Gaussian Mixture Model, K-means, and EM Algorithm

Bonnie Yuan

Introduction

When working with large datasets, such as those used in Genome-Wide Association Studies (GWAS), researchers often analyze associations between genetic variants and diseases by clustering individuals based on genetic similarity.

Two widely used algorithms for identifying clusters are K-means and the Expectation-Maximization (EM) algorithm. Each method serves a distinct purpose and has its own strengths.

K-means is a simple and efficient algorithm that partitions data into K clusters by minimizing within-cluster distance. In contrast, EM algorithm is a more flexible and robust method that handles latent (unobserved) variables that represent hidden structures or clusters in the data. It iterates between estimating these hidden cluster memberships (E-step) and updating the model parameters (M-step) to maximize the likelihood of the observed data.

This document summarizes these two algorithms and provides accompanying R code examples.

Notation

Here is the notation used for the two algorithms:

- $X = \{x_1, x_2, \ldots, x_n\}$: Observed data points
- K: Number of clusters
- μ_k: Mean of the k-th Gaussian component
- Σ_k: Covariance matrix of the k-th Gaussian component
- π_k: Mixing coefficient for the k-th component, where $\sum_{k=1}^{K} \pi_k = 1$
- γ_{ik}: Latent (unobserved) variable indicating the probability that data point x_i belongs to cluster k

Gaussian Mixture Model (GMM)

Gaussian Mixture Models (GMM) assume that data points are generated from a mixture of several Gaussian distributions, each characterized by its own mean, covariance, and mixing coefficient. The Gaussian mixture model can be written as:

$$p(x) = \sum_{k=1}^{K} \pi_k \phi(x; \mu_k, \Sigma_k) \tag{1}$$

where Gaussian density $\phi(x; \mu_k, \Sigma_k)$ is the kth component of the mixture with mean μ_k, and covariance Σ_k, and π_k is mixing coefficient, where $\sum_{k=1}^{K} \pi_k = 1$ and $0 \leq \pi_k \leq 1$

Below is an example scatter plot showing Gaussian mixture model with two clusters (Fig.1).

