Using Gaussian Mixture Models to Deconstruct Data With Unusual Distributions

Cassandra Jin [*]
Department of Mathematics and Statistics, Amherst College

December 9, 2023

Abstract

This paper explores the fundamental concepts that underpin Gaussian mixture models (GMM), offering a comprehensive overview of their mathematical formulation, the Expectation-Maximization (EM) algorithm employed for parameter estimation, and the principles that govern their probabilistic nature. The GMM is a generalization of k-means clustering and is used for decomposing multimodal data into multiple component distributions, namely Gaussian ones. This paper will thus include an overview of the Gaussian distribution and go deeper into the utility of covariance matrices. It will demonstrate how to implement GMM functions in R and how these apply to real-world applications where GMMs have demonstrated remarkable efficacy, illustrating their relevance and versatility across diverse domains.

Keywords: Probability density function; Expectation maximization algorithm; Multimodality; Pattern recognition; Covariance; k-means clustering

[*]The author gratefully acknowledges Professor Amy Wagaman for her enriching guidance and patient support in STAT-495. Professor Wagaman's instruction throughout the course has allowed for an incredible amount of technical and personal growth, and this project is has been a wonderful opportunity to apply my skills as well as to learn more invaluable ones.

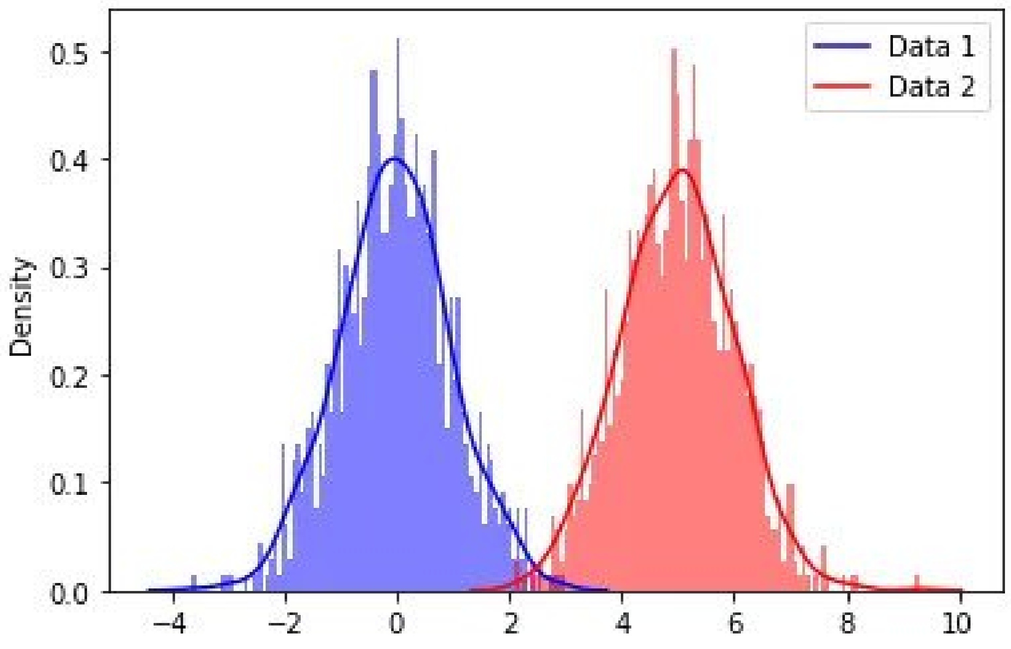


Figure 1: A multimodal distribution, where each peak represents a different Gaussian distribution or the cluster in the dataset (Ravihara, 2023).

1 Introduction

In the landscape of machine learning, pattern recognition, and data analysis, the objective and the challenge often lie in capturing intricate data distributions as effectively as possible. It may be convenient to assign a single distribution that seems to fit well enough to a large data set, but it is much more insightful to identify patterns and clusters within the data. Clustering algorithms such as k-means clustering provide solutions to finding the patterns within observations, but such traditional methods still have limitations when it comes to identifying clusters of different shapes and sizes. For working with a diverse data set, the Gaussian mixture model (GMM) employs a combination of multiple Gaussian distributions and, as a result, excels at capturing the various latent structures within the data [Kumar, 2022].

We understand k-means clustering to be a method that partitions observations into

k clusters in which each observation belongs to the cluster with the nearest mean. Like k-means clustering, the GMM is a type of machine learning algorithm, but instead, it groups data into different categories based on any underlying probability distributions. By identifying and merging multiple Gaussian components, the GMM achieves a parametric probability density function represented as a weighted sum of the densities corresponding to each Gaussian component [Reynolds et al., 2009]. The parameters in the method are estimated from training data using the iterative expectation-maximization (EM) algorithm, and in maximizing likelihood, GMM allows a representation of data that goes beyond the limitations of single-distribution models [Bouguila and Fan, 2020]. The method's ability to model a wide array of distributions (multimodal, non-Gaussian, etc.) has made GMMs essential in various domains such as computer vision, speech recognition, anomaly detection, finance, marketing, and more.

This paper aims to guide readers through the fundamental concepts that underpin GMMs, offering a comprehensive overview of their mathematical formulation, the Expectation-Maximization (EM) algorithm employed for parameter estimation, and the principles that govern their probabilistic nature. Furthermore, we will explore real-world applications where GMMs have demonstrated remarkable efficacy, illustrating their relevance and versatility across diverse domains.

2 Gaussian Mixture Models (GMMs)

2.1 The Gaussian Distribution

The most basic concept in probability theory and in statistics is the random variable. A random variable can be understood as a mapping from a random experiment to a variable, and depending on the nature of the experiment and the design of the mapping, a random variable can take on discrete values, continuous values, or a mix of discrete and continuous values. One possible distribution that the resulting values may follow is the singular Gaussian distribution, which has probability density function (PDF)

$$R_X(t,s) = E[A\cos\omega t \cdot A\cos\omega s] \tag{1}$$

$$= A^2 E[\cos\omega t \cos\omega s] \tag{2}$$

$$p(x) = \frac{1}{(2\pi)^{1/2}\sigma} exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2) \doteq N(x;\mu,\sigma^2), \tag{3}$$

or $x \sim N(\mu, \sigma^2)$, where x is a continuous random variable obeying a normal distribution with mean μ and variance σ^2.

