Exam 2

Alex Whang

2024-11-25

Part 1

a. The E-step calculates the probability of each data point belonging to a particular component of the mixture based on observed data and current estimates of model parameters.

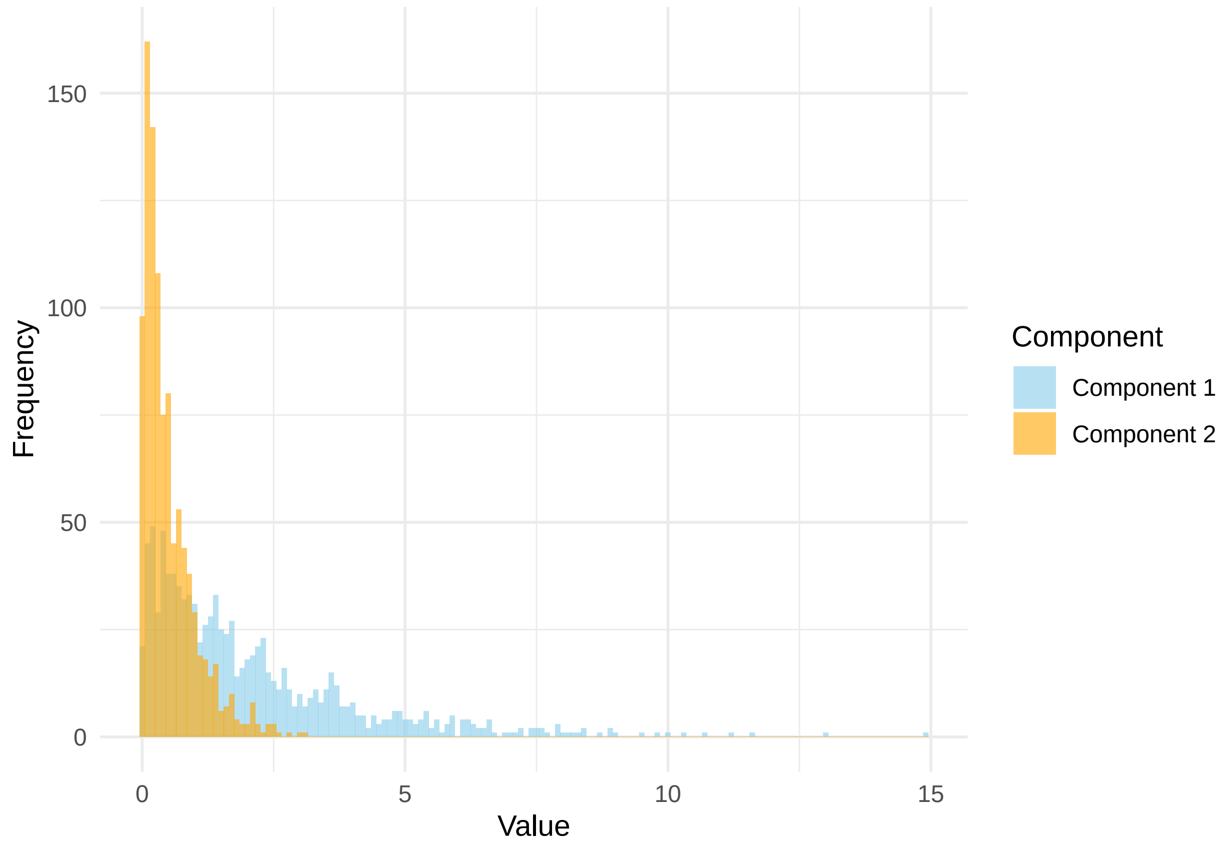
b. The M-step updates the parameters of the mixture model using the probabilities calculated in the E-step. This allows the model to better fit the data based on adjusting the parameter estimations of individual distributions.

c. The clusters are overlapping because all of the components are modeled as probabilistic distributions, and data points may have high probabilities of belonging to more than one cluster.