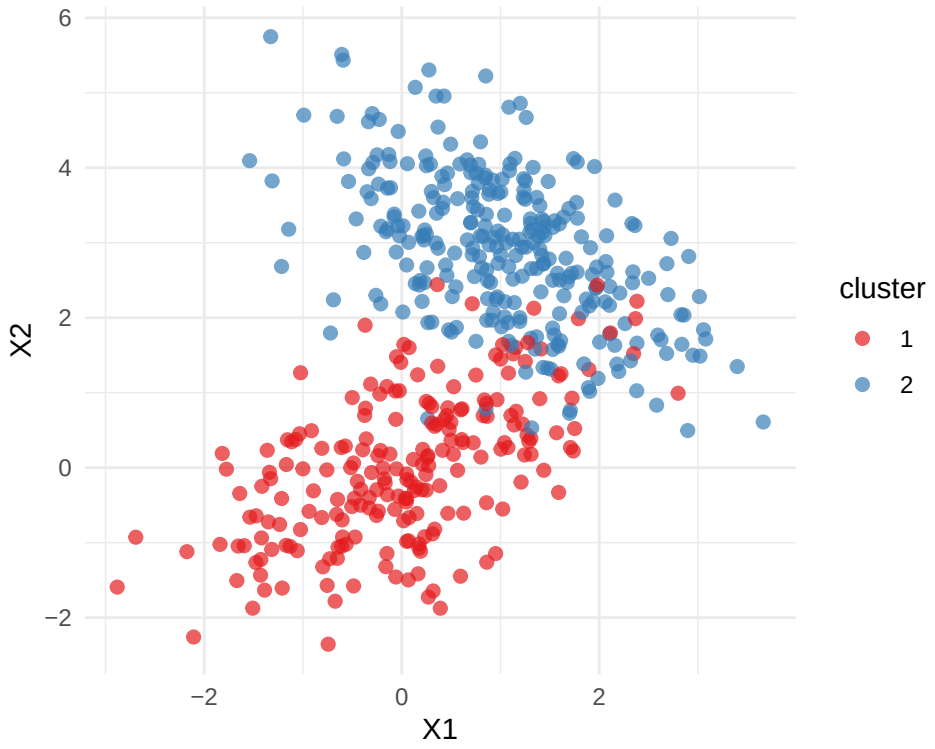


Figure 1: Gaussian mixture model

K-Means

The K-means algorithm partitions data into K clusters. To achieve this, the algorithm defines a new binary variable, $\gamma_{ik} \in \{0, 1\}$, which indicates whether the data point x_i is assigned to the kth cluster. The goal of the algorithm is to find a set of centroids, $\{\mu_k\}$ such that the distance between the data points and their closest centroids is minimized. This is done by minimizing the following objective function, which is the sum of squared Euclidean distances:

$$J = \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_{ik} ||x_i - \mu_k||^2 \tag{2}$$

Steps of the K-means Algorithm:

- Initialize μ_k, usually by randomly selecting K data points.
- Assign each data point to the nearest centroid by minimizing the distance: $\gamma_{ik} = argmin_{\gamma_{ik} J}$. In this step, μ_k are kept fixed, and $\gamma_{ik} = 1$ if the objective function is minimized.
- Update $\mu_k = argmin_{\mu_{ik}} J$. In this step, the cluster assignments γ_{ik} are kept fixed. And the μ_k is calculated as:

$$\hat{\mu_k} = \frac{\sum_{i=1}^{n} \gamma_{ik} x_i}{\sum_{i=1}^{n} \gamma_{ik}} \tag{3}$$

These steps are repeated iteratively until the algorithm converges.

Here is an example implementation of the K-means algorithm.

```
k_means <- function(data,K, max_iter=100,tol = 1e-6) {
  n <- nrow(data)  # Number of data points
  d <- ncol(data)  # Dimension of the data
```

```r
  # initialization mu_k and gamma_ik
  mu_k<-data[sample(1:n, K),]
  gamma_ik <- rep(0, n)
  old_muk <- mu_k
  for (iter in 1:max_iter) {
    # first update gamma_ik
    for( i in 1:n){
      # objective function
      obj <- apply(mu_k, 1, function(mu_k) sum((data[i, ] - mu_k)^2))
      gamma_ik[i] <- which.min(obj)
    }
    # update mu_k: calculate mean for each cluster
    for( k in 1:K){
      subpoints_k=data[gamma_ik==k,]
      if (nrow(subpoints_k) > 0) {
       mu_k[k,]<-colSums(subpoints_k) / nrow(subpoints_k)
      }
    }
    # Check for convergence
    if (sum((mu_k - old_muk)^2) < tol) {
      cat("Converged after", iter, "iterations.\n")
      break
    }
    old_muk <- mu_k
  }
  return(list(mu_k = mu_k, gamma_ik = gamma_ik))
}
```

EM Algorithm

Maximize Likelihood Estimate (MLE)

Recall that the GMM assumes that data is generated from a mixture of several Gaussian distributions. The log-likelihood for Gaussian mixtures is given by:

$$l(\mu_k, \Sigma_k) = \sum_{i=1}^{n} log \sum_{k=1}^{K} \pi_k \phi(x; \mu_k, \Sigma_k)$$

To estimate the parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}$, we can take derivative w.r.t each parameter, for example, for μ_k, it becomes:

$$\frac{d}{d\mu_k} l(\mu_k, \Sigma_k) = \sum_{i=1}^{n} \frac{\pi_k \phi(x; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \phi(x; \mu_k, \Sigma_k)} (\Sigma_k^{-1}(x_i - \mu_k))$$

Here we define the responsibility γ_{ik} which represents the probability that data point x_i belongs to the k-th cluster:

$$\gamma_{ik} = \frac{\pi_k \phi(x; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \phi(x; \mu_k, \Sigma_k)}$$

By setting the derivative to 0 and solve for μ_k, we can obtain the MLE for μ_k:

$$\begin{aligned}\hat{\mu_k} &= \frac{\sum_{i=1}^{n} \gamma_{ik} x_i}{\sum_{i=1}^{n} \gamma_{ik}} \\ &= \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{ik} x_i\end{aligned}$$

where $N_k = \sum_{i=1}^{n} \gamma_{ik}$, the effective number of points assigned to the k-th cluster. This formula closely resembles the update step in K-means clustering, except that in K-means γ_{ik} is binary, whereas in here, it is a probability between 0 and 1.