The Gaussian distribution is commonly used in many engineering and science disciplines due to its highly desirable computational properties, but also from its ability to approximate many naturally occurring real-world distributions, due to the law of large numbers (Ravihara, 2023).

Expanding this definition to incorporate the weights of component distributions, c_m, we have that a continuous random variable has a Gaussian-mixture distribution if its PDF is specified by

$$p(x) = \sum_{m=1}^{M} \frac{c_m}{(2\pi)^{1/2}\sigma_m} exp(-\frac{1}{2}(\frac{x-\mu_m}{\sigma_m})^2) = \sum_{m=1}^{M} c_m N(\mu_m, \sigma_m^2), \tag{4}$$

where $\infty < x < \infty$, $\sigma_m > 0$, and $c_m > 0$ [Yu et al., 2015].

These positive mixture weights must sum to 1: $\sum_{m=1}^{M} c_m = 1$, i.e. the component distributions combine to describe the full mixture model.

The most obvious property of Gaussian mixture distribution is its multimodality ($M > 1$ in Equation 4), in contrast to the unimodal property of the Gaussian distribution where $M = 1$. This makes it possible for a GMM to adequately describe many types of physical data exhibiting multimodality, which are poorly suited for a single Gaussian distribution.

Based on Equation 4, the expectation of a random variable x with the mixture Gaussian PDF is $E(x) = \sum_{m=1}^{M} c_m \mu_m$. But unlike a singular, unimodal Gaussian distribution, this simple summary statistic is not very informative unless all the component means, μ_m for $m = 1, ..., M$, in the Gaussian-mixture distribution are close to each other.

Instead, we use the multivariate generalization of the mixture Gaussian distribution, which has joint PDF

$$p(x) = \sum_{m=1}^{M} \frac{c_m}{(2\pi)^{D/2}|\Sigma_m|^{1/2}} exp(-\frac{1}{2}(x-\mu_m)^T \Sigma_m^{-1}(x-\mu_m)) = \sum_{m=1}^{M} c_m N(x; \mu_m, \Sigma_m), \quad (5)$$

where D is random variable x's dimensionality, Σ_m holds the covariance matrices, and $c_m > 0$ still.

In using the multivariate mixture Gaussian distribution of Equation 4, if D is large (this depends on the context), then the use of full (nondiagonal) covariance matrices (Σ_m) would involve a large number of parameters [Bouguila and Fan, 2020]. To reduce the number of parameters, one can instead use diagonal covariance matrices for Σ_m. Alternatively, when M is large, one can also constrain all covariance matrices to be the same. The advantage of using diagonal covariance matrices is significant simplification of computations needed for the applications of the Gaussian-mixture distributions [Wan et al., 2019].

2.2 Parameter Estimation

The Gaussian-mixture distributions discussed above contain a set of parameters. In the multivariate case of Equation 4, the parameter set consists of $\Theta = \{c_m, \mu_m, \Sigma_m\}$. The parameter estimation problem is motivated by wanting to determine the values of these parameters from data set that we assume to be drawn from the Gaussian-mixture distribution.

It is common to think of Gaussian mixture modeling and the related parameter estimation as a missing data problem. We want to "learn" appropriate parameters for the distribution, with the connection to the data points being represented as their membership in the individual Gaussian distributions.

Here, we focus on the expectation maximization (EM) algorithm as the maximum likelihood method of choice for parameter estimation of the Gaussian-mixture distribution. The EM algorithm is the most popular technique used to estimate the parameters of a mixture given a fixed number of mixture components, and it can be used to compute the parameters of any parametric mixture distribution. The EM algorithm finds the maximum likelihood of a model through two iterative steps: an expectation or E-step and a maximization or M-step. It alternates between performing an expectation step and a maximization step to achieve maximum likelihood until a certain stop condition is satisfied or a specified number of iterations is completed [Wan et al., 2019].

The EM algorithm is especially fitting to use for GMM, as we can express the parameters in their closed forms and iterate through the M-step [Do and Batzoglou, 2008]. Given the current estimate for the parameters, denoted by just a superscript in the closed forms, the conditional probability for a given observation $x^{(t)}$ generated from a mixture component m is determined for each data sample point at $t = 1, ..., N$, where N is the sample size. The parameters then update such that the new component weights correspond to the average conditional probability, and each component mean and covariance is the component specific weighted average of the mean and covariance of the entire sample set.

A particular advantage of the EM algorithm is that each successive iteration will not decrease the likelihood, a property not shared by many other maximization techniques [Reynolds et al., 2009]. Additionally, the EM algorithm naturally embeds constraints on the probability vector while avoiding extra computational costs to check and maintain appropriate values [Yu et al., 2015]. On the other hand, the EM algorithm can be too sensitive to initial values and may falsely identify local maxima. This problem can be addressed by evaluating at several initial points in the parameter space, but this may become computationally costly and undo the advantageous efficiency of the algorithm.

2.3 Summary

Overall, there are three key steps to using Gaussian mixture models for clustering:

1. Determine a covariance matrix that defines how each Gaussian is related to one another. The more similar two Gaussian component distributions are, the closer their means will be and vice versa if they are far away from each other in terms of similarity. A GMM can have a covariance matrix that is diagonal or symmetric (non-diagonal).

2. Decide the number of Gaussian component distributions, which dictates how many clusters are present in the model.

3. Select the hyperparameters that define how the data are optimally separated using GMMs.

3 Connections and Comparisons to Other Models

Compared with traditional k-means clustering algorithm, GMM has two primary advantages. Firstly, it takes into account covariance, which determines the shape of the distribution, so it can fit well when the distribution of the data presents different shapes. Secondly, while k-means results in hard clustering, GMM is able to perform a more flexible clustering on data, called soft clustering ("Cluster Using Gaussian Mixture Model", 2023). Soft clustering methods assign a score to each data point in a cluster to represent the association strength of the data point to the cluster. As opposed to how hard clustering (k-means) outputs a specific category for each data point, soft clustering (GMM) is flexible in that it can assign a data point to more than one cluster and it outputs the probability of each specific category the data points belong to. Thus, the GMM yields an estimated uncertainty measure of the degree of association between data points and specific categories [Huang and Gou, 2023].

