Clustering Algorithms

Comparison of K-Means and EM

Derek Wayne

17 April 2019

Abstract

This is meant to be a short overview of the implementation of the K-means and EM algorithms for clustering generated multivariate normal data. Performance of the two procedures is compared to gain a deeper understanding of their advantages and limitations.

K-means

In order to explore the K-means algorithm we generate data from a mixture of two bivariate normal distributions with parameters $\mu_1 = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 6.0 \\ 0.1 \end{bmatrix}$, and $\sum_1 = \sum_2 = \begin{bmatrix} 10 & 7 \\ 7 & 10 \end{bmatrix}$.

```
library(MASS)
library(ggplot2)
set.seed(101)
mu_1 <- c(0.1, 0.1); mu_2 <- c(6.0, 0.1)
Sigma <- matrix(c(10, 7, 7, 10), nrow = 2)
# Generate two samples
sample_1 <- mvrnorm(n=200, mu=mu_1, Sigma = Sigma)
sample_2 <- mvrnorm(n=200, mu=mu_2, Sigma = Sigma)

sample <- rbind(sample_1, sample_2)
colnames(sample) <- c("X1", "X2")
classes <- c(rep(1, 200), rep(2, 200))
data <- as.data.frame(sample)
data["Class"] <- as.factor(classes)
```

Assume that the true class labels are not known. The cost function is defined as,

$$\sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \mu\|_2^2$$

Below we implement the K-means algorithm below with $K = 2$.

```
cost <- function(x, centroid, R) {
  sum((x - t(centroid) %*% t(R)) ** 2)
}

km_e_step <- function(x, centroid, A) {
  norm_vec <- function(y) sqrt(sum(y^2)) # euclidean norm
  for (i in 1:nrow(A)) {
    dist_1 <- norm_vec(x[,i] - centroid[1,])
    dist_2 <- norm_vec(x[,i] - centroid[2,])
    k <- which.min(c(dist_1, dist_2))
    # assign r[i,k]
      if (k == 1) {
        A[i,] <- c(1,0)
```

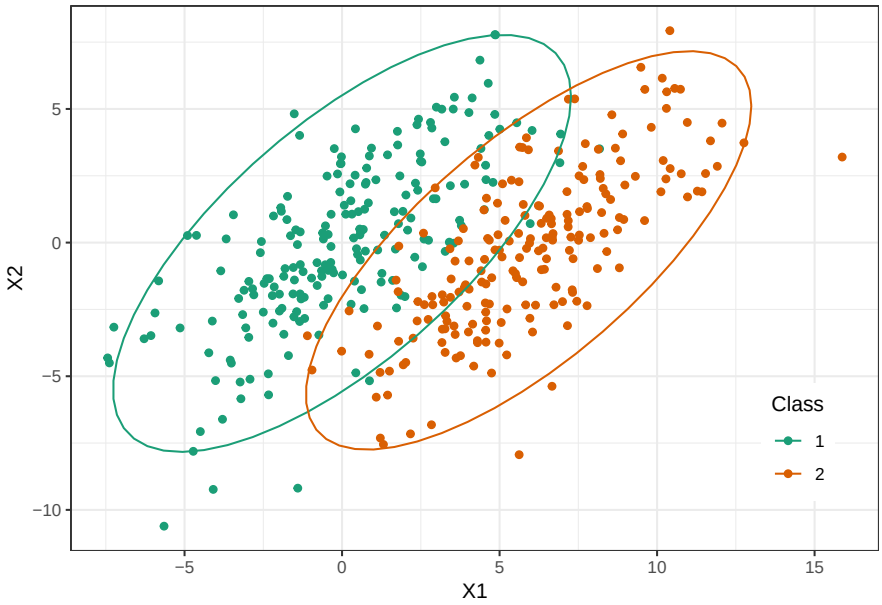

Figure 1: Generated Bivariate Normal Data

```
      } else {
        A[i,] <- c(0,1)
      }
  }
  A
}

km_m_step <- function(x, centroid, A) {
  C1 <- x %*% A[,1] / sum(A[,1])
  C2 <- x %*% A[,2] / sum(A[,2])
  centroid <- matrix(c(C1,C2),ncol=2,byrow = TRUE)
  centroid
}

km_ <- function(x, centroid, R, max.iter) {
  cost_old <- 1L
  cost_new <- 0
  i <- 0
  cost_vec <- c()
  while ((cost_new != cost_old) & (i < max.iter)) {
    R <- R <- km_e_step(x, centroid, R)
    cost_old <- cost(x, centroid, R)
    cost_vec <- c(cost_vec, cost_old)
    centroid <- km_m_step(x, centroid, R)
    cost_new <- cost(x, centroid, R)
    cost_vec <- c(cost_vec, cost_new)
    i <- i + 1
  }
  cluster <- apply(t(R), FUN = which.max, MARGIN = 2)
  r <- list(centroid=centroid, R=R, cluster=cluster, cost=cost_vec)
  r
```

