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).

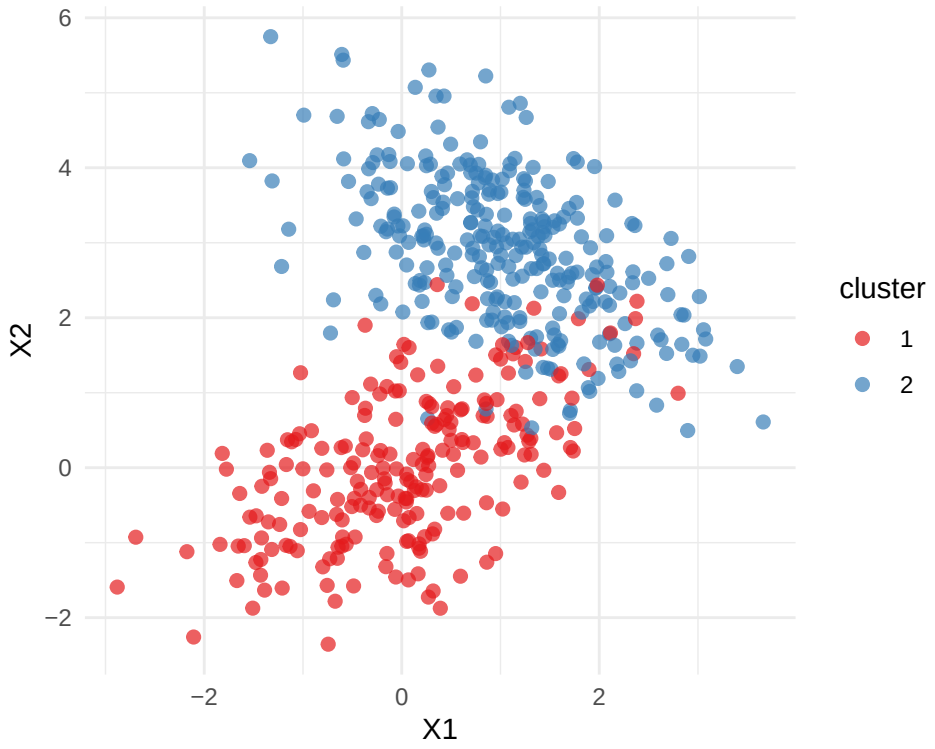

Figure 1: Gaussian mixture model

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
```

```
  # 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,]<-colMeans(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

Below is an example R code for implementing EM algorithm.

Simulation

Now let's compare the two algorithms. We first similate two clusters with overlap data points that we showed in Figure 1.

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

Here we can see the results from EM is pretty similar to the true means.

```
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.
```

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

```
##          [,1]       [,2]
## [1,] 4.437115  2.8179145
## [2,] 2.143161 -0.2089053
```

Conclusion

K-means assumes clusters are spherical and equally sized, and it works well when the data is relatively simple and separable. However, EM is suited for more complex tasks which identifies sub-populations in heterogeneous datasets, where overlapping or irregularly shaped clusters are common.