RAJAOBELINA_EM_ALGORITHM

Fitahiry RAJAOBELINA

2023-12-17

EM algorithm (for Gaussian Mixture Model)

Initialization

Goal: The initialization consist in setting random initial values for the means, variances, and mixing coefficients.

Assume that the data are in the variable "data". In order to make the compilation possible, we've choose default value number of cluster and default value of "data". Those values can be changed after.

```
# Number of cluster
K <- 1 # can be changed after

# Data
data <- data <- c(rnorm(100, mean = 3, sd = 1), rnorm(100, mean = 8, sd = 1)) # can be changed after

# Initialization of means and variances
initial_means <- runif(K, min(data), max(data))
initial_variances <- runif(K, 0.1, 1)

# Initialization of mixing coefficients
initial_pi <- rep(1/K, K)
```

E-step

Goal: To compute the expected value of the latent variables (the responsibilities) given the observed data and the current parameter estimates.

The following code calculates the responsibilities of each data point belonging to each cluster.

```
# E-step function
E_step <- function(data, means, variances, pi_values) {
  N <- length(data)  # Number of data points
  K <- length(means)  # Number of clusters

  responsibilities <- matrix(0, nrow = N, ncol = K)

  for (k in 1:K) {
    responsibilities[, k] <- pi_values[k] * dnorm(data, mean = means[k], sd = sqrt(variances[k]))
  }

  responsibilities <- responsibilities / rowSums(responsibilities)

  return(responsibilities)
}
```

M-step

Goal: To maximize the likelihood function with respect to the model parameters given the current responsibilities (expectation) computed in the E-step.

Given the theoric formula for Gaussian Mixture Model, the following code is the M-step:

```r
# M-step function
M_step <- function(data, responsibilities) {
  N <- nrow(responsibilities)  # Number of data points
  K <- ncol(responsibilities)  # Number of clusters

  updated_means <- rep(0, K)
  updated_variances <- rep(0, K)
  updated_pi <- rep(0, K)

  for (k in 1:K) {
    updated_means[k] <- sum(responsibilities[, k] * data) / sum(responsibilities[, k])
    updated_variances[k] <- sum(responsibilities[, k] * (data - updated_means[k])^2) / sum(responsibili
    updated_pi[k] <- mean(responsibilities[, k])
  }

  return(list(means = updated_means, variances = updated_variances, pi_values = updated_pi))
}
```

Test

Goal: To test the algorithm on a mixture of two Gaussians.

First test At first, we initialize the data

```r
# Parameters of the two Gaussians
mu1 <- 0
mu2 <- 4
sigma1 <- 1
sigma2 <- 0.5
pi1 <- 1/3

# Sample's length
N <- 1000

# Generating data from the mixture model
mixture_indicator <- rbinom(N, 1, pi1) + 1  # Indicator for which Gaussian each data point comes from
data <- c(rnorm(sum(mixture_indicator == 1), mean = mu1, sd = sigma1),
          rnorm(sum(mixture_indicator == 2), mean = mu2, sd = sigma2))
```

The we initialize according to the first part

```r
# Number of mixture
K <- 2


# Initialization of means and variances
initial_means <- runif(K, min(data), max(data))
initial_variances <- runif(K, 0.1, 1)
```

```r
# Initialization of mixing coefficients
initial_pi <- rep(1/K, K)
```

Then we code the EM-algorithm iteration

```r
# EM algorithm iteration
EM_algorithm <- function(data, K, max_iter = 100, tol = 1e-6) {
  # Initialization of parameters
  means <- initial_means
  variances <- initial_variances
  pi_values <- initial_pi

  likelihood_prev <- -Inf  # Initialize previous log-likelihood

  for (iter in 1:max_iter) {
    # E-step
    responsibilities <- E_step(data, means, variances, pi_values)

    # M-step
    updated_params <- M_step(data, responsibilities)
    means <- updated_params$means
    variances <- updated_params$variances
    pi_values <- updated_params$pi_values

    # Calculate log-likelihood to check for convergence
    log_likelihood <- sum(log(rowSums(responsibilities * matrix(pi_values, nrow = length(data), ncol = K
                                      sapply(1:K, function(k) dnorm(data, means[k], sqrt(variances[k])))

    # Check for convergence
    if (abs(log_likelihood - likelihood_prev) < tol) {
      cat("Converged after", iter, "iterations.\n")
      break
    }

    likelihood_prev <- log_likelihood  # Update previous log-likelihood
  }

  return(list(means = means, variances = variances, pi_values = pi_values))
}
```

```r
result <- EM_algorithm(data, K)
```

```
## Converged after 29 iterations.
```

```r
cat("Estimated Means:", result$means, "\n")
```

```
## Estimated Means: -0.03486875 3.97518
```

```r
cat("Estimated Variances:", result$variances, "\n")
```

```
## Estimated Variances: 1.07248 0.2621139
```

```r
cat("Estimated Mixing Coefficients:", result$pi_values, "\n")
```

```
## Estimated Mixing Coefficients: 0.6774292 0.3225708
```

The result are:

$$\hat{\mu_1} = -0.04989959 \hat{\mu_2} = 3.977022 \hat{\sigma_1} = 1.095812 \hat{\sigma_2} = 0.2476 \hat{\pi_1} = 0.3466856 \hat{\pi_2} = 0.6533144$$

Conclusion

The EM algorithm implementation seems to gives good estimates of the real parameters, however, the estimated variance $\hat{\sigma_2}$ is far from the real variance (which is 0.5). This can be improved by improving the initialization using K-means clustering or hierarchical clustering.