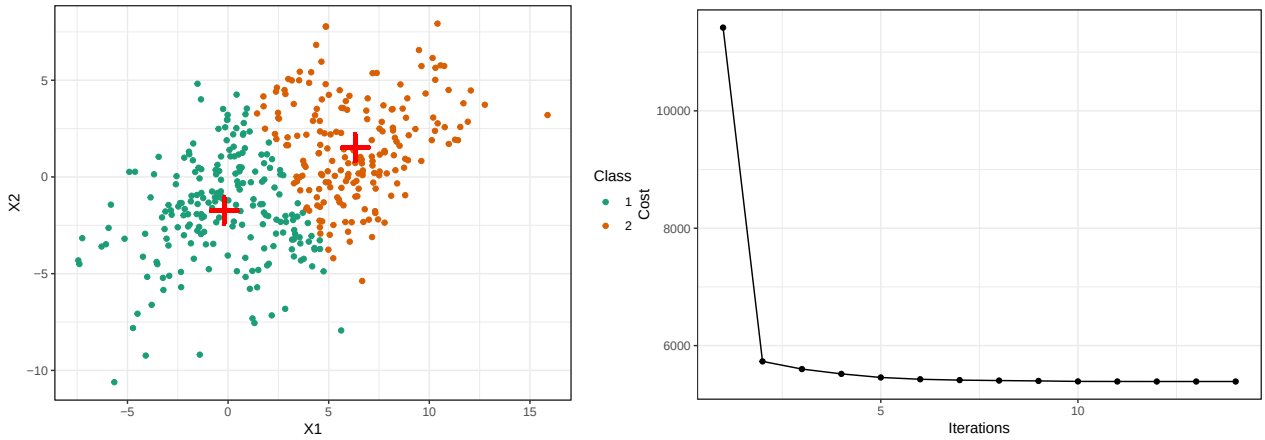

Figure 2: Results of K-means

```
}

X <- t(sample)
R <- matrix(0, nrow=ncol(X), ncol=2)
centroid <- matrix(c(0,0,1,1),ncol=2,byrow=TRUE)
r <- km_(X, centroid, R, max.iter = 20)
mc_err <- mean(classes != r$cluster)
```

The misclassification error is then 0.25.

EM

Modeling the data used above as a mixture of two bivariate Gaussians; then for a given datapoint $\mathbf{x}$,

$$p(\mathbf{x}) = \pi_1 \mathcal{N}(\mathbf{x}|\mu_1, \Sigma) + \pi_2 \mathcal{N}(\mathbf{x}|\mu_2, \Sigma)$$

For each datapoint observed there is a corresponding latent variable $\mathbf{z}_n$. Bayes' Theorem leads us to the following formulation,

$$\gamma(z_k) = P\{z_k = 1 \,|\, \mathbf{x}\} = \frac{\pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma)}{\pi_1 \mathcal{N}(\mathbf{x}|\mu_1, \Sigma) + \pi_2 \mathcal{N}(\mathbf{x}|\mu_2, \Sigma)}$$

Thus the parameters to be estimates are denoted by $\theta = \{\mu_1, \mu_\mathbf{2}, \pi_1, \pi_2, \Sigma\}$. Under the assumption of latent variables we consider the complete data likelihood to be

$$p(\mathbf{X}, \mathbf{Z} \,|\, \theta) = \prod_{n=1}^{N} \left\{ \left[\pi_1 \mathcal{N}(\mathbf{x}|\mu_1, \Sigma) \right]^{z_{n1}} \left[\pi_2 \mathcal{N}(\mathbf{x}|\mu_2, \Sigma) \right]^{z_{n2}} \right\}$$

and so the complete data log-likelihood is

$$log\, p(\mathbf{X}, \mathbf{Z} \,|\, \theta) = \sum_{n=1}^{N} \left\{ z_{n1} log \pi_1 + z_{n1}\, log\, \mathcal{N}(\mathbf{x}|\mu_1, \Sigma) + z_{n2} log \pi_2 + z_{n2}\, log\, \mathcal{N}(\mathbf{x}|\mu_2, \Sigma) \right\}$$

leading to the maximum likelihood estimates

$$\widehat{\pi}_k = N_k/N \quad w.\ N_k = \sum_{n=1}^{N} \gamma_k(z_{nk})$$

$$\widehat{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k(z_{nk}) x_n$$

$$\Sigma_k^{\dagger} = \frac{1}{N_k} \gamma_k(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T$$