As a trade-off of GMMs being more flexible than k-means, they can be more difficult to train. K-means is typically faster to converge and more accurate, so it may be preferred in cases where the runtime is an important consideration or generally when the data set is large and the clusters are well-separated. On the other hand, GMMs are more accurate when the data set is small or the clusters are not well-separated [Kumar, 2022].

We know that the EM algorithm enables effective parameter estimation in probabilistic models with incomplete data. By implementing this algorithm, Gaussian mixture models can handle missing data, whereas K-means cannot. This difference makes GMMs more effective in certain applications where the data contains considerable noise or is not well-defined.

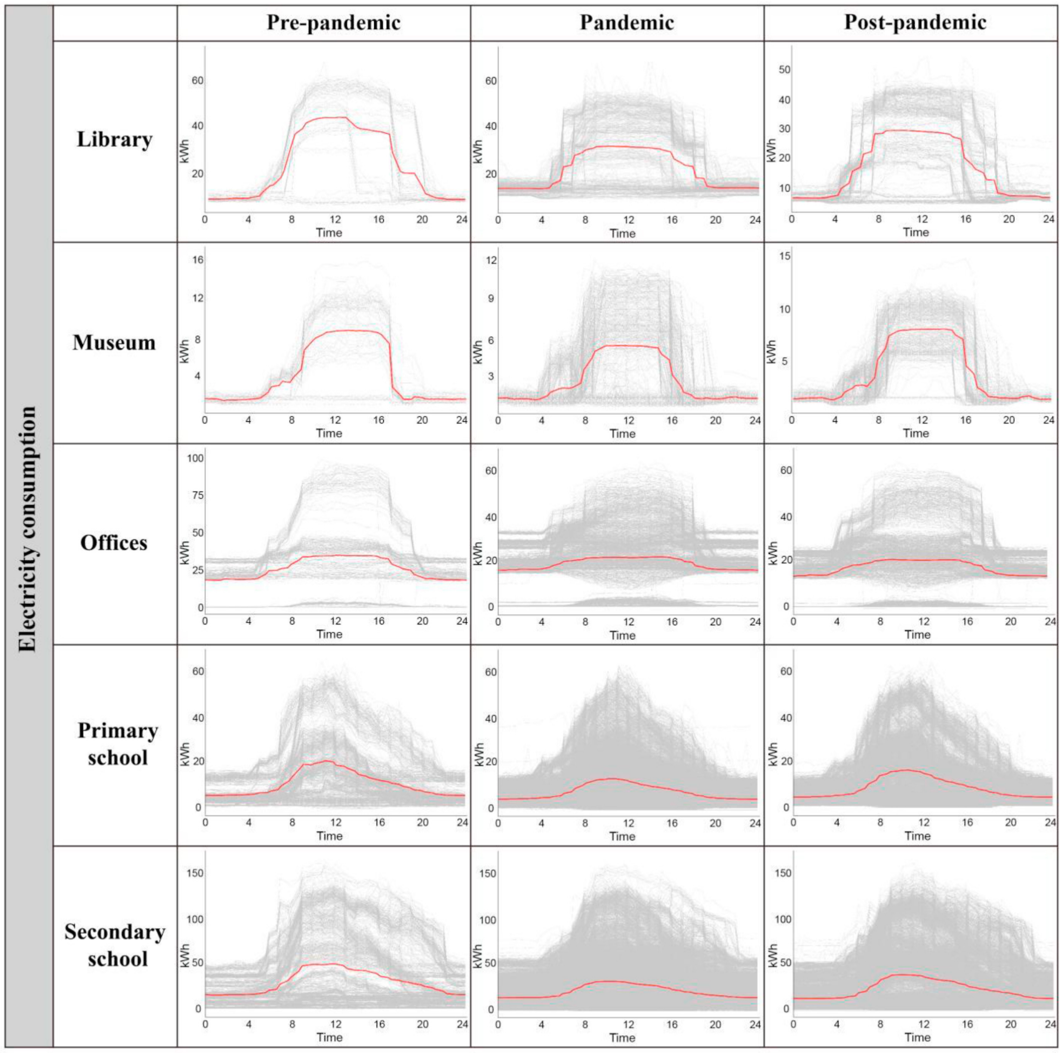


Figure 2: Daily electricity consumption trends of various public buildings in different periods [Huang and Gou, 2023].

4 Applications in Literature

- Understanding the impact of COVID-19 on energy consumption: Researchers used the electricity data set of public buildings in Scotland and applied Gaussian Mixture Model (GMM) to explore the changes in electricity usage patterns throughout the pandemic, so as to understand the long-term impact of COVID-19 on energy consumption of public buildings [Huang and Gou, 2023].

