, ]) - mu_1) %*% t(as.numeric(data[i, ]) - mu_1) +
        a[i] * (as.numeric(data[i, ]) - mu_2) %*% t(as.numeric(data[i, ]) - mu_2)
    )
  }

  # Update 'beta_optimal' based on 'cov'
  Clambda <- 7# Adjust as needed
  k<-0.1
  lambda_0 <- k*lambda_0+Clambda * sqrt(log(p) / n)
  beta <- Variable(p)

  # Define the objective function with L1 regularization
  objective <- Minimize(sum(0.5 * quad_form(beta, cov) - t(beta) %*% (mu_1 - mu_2) + lambda_
      0 * sum(p_norm(beta, 1))))

  # Define the problem
  problem <- Problem(objective)

  # Solve the problem
  result <- solve(problem)

  # Get the updated 'beta_optimal'
  beta_optimal <- result$getValue(beta)


  # Check for convergence
  change_in_mu_1<- abs(mu_1 - prev_mu_1)
  if (sum(change_in_mu_1)<convergence_threshold) {
    break
  }

  # Increment the iteration counter
  iteration <- iteration + 1
}



#
pdf1<-numeric(120)
# Calculate density estimates and PDF values for each data point
for (i in 1:120) {
  data_point <- test_data[i,]
  density_estimations_cluster_1 <- dmvnorm(data_point, mean = mu_1, sigma = cov)
  density_estimations_cluster_2 <- dmvnorm(data_point, mean = mu_2, sigma=cov)

  pdf1[i] <- (1 - w) * density_estimations_cluster_1 + w * density_estimations_cluster_2
}



```

```r






# Create vectors to store probabilities
probabilities <- numeric(nrow(test_data))

# Iterate through each data point
for (i in 1:nrow(test_data)) {
  pdf_value <- pdf[i]  # Density from the first GMM
  pdf1_value <- pdf1[i]  # Density from the second GMM
  y <- sum(testing_labels == 1)
  z<- sum(testing_labels == 0)

  # Calculate probability using the formula
  probability <- y * pdf_value / (z* pdf_value + y* pdf1_value)

  # Store the probability in the vector
  probabilities[i] <- probability
}

threshold<-0.5
# Classify instances based on the threshold
classification <- ifelse(is.na(probabilities), NA, ifelse(probabilities > threshold, 1, 0))
classification<- na.omit(classification)
testing_labels
classification

Chime_correct_rates_5<-sum(testing_labels==classification)/120



## Load the glmnet package if not already loaded
if (!require(glmnet)) {
  install.packages("glmnet")
  library(glmnet)
}

# Define the response variable (Y) and predictor variables (X) for training data
training_response <- as.numeric(training_labels)
training_predictors <- as.matrix(train_data)  # Exclude the first column (Y)

```

```r
# Fit a generalized linear model with elastic net regularization
alpha <- 0.1 # Adjust this value for the desired balance between L1 and L2 regularization
lambda <- 0.5 # Adjust the regularization strength (tune this parameter)

# Create the glmnet model
model <- glmnet(training_predictors, y = training_response, alpha = alpha, lambda = lambda)

# Print a summary of the model
print(model)

# Plot the coefficients along the regularization path
plot(model)

# To make predictions on test data, you can use the predict() function
# Define the predictor variables for test data
testing_predictors <- as.matrix(test_data)  # Exclude the first column (Y)

# Make predictions on the test data
predictions <- predict(model, newx = testing_predictors, s = lambda)

threshold<-0.5
# Classify instances based on the threshold
classification1 <- ifelse(is.na(predictions), NA, ifelse(predictions > threshold, 0, 1))
classification1<- na.omit(classification1)
testing_labels
classification1

glmnet_correct_rates_5<-sum(testing_labels==classification1)/120






#
# Load data
X <- as.matrix(train_data)
Y <- as.matrix(training_labels)


new_column <- matrix(1-training_labels, nrow = nrow(Y), ncol = 1)  # Creating a new column
    with zeros
colnames(new_column) <- "NewColumn"  # Setting the column name (change "NewColumn" to your
    desired name)

```

```
  # Combine Y and the new column using cbind
  Y <- cbind(Y, new_column)

  colnames(Y) <- c("0",
                   "1")
  ## training data
  Xtr<-X
  k<-2
  n<-dim(Xtr)[1]
  ## Normalize data
  Xc<-normalize(Xtr)
  Xn<-Xc$Xc
  p<-dim(Xn)[2]
  ## Perform SDA with one non-zero loading for each discriminative
  ## direction with Y as matrix input
  out <- sda(Xn, Y,
             lambda = 1e-6,
             stop = -1,
             maxIte = 25,
             trace = TRUE)
  ## predict training samples
  train <- predict(out, Xn)
  ## testing
  Xtst<-as.matrix(test_data)
  Xtst<-normalizetest(Xtst,Xc)

  test <- predict(out, Xtst)
  print(test$class)


  Sparse_lda_correct_rates_5<-sum(testing_labels==test$class)/120


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

glmnet_correct_rates<-(glmnet_correct_rates_1+glmnet_correct_rates_2+glmnet_correct_rates_3+
    glmnet_correct_rates_4+glmnet_correct_rates_5)/5


Chime_correct_rates<-(Chime_correct_rates_1+Chime_correct_rates_2+Chime_correct_rates_3+Chime_
    correct_rates_4+Chime_correct_rates_5)/5

# Store the correctness rates for this simulation
```

```r
Chime_error_rates[sim] <- 1-Chime_correct_rates # Replace with the correct variable
glmnet_error_rates[sim] <- 1-glmnet_correct_rates  # Replace with the correct variable
Sparse_lda_error_rates[sim] <- 1-Sparse_lda_correct_rates#Replace with the correct variable
}

# Calculate summary statistics for correctness rates (e.g., mean, standard deviation) for
    Chime, glm, and lda
chime_summary_stats <- summary(Chime_error_rates)
glm_summary_stats <- summary(glmnet_error_rates)
lda_summary_stats <- summary(Sparse_lda_error_rates)


# Create a data frame to combine the summary statistics
summary_table <- data.frame(
  Model = c("Chime", "glmnet", "Sparse_lda"),
  Mean = c(mean(Chime_error_rates), mean(glmnet_error_rates), mean(Sparse_lda_error_rates)),
  SD = c(sd(Chime_error_rates), sd(glmnet_error_rates), sd(Sparse_lda_error_rates))
)

# Print the summary table
print(summary_table)



Chime_error_rates_model1<-Chime_error_rates
Sparse_lda_error_rates_model1<-Sparse_lda_error_rates
High_dimension_logistic_error_rates_model1<-glmnet_error_rates
```