Assignment2

Florencia Luque and Simon Schemtz

2025-03-14

```
library(ggplot2)
library(ellipse)
```

```
## Warning: package 'ellipse' was built under R version 4.4.3
```

```
##
## Adjuntando el paquete: 'ellipse'
```

```
## The following object is masked from 'package:graphics':
##
##     pairs
```

```
library(jpeg)
library(mvtnorm)
```

Introduction

This document presents the work done for Assignment 3 in the Statistical Learning course at Universidad Carlos III de Madrid. The objective of this assignment is to implement the Expectation-Maximization (EM) algorithm for estimating the parameters of a Gaussian Mixture Model (GMM) across different data dimensions.

Specifically, the assignment requires:

- Implementing the EM algorithm for Gaussian mixtures in 1D, 2D, and higher dimensions.
- Validating the implementation on synthetic datasets, providing both visual and analytical results.
- Applying the algorithm to image segmentation, demonstrating its ability to classify different regions within an image.

This report details the methodology, results, and insights gained from the implementation and testing of the EM algorithm. # Functions ## Initialization function Initialization is the first step in the EM algorithm and is crucial because convergence depends heavily on its setup, while the quality of the initial clusters influences the final solution. In this implementation, we initialize the cluster means by randomly selecting KK data points, ensuring that the means start within a reasonable range. The covariance matrices are set to identity matrices to maintain numerical stability and prevent singularities. The mixture coefficients are uniformly initialized to ensure that no cluster dominates the process at the start. This method provides a simple and effective way to initialize EM without relying on K-means, making the procedure fully autonomous.

```r
initialize_em_parameters = function(X, K) {
  N = nrow(X)  # Data rows
  D = ncol(X)  # Data columns

  # Randomly select K data points as initial means
  mu = X[sample(1:N, K), ]

  # Initialize covariance matrices as identity matrices
  sigma = array(diag(D), dim = c(D, D, K))

  # Initialize equal mixing coefficients
  pi_k = rep(1/K, K)

  return(list(mu = mu, sigma = sigma, pi_k = pi_k))
}
```

Plot 1D and 2D functions

Generate plot for 1D and 2D image

```r
plot_1D_2D = function(X, gamma, mu, sigma) {
  D = ncol(X)  # Dimension of data (1D or 2D)
  K = nrow(as.matrix(mu))  # Ensure mu is treated as a matrix
  cluster_labels = apply(gamma, 1, which.max)  # Assign each point to a cluster

  data_plot = as.data.frame(X)
  data_plot$cluster = as.factor(cluster_labels)  # Convert to categorical

  if (D == 1) {
    ggplot(data_plot, aes(x = V1, fill = cluster)) +
      geom_histogram(alpha = 0.5, bins = 30, position = "identity") +
      geom_vline(xintercept = as.vector(mu), col = "black", linetype = "dashed", size = 1) +  # Ensure
      labs(title = "1D Gaussian Mixture Clustering", x = "Value", y = "Frequency") +
      theme_minimal()

  } else if (D == 2) {
    ggplot(data_plot, aes(x = V1, y = V2, color = cluster)) +
      geom_point(alpha = 0.6) +
      geom_point(data = as.data.frame(mu), aes(x = V1, y = V2),
                 color = "black", size = 4, shape = 4) +
      labs(title = "2D Gaussian Mixture Clustering", x = "X", y = "Y") +
      theme_minimal()
  } else {
    stop("This function only supports 1D and 2D data!")
  }
}
```

Plot 3D or more function (for images)

```
reconstruct_image = function(original_image_path, em_result) {
  library(grid)  # Load the grid package for plotting images

  # Read the image
  img = readJPEG(original_image_path)
  dim_img = dim(img)  # Original dimensions

  # Get values
  gamma = em_result$gamma

  # Assign pixel to cluster
  labels = apply(gamma, 1, which.max)

  # Replace each pixel with the cluster mean
  segmented_img = em_result$mu[labels, ]

  # Reshape the segmented image back to its original dimensions
  segmented_img = array(segmented_img, dim = dim_img)
  # Return the segmented image array (optional, for further use)
  return(segmented_img)
}
```

function for log likelihood convergency

```
plot_log_likelihood = function(log_likelihood_values) {
  iterations = 1:length(log_likelihood_values)
  log_likelihood_df = data.frame(Iteration = iterations, LogLikelihood = log_likelihood_values)

  ggplot(log_likelihood_df, aes(x = Iteration, y = LogLikelihood)) +
    geom_line(color = "blue", size = 1) +
    geom_point(color = "red", size = 2) +
    labs(title = "EM Algorithm Log-Likelihood Convergence",
         x = "Iteration",
         y = "Log-Likelihood") +
    theme_minimal()
}
```

