The learning process for and development of Mixture Discriminant Analysis for classification

Jaxon O'Connell - 200003753 - MSc Artificial Intelligence - Supervisor: Lei Fang

April 2022

1 Abstract

In this project, we set out to develop a Mixture Discriminant Analysis algorithm for classification, for which we sample the model parameters using Gibbs Sampling. We additionally implement simpler Linear and Quadratic discriminant analysis algorithms that we use to compare the performance of the algorithms we develop. We do this without the use of any Machine Learning packages, and instead implement all code and mathematics from scratch. To achieve this, we study the necessary knowledge of statistics from a basic level; we explain each of the steps involved in this learning process in this dissertation, up to a point where we can implement our algorithm from scratch.

2 Introduction

With the increased interest in Machine Learning and its application over recent years, it has become easier to implement these Machine Learning algorithms using programming packages such as SciPy for Python [7]. This move away from traditional methods means that mathematical models that would once be defined by hand can now be defined automatically [1]. However, the roots of Machine Learning is still firmly planted in mathematics, particularly linear algebra and statistics. Given the increase in automation and ease of use of Machine Learning in the present, it is no surprise that fewer ML practitioners have a complete understanding of the mathematics beneath the algorithms they are using. However, such an understanding is a powerful tool, enabling practitioners to develop more advanced Machine Learning algorithms and improve the performance of these algorithms.

The purpose of this project is to attempt to develop some of the statistical knowledge necessary in understanding many Machine learning techniques, while having very little background in statistics initially. We apply this knowledge by developing a generative classifier, specifically Mixture Discriminant Analysis with Mixture of Gaussians, using Bayesian statistics to sample the variables using Gibbs sampling. We do so without using any pre-built methods or packages, and instead implement the mathematics ourselves, breaking down the steps as we go, and comparing its performance to similar Discriminant Analysis algorithms. For this, we use the Julia programming language in the Pluto reactive notebook environment [4]. We use Kevin P. Murphys, Machine Learning : A Probabilistic Perspective book [3] as the teaching material and reference for this project.

We aim to develop a Mixture Discriminant Analysis algorithm as the steps to achieve this can be broken down and provide a path for us to follow, so we always have something to aim towards in the learning process. Since it is a generative classifier, we are able to apply Bayesian inference, an important part of statistics and its application to Machine Learning. Part of the MDA algorithm also involved unsupervised learning by using Gibbs sampling to cluster Mixture of Gaussians that become the subclasses of the MDA algorithm. Therefore, by implementing this algorithm, we are able to see how the statistics behind unsupervised learning, supervised learning and Bayesian inference work all in one model. Not to mention that there is not as much material on MDA compared to other ML algorithms, meaning this dissertation can also serve the purpose of learning new things about the algorithm, while also learning the necessary knowledge of statistics.

As for why we choose to investigate Bayesian statistics and inference rather than other types of statistics, since the Bayesian approach is concerned with uncertainty, it is much more common in Machine Learning applications. This is as it allows us to predict new values from data, and give a metric as to how sure we are about our estimated values. This is especially useful in cases where we cannot afford to be wrong about our predictions in applications such as medicine.

As each step of the learning process we describe builds on the previous, the code created at each learning step was often incorrect and was remodeled multiple times before being able to correctly classify data in the way that we wanted. This was all part of the learning process, which we try to capture in the *Background* section. However, this code is not used and the final piece of code is separate from these prior attempts and should work without going into detail about the learning process.

3 Background

To be able to design a Mixture Discriminant Analysis model, we first need to understand the fundamentals of Bayesian statistics, and build upon it sequentially, consistent with the learning process undertaken as part of the project. As such, the following section will provide an explanation in understanding each part necessary in developing a generative classifier with Mixture Discriminant Analysis.

3.1 Probability Theory

The field of statistics is often interpreted as one of either frequentist or Bayesian. In the frequentist approach, probabilities represent repeated trials and what we expect to happen given these trials; given we flip a coin many times, we would expect heads half the time. In Bayesian statistics, we concern ourselves with uncertainty; how likely is a coin to land on heads? Measuring this uncertainty gives us two major advantages over frequentist statistics. Firstly, because of the emphasis on uncertainty, the Bayesian interpretation of statistics can handle situations well where data is sparse or missing. Secondly, we can use this knowledge of uncertainty to predict the likelihood of future events. Both these properties of Bayesian statistics prove very useful in Machine Learning, as data is not always complete or plentiful, but we still want to use all useful data available to make useful predictions and inferences on our data. Doing any less would mean sacrificing the effectiveness of our models.

3.2 Bayes rule

Given that we have two events, A and B, and their probabilities of occurring, p(A) and p(B), we can use basic probability theory to define the probabilities of various events involving A and B. For example, if A is the probability of a die landing on 6, and B is the probability of a coin landing on heads, then we would say the joint probability of A and B is the probability that both of these events occur together, denoted by $p(A, B)$.

$$p(A, B) = p(A|B) * p(B) = p(B|A) * p(A) \tag{1}$$

Where $p(A|B)$ is the conditional probability of A given B. In other words, given that we have observed event B, such that we know with certainty it has occurred, then what is the probability that A also occurs; given that the coin landed on heads, what is the probability the die lands on 6.

$$p(A|B) = \frac{p(A, B)}{p(B)} \tag{2}$$

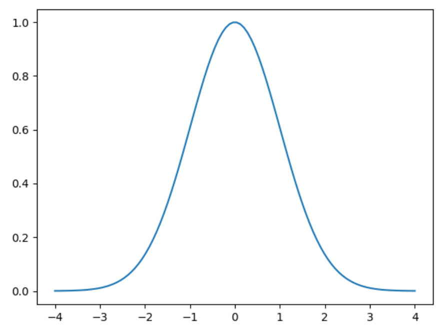
What if we want to find the probability that the die lands on 6 independently ($p(A)$), but we only have probabilities of the die landing on 6 and some value of the coin. If we have the joint probability of the die landing on 6 and the coin landing on heads, as well as the joint probability of the die landing on 6 and the coin landing on tails, then we can effectively cancel out the coin by summing across all joint values of the coin, since it can only land on heads or tails. Using this logic, we can calculate the marginal probability of event A as the sum of joint probabilities $p(A, B)$, across all possible events for B.

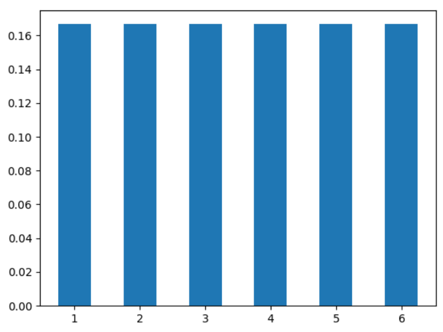
$$p(A) = \sum_b p(A, B) \tag{3}$$

Plugging equation 1 into the numerator of equation 2 in place of the joint probability, whilst also plugging equation 3 into the denominator for equation 1 gives us the following:

$$p(X = x|Y = y) = \frac{p(X = x)p(Y = y|X = x)}{\sum_{x'} p(X = x')p(Y = y|X = x')} \tag{4}$$

Figure 1: Examples of probability functions, generated by Figures.py


(a) Probability density function for a Normal Gaussian Distribution


(b) Probability mass function for a die roll

The above equation is Bayes rule, and is fundamental to probability theory, as well as this project. It allows us to better use the probabilities we have to infer related probabilities. Notice the change in notation also, as A and B becomes $X = x$ and $Y = y$. Here, X and Y are random variables that can take a range of values, where x and y are one set of those particular values. Quoting the example above, X could represent the number on the die, and X=6 represents it being 6 (similarly for Y and the coin). We use this equation later in defining class-conditional probabilities and posterior distributions of variables.

3.3 Gaussian Distributions

As we mentioned above, we can use Bayes rule to predict the probability of a variable having a particular value under certain conditions, and that these variables have a domain of values that each correspond to a probability. This mapping of values to probabilities is the probability distribution of that variable. If the variable is discrete and one dimensional, then we have a probability mass function, which can be visualised as a bar chart, where each bar represents a value and the height represents the probability (figure 1b). If the variable is continuous, then we have a probability density function, which can be visualised as the plotting of some continuous function (figure 1a).

In reality, most things can be seen as a variable with some underlying probability distribution that defines it, and with many other variables that it is conditional upon, even if only slightly. The chances it rains at a given time can be thought of as a probability distribution, conditional on other variables such as the wind speed, temperature and whether it rained yesterday, all of which could be modeled as variables with a probability distribution in the same way. The problem is we have no way of knowing what these underlying distributions might look like, and no way of knowing what variables it is dependent on. To address this, we attempt to model these true distributions by using various distributions that have a defined probability density function, or pdf, that take certain parameters and make various assumptions. One such distribution is the Gaussian distribution, also called the bell-curve distribution (figure 1a). In the univariate case, the pdf is defined as:

$$N(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{5}$$

This says that given data x, we can estimate its probability of occurring inside this distribution as long as we know the mean, μ, and variance, σ^2, of this distribution. By using the Gaussian distribution, we assume any data defined by it has maximum probability of occurring at the mean of the distribution, which decreases symmetrically either side of the mean the further out we go, and the rate that this falls is defined by the variance.

There are many reasons why the Gaussian distribution is so popular: since it only has two parameters which are easy to interpret, because it makes very few assumptions or because it has a simple mathematical form which is easy to implement. Among these reasons, it also follows the central limit theorem (figure [3] p.39), which says that for any population with sufficient values, if we sample the population and take a mean of some samples, if we do this multiple times and combine the mean samples together, then we will get a Gaussian distribution about the population mean. To explain this, think back to our example of rolling a 6-sided die. If we were to roll this die many times, we would find that the probabilities are all equally likely as in figure 1b. However, if we took a group of 5 die rolls and found the mean of those die rolls, and did this 1000 times and plotted a histogram of the results, we would notice that the histogram begins to form a Gaussian distribution centred around 3.5 as the mean. This would hold true for any population, which makes the Gaussian distribution a very simple yet powerful generalisation in most cases. We will also use this property later in defining the posterior distribution of the mean.