Part 2

```r
library(ggplot2)
set.seed(1234)
# sample
n_samples <- 2000
true_rates <- c(0.5, 2.0)
true_proportions <- c(0.5, 0.5)

# Sample group assignments (1 or 2) based on mixing proportions
component_labels <- sample(1:2, n_samples, replace = TRUE, prob = true_proportions)

# Stage 2: Generate data from the corresponding exponential
# distribution based on the group assignment
data <- numeric(n_samples) # Initialize the data vector

# Generate data for group 1 (rate = 0.5)
data[component_labels == 1] <- rexp(sum(component_labels == 1), rate = true_rates[1])

# Generate data for group 2 (rate = 2.0)
data[component_labels == 2] <- rexp(sum(component_labels == 2), rate = true_rates[2])

# Create a data frame for plotting
plot_data <- data.frame(
  Value = data, # Data points
  Component = factor(component_labels, labels = c("Component 1", "Component 2")) # Component labels
)
```

```r
# Plot the data using ggplot2
ggplot(plot_data, aes(x = Value, fill = Component)) +
  geom_histogram(binwidth = 0.1, position = "identity", alpha = 0.6) +
  scale_fill_manual(values = c("skyblue", "orange")) + # Set colors for the components
  labs(
    x = "Value",
    y = "Frequency",
    fill = "Component"
  ) +
  theme_minimal() +
  theme(plot.title = element_blank()) # Remove the title
```



```r
exponentialMixture <- function(data, K, max_iter = 1000, tol = 1e-5) {
  n <- length(data)
  pi <- rep(1/K, K) # mixing proportions
  lambda <- runif(K, 0.1, 1) # rate parameters
  log_likelihoods <- numeric(max_iter) # store log-likelihood values

  for (iter in 1:max_iter) { # E-step: numerator of the gammas
    gamma <- matrix(NA, nrow = n, ncol = K)
    for (k in 1:K) {
      gamma[, k] <- pi[k] * dexp(data, rate = lambda[k])
    }
    row_sums <- rowSums(gamma) # denominator of the gammas
    gamma <- gamma / row_sums # normalize probabilities
    pi_old <- pi # M-step: Update mixing proportions and rate parameters
    lambda_old <- lambda

    pi <- colMeans(gamma) # update mixing proportions
    for (k in 1:K) {
      lambda[k] <- sum(gamma[, k]) / sum(gamma[, k] * data) # update rate parameters
    }

    log_likelihoods[iter] <- sum(log(row_sums)) # calculate log-likelihood

    if (max(abs(pi - pi_old)) < tol && max(abs(lambda - lambda_old)) < tol) {
      log_likelihoods <- log_likelihoods[1:iter] # trim to the number of iterations
      cat("Convergence reached at iteration", iter,
          "with log-likelihood:", log_likelihoods[iter], "\n")
    break
    }
  }
  return(list(pi = pi, lambda = lambda, log_likelihood = log_likelihoods))
}
```

a.

```
# Code taken from in class exercise mixture models
set.seed(1234)
library(knitr)

# Fit the mixture model
result <- exponentialMixture(data, K = 2)
```

```
## Convergence reached at iteration 229 with log-likelihood: -2390.612
```

```
# Create a data frame to display the results in a table
results_table <- data.frame(
  "Component" = 1:2,
  "Estimated Mixing Proportion" = round(result$pi, 4),
  "Estimated Rate Parameter (lambda)" = round(result$lambda, 4)
)

# move results to a table
kable(results_table, caption = "Estimated Parameters for the Mixture Model", format = "pipe")
```

Table 1: Estimated Parameters for the Mixture Model

Component	Estimated.Mixing.Proportion	Estimated.Rate.Parameter..lambda.
1	0.5135	0.4998
2	0.4865	1.9721