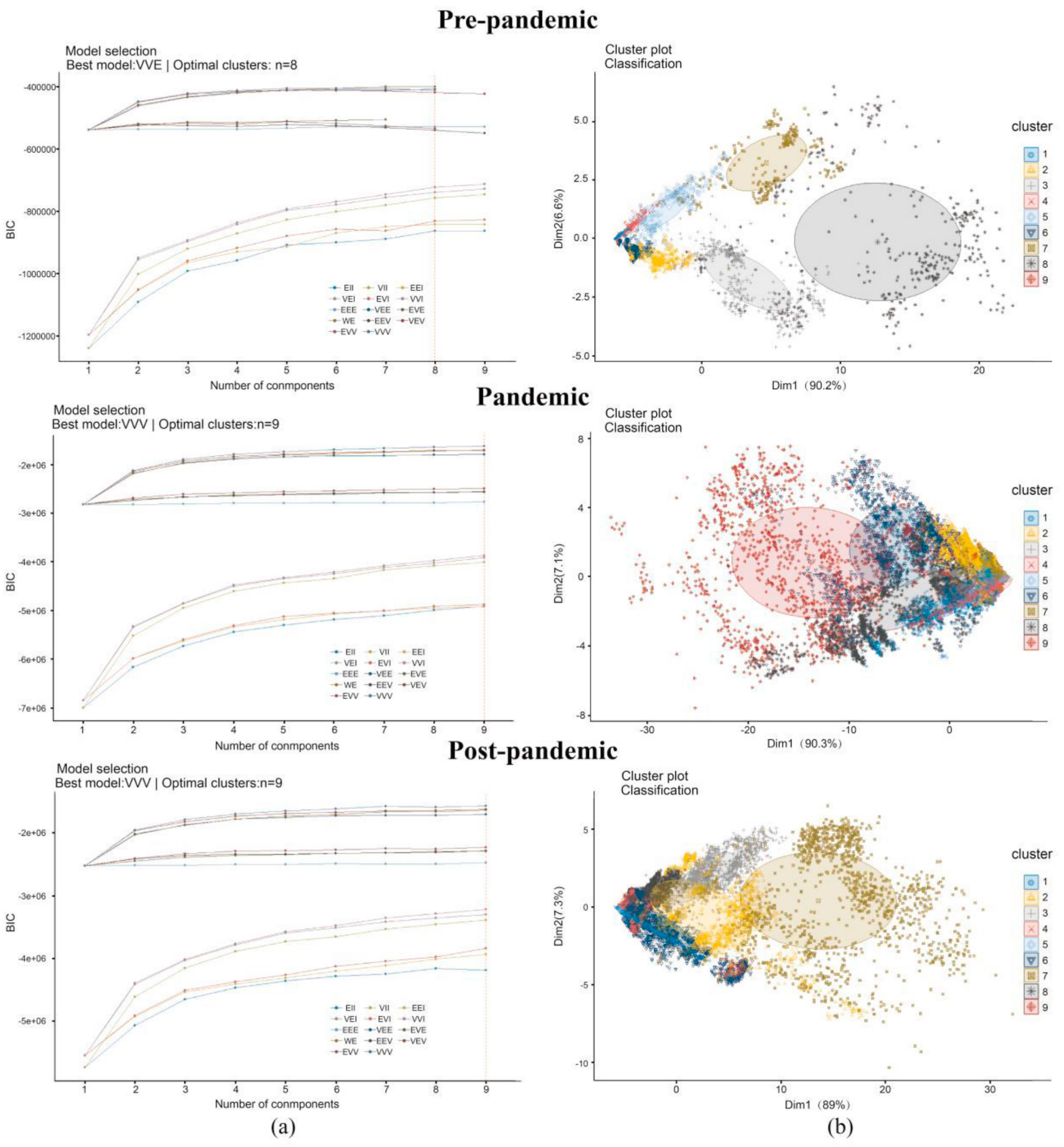


Figure 3: On the right, (b), the visualized Gaussian mixture model for each condition [Huang and Gou, 2023].

- Finding patterns in medical datasets: Medical researchers use GMMs to segment images into multiple categories based on their content or to find specific patterns in medical datasets. They can determine clusters of patients with similar symptoms, identify disease subtypes, and even predict outcomes [Riaz et al., 2020].

- Modeling natural phenomena: GMMs can be used to model natural phenomena where it has been found that noise follows Gaussian distributions. This model of probabilistic modeling relies on the assumption that there exists some underlying continuum of unobserved entities or attributes, and that each member is associated with measurements taken at equidistant points in multiple observation sessions [Xi et al., 2023].

- Customer behavior analysis: GMMs can be used for performing customer behavior analysis in marketing to make predictions about future purchases based on historical data [Melzi et al., 2017].

- Stock price prediction: In finance, GMMs can be applied to a stock's price time series. They can detect changepoints in time series data and help find turning points of stock prices or other market movements that are otherwise difficult to spot due to volatility and noise [Gopinathan et al., 2023].

- Gene expression data analysis: GMMs can be used to detect differentially expressed genes between two conditions and to identify which genes might contribute toward a certain phenotype or disease state [McNicholas and Murphy, 2010].

5 Implementation in R

The function `rGMM` simulates observations from a GMM. The number of observations is specified by n, and the dimension of each observation by d, as seen in Equation 5. The number of clusters is set using k, which defaults to 1. We introduce the parameter of missingness, which is the proportion of elements in the $n \times d$ data matrix that are missing and which defaults to zero [McCaw et al., 2020]. The cluster means and covariances (`covs`) are provided as numeric prototype vectors, or lists of such vectors. In settings where `miss` > 0 (it is possible for all elements of an observation to be missing), these vectors are used as the mean and covariance of all clusters. By default, however, the zero vector and the identity matrix are adopted for the mean and covariance, respectively. We will simulate a few cases of data to demonstrate the functionality of the `rGMM` function.

5.1 Single Component Without Missingness

For n observations on d dimensional data, we can introduce a fraction of missing values m completely at random by setting elements of the data set to NA.

Here, $n = 1000$ observations are simulated from a single $k = 1$ bivariate normal distribution $d = 2$ without missingness (miss $= 0$, by default). The mean is $\mu = (2, 2)$, and the covariance is an exchangeable correlation structure with off-diagonal $\rho = 0.5$.

```r
set.seed(495)

sigma <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)

data = rGMM(n = 1e3, d = 2, k = 1, means = c(2, 2), covs = sigma)

fit <- FitGMM(data, k = 1)

show(fit)
```

```
## Multivariate Normal Model.
##
## Estimated mean:
##   y1   y2
## 2.01 2.00
##
## Estimated covariance:
##       y1    y2
## y1 0.963 0.475
## y2 0.475 1.020
##
## Final Objective:
## [1] -1720
```

The `Estimated covariance` table contains values obtained through fitting a GMM with the `rGMM` function on generated data, and we see that they are close to the target covariance values of $\{1, 0.5, 0.5, 1\}$.

5.2 Single Component With Missingness

$n = 1000$ observations are simulated from a single $k = 1$ trivariate normal distribution $d = 3$ with 20% missingness (miss $= 0.2$). The mean defaults to the zero vector, and the covariance to the identity matrix.

```
set.seed(495)
sigma <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)
```

```r
data = rGMM(n = 1e3, d = 3, k = 1, miss = 0.2)
fit <- FitGMM(data, k = 1)
```

```
## Objective increment:  2.76
## Objective increment:  0.135
## Objective increment:  0.00904
## Objective increment:  0.000855
## Objective increment:  0.000101
## Objective increment:  1.33e-05
## Objective increment:  1.82e-06
## Objective increment:  2.55e-07
## 7 update(s) performed before reaching tolerance limit.
```

```r
show(fit)
```

```
## Multivariate Normal Model.
##
## Estimated mean:
##      y1      y2      y3
## 0.01740 0.02460 0.00185
##
## Estimated covariance:
##         y1      y2     y3
## y1  0.9700 -0.0303 0.0227
## y2 -0.0303  1.0400 0.0363
## y3  0.0227  0.0363 1.0700
```

```
##
## Final Objective:
## [1] -3040
```

Again, we see that the values of the `Estimated mean` vector are near-zero and those of the `Estimated covariance` matrix closely follow the identity matrix, as expected in this case.