Since we need the mean and variance to model a distribution as Gaussian, we first need data from the distribution we want to model. Using this data, we find the mean by taking the average over all values, and the variance by measuring the average distance between the mean and each data point, squared. The more data points from the distribution we have, the more accurate these values and thus our approximation of the distribution.

$$\mu = \frac{1}{n}\sum_{n} x \tag{6}$$

$$\sigma^2 = \frac{1}{n}\sum_{n}(x-\mu)^2 \tag{7}$$

3.4 Multivariate Gaussian Distribution

When we discussed Gaussian distributions in the previous section, we looked at the univariate case, for when there is only one variable we wish to model. However, this alone is not very useful in practice, as rarely can a distribution be described by one variable alone. Luckily, it is simple to modify the univariate pdf into a multivariate one, as the concept is the same. We still use the mean and variance to define the multivariate Gaussian, however the mean becomes a vector, with each entry in the mean vector being the mean for a specific class or dimension. The variance is replaced with a covariance matrix, since variance is defined with respect to a single variable, and we now have multiple variables. The covariance matrix is a square matrix where the diagonal entries are the variances of each variable, and the other entries are the covariances between variables, which describe the degree to which the two variables are related (Murphys book [3] p.45). Applying this and adjusting the univariate Gaussian pdf to a multivariate one gives us:

$$N(x|\mu,|\Sigma|) = \frac{1}{\sqrt{(2\pi)^{D/2}|\Sigma|^{1/2}}} exp[-\frac{1}{2}(x-\mu)^T|\Sigma|^{-1}(x-\mu)] \tag{8}$$

Where D is the number of dimensions. Similarly as we did in the univariate case, we can estimate the mean and covariance parameters by finding their empirical values across the data we have. This is also known as the maximum likelihood estimate, or MLE (Murphys book [3] p.101):

$$\hat{\mu}_{mle} = \frac{1}{N}\sum_{i=1}^{N} x_i = \bar{x} \tag{9}$$

$$\hat{\Sigma}_{mle} = \frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})(x_i-\bar{x})^T \tag{10}$$

Using MLE can work well on simple datasets. However, as our distributions become more complicated, MLE becomes insufficient to estimate the parameters: It is heavily effected by outliers and has no measure of uncertainty, which can cause overfitting of incorrect parameters.

Using the multivariate Gaussian distribution on its own also has its flaws. Imagine a distribution is multimodal for any given dimension, such that it peaks more than once. For example, the height of a population would be multimodal, as it would peak at the average height of men, and again for the average height of women (assuming they cannot be differentiated).

We address both these problems by using a sampling method to generate estimates of the parameters (Gibbs Sampling), and using the multivariate Gaussian to define a new kind of distribution (Mixture of Gaussian's) respectively.

3.5 Bayesian Inference

When we have a group of samples for some population, we can find point estimates on these samples using the MLE. We call it MLE as we are estimating the true parameters by maximising the *likelihood* function. This likelihood distribution defines the probability of observing data D given parameters θ; how likely our data is to be observed, given the parameters. In frequentist

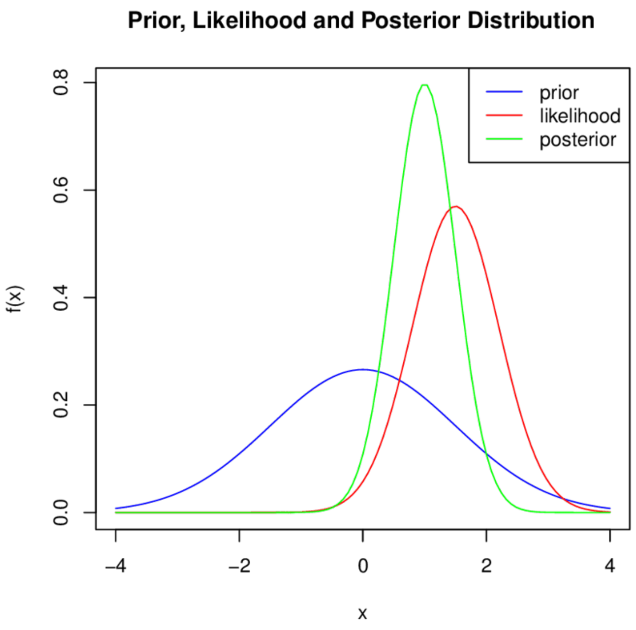


Figure 2: Prior, Likelihood and Posterior Trio for Gaussian. Credit to `https://www.researchgate.net/publication/322299460_Decision_Theory_and_Bayesian_Analysis`

statistics, we want to maximise this likelihood distribution, defined as $p(D|\theta)$. However, the Bayesian approach is to maximise the posterior distribution, or to find the value of θ that maximises θ given data D: $p(\theta|D)$. In doing so as we did with MLE in the likelihood function, we obtain the Maximum a posteriori estimate of θ, or the MAP estimate. The difference between the posterior and likelihood is that the posterior takes into account the prior distribution of the parameters θ, $p(\theta)$. We can see this by defining the posterior $p(\theta|D)$ using Bayes theorem.

$$p(\theta|D) = \frac{p(D|\theta) * p(\theta)}{p(D)} = \frac{likelihood * prior}{evidence} = posterior \tag{11}$$

Here, the evidence is the probability that the observed data is true, and acts as a normalisation factor that maintains the probability simplex that all probabilities must sum to 1.

The prior distribution is our initial belief about the parameters; it is the probability of an event before observing the data. In the case of distribution of height, if we initially believe men are taller than women, then we can implement a prior pdf that places higher density on the suspected peak of mens height than on women. If we know nothing about the parameters, we can use a uniform distribution, and the posterior becomes the likelihood. A well-fitting prior is especially useful when data is sparse, as it can shape our distribution towards the correct direction, as shown in figure 2. As we get more and more data, the effect of this prior will shrink as the likelihood becomes denser and the model nears closer to the true distribution. This prior is often subjective, however, as it is often up to us to define. It is not modelled through data like the likelihood function.

Another property of Bayesian inference is that both data and parameters are considered to be random variables. This means not only do we have a distribution across all possible data values, but also across all possible parameters. This allows us to use Bayes theorem with Bayesian inference interchangeably to infer the values we need. We can model the posterior distribution of the model parameters this way, where any sample from this distribution is a possible value of θ. Since θ is meant to represent all parameters of the distribution, in the case of multivariate Gaussian, $\theta = \mu, \Sigma$. Then the posterior becomes the joint probability of parameters conditional on feature vectors $X = x_1, ..., x_D$, where x_i is the input parameter in dimension i, such that $p(\mu, \Sigma|X)$ is the posterior distribution we want to model. Using sampling methods, we can then find potential values for our parameters, an alternative to using MLE or MAP to find them. This has the advantage over point estimates in that we can model uncertainty about the parameters, which is the goal of Bayesian statistics. In reality, we cannot be 100% certain about our parameters. How can we know the exact sufficient statistics that generate a complex distribution? Modelling the uncertainty around this parameter in the form of a distribution and sampling from it can prevent us from overfitting to a locally optimal solution in the case that our distribution is multimodal or we simply don't have enough data to describe the distribution well enough. Combining the ability of Bayesian inference to predict uncertainty and apply a prior distribution which is more effective with a sparse likelihood function, we can see how Bayesian inference works very well when we have small amounts of data or missing values.

Now assume that we have the likelihood and prior functions, it is not immediately obvious what that posterior function looks like. We can calculate this using Bayes theorem, as described in equation 11. However, having to calculate this every time can be inefficient, and the integration involved can be intractable. If we had a simplified form of the posterior, this process would be much faster. We can achieve this by having the likelihood and prior be of the same form, such that by multiplying the two functions together as in equation 11 allows us to add the exponentials together, meaning the posterior also shares the same simplified form. This is known as a conjugate prior and is very useful in making efficient Bayesian models.

3.6 Mixture of Gaussians

So far, we have described how to predict the underlying distribution that generates some data, based on a number of samples. Now, we extend this further. Suppose each sample belongs to one of K classes, where each class has its own distribution and we don't know which class each data point belongs to. Let z_i be the unknown class for sample i. We call this the hidden or latent variable. Since there are multiple distributions contained within the data, we can't just use the MLE or sample the posterior from the data we have to estimate the parameters, as this would assume that all data belongs to the same distribution, which would be wrong; the mean would be centered between the distributions. Instead, we need to consider each class as a separate distribution, yet we don't know which class each data point belong to. In this case, we have something similar to what is shown in figure 3a. Here, we see three groups of samples that are clearly from separate distributions. Our job now is to colour each sample correctly to its true class, as in figure 3b. This is known as clustering.

Since in this example we have three classes, we have three sets of parameters, one for each class: θ_1, θ_2 and θ_3. We have data D, which contains a mixture of samples from each class, as well as the hidden variable telling us which class we believe each sample belongs to, Z. Our goal is to find a good assignment of each sample to a corresponding z_i label, and once the samples have been separated, calculate the parameters of each class. Then we can join all these distributions together in a new distribution, known as a mixture model.

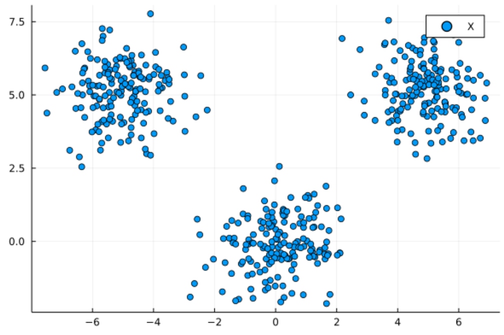
Given that all classes in figure 3b have a Multivariate Gaussian distribution, then we have $p(x_i|z_i = k) = N(x_i|\mu_k, \Sigma_k)$ as the likelihood for each class k. However, these classes are not connected, and so we cannot estimate the probability that any sample belongs to a given class in relation to others correctly, since one class may have a greater weight / more samples than the others which isn't reflected in the pdf; we can't assume that all are equal. In addition, if we sum the individual probabilities, we will break the probability simplex as we can get a probability more than one.