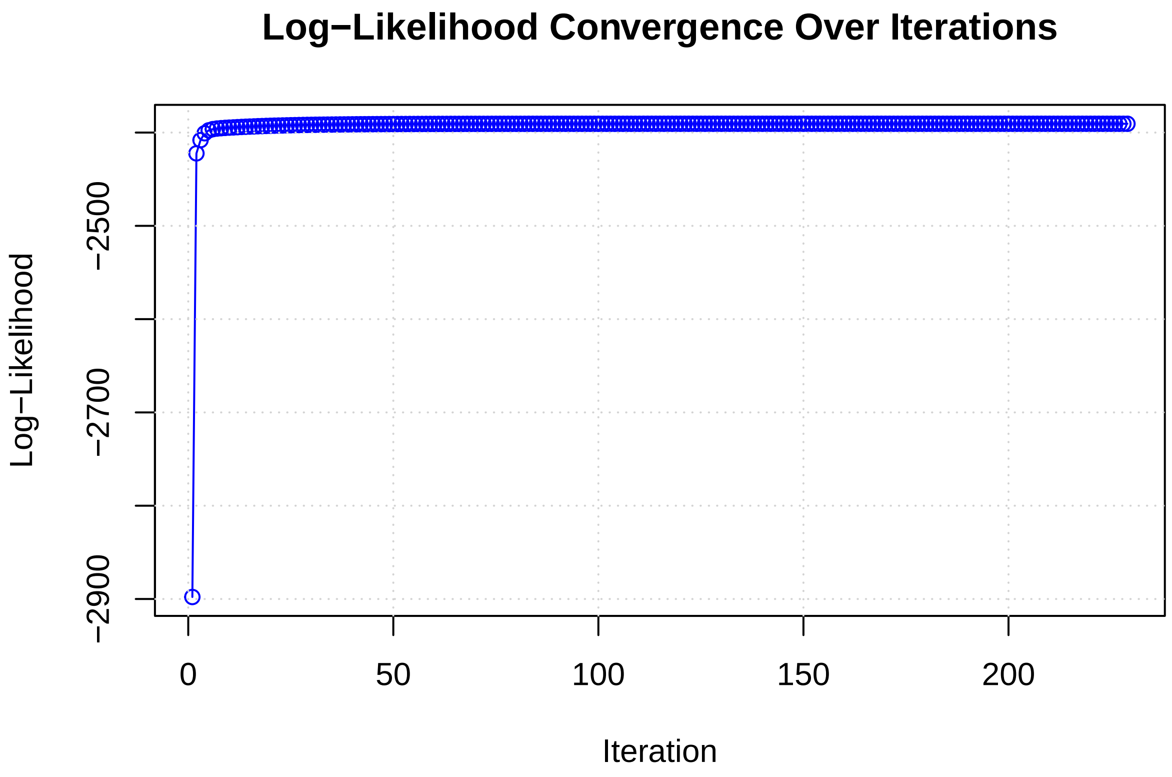
The estimated mixing proportions of component 1 and 2 are 0.5135 and 0.4865 respectively. These estimations are both 0.0135 units away from the true proportion values of 0.5 for component 1 and 0.5 for component 2.

The estimated rate parameters of component 1 and 2 are 0.4998 and 1.9721 respectively. The estimation for the first component is just 0.0002 units away from the true rate parameter of 0.5. The estimation for the second component is 0.0279 units away from the true rate parameter of 2.0.

b. Yes, the parameter estimates appear close to the true estimates. Because the algorithm converges successfully after a sufficient amount of iterations, it is ok to say that the mixture model recovers the true parameters well, as they are very similar to the estimates.

c.

```r
# Get log-likelihoods from result
log_likelihoods <- result$log_likelihood

# Plot log-likelihoods vs. number of iterations
plot(log_likelihoods, type = "o", col = "blue", xlab = "Iteration",
     ylab = "Log-Likelihood", main = "Log-Likelihood Convergence Over Iterations")
grid()
```



As the number of iterations increase, the log-likelihood always increases as well. As the model converges, the log-likelihood plateaus. This plot helps us monitor convergence of the EM algorithm and helps us determine if the algorithm needs more iterations or if it get stuck at a point.

d. You are monitoring the expected log-likelihood. From lecture slides 21 and 22 of 08 exam review of clustering review, it is shown that the iterative calculations of Q allows us to monitor convergence through expected log-likelihood.