5.3 Cluster Number Selection

The function `ClustQual` provides several metrics for internally assessing the quality of cluster assignments from a fitted GMM (Arquez, 2020). The output is a list containing the following metrics:

- BIC (Bayesian information criterion): a penalized version of the negative log likelihood. A lower value indicates better clustering quality.
- CHI (Calinski-Harabaz index): a ratio of the between-cluster to within-cluster variation. A higher value indicates better clustering quality.
- DBI (Davies-Bouldin index): an average of similarities across clusters. A lower value indicates better clustering quality.
- SIL: average silhouette width, a measure of how well an observation matches its assigned cluster. A higher value indicates better clustering quality.

```
set.seed(495)
mu <- list(
  c(2, 2),
```

```r
  c(2, -2),
  c(-2, 2),
  c(-2, -2)
)
sigma <- 0.5 * diag(2)
data = rGMM(n = 100, d = 2, k = 4, means = mu, covs = sigma)
fit <- FitGMM(data, k = 4, maxit = 100, eps = 1e-8, report = F)

# Quality metrics
clust_qual = ClustQual(fit)
cat("BIC:\n")
```

```
## BIC:
```

```r
clust_qual$BIC
```

```
## [1] 221.9521
```

```r
cat("\nCHI:\n")
```

```
##
## CHI:
```

```r
clust_qual$CHI
```

```
## [1] 10.17547
```

```
cat("\nDBI:\n")
```

```
##
## DBI:
```

```
clust_qual$DBI
```

```
## [1] 0.470887
```

```
cat("\nSIL:\n")
```

```
##
## SIL:
```

```
clust_qual$SIL
```

```
## [1] 0.6393902
```

The biggest question in applying the GMM lies in the unknown number of clusters k. Using the inputs of the data matrix, the minimum number of clusters to assess `k0`, the maximum number of clusters to assess `k1`, and the number of bootstrap replicates at each cluster number boot, the function `ChooseK` provides guidance on the number of clusters likely present in the data [McCaw et al., 2020]. For each cluster number k from `k0` to `k1`, boot bootstrapped data sets are generated. A GMM with k components is fit, and the quality metrics from above are calculated. The bootstrap replicates are summarized by their mean and standard error (SE). Thus, for each quality metric, we obtain the cluster number `k_opt` that provided the optimal quality and the smallest cluster number whose quality was within 1 SE of the optimum `k_1se`.

```
choose_k <- ChooseK(data, k0 = 2, k1 = 6, boot = 10)
```

```
## Cluster size 2 complete. 11 fit(s) succeeded.
## Cluster size 3 complete. 11 fit(s) succeeded.
## Cluster size 4 complete. 11 fit(s) succeeded.
## Cluster size 5 complete. 11 fit(s) succeeded.
## Cluster size 6 complete. 10 fit(s) succeeded.
```

```
cat("\nCluster number choices:\n")
```

```
##
## Cluster number choices:
```

```
choose_k$Choices
```

```
##   Metric k_opt  Metric_opt k_1se  Metric_1se
## 1    BIC     6 151.3227726     5 157.3881681
## 2    CHI     6  15.7701771     6  15.7701771
## 3    DBI     4   0.4754451     4   0.4754451
## 4    SIL     4   0.6443205     4   0.6443205
```

```
cat("\nAll results:\n")
```

```
##
## All results:
```

```r
head(choose_k$Results)
```

```
##   Clusters Fits Metric        Mean          SE
## 1        2   11    BIC 331.1585430  7.83821234
## 2        2   11    CHI   1.9099133  0.18458241
## 3        2   11    DBI   0.9500322  0.05248028
## 4        2   11    SIL   0.4765720  0.01810555
## 5        3   11    BIC 273.7737692 16.45488885
## 6        3   11    CHI   3.9120636  0.26041417
```