To treat this, we introduce a discrete prior, $p(z_i) = Cat(\pi)$. This can be thought of as the ratio of samples in class k to samples in all classes, for each k. This will sum to one, meaning multiplying each class' likelihood by its corresponding prior weight will give us the probability that a given sample belongs to that class. Doing this across all classes gives us a probability for each, which will sum to 1. We can treat these probabilities as parameters in a Bernouli distribution, which we can sample a value from to decide which class to assign each data to. Therefore, a Mixture of Gaussians has the form:

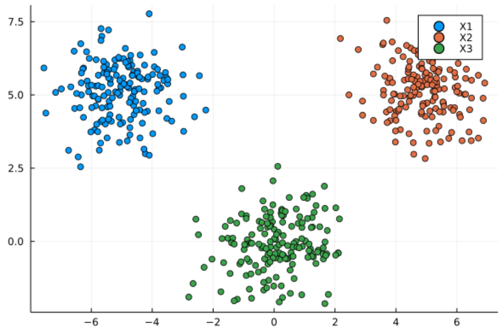
$$p(x_i|\theta) = \sum_{k=1}^{K} \pi_k N(x_i|\mu_k, \Sigma_k) \tag{12}$$

In summary, a Mixture of Gaussian's model contains a mixture of K multivariate Gaussian distributions that have their own parameters and weight, and any data point belonging to this distribution is assigned to only one distribution, but we can use the model to predict the probability that any sample belongs to any of the classes.

Figure 3: Samples from three Multivaraite Gaussian distributions with $\mu = [-5, 5], [0, 0], [5, 5]$ and shared covariance identity matrix, generated in Julia



(a) Unclustered example with z_i unknown



(b) Clustered example with true z_i shown

3.7 Gibbs Sampling

We previously mentioned that the Bayesian approach to parameter estimation is to model the posterior distribution and sample values for the parameter from it. There are many ways to go about this; we will focus on Gibbs sampling.

Gibbs sampling is a type of Monte Carlo Markov Chain (MCMC) inference. The Markov Chain refers to how the next event in the sequence depends only on the previous event, while Monte Carlo refers to using repeated random sampling to obtain values.

In accordance with Bayesian inference, we are interested in finding the joint probability of variables. However, this can involve calculating integrals that are very difficult to solve. To avoid this, Gibbs sampling samples each variable on its own, conditional on all other variables. That is, given variables x_1, x_2 and x_3, rather than sampling $p(x_1, x_2, x_3)$ directly to obtain a value for x_1, x_2 and x_3, we sample the value of x_1 conditional on x_2 and x_3. We then use this new sample of x_1 in calculating x_2 and x_3 conditional on the other variables, as follows ([3] p.840):

$$x_1^{s+1} \sim p(x_1 | x_2^s, x_3^s) \tag{13}$$

$$x_2^{s+1} \sim p(x_2 | x_1^{s+1}, x_3^s) \tag{14}$$

$$x_3^{s+1} \sim p(x_3 | x_1^{s+1}, x_2^{s+1}) \tag{15}$$

Where s is the sample number in the chain of iterations. Doing this process once will slightly improve the sampled values of the variables. However, with Gibbs, we repeat this process iteratively. Over time, our samples will improve. After enough iterations, our estimated samples for the variables will be close to the true parameters if Gibbs sampling is used correctly. It will take some time to reach this point from the initial sample however, meaning that a large amount of samples at the beginning are not useful to us. We call this process burn-in, and remove all samples below this burn-in point. The speed that the useful samples are burnt-in depends heavily on the starting sample, which could be picked at random or from an existing sample.

We are also not restricted to unimodal distributions, such that we could get samples that seem different but are both correct as they represent a different optimal value from the distribution, such as the mean height of men and women in the example we used earlier. This addresses the problem of overfitting that is present in many other machine learning techniques, as it is easy to get stuck in a local optima and become stuck there.

We can adapt this to other variables as long as we have the full conditional of each variable with respect to all other variables (the full conditional for a variable is the probability of that variable conditional on all others).

3.8 Gibbs Sampling for Mixture of Gaussians

In our description of Mixture of Gaussians, we explained how the distribution works, but we never explained how we can estimate the parameters. If we don't know which classes the samples belong to, then we cant predict the parameters for each Gaussian. If we don't have the parameters for each Gaussian, then we can't predict which samples may belong to it.

We are trying to predict the joint probability of all parameters $p(x, z, \pi, \mu, \Sigma)$ at the same time, but the number of unknown variables can cause us issues. We can apply Gibbs sampling to this joint distribution as we did above, but first we need the joint distribution in terms of conditional distribution. First, applying the chain rule of probability to the joint probability gives us equation 16 (defined in Murphys book [3] p.842):

$$p(x, z, \mu, \Sigma, \pi) = p(x|z, \mu, \Sigma)p(z|\pi)p(\pi) \prod_{k=1}^{K} p(\mu_k)p(\Sigma_k) \tag{16}$$

Now, we can simplify this by using the same prior for each mixture component, giving us the following conditional probabilities:

$$p(z_i = k|x_i, \mu, \Sigma, \pi) \tag{17}$$

$$p(\pi|z) \tag{18}$$

$$p(\mu_k|\Sigma_k, z, x) \tag{19}$$

$$p(\Sigma_k|\mu_k, z, x) \tag{20}$$

Next, we need to define these probabilities for a Mixture of Gaussians. We have already explained the process for defining equation 17 in the section on Mixture of Gaussians. As the probability of a given data point belonging to a class is the weighted probability of that class, which is a multivariate Gaussian, its definition is trivial:

$$p(z_i = k|x_i, \mu, \Sigma, \pi) \propto \pi_k N(x_i|\mu_k, \Sigma_k) \tag{21}$$

We use the proportional sign as we are excluding the denominator. However, as this acts as a normalisation constant, we can disregard it, since we do not care for the scale of this value, only the ratio between classes. In doing so we can speed up the process significantly.

Secondly, we consider the pdf $p(\pi|z)$, which is the mixture weights for the classes. As such, when we sample π, we have K probabilities, one for each class, and we consider these probabilities across all z values of each data point. Then, assuming that each event is independent, we can calculate the likelihood as $\prod_{k=1}^{K} \pi_k^{N_k}$, where N_k in the number of samples assigned the class k, and π_k is the mixture weight of class k. This describes a multinomial model, the same as we would use if we were rolling a K-sided die N_K times. However, we need a prior distribution in order to obtain the posterior. As we mentioned, we want to find a conjugate prior that is the same form as the likelihood to simplify things. Such a distribution exists, called the Dirichlet distribution, with the form $Dir(\theta|\alpha) = \frac{1}{B(\alpha)} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1} \mathbb{I}(x \epsilon S_K)$. Notice that both share the same form, such that we can add the exponents together to gain the posterior distribution. Before doing so, we define $\mathbb{I}$ as the indicator function, such that anything that meets the condition inside this function returns 1, otherwise 0. Given that we have $\mathbb{I}(z_i = k)$, this would be 1 if z_i belongs to class k, and 0 otherwise. Summing this across all data points for class k would give us N_k (the number of samples belonging to class k). Applying all of this, we have that the posterior conditional of class weights is as follows (ignoring the constants since we care for the ratio)

$$p(\pi|z) \propto p(z|\pi)p(\pi) = \prod_{k=1}^{K} \pi_k^{\sum_{i=1}^{N} \mathbb{I}(z_i = k)} \prod_{k=1}^{K} \pi_k^{\alpha_k} = Dir(\{\alpha_k + \sum_{i=1}^{N} \mathbb{I}(z_i = k)\}_{k=1}^{K}) \tag{22}$$

Where α_k is the prior hyperparameter of the mixture weights.

Computing the posterior for μ is similar to this process. We start with the likelihood $p(D|\mu) = N(\bar{x}|\mu, \frac{1}{N}\Sigma)$, where $\bar{x}$ is the empirical mean we obtain from the data D, and we estimate the probability of this empirical mean relative to μ, thus estimating the probability of the data that generated it. Now that we have the likelihood, we obtain the posterior after finding a conjugate prior, for which we can use another multivariate Gaussian distribution, $p(\mu) = N(\mu|m_0, V_0)$, where

m_0 and V_0 are the hyperparameters (mean and covariance respectively) of the prior of μ that we define. We are then left with a posterior distribution of the form $p(\mu|D,\Sigma) = N(\mu|m_N, V_N)$, where we must define the value of the posterior parameters, m_N and V_N.

First we consider V_N, which is the covariance of the posterior. This describes the spread of the distribution. We will focus on the inverse of this value, which is the precision, V_N^{-1}. The higher this value, the more precise our distribution is about its mean, suggesting we are more certain on our samples of μ. Since this posterior precision depends on the likelihood and prior precision, it should be a mixture of both terms. Given that as our data increases, our likelihood precision becomes more reliable, whilst the prior precision stays the same, we can express the precision as $V_N^{-1} = V_0^{-1} + N * \Sigma^{-1}$, which allows us to define our prior covariance in such a way that when we have a small amount of data, it takes a greater weight in the posterior covariance, but is overwhelmed by the likelihood covariance as we got more and more data, in line with Bayesian inference.

By the same logic for the posterior mean parameter, we can use this posterior covaraince value to define μ_N as a convex combination of the MLE and the prior mean (Murphys book [3] p.123): $\mu_N = \frac{\Sigma^{-1}(N\bar{x})+V_0^{-1}m_0}{V_N^{-1}}$, giving us the posterior mean as the following:

$$p(\mu_k|\Sigma_k, z, x) = N(\mu_k|m_k, V_k) \tag{23}$$

$$V_k^{-1} = V_0^{-1} + N_k\Sigma_k^{-1} \tag{24}$$

$$m_k = V_k(\Sigma_k^{-1}N_k\bar{x}_k + V_0^{-1}m_0) \tag{25}$$

Finally, we give the posterior for the covariance. The likelihood has the form $p(D|\mu,\Sigma) \propto |\Sigma|^{-\frac{N}{2}}exp(-\frac{1}{2}tr(S_\mu\Sigma^{-1}))$ if we ignore the constants in equation 8 and use the property of the trace function that $tr(A) = tr(C^{-1}AC)$. In doing so, we have changed the form of the likelihood to match that of an Inverse Wishart distribution which takes input in the size of Σ, allowing us to use an Inverse Wishart prior, $p(\Sigma) = IW(\Sigma|S_0^{-1}, v_0)$. Therefore, we have (see Murphys book [3] p.131):

$$p(D|\mu,\Sigma) = N(D|\mu,\Sigma) \propto |\Sigma|^{-\frac{N}{2}}exp(-\frac{1}{2}tr(S_\mu\Sigma^{-1})) \tag{26}$$

$$p(\Sigma) = IW(\Sigma|S_0^{-1}, v_0) \propto |\Sigma|^{-(v_0+D+1)/2}exp(-\frac{1}{2}tr(S_0\Sigma^{-1})) \tag{27}$$

$$p(\Sigma|D,\mu) = IW(\Sigma|S_N^{-1}, v_N), \quad v_N = v_0 + N, \quad S_N = S_0 + S_\mu \tag{28}$$

Where v is the degrees of freedom, which describes the datas ability to vary without breaking other constraints, and S is the posterior scatter matrix, a positive definite matrix similar to Σ, but defines the overall sum of data variance rather than the average.