e.

```
# sample
n_samples <- 2000
true_rates <- c(0.5, 2.0)
true_proportions <- c(0.5, 0.5)

# Sample group assignments (1 or 2) based on mixing proportions
component_labels <- sample(1:2, n_samples, replace = TRUE, prob = true_proportions)

# Stage 2: Generate data from the corresponding exponential
# distribution based on the group assignment
data <- numeric(n_samples) # Initialize the data vector

# Generate data for group 1 (rate = 0.5)
data[component_labels == 1] <- rexp(sum(component_labels == 1), rate = true_rates[1])

# Generate data for group 2 (rate = 2.0)
data[component_labels == 2] <- rexp(sum(component_labels == 2), rate = true_rates[2])

# Fit the mixture model
result <- exponentialMixture(data, K = 2)
```

```
## Convergence reached at iteration 202 with log-likelihood: -2387.018
```

```
# Create a data frame to display the results in a table
results_table <- data.frame(
  "Run" = 1:1,
  "Component" = 1:2,
  "Estimated Mixing Proportion" = round(result$pi, 4),
  "Estimated Rate Parameter (lambda)" = round(result$lambda, 4)
)

for (x in 2:5) {
  # Stage 2: Generate data from the corresponding exponential
  # distribution based on the group assignment
  data <- numeric(n_samples) # Initialize the data vector

  # Generate data for group 1 (rate = 0.5)
  data[component_labels == 1] <- rexp(sum(component_labels == 1), rate = true_rates[1])

  # Generate data for group 2 (rate = 2.0)
  data[component_labels == 2] <- rexp(sum(component_labels == 2), rate = true_rates[2])

  # Fit the mixture model
  result <- exponentialMixture(data, K = 2)

  results_table <- rbind(results_table, list(x, 1, round(result$pi, 4)[1], round(result$lambda, 4)[1]))
  results_table <- rbind(results_table, list(x, 2, round(result$pi, 4)[2], round(result$lambda, 4)[2]))
}
```

```
## Convergence reached at iteration 164 with log-likelihood: -2361.567
## Convergence reached at iteration 181 with log-likelihood: -2268.558
## Convergence reached at iteration 102 with log-likelihood: -2324.903
## Convergence reached at iteration 223 with log-likelihood: -2363.661
```

```r
# move results to a table
kable(results_table, caption = "Estimated Parameters for the Mixture Model", format = "pipe")
```

Table 2: Estimated Parameters for the Mixture Model

Run	Component	Estimated.Mixing.Proportion	Estimated.Rate.Parameter..lambda.
1	1	0.4879	1.9708
1	2	0.5121	0.4999
2	1	0.5532	1.8361
2	2	0.4468	0.4636
3	1	0.5253	0.5375
3	2	0.4747	2.1498
4	1	0.4384	0.4722
4	2	0.5616	1.8158
5	1	0.4959	0.4942
5	2	0.5041	1.9580

By running the algorithm 5 times with the same data and initialization settings, we can see that the estimates vary slightly for each run due to the random initialization. However, the estimations still converge close to the true values for each run.

f. When coding the function, I found it most challenging to understand what exactly I'm supposed to be coding for each step of the algorithm. Because the algorithm is very math heavy, it was necessary to understand the math behind the algorithm before trying to code it.

The step by step lecture notes were very helpful in explaining the intuition behind the EM algorithm. Attending class was also helpful because I was able to hear questions I didn't think about and helped my understand the content more.

Part 3

```r
# Simulation parameters
set.seed(123) # For reproducibility
n_samples <- 1000 # Number of samples
true_rates <- c(0.5, 1.5) # True rate parameters for two components
true_proportions <- c(0.6, 0.4) # True mixing proportions

# Generate synthetic data
data <- c(rexp(n_samples * true_proportions[1], rate = true_rates[1]),
          rexp(n_samples * true_proportions[2], rate = true_rates[2]))
```