6 Application to Data

6.1 Iris

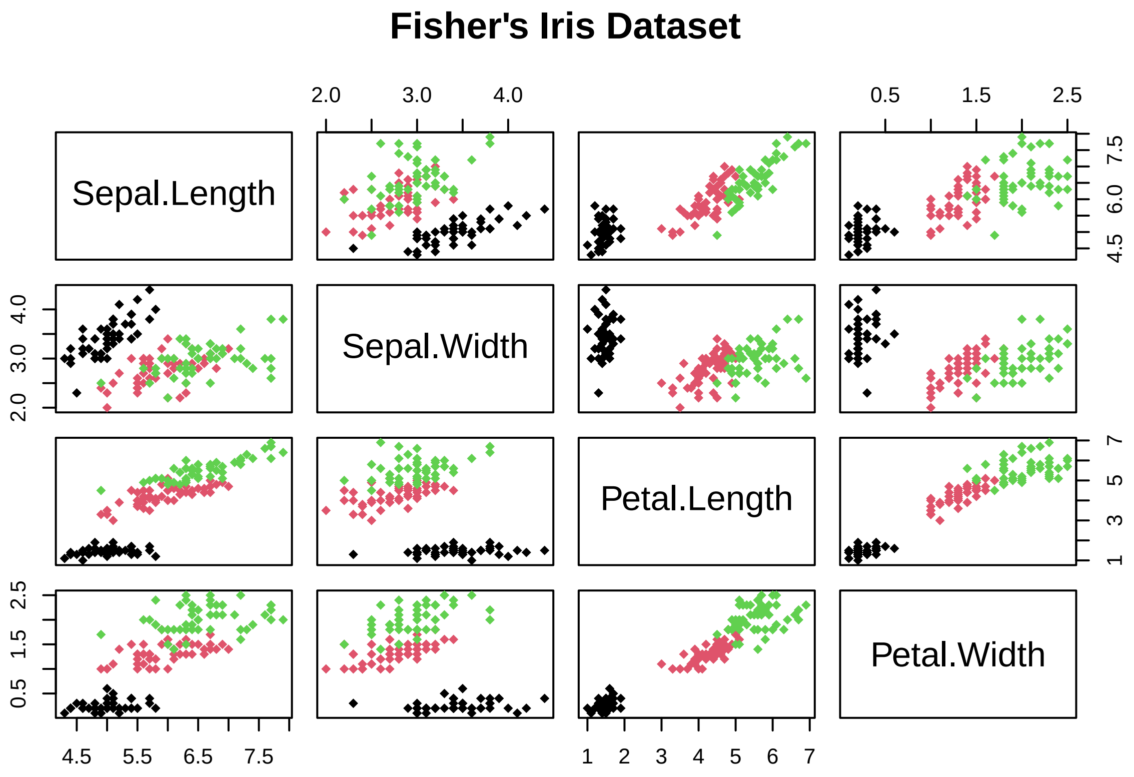
First, let us observe the multivariate normal PDF function and apply the GMM process on a familiar dataset, `iris`. Given a matrix `x`, `mu` (mean), and `sigma` (covariance), we write a function that calculates the probability density for each row, and this function is used to fill in the components of the mean vector and covariance matrix for the GMM.

```r
data(iris)


max_iter <- 100
bic_values <- numeric(max_iter)


mvpdf <- function(x, mu, sigma) {
    if (det(sigma) == 0) {
        warning("Determinant is equal to 0.")
    }
    apply(x, 1, function(x) exp(-(1/2) * (t(x) - mu) %*% MASS::ginv(sigma) %*%
        t(t(x) - mu))/sqrt(det(2 * pi * sigma)))
}


# plot dataset
plot(iris[, 1:4], col = iris$Species, pch = 18, main = "Fisher's Iris Dataset")
```



```
# Mclust comes with hierarchical clustering -> initialize 3 different classes
initialk <- hc(data = iris, modelName = "EII")
initialk <- hclass(initialk, 3)
# split by class, calculate column-means for each class
mu <- split(iris[, 1:4], initialk)
mu <- t(sapply(mu, colMeans))
# covariance matrix for each initial class
cov <- list(diag(4), diag(4), diag(4))
# mixing components
a <- runif(3)
a <- a/sum(a)

for (iter in 1:max_iter) {
# calculate PDF with class means and covariances
z <- cbind(mvpdf(x = iris[, 1:4], mu = mu[1, ], sigma = cov[[1]]), mvpdf(x = iris[,
    1:4], mu = mu[2, ], sigma = cov[[2]]), mvpdf(x = iris[, 1:4], mu = mu[3,
    ], sigma = cov[[3]]))

# expectation step for each class
r <- cbind((a[1] * z[, 1])/rowSums(t((t(z) * a))), (a[2] * z[, 2])/rowSums(t((t(z) *
    a))), (a[3] * z[, 3])/rowSums(t((t(z) * a))))

# choose highest row-wise probability
eK <- factor(apply(r, 1, which.max))

# total responsibility
mc <- colSums(r)

# update mixing components
a <- mc/NROW(iris)


# update means

mu <- rbind(colSums(iris[, 1:4] * r[, 1]) * 1/mc[1], colSums(iris[, 1:4] *

    r[, 2]) * 1/mc[2], colSums(iris[, 1:4] * r[, 3]) * 1/mc[3])


# update covariance matrix

cov[[1]] <- t(r[, 1] * t(apply(iris[, 1:4], 1, function(x) x - mu[1, ]))) %*%

    (r[, 1] * t(apply(iris[, 1:4], 1, function(x) x - mu[1, ]))) * 1/mc[1]


cov[[2]] <- t(r[, 2] * t(apply(iris[, 1:4], 1, function(x) x - mu[2, ]))) %*%

    (r[, 2] * t(apply(iris[, 1:4], 1, function(x) x - mu[2, ]))) * 1/mc[2]


cov[[3]] <- t(r[, 3] * t(apply(iris[, 1:4], 1, function(x) x - mu[3, ]))) %*%

    (r[, 3] * t(apply(iris[, 1:4], 1, function(x) x - mu[3, ]))) * 1/mc[3]


# compute sum of the mixture densities, take log, add the column vector

loglik <- sum(log(apply(t(t(z) * a), 1, sum)))


# calculate BIC

bic_values[iter] <- -2 * loglik + 14 * log(NROW(iris))

}


# plot after each iteration
```

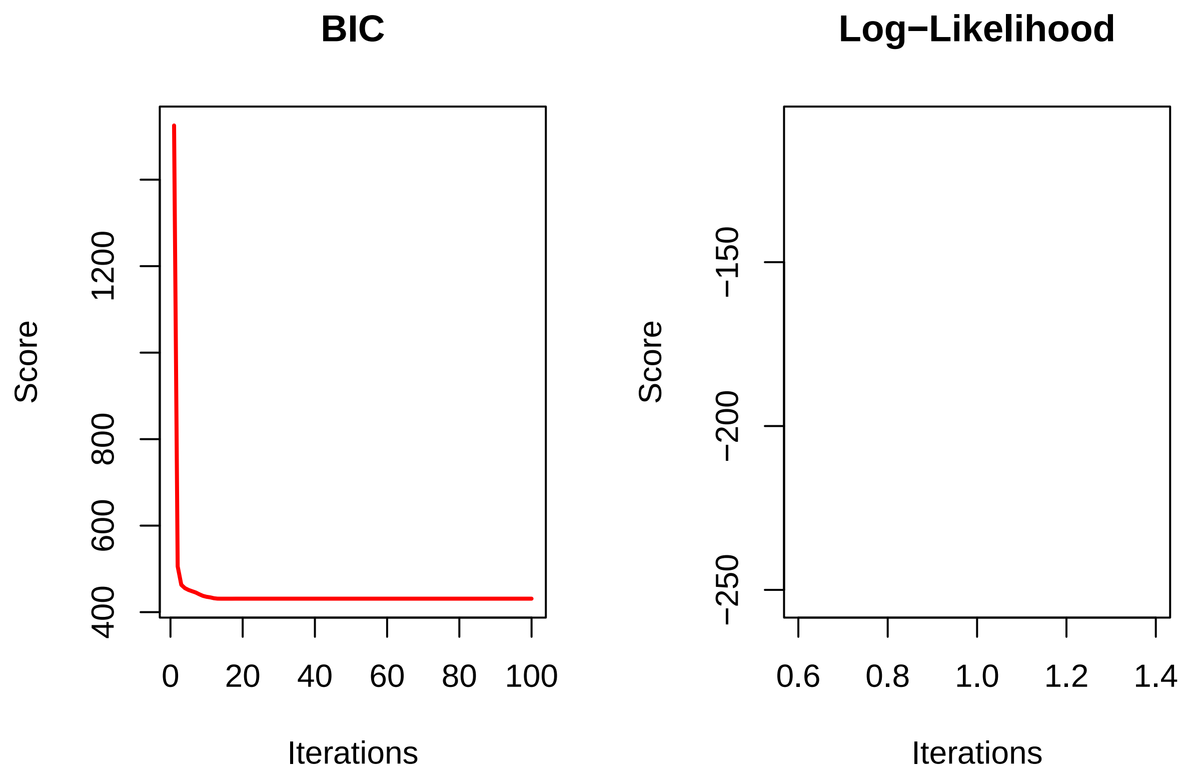
```r
par(mfrow = c(1, 2))

plot(bic_values, type = "l", lwd = 2, col = "red", main = "BIC", xlab = "Iterations",

    ylab = "Score")

plot(loglik, type = "l", lwd = 2, col = "blue", main = "Log-Likelihood",

    xlab = "Iterations", ylab = "Score")
```