The following function is to compute the multivariate Gaussian density. However, care must be taken as it involved inverting a covariance matrix. The base solve function in R can provide results that are ruined by rounding error. Since the covariance matrix is an SPD (semi) matrix the Cholesky factorization is a natural choice for speed and numerical stability.

```
normal_density <- function(x, mean, sigma) {
  cf <- chol(sigma)
  tmp <- backsolve(cf, t(x) - mean, transpose = TRUE)
  rss <- colSums(tmp^2)
  lv <- -sum(log(diag(cf))) - 0.5 * ncol(x) * log(2 *
            pi) - 0.5 * rss
  exp(lv)
}
```

Below is the implementation of the EM algorithm.

```
e_step <- function(x, theta) {
  mix.p <- c(theta[[1]], theta[[2]])
  mu_1 <- theta[[3]]; mu_2 <- theta[[4]]
  Sigma_1 <- theta[[5]]
  Sigma_2 <- theta[[6]]

  a <- mix.p[1] * normal_density(x, mean=mu_1, sigma=Sigma_1)
  b <- mix.p[2] * normal_density(x, mean=mu_2, sigma=Sigma_2)
  resp_1 <- a / (a+b)
  resp_2 <- b / (a+b)
  r <- list("loglik"=sum(log(a+b)), "responsibilities"=cbind(resp_1, resp_2))
  r
}

m_step <- function(x, resps) {
  mu_1.dagger <- as.vector(crossprod(resps[,1],x) / sum(resps[,1]))
  mu_2.dagger <- as.vector(crossprod(resps[,2],x) / sum(resps[,2]))

  Sigma_1.dagger <- t(resps[,1] * t(apply(x, 1, function(x) x - mu_1.dagger))) %*%
    (resps[,1] * t(apply(x, 1, function(x) x - mu_1.dagger))) * 1/sum(resps[,1])
  Sigma_2.dagger <- t(resps[,2] * t(apply(x, 1, function(x) x - mu_2.dagger))) %*%
    (resps[,2] * t(apply(x, 1, function(x) x - mu_2.dagger))) * 1/sum(resps[,2])

  pi_1.dagger <- sum(resps[,1]) / 400
  pi_2.dagger <- sum(resps[,2]) / 400

  r <- list(p1=pi_1.dagger, p2=pi_2.dagger, mu1=mu_1.dagger, mu2=mu_2.dagger,
            sig1=Sigma_1.dagger, sig2=Sigma_2.dagger)
  r
}
```