Similarly the MLE for Σ_k is:

$$\hat{\Sigma_k} = \frac{1}{N_k} \sum_{i=1}^{n} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T$$

Finally, we update the mixing coefficient π_k. Using a Lagrange multiplier to enforce the constraint $\sum_k \pi_k = 1$ we obtain:

$$L(\lambda) = log\ p(x|\theta) + \lambda(\sum_k \pi_k - 1)$$

Then we take deravative of $L(\lambda)$ w.r.t π_k which gives

$$\sum_{i=1}^{n} \frac{\phi(x_i; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \phi(x_i; \mu_k, \Sigma_k)} + \lambda = 0$$

And we sum over k, this gives $\lambda = -N$, and recall $N_k = \sum_{i=1}^{n} \gamma_{ik} = \sum_{i=1}^{n} \frac{\pi_k \phi(x; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} \pi_k \phi(x; \mu_k, \Sigma_k)}$.

this gives

$$\hat{\pi}_k = \frac{N_k}{N}$$

Alternative view of EM

Another way to understand the EM algorithm is through the concept of latent variables and the complete-data log-likelihood. In a GMM, the data X , but the cluster membership of each point is unknown. We introduce a latent variable $Z_k = \mathbf{I}\{\gamma_{ik} = 1\}$, which indicates whether the data point x_i belongs kth cluster.

The marginal log-likelihood can then be written as:

$$\begin{aligned}\log\ p(x; \theta) &= \log \sum_z p(x, z; \theta) \\ &= \log \sum_{q(z)} \frac{p(x, z; \theta)}{q(z)}(z) \\ &= \log\ E_{q(z)}[\frac{p(x, z; \theta)}{q(z)}]\end{aligned}$$

Applying Jensen's inequality and introducing a distribution q(z), we obtain a lower bound on the log-likelihood known as the Evidence Lower Bound (ELBO):

$$\begin{aligned}\log\ p(x; \theta) &\geq E_{q(z)}[\log\ \frac{p(x, z; \theta)}{q(z)}] \\ &= E_{q(z)}[\log\ p(x, z; \theta)] - E_{q(z)}[\log\ q(z)] \\ &= Q(\theta) + H(q(z))\end{aligned}$$

Here, H(q(z)) is the entropy of q(z), which measures the uncertainty in q(z), and $Q(\theta)$ is the expected complete-data log-likelihood under the distribution q(z).

- We further define $E_{q(z)}[\log \frac{p(x,z;\theta)}{q(z)}] = ELBO(\theta, q)$, Evidence Lower Bound based on the evidence of observed data.

Further investigate $ELBO(\theta, q)$ we'll have

$$\begin{aligned}
ELBO(\theta, q) &= E_{q(z)}[\log \frac{p(x,z;\theta)}{q(z)}] \\
&= \sum_{q(z)} \log \frac{p(x,z;\theta)}{q(z)} \\
&= \sum_{q(z)} \log \frac{p(x,z;\theta)}{p(z|x;\theta)} \frac{p(z|x;\theta)}{q(z)} \\
&= \sum_{q(z)} \log \frac{p(z|x;\theta)p(x;\theta)}{p(z|x;\theta)} \frac{p(z|x;\theta)}{q(z)} \\
&= \sum_{q(z)} \log p(x;\theta) \frac{p(z|x;\theta)}{q(z)} \\
&= E_{q(z)}[\log p(x;\theta)] - E_{q(z)}[\log \frac{p(z|x;\theta)}{q(z)}] \\
&= \log p(x;\theta) - E_{q(z)}[\log \frac{q(z)}{p(z|x;\theta)}]
\end{aligned}$$

We can observe that the ELBO is equal to the marginal log-likelihood minus the expected difference between the prior q(z) and the posterior distribution $p(z|x;\theta)$,which is the Kullback-Leibler (KL) divergence.This KL divergence measures how close the two distrisbution. Therefore, to optimize the ELBO effectively, the goal is to set q(z) as close as possible to the posterior $p(z|x;\theta) = \frac{\pi_k \phi(x;\mu_k,\Sigma_k)}{\sum_{k=1}^{K} \pi_k \phi(x;\mu_k,\Sigma_k)}$.