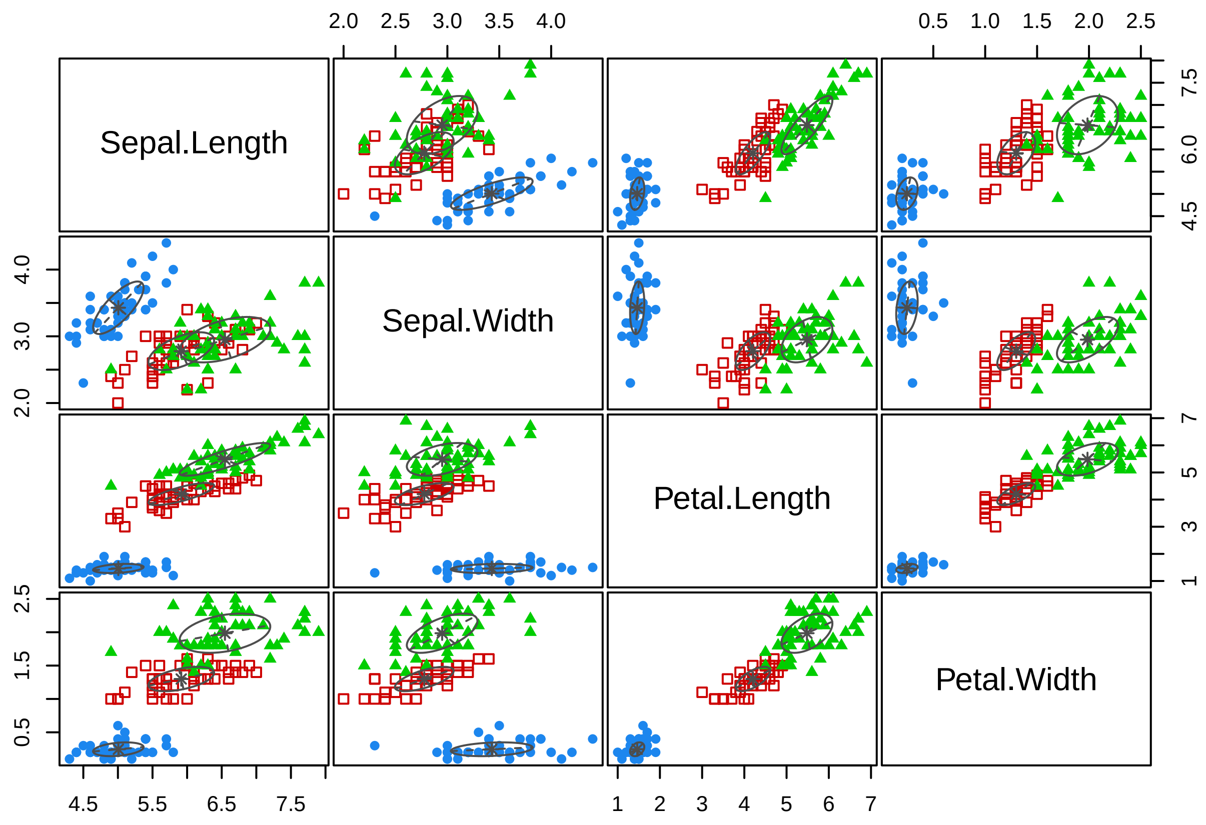
```r
# select 4 continuous variables, look for three distinct groups

mcl.model <- Mclust(iris[, 1:4], 3)



# plot results

plot(mcl.model, what = "classification", main = "Mclust Classification")
```



6.2 Breast Cancer in Wisconsin

Now, we move onto a new dataset, the Breast Cancer Wisconsin (Diagnostic) Dataset. The observations are obtained by processing digitized images of a fine needle aspirate (FNA) of breast mass. With 569 instances of either benign or malignant tumors, each feature in the dataset describes more than 30 characteristics of the cell nuclei that are found in the digitalized images. We will consider just four variables: `radius_mean`, `texture_mean`, `area_mean`, `concave_points_mean` (chosen by observing term significance in a regression model for predicting `diagnosis`).

```
head(breast_cancer[, 1:5])
```

```
##   radius_mean texture_mean area_mean concave_points_mean diagnosis
## 1       17.99        10.38    1001.0             0.14710         M
## 2       20.57       17.77    1326.0             0.07017         M
## 3       19.69       21.25    1203.0             0.12790         M
## 4       11.42       20.38     386.1             0.10520         M
## 5       20.29       14.34    1297.0             0.10430         M
## 6       12.45       15.70     477.1             0.08089         M
```

```
max_iter <- 100
bic_values <- numeric(max_iter)

mvpdf <- function(x, mu, sigma) {
    if (det(sigma) == 0) {
        warning("Determinant is equal to 0.")
    }
    apply(x, 1, function(x) exp(-(1/2) * (t(x) - mu) %*% MASS::ginv(sigma) %*%
        t(t(x) - mu))/sqrt(det(2 * pi * sigma)))
}

initialk <- hc(data = breast_cancer, modelName = "EII")
initialk <- hclass(initialk, 3)
# mean for each class
mu <- split(breast_cancer[, 1:4], initialk)
mu <- t(sapply(mu, colMeans))
# covariance matrix for each initial class
cov <- list(diag(4), diag(4), diag(4))
# mixing components
```