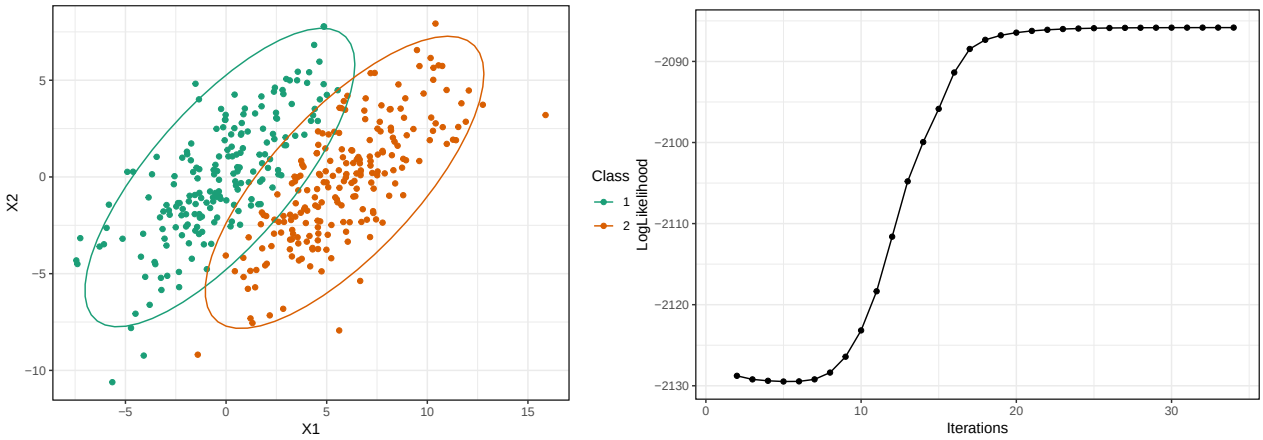

Figure 3: Clustering by EM

```
EM <- function(x, init, max.iter, tol=10e-4) {
  for (i in 1:max.iter) {
    if (i == 1) {
      E_ <- e_step(x, init)
      M_ <- m_step(x, E_$responsibilities)
      old.loglik <- E_$loglik
      loglik.vec <- E_$loglik
    } else {
      E_ <- e_step(x, M_)
      M_ <- m_step(x, E_$responsibilities)
      loglik.vec <- c(loglik.vec, E_$loglik)
      condition <- abs((E_$loglik - old.loglik))
      if (condition < tol) {
        break
      } else {
        old.loglik <- E_$loglik
      }
    }
  }
  list(p1=M_$p1, p2=M_$p2, mu1=M_$mu1, mu2=M_$mu2,
       sig1=M_$sig1, sig2=M_$sig2,
       responsibilities = E_$responsibilities, loglik = loglik.vec)
}
```

```
theta.init <- list(0.5, 0.5, c(0,0), c(1,1), diag(2), diag(2)) # initializations
res <- EM(sample, theta.init, 100)
pred <- apply(res$responsibilities, 1, function(row) which.max(row))
mc_err2 <- mean(pred != classes)
```

Difference in misclassification error: kmeans: 0.25, EM: 0.0725.

Results

The K-Means procedure converges in 10 iterations whereas the EM converges very quickly. The K-Means is very clearly biased with respect to the L_2 norm. It uses hard partitioning to assign points to each cluster

vs. the EM, which assigns based on the expectation of a point belonging to a particular cluster. Therefore, it can be seen that K-Means does not take into account the covariance structure of the data like the EM does. This produces difference in the misclassification errors between the two methods. To demonstrate that this conclusion is not dependent of the data we will generate new multivariate Gaussian data and run the procedures again.

```
data <- list()
for (i in 1:10) {
  sample_1 <- mvrnorm(n=200, mu=mu_1, Sigma = Sigma)
  sample_2 <- mvrnorm(n=200, mu=mu_2, Sigma = Sigma)
  sample <- rbind(sample_1, sample_2)
  colnames(sample) <- c("X1", "X2")

  data_i <- as.data.frame(sample)
  data_i["Class"] <- as.factor(c(rep(1, 200), rep(2, 200)))
  data[[i]] <- data_i
}
new_results <- data.frame("K_Means"=numeric(10),
                          "EM"=numeric(10),
                          "Absolute_Diff"=numeric(10))
for (i in 1:10) {
  X <- t(data[[i]][,c("X1", "X2")])
  r1 <- km_(X, centroid, R, max.iter=20)
  mc_err1 <- mean(classes != r1$cluster)
  new_results$K_Means[i] <- mc_err1

  r2 <- EM(t(X), theta.init, 20)
  pred <- apply(r2$responsibilities, 1, function(row) which.max(row))
  mc_err2 <- mean(classes != pred)
  new_results$EM[i] <- mc_err2
}
new_results$Absolute_Diff <- abs(new_results$K_Means - new_results$EM)
new_results
```

```
##    K_Means     EM Absolute_Diff
## 1   0.2500 0.3975        0.1475
## 2   0.2325 0.1125        0.1200
## 3   0.2425 0.4925        0.2500
## 4   0.2375 0.2250        0.0125
## 5   0.2500 0.1475        0.1025
## 6   0.2550 0.2525        0.0025
## 7   0.2600 0.2825        0.0225
## 8   0.2575 0.2000        0.0575
## 9   0.2875 0.3950        0.1075
## 10  0.2900 0.1350        0.1550
```

```
sum(new_results$K_Means>new_results$EM)/10
```

```
## [1] 0.6
```

By applying the two algorithms for 10 independent realizations of the data we can see that performance depends greatly on the data. However, EM is much better suited for capturing the structure since the data was generated from a mixture of Gaussians. If the clusters were better seperated, both methods would give similair results but K-means converges faster. It is also worth noting that the generative EM model, is able to produce new observations.