With this, we have all the conditional probabilities needed to estimate the joint probability for a Mixture of Gaussians through Gibbs sampling. We now iteratively update our parameter values using these probabilities in an MCMC fashion.

4 Quadratic and Linear Discriminant Analysis

Suppose that for all possible classes y, we want to find the probability of data x coming specifically from class c. We have described this when we defined Bayes theorem and applied it to find the assignments of the hidden values, z_i, in equation 21. This is called the class-conditional density, defined as:

$$p(y = k|x) = \frac{\pi_k L(Y = k|x)}{\sum_{k'} \pi'_k L(Y = k'|x)} \tag{29}$$

We have seen and explained this equation previously; π_k is the class weight for class k and L(Y=k—x) is the likelihood of class k given the data x (excluding the prior), which is then normalised.

If we apply the pdf of a Multivariate Gaussian to the likelihood, we get the following:

$$p(y = k|x) = \frac{\pi_k |2\pi\Sigma_k|^{1/2} exp[-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1}(x - \mu_k)]}{\sum_{k'} \pi_{k'} |2\pi\Sigma_{k'}|^{1/2} exp[-\frac{1}{2}(x - \mu_{k'})^T \Sigma_{k'}^{-1}(x - \mu_{k'})]} \tag{30}$$

With this, we have the posterior over class labels, thresholding the results into a quadratic function, due to the quadratic form of the Multivariate Gaussian. What we are left with is the equation for Quadratic Discriminant Analysis; assuming that each class we wish to classify is Gaussian, we predict its probability given data x with equation 30. We do this for every class for each data point and classify the data point to the class which gives the highest probability.

To find the parameter values of μ_k and Σ_k, we calculate the MLE across our training data, splitting the data per class. With all this, we can use Quadratic Discriminant Analysis to classify data.

If we assume that the covariance matrix is shared across all classes, such that $\Sigma = \Sigma_k$, then we can expand the quadratic form of equation 30 into a linear one, giving us the form for Linear Discriminant Analysis (we say it is proportional to disregard the denominator as we care only for the ratio between classifications):

$$p(y = k|x) \propto \pi_k exp[\mu_k^T \Sigma^{-1} x - \frac{1}{2} x^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k] \tag{31}$$

Then we calculate the covariance by finding the MLE of the covariance for each class, averaging this across all classes and assuming each class has this covariance. This causes the decision boundary when using LDA to be linear, as oppose to QDA where it is quadratic.

Both QDA, LDA and MDA are forms of discriminant analysis. The difference is that we assume for MDA that each Gaussian is made from a further Mixture of Gaussians that we don't know the parameters for, so we use Gibbs sampling to cluster these sub-classes and assign each data a probability from each sub-class, and use the best estimate to classify.

5 Mixture Discriminant Analysis

Using what we have described so far, we are able to model different distributions based on some data, then sample values from these distributions. What we have described by separating the data by classes without labels in Gibbs sampling is unsupervised learning in the form of clustering. The supervised version of this clustering would be to perform classification in the case that labels are available, which is usually done using a discriminative classifier, which learns the conditional probability directly and uses this to differentiate between classes. When we have described algorithms like Gibbs sampling previously, our goal was to describe the joint probability. This is relevant to the approach that generative classifiers take; first learn the joint probability distribution and use Bayes theorem to transform this to the conditional probability. In doing so, we learn the underlying distribution, which we can use to generate new samples from, which is why we call it a generative classifier.

While generative classifiers are able to model the underlying distribution, which can be useful for testing and updating the model in the future with new data, it is not as fast as discriminative classifiers, which is why generative models are mostly used for unsupervised problems and discriminative models are mostly used for supervised models. Despite this, we define a generative classifier as such models rely on Bayes theorem and are based in probability theory, which is the purpose of this project to learn and investigate.

Imagine we have a Gibbs sampler working to sample a Multivariate Gaussian distribution, as we have described previously. We iteratively improve our estimated samples for each variable.

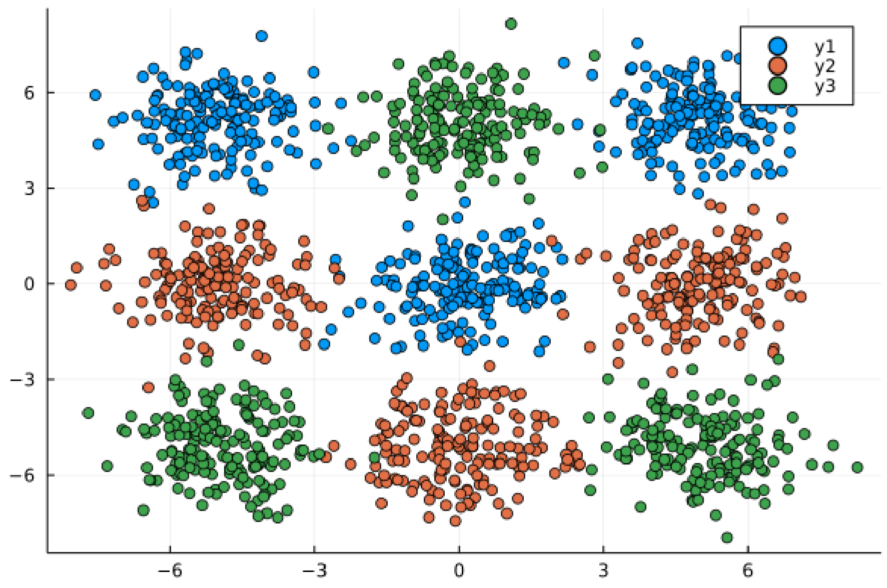


Figure 4: Mixture of Gaussians within a mixture of Gaussians, with the first mixture being labeled and the second being hidden. Plotted in Julia

In this case, we don't have the class labels, which is why this problem is unsupervised. However, suppose we did have the labels, we could split the data according to its class. Mixture discriminant analysis makes the assumption that for each class that is labeled, it is made up of further hidden subclasses, all of which are Multivariate Gaussian. Now, if we isolate each class of data, then we can use Gibbs sampling to sample across the subclasses. In this way, the sampling process remains the same, only we do this K times across K classes, each of which have R subclasses. We can think of this as sampling from a Mixture of Mixture of Gaussians, where the first Mixture is labeled, simplifying the process. This assumes that our data is Gaussian distributed, which can be very useful if this assumption is true. This is still useful if not as it gives a greater level of flexibility to our model. As the subclasses model themselves on the class they are based on, if it is made up of a singular multivariate Gaussian, then all of but one subclasses may 'switch off', setting their weights to zero, or all become equally weighted about the same parameter, in which case this is no different in principle than using Gaussian discriminant analysis.

Once we have sampled the parameters for each subclass, we can classify any new samples by evaluating their probability against each subclass from each class. We do this by taking all of the burnt-in parameter samples and calculating the log likelihood of each data point with these parameters, where the likelihood equation is the same as equation 21, only we replace the parameters with the subclass parameters, μ_{kr} and Σ_{kr}. We apply a logarithm across the likelihood as it prevents underflow, as multiplying many decimal probabilities together may lead to a value with more decimal digits than the variables in the Julia language can store. It also increase the speed of our calculations. We are able to do this as the logarithm function is monotonically increasing, so we will not lose any information regarding the ratio between likelihoods that we use to classify.

$$logLikelihood = log(\pi_{kr}) + log(N(x_i|\mu_{kr}, \Sigma_{kr})) \tag{32}$$

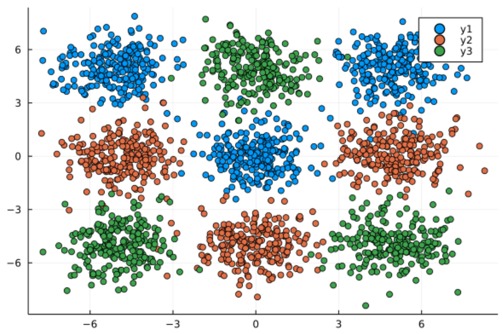
Once we have the log-likelihood for each data point belonging to each subclass with parameter values from each one of the burnt-in samples, we take the sum of the likelihoods across all burnt-in values and for each data point, the class-subclass combination that has the highest log-likelihood is classified. In summary, we classify each data point based on the class-subclass grouping that maximises the likelihood across all burnt-in sample values of the parameters.

This will then be assigned to a class and a subclass based on this probability, and we use the assignment of the class to classify this new sample. The increased flexibility of this model comes from letting a sample be one of many subclasses for each class. To visualise this, see figure 4, in which we can see the different classes have been labeled by different colours. However, we can also see that for each class, there is a further unlabeled Mixture of Gaussians, similar to figure 3a. This is the ideal case for the application of Mixture of Gaussians, as many other models would fail to see the second Mixture of Gaussians, where as MDA should recognise the parameters for each subclass and classify new data points belonging to the same distribution almost perfectly.

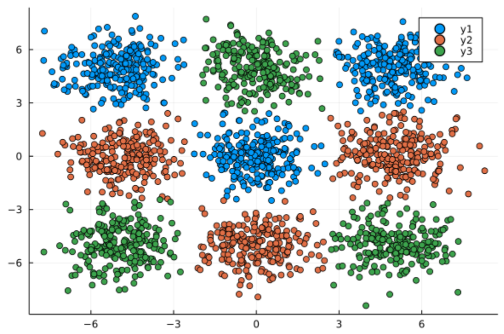
Since these discriminant analysis algorithms are so similar, we implement the QDA and LDA algorithms to perform the same experiments that we performed on MDA and compare the results. In the following sections, we first run experiments with MDA only and discuss the results. We then repeat the same experiments again, but also with QDA and LDA, and compare the results. We have listed these results below.