a.

```r
# Fit the mixture model
result <- exponentialMixture(data, K = 2)
```

```
## Convergence reached at iteration 720 with log-likelihood: -1383.459
```

```r
# Create a data frame to display the results in a table
results_table <- data.frame(
  "Run" = 1:1,
  "Component" = 1:2,
  "Estimated Mixing Proportion" = round(result$pi, 4),
  "Estimated Rate Parameter (lambda)" = round(result$lambda, 4)
)

for (x in 2:5) {
  # Stage 2: Generate data from the corresponding exponential
  # distribution based on the group assignment
  data <- numeric(n_samples) # Initialize the data vector

  # Generate data for group 1 (rate = 0.5)
  data[component_labels == 1] <- rexp(sum(component_labels == 1), rate = true_rates[1])

  # Generate data for group 2 (rate = 2.0)
  data[component_labels == 2] <- rexp(sum(component_labels == 2), rate = true_rates[2])

  # Fit the mixture model
  result <- exponentialMixture(data, K = 2)

  results_table <- rbind(results_table, list(x, 1, round(result$pi, 4)[1], round(result$lambda, 4)[1]))
  results_table <- rbind(results_table, list(x, 2, round(result$pi, 4)[2], round(result$lambda, 4)[2]))
}
```

```
## Convergence reached at iteration 266 with log-likelihood: -2531.141
## Convergence reached at iteration 582 with log-likelihood: -2500.91
## Convergence reached at iteration 318 with log-likelihood: -2514.456
## Convergence reached at iteration 417 with log-likelihood: -2513.783
```

```r
# move results to a table
kable(results_table, caption = "Estimated Parameters for the Mixture Model", format = "pipe")
```

Table 3: Estimated Parameters for the Mixture Model

Run	Component	Estimated.Mixing.Proportion	Estimated.Rate.Parameter..lambda.
1	1	0.2327	1.9504
1	2	0.7673	0.5624
2	1	0.5073	1.5125
2	2	0.4927	0.4913
3	1	0.5218	0.5364
3	2	0.4782	1.4368
4	1	0.5569	0.5173
4	2	0.4431	1.7451
5	1	0.4131	1.6852
5	2	0.5869	0.5463

By running the algorithm 5 times with the same data and initialization settings, we can see that the estimates vary moderately for each run due to the random initialization. It is worth noting that the estimates are further away from the true values in some runs in part 3 compared to part 2.

b. The estimated mixing proportions and rate parameters differ significantly between runs. For example, in run 1, the mixing proportions are 0.2327 and 0.7673 for components 1 and 2 respectively, whereas the mixing proportions in run 3 are 0.5218 and 0.4782. This variability could indicate that the EM algorithm's results are sensitive to the starting values for mixing proportions and rate parameters.

The algorithm appears to converge to a solution that does not closely align with the true values in some runs. This variability suggests convergence to different local maxima depending on initialization. Additionally, clusters may potentially overlap as estimates overlap leading to less distinct clusters.

c. The root problem might be poor initialization of mixing proportions and rate parameters. The algorithm's reliance on random starting values without any apparent strategy increases the susceptibility to local optima. Moreover, this shows that the algorithm is sensitive to starting values. This increases the likelihood of convergence to suboptimal solutions.