The general steps of the EM algorithm are:

- Initialize $\theta = \theta^{(0)}$
- (E-step) At step t, calculate E[q(z|x,$\theta^{(t)}$)]/responsibility and compute expected value of complete log-likelihood over q(z) $Q(\theta^{(t)}) = E_{q(z)}[log\ p(x, z|\theta)]$
- (M-step) Maximize $\theta^{(t+1)} = argmax_\theta Q(\theta)$

Below is an example R code for implementing EM algorithm.

Simulation

Simulation # 1:

Now let's compare the two algorithms using the data from Figure 1. We simulate 400 data points from two Gaussian distributions with the following characteristics:

- **Mixing coefficients:** $\pi_1 = 0.4$ and $\pi_1 = 0.6$.
- **Means:** $mu_1 = (0, 0)$, and $mu_1 = (1, 3)$
- **Covariance matrices:** The covariance matrix for the two clusters are:

$$\Sigma_1 = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

$$\Sigma_2 = \begin{pmatrix} 1 & -0.5 \\ -0.5 & 1 \end{pmatrix}$$

```
set.seed(2024)
n <- 500
mu <- matrix(c(0,0,
               1,3),byrow=T,ncol=2)
covar <- array(rep(NA,2*2*2), c(2,2,2))  # 3D matrix
covar[,,1] <- matrix(c(1, 0.5,
                       0.5,1),
                     nrow=2,
                     byrow=TRUE) # positive cov
covar[,,2] <- matrix(c(1,-0.5,
                       -0.5,1),
                     nrow=2,
                     byrow=TRUE) # negative cov

# mixing coefficients
pi_k      <- c(0.4,0.6)
class <- sample(1:2, n, replace=TRUE, prob=pi_k)
data<-matrix(rep(NA,n*2), ncol=2)
for(i in 1:n){
  data[i,]=mvrnorm(1,mu=mu[class[i],],Sigma=covar[,,class[i]])
}

head(data)
```

```
##             [,1]        [,2]
## [1,]  1.41128205  1.58367229
## [2,]  0.25034892  4.02736931
## [3,]  1.02268630 -0.55146315
## [4,]  0.20864798  0.04754757
## [5,]  1.09703658  4.05383668
## [6,] -0.03497553 -0.37851220
```

Now let's compare the performance of k-means and EM

In this simulation, we can see that the results from EM are quite similar to the true means. EM slightly outperforms K-means in estimating μ_1.

```
result_kmeans = k_means(data,K=2)
```

```
## Converged after 7 iterations.
```

```
print(result_kmeans$mu_k)   # means
```

```
##           [,1]       [,2]
## [1,] -0.119947 -0.2009369
## [2,]  1.066903  2.8227336
```

```
result_em = EM(data,K=2)
```

```
## Converged after 6 iterations.
```

```
print(result_em$mu_k)
```

```
##             [,1]        [,2]
## [1,]  1.05512479  2.87500244
## [2,] -0.01430165 -0.04354889
```

```r
print(result_em$pi_k) # mixing coefficients
```

```
## [1] 0.5759719 0.4240281
```