Accuracy from experiments comparing MDA, QDA and LDA			
Experiment	MDA	QDA	LDA
1 - (K=3, R=3)	96.1%	58.8%	48.4%
2 - (K=4, R=3)	91.9%	48%	38%
3 - (Single MVG per class)	94.9%	94.6%	94.65
4 - (Different covariances)	85.35%	56.15%	50.75%
5.1 - (High dimensional data (easy))	99.8%	97.2%	86.6%
5.2 - (High dimensional data (hard))	79%	76.2%	61.7%
6.1 - (Iris dataset)	100%	100%	100%
6.2 - (Pima dataset)	68.1%	72.5%	71%

Figure 5: Results of MDA on Multivariate Gaussians (experiment 1)



(a) Correct assignment of the test data



(b) Predicted assignment of the test data

6 Experiments on Mixture Discriminant Analysis

6.1 Experiment 1: Mixture of Gaussians with K = R = 3

Since our Mixture Discriminant analysis algorithm makes the assumption that each class has the form of a Mixture of Gaussians, the case in which the data matches our assumption should provide the best results. To test this, we define a dataset inspired by John A. Ramey's Overview on MDA [5], in which he shows MDA's ability to classify an easily distinguishable Mixture of Gaussians distribution within a Mixture of Gaussians, which other discriminant analysis algorithms failed at. We use a training set of N=10000 samples, and a test set of N=3000 samples to evaluate. All classes and subclasses have covariance equal to the identity matrix and different means (same parameters used in figure 4). There are K=3 classes and R=3 subclasses for each class. All classes are equally weighted. We perform 2000 iterations of Gibbs sampling, with a burn-in point at 1000 samples. As such, we have the following parameters:

$$\mu_1 = [(-5,5),(0,0),(5,5)], \quad \mu_2 = [(-5,0),(0,-5),(5,0)], \tag{33}$$

$$\mu_3 = [(-5,-5),(0,5),(5,-5)], \quad \pi = [1/3,1/3,1/3], \tag{34}$$

$$N\_train = 10000, \quad N\_test = 2000, \quad iters = 2000, \quad burnIn = 1000 \tag{35}$$

$$v_0 = dimension = 2, \quad m_0 = (0,0), \quad \alpha_0 = 50 \tag{36}$$

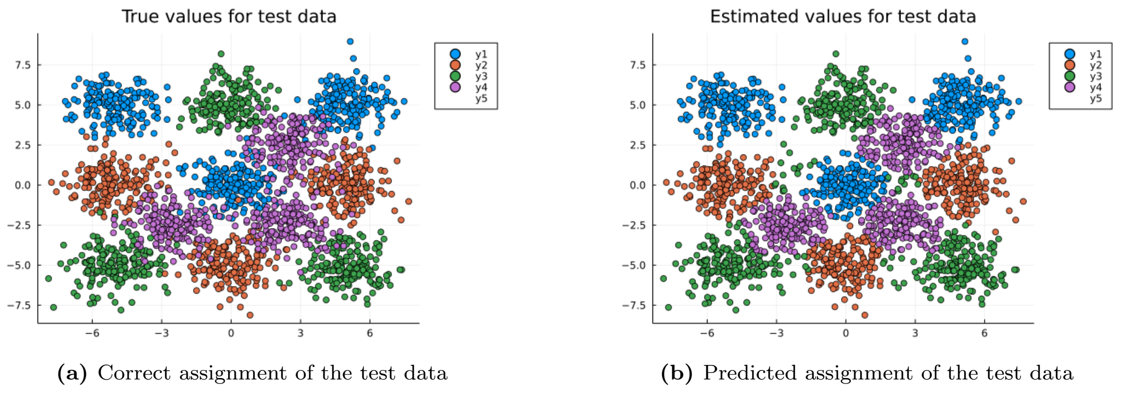
$$V_0 = S_0 = \Sigma_0 = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \tag{37}$$

We use these as the default parameters, and any changes made in further experiments will be defined (parameters are saved in our code as a preset, numbered accordingly to the experiment).

Once the algorithm was ran with these parameters, out of 2000 test samples, 1960 were classified correctly, giving us an accuracy of 98% (98.8% when tested again on a new dataset with the same distribution). Looking at the correct assignments of this test data in figure 5a, we can see that outliers of some classes overlap into the center of others. For example, there are blue samples from class one in the orange samples of class 2 in the middle right Gaussian. Looking over to figure 5b, we see the predictions for these values, and how these outliers disappear, which is likely our main cause of misclassifications. This is to be expected, as one of the weaknesses of the Gaussian distribution is its inability to handle outliers; we would have a hard time in creating an algorithm that correctly predicts outliers without misclassifying input from other classes, especially with this dataset.

This shows our hypothesis of MDA being highly accurate for a Mixture of Gaussians dataset to be correct, whilst highlighting a weakness of the model in classifying outliers.

Figure 6: Results of MDA on MVG with 4 classes and overlap (experiment 2)



(a) Correct assignment of the test data **(b)** Predicted assignment of the test data

6.2 Experiment 2: MoG with K = 4, R = 3 with more overlapping classes

Since the previous experiment focused on a Gaussian dataset that was easily distinguishable, we add another MoG class which overlaps with the other Gaussians. This is testing the algorithms ability to distinguish between more difficult overlapping datasets whilst also showing its dynamic nature, to be easily able to switch the number of classes. We expect the classification accuracy to be much lower in this case, as it is harder to tell which class a data point belongs to.

We update the values from the default parameters in equation 33 - 37 by setting K = 4 and setting the mean for this new class as $\mu_4 = [(-2.5, -2.5), (2.5, 2.5), (2.5, -2.5)]$ with the covariance matrix set to the identity matrix. We maintain that all classes have equal weight, so now $\pi_k = [1/4, 1/4, 1/4, 1/4]$.

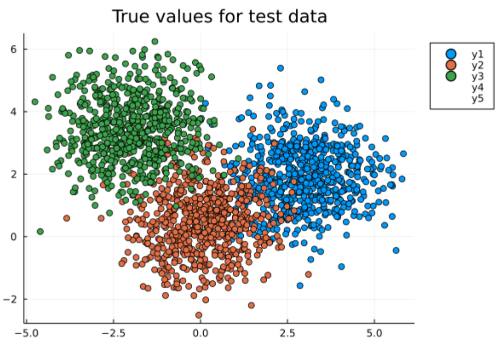
When we run the algorithm on this new dataset, we find it has an accuracy of 91.5%, which is significantly lower than experiment 1 as we expected, but considering the amount of overlap we find in this dataset, this is still a good performance. Furthermore, we find something very interesting when we investigate the results. In figure 6a, we see the correct classifications for the test values, in which we see class 4 overlapping. In figure 6b, we see the results of our classification. If we analyse this figure closely, we see that at around (3,0), there is a cluster of class 3 labels, despite this area being surrounded by samples from classes 1, 2 and 4. We also see that the labels for the top subclass of class 3 stretch down and left, when they shouldn't. Further investigation shows that the burnt-in parameter samples seem correct for classes 1, 2 and 4, but incorrect for subclasses 2 and 3 for class 3. The mean values for these subclasses are $\mu_{3,2} = (2.52, 0.28)$ and $\mu_{3,3} = (-2.70, 0.08)$ but the true values are $\mu_{3,2} = (5, -5)$ and $\mu_{3,3} = (-5, -5)$. These classes also have the following covariance matrices, when this should be an identity matrix:

$$\Sigma_{3,2} = \begin{matrix} 6.8 & -12.4 \\ -12.4 & 25.1 \end{matrix} \quad , \Sigma_{3,3} = \begin{matrix} 6.8 & 12.4 \\ 12.4 & 25.1 \end{matrix} \tag{38}$$

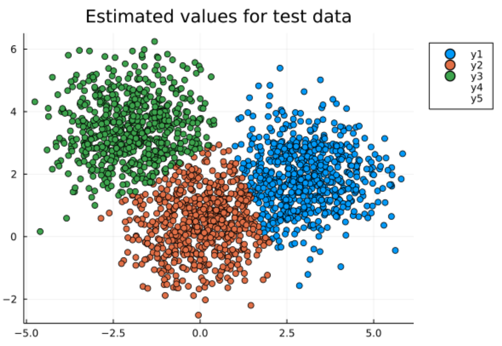
This explains what we see in figure 6b; the small cluster of green samples near (3,0) corresponds with class 3, subclass 2's mean. The larger cluster to the bottom right of it is classified as the same subclass due to the large covariance of this subclass, making it seem reasonable that such values would occur at a large distance from the mean, whereas all other classes have the covariance matrix equal to an identity matrix, meaning little deviation from their means.

When we replace the burnt-in parameter samples with the true parameters on the same test data, we get an accuracy of 93%. This tells us that while we sampled the parameters sub-optimally for subclass 2 and 3 of class 3, it is still a good assumption that was made, as it allows us to classify close to the optimal possible value. Taking into account things such as noise introduced by the prior, this solution is likely very close to the performance that would be achieved if we sampled the parameters almost perfectly. Because of this, it is likely that Gibbs sampling did not interpret this move towards the true values as a positive gain from the previous sample. This short-sightedness caused by the markov chain part of MCMC makes it a double-edged sword. It makes algorithms using it much faster and more efficient, but also means that less possibilities are explored, since if there is a loss in the performance when sampling a new value from the previous one, that this action will never be taken. So while some parameters are incorrect, the classification is still effective.

Figure 7: Results of MDA with a single MVG per class (experiment 3)



(a) Correct assignment of the test data



(b) Predicted assignment of the test data

6.3 Experiment 3: Setting each class input to a single Multivariate Gaussian

As we suggested earlier that if a single Multivariate Gaussian was used per class, either all but one of the subclasses would 'switch off' or all subclasses would have equal weight about the mean of the class, we should test this hypothesis. We can do this by setting R = 3, whilst using a separate R variable, dataR, to generate the test and training data. By setting dataR to 1, we generate only one MVG distribution per class. We set the mean of each class to (3, 2), (0, 0.5) and (-2, 3.5) respectively and we get the test data displayed in figure 7a. Classifying this data gives us figure 7b, where we can clearly notice linear decision boundaries between classes, with an accuracy of 93.3%. We expect to see this linear decision boundary, as the covariance matrices are all equal, so there should be no curvature in the decision boundary. Looking at the sampled parameters, the means for each subclass are all very close to the true value of the class mean, suggesting that *'all subclasses will have equal weight about the mean'* was correct.