d. I propose performing k-means clustering on the data to initialize mixing proportions and rate parameters. This provides a much better starting point for the EM algorithm which would help reduce variability and improve converging to an optimal solution.

```r
# Simulation parameters
set.seed(123) # For reproducibility
n_samples <- 1000 # Number of samples
true_rates <- c(0.5, 1.5) # True rate parameters for two components
true_proportions <- c(0.6, 0.4) # True mixing proportions

# Generate synthetic data
data <- c(rexp(n_samples * true_proportions[1], rate = true_rates[1]),
          rexp(n_samples * true_proportions[2], rate = true_rates[2]))

set.seed(123)
K <- 2 # how many clusters
kmeans_result <- kmeans(data, centers = 2)

# Calculate initial mixing proportions
component_labels <- kmeans_result$cluster
mixing_proportions <- table(component_labels) / length(data)

# Calculate initial rate parameters
rate_parameters <- numeric(K)
for (k in 1:K) { # k = 2 in this instance
  cluster_data <- data[component_labels == k] # data points in component k
  rate_parameters[k] <- 1 / mean(cluster_data) # rate as reciprocal of mean
}

# New exponentialMixture function with initial estimate parameters
exponentialMixture <- function(data, K, max_iter = 1000, tol = 1e-5,
                               init_pi = NULL, init_lambda = NULL) {
  n <- length(data)
  pi <- if (is.null(init_pi)) rep(1/K, K) else init_pi # mixing proportions
  lambda <- if (is.null(init_lambda)) runif(K, 0.1, 1) else init_lambda # rate parameters
  log_likelihoods <- numeric(max_iter) # store log-likelihood values

  for (iter in 1:max_iter) { # E-step: numerator of the gammas
    gamma <- matrix(NA, nrow = n, ncol = K)
    for (k in 1:K) {
      gamma[, k] <- pi[k] * dexp(data, rate = lambda[k])
    }
    row_sums <- rowSums(gamma) # denominator of the gammas
    gamma <- gamma / row_sums  # normalize probabilities

    # M-step: Update mixing proportions and rate parameters
    pi_old <- pi
    lambda_old <- lambda
    pi <- colMeans(gamma) # update mixing proportions
    for (k in 1:K) {
      lambda[k] <- sum(gamma[, k]) / sum(gamma[, k] * data) # update rate parameters
    }

    log_likelihoods[iter] <- sum(log(row_sums)) # calculate log-likelihood

    if (max(abs(pi - pi_old)) < tol && max(abs(lambda - lambda_old)) < tol) {
      log_likelihoods <- log_likelihoods[1:iter] # trim to the number of iterations
      cat("Convergence reached at iteration", iter, "with log-likelihood:", log_likelihoods[iter], "\n"
      break
    }
  }

  return(list(pi = pi, lambda = lambda, log_likelihood = log_likelihoods))
}

# Run EM function with k-means initialization
results <- exponentialMixture(data, K = K, init_pi = as.numeric(mixing_proportions), init_lambda = rate
```

```
## Convergence reached at iteration 856 with log-likelihood: -1383.459
```

```r
# Create a data frame to display the results in a table
results_table <- data.frame(
  "Run" = 1:1,
  "Component" = 1:2,
  "Estimated Mixing Proportion" = round(result$pi, 4),
  "Estimated Rate Parameter (lambda)" = round(result$lambda, 4)
)

for (x in 2:5) {
  # Stage 2: Generate data from the corresponding exponential
  # distribution based on the group assignment
  data <- numeric(n_samples) # Initialize the data vector

  # Generate data for group 1 (rate = 0.5)
  data[component_labels == 1] <- rexp(sum(component_labels == 1), rate = true_rates[1])

  # Generate data for group 2 (rate = 2.0)
  data[component_labels == 2] <- rexp(sum(component_labels == 2), rate = true_rates[2])

  # Fit the mixture model
  result <- exponentialMixture(data, K = 2)

  results_table <- rbind(results_table, list(x, 1, round(result$pi, 4)[1], round(result$lambda, 4)[1]))
  results_table <- rbind(results_table, list(x, 2, round(result$pi, 4)[2], round(result$lambda, 4)[2]))
}
```

```
## Convergence reached at iteration 601 with log-likelihood: -935.1917
## Convergence reached at iteration 523 with log-likelihood: -894.4343
## Convergence reached at iteration 360 with log-likelihood: -901.898
## Convergence reached at iteration 378 with log-likelihood: -860.606
```

```
# Move results to a table
kable(results_table, caption = "Estimated Parameters for the Mixture Model", format = "pipe")
```

Table 4: Estimated Parameters for the Mixture Model

Run	Component	Estimated.Mixing.Proportion	Estimated.Rate.Parameter..lambda.
1	1	0.4131	1.6852
1	2	0.5869	0.5463
2	1	0.2143	0.5459
2	2	0.7857	1.4029
3	1	0.3461	0.6495
3	2	0.6539	1.7023
4	1	0.5025	0.7548
4	2	0.4975	1.9570
5	1	0.8303	1.4952
5	2	0.1697	0.5087

The above code assigns initial cluster labels to data points based on k-means. After calculating proportions and rate parameters, these initial values are passed as arguments to the EM algorithm for better starting points for parameter estimation. The results are much more consistent across runs in this part compared to part a. Both the estimated mixing proportions and estimated rate parameters are now closer to the true values compared to previous results.

e. I learned that poor initialization can significantly impact convergence and accuracy of EM algorithms. The starting points of these algorithms can greatly impact convergence results, and that's why it's so important to use good initialization methods such as k-means, as these methods can greatly improve results.

When using mixture models and the EM algorithm on a real data set, it's very important to use multiple runs with different initializations to confirm that the algorithm converges to the optimal solution. When working with real data, it is also good to use precaution when selecting the number of clusters for the model. It is important to have methods to determine the optimal number of clusters.