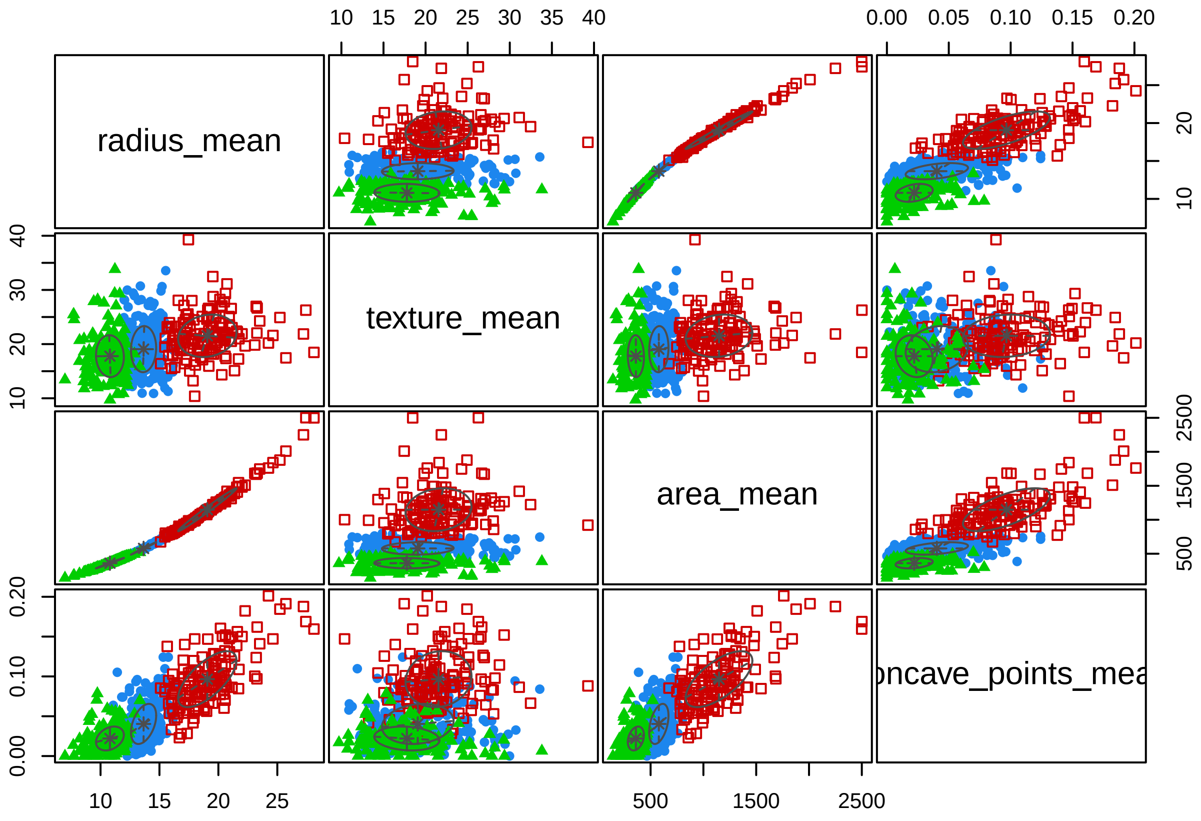
```r
a <- runif(3)

a <- a/sum(a)



mcl.model <- Mclust(breast_cancer[, 1:4], 3)

plot(mcl.model, what = "classification", main = "Mclust Classification")
```



Conclusion

Nizar Bouguila and Wentao Fan. *Mixture models and applications*. Springer, 2020.

Chuong B Do and Serafim Batzoglou. What is the expectation maximization algorithm? *Nature biotechnology*, 26(8):897–899, 2008.

Kala Nisha Gopinathan, Punniyamoorthy Murugesan, and Joshua Jebaraj Jeyaraj. Stock price prediction using a novel approach in gaussian mixture model-hidden markov model. *International Journal of Intelligent Computing and Cybernetics*, 2023.

Zefeng Huang and Zhonghua Gou. Gaussian mixture model based pattern recognition for understanding the long-term impact of covid-19 on energy consumption of public buildings. *Journal of Building Engineering*, 72:106653, 2023.

A Kumar. Gaussian mixture models: What are they & when to use. *Online, Apr*, 2022.

Zachary R. McCaw, Hanna Julienne, and Hugues Aschard. Mgmm: An r package for fitting gaussian mixture models on incomplete data. *bioRxiv*, 2020. doi: 10.1101/2019.12.20.884551. URL `https://www.biorxiv.org/content/early/2020/10/03/2019.12.20.884551`.

Paul D McNicholas and Thomas Brendan Murphy. Model-based clustering of microarray expression data via latent gaussian mixture models. *Bioinformatics*, 26(21):2705–2712, 2010.

Fateh Nassim Melzi, Allou Same, Mohamed Haykel Zayani, and Latifa Oukhellou. A dedicated mixture model for clustering smart meter data: identification and analysis of electricity consumption behaviors. *Energies*, 10(10):1446, 2017.

Douglas A Reynolds et al. Gaussian mixture models. *Encyclopedia of biometrics*, 741 (659-663), 2009.

Farhan Riaz, Saad Rehman, Muhammad Ajmal, Rehan Hafiz, Ali Hassan, Naif Radi Aljohani, Raheel Nawaz, Rupert Young, and Miguel Coimbra. Gaussian mixture model based probabilistic modeling of images for medical image segmentation. *IEEE Access*, 8: 16846–16856, 2020.

Huan Wan, Hui Wang, Bryan Scotney, and Jun Liu. A novel gaussian mixture model for classification. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3298–3303. IEEE, 2019.

Ziyi Xi, Songqiao Shawn Wei, Weiqiang Zhu, Greg Beroza, Yaqi Jie, and Nooshin Saloor. Deep learning for deep earthquakes: Insights from obs observations of the tonga subduction zone. 2023.

Dong Yu, Li Deng, Dong Yu, and Li Deng. Gaussian mixture models. *Automatic Speech Recognition: A Deep Learning Approach*, pages 13–21, 2015.