6.4 Experiment 4: Changing the covariance matrix

Up until now, we have experimented with data which has a covariance matrix equal to the identity matrix, meaning that all data is clustered about its mean. We now generate a dataset with different values for the covariance matrix to see how Mixture Discriminant Analysis handles this. We use the default parameters and change the covariance matricies as follows:

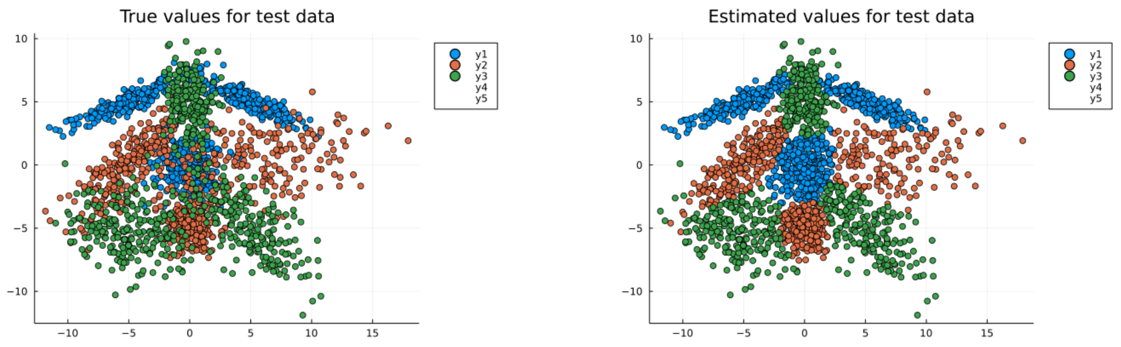
$$\Sigma_{1,1} = \begin{matrix} 5 & 2 \\ 2 & 1 \end{matrix} \qquad \Sigma_{1,2} = \begin{matrix} 2 & 0 \\ 0 & 3 \end{matrix} \qquad \Sigma_{1,3} = \begin{matrix} 5 & -2 \\ -2 & 1 \end{matrix} \qquad \Sigma_{2,1} = \begin{matrix} 5 & 4 \\ 4 & 5 \end{matrix} \tag{39}$$

$$\Sigma_{2,3} = \begin{matrix} 20 & 4 \\ 4 & 5 \end{matrix} \qquad \Sigma_{3,1} = \begin{matrix} 5 & -1 \\ -1 & 3 \end{matrix} \qquad \Sigma_{3,2} = \begin{matrix} 1 & 0 \\ 0 & 3 \end{matrix} \qquad \Sigma_{3,3} = \begin{matrix} 7 & -5 \\ -5 & 6 \end{matrix} \tag{40}$$

The resulting data is spread much more in different directions, with lots of overlapping datapoints (see figure 8a). After classification, we have an accuracy of 83.7%. When we perform classification on this test data again with the true parameters, we get an accuracy of 83.9%, so while the accuracy is relatively low, this is caused by the significant overlapping of classes which cannot be avoided. Hence the classification is still very close to the optimal accuracy for this dataset. Looking at the classified results in figure 8b, we see that there are hard boundaries between the classification of each subclass, where as in figure 8a, there is much more overlap. We see here again the difficulty that Gaussian distributions have with handling outliers.

When analysing the covariance samples, we see no issue in their values. All seem to be within a reasonable margin of what we would expect. For example, for $\Sigma_{1,1}$, it equals [[4.83, -1.95], [-1.95, 0.98]]. While there is slight deviation from the true values, this is to be expected due to the large overlapping of data, and explains why our accuracy is low.

Figure 8: Results of MDA with varying covariance matrices (experiment 4)



(a) Correct assignment of the test data **(b)** Predicted assignment of the test data

6.5 Experiment 5: High dimensional data

Up until now, all of our data has been 2-dimensional; this is done to visualise the data better. We will now show that Mixture Discriminant Analysis also works well on high dimensional data. We set dim = 5, meaning there are 5 input values that determine the class of each data sample. We set the mean values according to figure 9a. This gives us an accuracy of 100% when tested against 1000 values. Since we have more dimensions, this gives us more features to discriminate the samples against, making it easier to differentiate classes. While there may be overlapping of the mean for classes in one dimension, in another they may be very different. This difference makes it easier to classify correctly.

In figure 9b, the final sample value for the mean parameter is almost identical to the true values. As such, we provide a more difficult example. This time, we set the mean parameters as in equation 41 - 43.

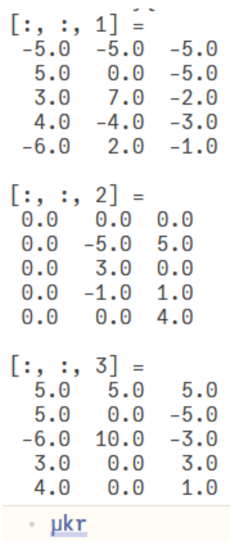
$$\mu_{1,1} = (-1, 1.5, 3, 0, -1) \qquad \mu_{1,2} = (0, 0, 0, 0, 0) \qquad \mu_{1,3} = (1, -1, -0.5, 2, 1) \tag{41}$$

$$\mu_{2,1} = (-1, 0, 2, -2, 2) \qquad \mu_{2,2} = (0, -0.5, 1.5, -0.5, 0) \qquad \mu_{2,3} = (0.5, 0, 1, 0, 0) \tag{42}$$

$$\mu_{3,1} = (1, -2, -1, -1, -1) \qquad \mu_{3,2} = (0, -1, 1.5, 1.2, 1.7) \qquad \mu_{3,3} = (0.3, -0.5, -2.3, 1.3, -2) \tag{43}$$

Here, the means are much closer together. If we were to use only 2 dimensions to attempt to classify the data, then this would not be plausible. For example, see figure 10, in which the training data for dimension 1 to all other dimensions is shown. In all of these figures, the data almost perfectly overlaps, with only some deviation at the borders. This would be impossible to achieve a reasonable classification for. Yet using all 5 dimensions, we are able to achieve an accuracy of 81.9%. When we use the true parameter values to classify the test data, we obtain a 82% accuracy. Thus, despite the data seeming impossible to classify in 2-dimensions, when we use all 5, we are able to achieve an accuracy rate only 0.1% lower than the best possible accuracy. Therefore, not only does our Mixture Discriminant Analysis algorithm work for higher dimensions, it provides better results than at lower dimensions, assuming the data is useful.This does come at a cost, however, as the higher the dimensions, the more memory is used and the longer the algorithm takes to run. This is one of the disadvantages of using a generative classifier.

Figure 9: Assigned and sampled mean values from experiment 5



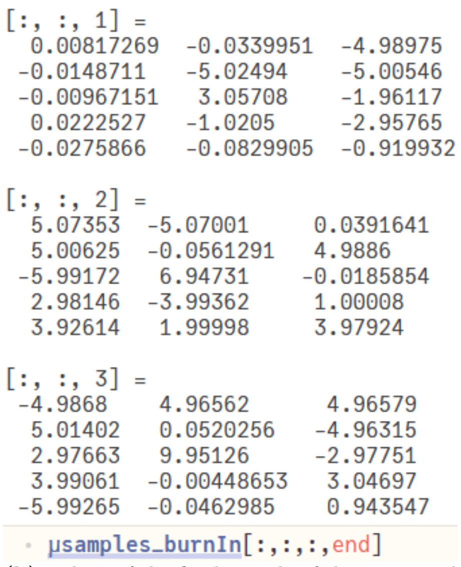
```
[:, :, 1] =
 -5.0  -5.0  -5.0
  5.0   0.0  -5.0
  3.0   7.0  -2.0
  4.0  -4.0  -3.0
 -6.0   2.0  -1.0

[:, :, 2] =
 0.0   0.0  0.0
 0.0  -5.0  5.0
 0.0   3.0  0.0
 0.0  -1.0  1.0
 0.0   0.0  4.0

[:, :, 3] =
  5.0   5.0   5.0
  5.0   0.0  -5.0
 -6.0  10.0  -3.0
  3.0   0.0   3.0
  4.0   0.0   1.0
```

```
μkr
```

(a) Mean values selected for data distribution



```
[:, :, 1] =
  0.00817269  -0.0339951   -4.98975
 -0.0148711   -5.02494     -5.00546
 -0.00967151   3.05708     -1.96117
  0.0222527   -1.0205      -2.95765
 -0.0275866   -0.0829905   -0.919932

[:, :, 2] =
  5.07353  -5.07001      0.0391641
  5.00625  -0.0561291    4.9886
 -5.99172   6.94731     -0.0185854
  2.98146  -3.99362      1.00008
  3.92614   1.99998      3.97924

[:, :, 3] =
 -4.9868    4.96562      4.96579
  5.01402   0.0520256   -4.96315
  2.97663   9.95126     -2.97751
  3.99061  -0.00448653   3.04697
 -5.99265  -0.0462985    0.943547
```

```
μsamples_burnIn[:,:,:,end]
```

(b) Values of the final sample of the estimated mean values

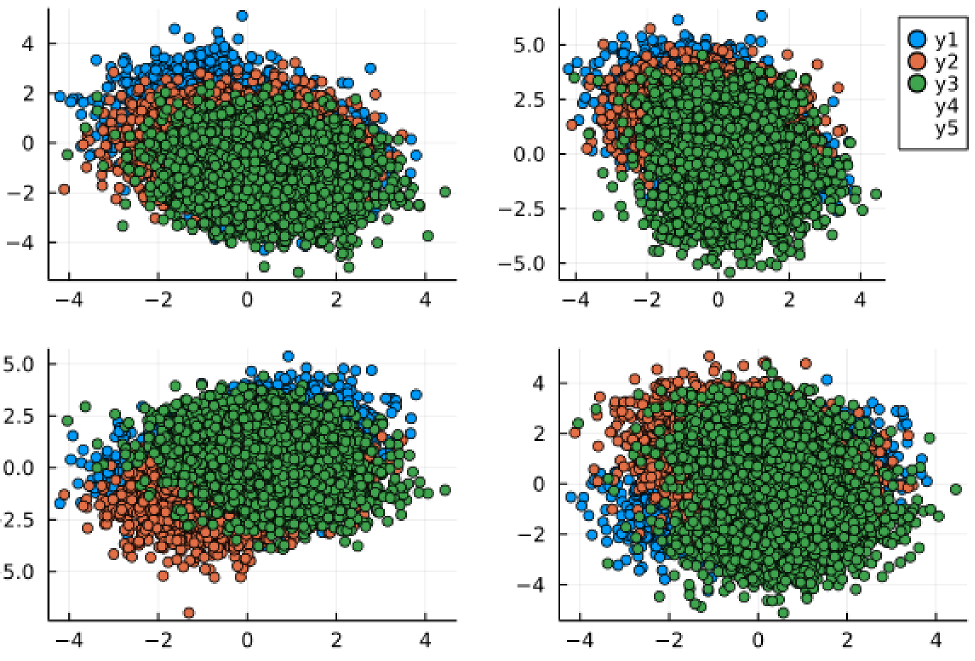


Figure 10: Scatterplot of training data from experiment 5 for dimensions 1 and 2 (top left), 1 and 3 (top right), 1 and 4 (bottom left), 1 and 5 (bottom right) using mean parameters from equation 41-43

6.6 Experiment 6: Testing MDA on the iris dataset

Up until now, we have used generated datasets that we know the distribution for. While these work well for the purpose of the project, which is to learn and explore the MDA algorithm, it is not a very practical application. We therefore apply the MDA algorithm to the iris dataset [2], which classifies 3 different types of flowers by 4 different inputs: sepal length, sepal width, petal length and petal width. It is a small dataset, made up of only 150 samples. We divide these samples into 120 for the training set (40 samples from each class) and 30 for the test set (10 samples from each class).