Simulation # 2

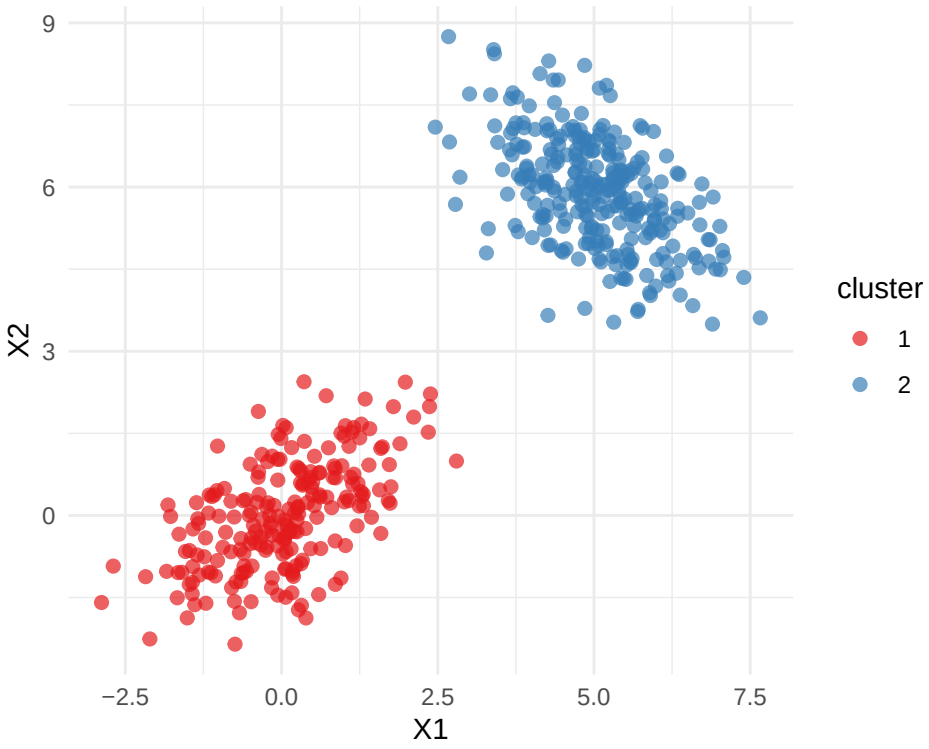
How about two clusters that are not overlap?

```r
set.seed(2024)
n <- 500
mu <- matrix(c(0,0,
               5,6),byrow=T,ncol=2)
covar <- array(rep(NA,2*2*2), c(2,2,2))  # 3D matrix
covar[,,1] <- matrix(c(1, 0.5,
                       0.5,1),
                     nrow=2,
                     byrow=TRUE) # positive cov
covar[,,2] <- matrix(c(1,-0.5,
                       -0.5,1),
                     nrow=2,
                     byrow=TRUE) # negative cov

# mixing coefficients
pi_k      <- c(0.4,0.6)
class <- sample(1:2, n, replace=TRUE, prob=pi_k)
data<-matrix(rep(NA,n*2), ncol=2)
for(i in 1:n){
  data[i,]=mvrnorm(1,mu=mu[class[i],],Sigma=covar[,,class[i]])
}

data_clustered <- data.frame(data, cluster = factor(class))

# scatter plot colored by cluster
ggplot(data_clustered, aes(x = X1, y = X2, color = cluster)) +
  geom_point(size = 2, alpha = 0.7) +
  labs(x = "X1", y = "X2") +
  theme_minimal() +
  scale_color_brewer(palette = "Set1")
```



In cases where the clusters are well-separated and spherical, both K-means and the EM algorithm are essentially the same.

```
result_kmeans = k_means(data,K=2)
```

```
## Converged after 2 iterations.
```

```
print(result_kmeans$mu_k)    # means
```

```
##            [,1]        [,2]
## [1,] 0.02944604 0.007303507
## [2,] 5.04400620 5.897642622
```

```
result_em = EM(data,K=2)
```

```
## Converged after 4 iterations.
```

```
print(result_em$mu_k)
```

```
##            [,1]        [,2]
## [1,] 5.04402535 5.897689757
## [2,] 0.02956534 0.007411772
```

```
print(result_em$pi_k) # mixing coefficients
```

```
## [1] 0.5639875 0.4360125
```

Referneces

1. http://www.di.fc.ul.pt/~jpn/r/PRML/chapter9.html#k-means-clustering-section-9.1
2. Bishop, Christopher M. (2006). Pattern recognition and machine learning. New York :Springer
3. https://teng-gao.github.io/blog/2022/ems/