Since this is a real world dataset, we do not set the parameters ourselves. We therefore cannot know if the parameters we get are correct or if the underlying distribution that created the samples is a Mixture of Gaussians. All that matters is that we learn a distribution that correctly estimates the values of new samples from the data.

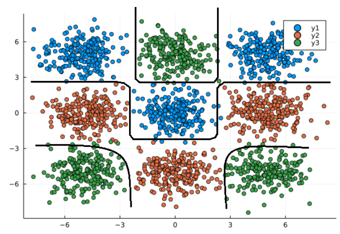
When we run MDA on the Iris dataset, we get an accuracy of 100%. While we mentioned that we cannot know the true parameters of the Iris distribution, we can perform MLE to get a reasonable estimate, as the dataset is quite simple. Doing this, we find the MLE for the mean to be $\mu_{mle} = (5.84, 3.06, 3.76, 1.20)$. Assuming that this value is a good approximation to the mean, then we might expect to see something similar from experiment 3, where each class consisted of a single Multivariate Gaussian. Hence we might expect all subclasses to have equal weight about μ_{mle}; This is not the case. We find that the mean of the first dimension for the last sample of our algorithm is:

$$\mu_{dim1} = \begin{matrix} 0.30 & 5.05 & -0.46 \\ -0.07 & 5.90 & 0.86 \\ 6.44 & 0.57 & -0.50 \end{matrix} \tag{44}$$

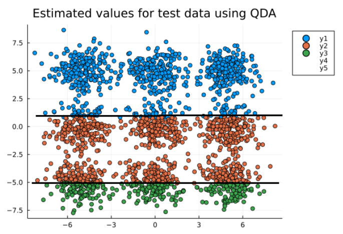
Comparing this to the MLE mean value of 5.84, there is no obvious connection between the two. While some values here resemble the mean, this resemblance is weak, and weaker still in other dimensions. Testing the averages across classes and subclasses doesn't provide any meaningful connections either. The same is true for the covariance. Despite this, MDA is able to approximate the distribution seemingly perfect for the data we have given it.

While the results obtained by MDA seem to be unpredictable in this case, they seem to be very effective at classifying various types of data.

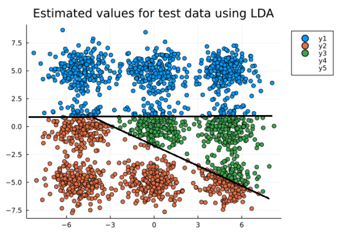
Figure 11: Results from experiment 1 with discriminant analysis algorithms, with decision boundaries marked



(a) Classified test data and decision boundary for MDA



(b) Classified test data and decision boundary for QDA



(c) Classified test data and decision boundary for LDA

7 Experiments on MDA, QDA and LDA with a comparison of results

7.1 Experiment 1 with QDA and LDA: Mixture of Gaussians with K = R = 3

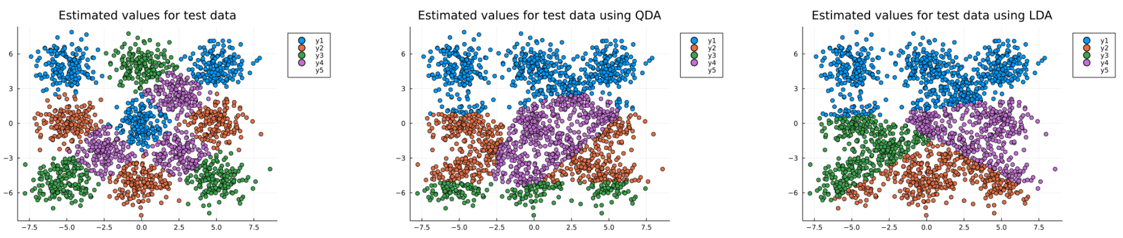
When we ran this experiment with Mixture Discriminant Analysis previously, we were able to obtain an accuracy of 98.8%. We run this experiment again with the same parameters and a newly generated dataset, to compare the performance of all discriminant analysis algorithms. We get the following results for accuracy of the classification:

- Mixture Discriminant Analysis: 96.1%
- Quadratic Discriminant Analysis: 58.8%
- Linear Discriminant Analysis: 48.4%

Given that this example was meant to represent the best conditions for using an MDA algorithm, for which QDA and LDA do not have the flexibility to classify, this is not surprising. Having only one Gaussian to classify 3, all of which are disjoint, is an impossible task. Still, QDA and LDA attempt to classify the test data the best they can, for which we see in figures 11b and 11c respectively. In these figures, we draw out the decision boundaries to understand the algorithms further. For MDA in figure 11a, we see that we are able to separate the data very well using three Mixture of Gaussians. However, in figure 11b, we see that the ability of the algorithm to separate the data is limited by the quadratic form of a Gaussian rather than a Mixture of Gaussians, which is why the decision boundary appears as two horizontal dividing lines. Separating the data in this way can be thought of logically. If we think of the data in figure 11b as three rows, then most of the data for class 1 (blue samples) is on the top row (2 blue clusters, 1 green), and is connected to another blue subclass in the row below, so we assign the top row and some of the middle row to class 1. The same applies for class 2 (orange samples). Most of the middle row is made from class 2 samples, so we assign most of the middle row to class 2. It is then connected to another cluster from class 2 in the bottom row, so some of the bottom row is also assigned to class 2. Finally, we are left with class 3 (green class). The bottom row is mostly green (in the training data), so we assign most of the bottom row to class 3. However, since the third subclass for class 3 is in the top row, we can't use the same technique as we did for the other 2 classes, so this is missed out from the classification of class 3 and it is given less weight as a result. For the possible decision boundaries that QDA has in this case, this strategy is the optimal for maximising classification accuracy.

We can see the same logic applies in the LDA case in figure 11c, only because the decision boundaries are linear, they cannot be disjoint like the quadratic decision boundaries in QDA. Therefore, the decision boundaries have an intersection point, which means that the model has less flexibility still and therefore a lower accuracy. This trend is exactly what is found in John Rameys *Overview of Mixture Discriminant Analysis* [5]

Figure 12: Results from experiment 2 with discriminant analysis algorithms



(a) Classified test data for MDA **(b)** Classified test data for QDA **(c)** Classified test data for LDA

7.2 Experiment 2 with QDA and LDA: MoG with K = 4, R = 3 with more overlapping classes

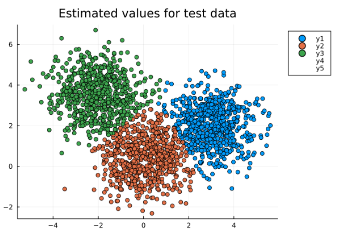
In experiment 2, we added a 4th class, which overlapped much more with the other classes. This overlapping makes it more difficult to distinguish between classes, so perhaps, QDA can perform better than it did in experiment 2. When we run the discriminant analysis, we find the accuracy is lower in all cases when compared to experiment 1:

- Mixture Discriminant Analysis:
  - Accuracy: 91.9%
- Quadratic Discriminant Analysis:
  - Accuracy: 48.0%
  - $\pi = [0.3215, 0.282, 0.1135, 0.283]$
  - $\mu = \begin{matrix} -0.009 & 0.008 & -0.018 & 0.798 \\ 3.335 & -1.672 & -1.648 & -0.854 \end{matrix}$
  - $\Sigma = \begin{bmatrix} 17.480 & -0.074 \\ -0.074 & 6.635 \end{bmatrix}, \begin{bmatrix} 17.555 & 0.094 \\ 0.094 & 6.578 \end{bmatrix}, \begin{bmatrix} 17.816 & 0.105 \\ 0.105 & 23.431 \end{bmatrix}, \begin{bmatrix} 6.494 & 2.719 \\ 2.719 & 6.598 \end{bmatrix}$
- Linear Discriminant Analysis:
  - Accuracy: 38.0%
  - $\pi = [0.3565, 0.2135, 0.2015, 0.2285]$
  - $\mu = \begin{matrix} -0.009 & 0.008 & -0.018 & 0.798 \\ 3.335 & -1.672 & -1.648 & -0.854 \end{matrix}$
  - $\Sigma = \begin{matrix} 14.836 & 0.711 \\ 0.711 & 10.810 \end{matrix}$

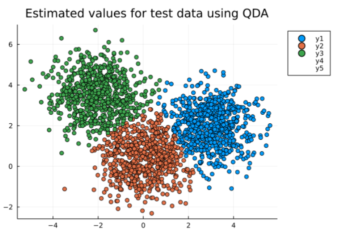
Therefore, adding further classes or increasing the overlap between data does not seem to make any of the discriminant algorithms more adaptable. We can also see the disadvantage to using LDA over QDA, as the covariance matrix for class 1,2 and 3 are similar in QDA, but the 4th class is significantly different. QDA is able to handle this as it doesn't assume Σ is shared. Since LDA takes the average of these values, it is swayed by class 4's covariance matrix acting like an outlier, and results in a value that doesn't accurately represent any of the classes. This shows that LDA is best suited when we can assume such a thing is true, otherwise it will effect our results.

We can visualise what is happening here between QDA and LDA to be similar to the difference between MDA and QDA. As the covariance in LDA is explained by multiple covariances in QDA, any covariance in QDA is broken down further into more covariance matrices in the subclasses of MDA.

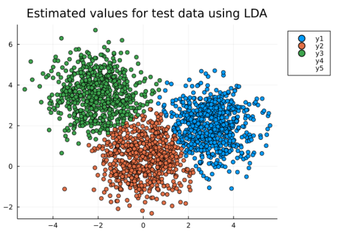
Figure 13: Results from experiment 3 with discriminant analysis algorithms



(a) Classified test data for MDA



(b) Classified test data for QDA



(c) Classified test data for LDA

7.3 Experiment 3 with QDA and LDA: Setting each class input to a single Multivariate Gaussian

Since we have previously tested on Mixture of Gaussians, for which QDA and LDA aren't able to classify, they should perform much better for the case where the data for each class is a single Multivariate Gaussian, since the formula of their class-conditional densities is of the form of a Multivariate Gaussian.

- Mixture Discriminant Analysis: 94.9%
- Quadratic Discriminant Analysis: 94.6%
- Linear Discriminant Analysis: 94.65%

We see that this is true, as the accuracy of all three discriminant analysis algorithms is within 94.6% - 95%. It is interesting that MDA is able to get a slightly better accuracy, even though the data is a single Multivariate Gaussian. If we were to predict this Gaussian correctly in QDA or LDA, we might expect this accuracy to be lower for MDA, as extra classes may add more uncertainty. Yet we see that this seems to benefit the model. One possibility is that with more subclasses per class, MDA is better able to predict the uncertainty in the data.

We also notice that LDA is slightly more accurate than QDA, which is opposite to what we have seen in the other two examples. This could be because the assumption that the covariance matrix is shared is true. This is unlikely, however, as the other experiments share this property. It is more likely that LDA works better than QDA in simple datasets where the covaraiance assumption is true.

7.4 Experiment 4 with QDA and LDA: Changing the covariance matrix

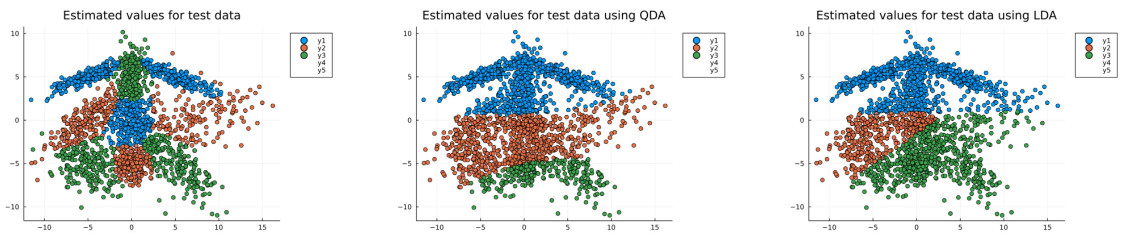
By changing the covariance matrix such that they are no longer sharing the identity matrix, we can see the impact that this has on QDA and LDA in respect to MDA, as LDA should perform poorly since it makes an incorrect assumption about the parameters being shared, and QDA should perform better than it as it can better model the different covariances. However, we expect both these models to perform poorly, since the dataset is complex, and we have shown already that QDA and LDA work best on simple datasets.

- Mixture Discriminant Analysis: 85.35%
- Quadratic Discriminant Analysis: 56.15%
- Linear Discriminant Analysis: 50.75%

Looking at the accuracy results, this appears to be true. MDA has an accuracy of 85.35%, which is the exact accuracy we receive when we use the true parameters that were used to generate the data, showing us that MDA found the optimal solution of assigning each of the 2000 test data points.

We notice that similar logic applies to QDA and LDA that we saw in experiment 2. Although both algorithms are incapable of fully classifying the test data, they use what they have to give an accurate solution. We see in figure 14c that the same linear decision boundary applies. In figure 14b, we see that QDA has made use of its ability to change its covariance to better classify the test data, which we can tell by the curvature in the decision boundaries. This allows QDA to obtain better accuracy than LDA did.

Figure 14: Results from experiment 4 with discriminant analysis algorithms



(a) Classified test data for MDA **(b)** Classified test data for QDA **(c)** Classified test data for LDA

7.5 Experiment 5 with QDA and LDA: High-dimensional data

When we initially performed this experiment on QDA, we split it in two parts. The first was classifying data that has significantly different means, making it easy for MDA to classify the data perfectly. We then used a dataset with very similar means to provide a more challenging and useful experiment. We replicate both of those experiments here. The first part has the following accuracy:

- Mixture Discriminant Analysis: 99.8%
- Quadratic Discriminant Analysis: 97.2%
- Linear Discriminant Analysis: 86.6%

Despite the data for each class being a Mixture of Gaussians, and the class-conditional density of the QDA algorithm being a single Multivariate Gaussian, it was able to achieve a high accuracy of 97.2%. LDA was also able to achieve a good accuracy of 86.6%, but this is significantly less than QDA, despite all covaraince matrices being shared in this case. Thus, the additional accuracy in QDA compared to LDA comes from QDA using its covariance to differentiate different subclasses of data by dimension, as there is significant variation between class covariances in the case of QDA, which is not possible in LDA.

Trying this experiment on the more difficult dataset gives us the following accuracy:

- Mixture Discriminant Analysis: 79%
- Quadratic Discriminant Analysis: 76.2%
- Linear Discriminant Analysis: 61.7%

After checking the accuracy when using the true parameters used to generate the data, we see it has an accuracy of 78.4% on the same test data. Here, MDA manages to outperform the true distribution of data. This is possible as there is much overlap in the data, as shown in the previous experiments with this data. MDA is likely able to exploit patterns in the overlapping of the data which is not considered in the true parameters. QDA is nearly as impressive also, despite modelling a Mixture of Gaussians with a single Gaussian. LDA is not as effective, however. We can see that at higher dimensions, QDA is able to adapt well to the data to gain an effective classification, although LDA still struggles.

7.6 Experiment 6 with QDA and LDA: Testing on real dataset

When we ran all three discriminant analysis algorithms on the iris dataset, we found that all were able to classify the data with 100% accuracy. We therefore extend the experiment to a more difficult dataset.

We will use the Pima dataset [6], a dataset on a population on Indian women older than 21 years old that were tested for diabetes. It contains 7 input variables such as age, bmi, blood-pressure and glucose, and one output variable: whether they have diabetes or not. Our goal is to classify these values as accurately as possible. We perform a 390:138 train-test split on the 528 samples. Here, K = 2 since the classification is binary. When we classify the test values, we find something interesting:

- Mixture Discriminant Analysis: 68.1%

- Quadratic Discriminant Analysis: 72.5%
- Linear Discriminant Analysis: 71.0%

We see that for the first time, MDA had an accuracy lower than QDA and LDA. There may be four reasons for this. Firstly, higher dimensions may cause MDA difficulty. Secondly, MDA may perform poorly in binary classification problems as it may over-complicate things. Thirdly, the samples in Gibbs sampling may not have burnt-in yet. Finally, the distribution of the data is not Gaussian and our assumption was incorrect.

Addressing the first point, we have already tested data at higher dimensions in experiment 5 and found MDA works well here. Addressing the third point, we run MDA again for 10000 iterations of Gibbs sampling, 5 times as many, and place the cut-off point at 5000; the result we get for the accuracy ends up worse, at 65.2%. As for the final point, if this were the case, QDA and LDA would have performed even worse since they are based off of a Gaussian distribution also. Therefore, it must be the case that in binary classification where we use K=2, MDA performs poorly.

8 Conclusion and Future Work

To conclude, we have been able to summarise the learning that has been undertaken for the purposes of this project, condensing only the important parts. This explains the journey undertaken that was necessary to understand statistics for the purpose of its application in Machine Learning. We have developed this knowledge up to a point where we are able to create a Mixture Discriminant Analysis algorithm from scratch, starting with only a basic knowledge of statistics. Hopefully, this dissertation has provided sufficient resources in order to comprehend the learning process involved in such a task, and has gone onto explain this in a practical application through various experiments performed on the code that has been developed along-side this newfound understanding of the field. Despite this, the process has shown statistics to be a large and complex field, for which this dissertation only touches the surface of.

In testing the code that was generated, we discovered numerous things about the Mixture Discriminant Analysis algorithm. When it is modelled correctly and the assumptions made by the model are satisfied, it can be a very powerful model. It has the ability to use its subclass distributions to approximate well to most distributions, even when these are not strictly Gaussian. As a result of fundamentally being a Gaussian distribution, it suffers from the same problems as Gaussian distributions, such as not being able to handle outliers well. However, this weakness is not major. It also seems to perform poorly in binary classification problems, and is best to be avoided with simple datasets where it can be approximated sufficiently using LDA or QDA instead. This is due to the amount of resources that are required in MDA.

Since we use Gibbs sampling to predict the parameters of the subclasses, we store these values over all iterations. We then use the burnt-in samples to classify the data. This requires all these sample values to be stored. If we have a large amount of iterations, and a high number of classes and subclasses, the amount of memory required to store this can be huge. As a result, MDA is powerful but costly also.

There are further optimisations that could be made to this current version of MDA in the future. Adding functionality to read and write these samples to and from memory would allow for MDA to be used on huge, very complex datasets, although this would be very computationally expensive.

We could also implement non-parametric methods alongside MDA such that we do not have to define the values of K and R. Instead, such an algorithm could pick its own K and R values for what it believes to be optimal values.

To summarise, QDA is a very powerful and expensive generative classifier. For simple datasets it may be best to use simpler models, and is best suited to classifications in which the output is not binary. It works well with high dimensional input for which it can model its subclasses to match patterns present in the data across dimensions.

References

[1] Keith D. Foote. *A Brief History of Machine Learning.* 2021. URL: https://www.dataversity.net/a-brief-history-of-machine-learning/ (visited on 05/04/2022).

[2] *Iris Dataset.* URL: https://gist.github.com/curran/a08a1080b88344b0c8a7#file-iris-csv.

[3] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* 1st ed. The MIT Press, 2012.

[4] *Pluto reactive notebook package for the Julia Programming Language.* URL: https://www.juliapackages.com/p/pluto.

[5] John A. Ramey. *An Overview of Mixture Discriminant Analysis.* 2013. URL: https://github.com/ramhiser/tech-reports/blob/master/mixture-discrim-analysis/mixture-discriminant-analysis.pdf.

[6] RDatasets. *PIMA Dataset.* URL: https://r-data.pmagunia.com/dataset/r-dataset-package-mass-pimatr.

[7] *SciPy.* URL: https://scipy.org/ (visited on 05/04/2022).