EM function

The **EM function** was implemented following the algorithm described in *Pattern Recognition and Machine Learning* by **Christopher Bishop (2006, Chapter 9)**. The Expectation-Maximization (EM) algorithm iteratively estimates the parameters of a **Gaussian Mixture Model (GMM)** by maximizing the likelihood function. In the **E-step**, it computes the **responsibilities** γ_{ik}, which represent the probability that each data point $\mathbf{x}_i$ belongs to cluster k:

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(\mathbf{x}_i | \mu_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_i | \mu_j, \mathbf{\Sigma}_j)}$$

where π_k is the **mixing coefficient**, μ_k is the **mean**, and $\mathbf{\Sigma}_k$ is the covariance matrix of cluster k. In the M-step, the parameters are updated using the computed responsibilities. The new cluster means are estimated as:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \mathbf{x}_i$$

where $N_k = \sum_{i=1}^{N} \gamma_{ik}$ represents the effective number of points assigned to cluster k. The covariance matrices are updated as:

$$\mathbf{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T + \epsilon I$$

where ϵI is a small regularization term added for numerical stability, as suggested in *Bishop (2006, Section 9.2.2)*. The algorithm iterates until the log-likelihood function:

$$\log L = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i | \mu_k, \mathbf{\Sigma}_k) \right)$$

converges. The function ensures an optimal and numerically stable implementation of EM for 1D, 2D, and high-dimensional data, including image segmentation. ## Function of EM for 1D

```
EM_GMM_1D = function(X, K, max_iter = 100, tol = 1e-6) {
  N = length(X)  # n data

  set.seed(234)
  mu = runif(K, min(X), max(X))  # Random means
  sigma = rep(var(X), K)  # Initialize  sample variance
  pi_k = rep(1/K, K)  # Equal mixture weights initially

  gamma = matrix(0, N, K)  # Responsibility matrix
  log_likelihood_values = numeric(max_iter)  # Store log-likelihood values

  log_likelihood = function() {
    sum(log(rowSums(sapply(1:K, function(k) {
      pi_k[k] * dnorm(X, mean = mu[k], sd = sqrt(sigma[k]))
    }))))
  }

  logL = log_likelihood()
  log_likelihood_values[1] = logL

  for (iter in 2:max_iter) {
    # E-step: Compute responsibilities
    for (k in 1:K) {
      gamma[, k] = pi_k[k] * dnorm(X, mean = mu[k], sd = sqrt(sigma[k]))
    }
    gamma = gamma / rowSums(gamma)

    # M-step: Update parameters
    Nk = colSums(gamma)  # Effective number of points in each cluster
```

```
    pi_k = Nk / N  # Update mixing coefficients
    mu = colSums(gamma * X) / Nk  # Update means
    sigma = colSums(gamma * (X - mu)^2) / Nk  # Update variances

    # Compute new log-likelihood
    new_logL = log_likelihood()
    log_likelihood_values[iter] = new_logL

    # Check for convergence
    if (abs(new_logL - logL) < tol) {
      message("Converged at iteration ", iter)
      log_likelihood_values = log_likelihood_values[1:iter]
      break
    }
    logL = new_logL
  }

  return(list(mu = mu, sigma = sigma, pi_k = pi_k, gamma = gamma, log_likelihood = log_likelihood_values
}
```

```
EM_GMM = function(X, K, max_iter = 100, tol = 1e-6) {
  # Check if X is a single-column matrix (1D case)
  if (ncol(X) == 1) {
    print("1D detected, calling EM_GMM_1D")
    return(EM_GMM_1D(as.vector(X), K, max_iter, tol))  # Convert to vector for 1D function
  }

  N = nrow(X)  # Number of data points
  D = ncol(X)  # Data dimensions

  params = initialize_em_parameters(X, K)
  mu = params$mu
  sigma = params$sigma
  pi_k = params$pi_k

  gamma = matrix(0, N, K)  # Responsibility matrix
  log_likelihood_values = numeric(max_iter)  # Store log-likelihood values

  log_likelihood = function() {
    sum(log(rowSums(sapply(1:K, function(k) {
      pi_k[k] * dmvnorm(X, mean = as.numeric(mu[k, , drop = FALSE]), sigma = sigma[, , k])
    }))))
  }

  logL = log_likelihood()
  log_likelihood_values[1] = logL

  for (iter in 2:max_iter) {
    for (k in 1:K) {
      gamma[, k] = pi_k[k] * dmvnorm(X, mean = as.numeric(mu[k, , drop = FALSE]), sigma = sigma[, , k])
    }
    gamma = gamma / rowSums(gamma)
```

```
    Nk = colSums(gamma)
    pi_k = Nk / N
    mu = matrix(t(gamma) %*% X / Nk, ncol = D, byrow = TRUE)

    if (K == 1) {
      mu = matrix(mu, nrow = 1, ncol = D)
    }

    for (k in 1:K) {
      X_centered = sweep(X, 2, mu[k, , drop = FALSE], FUN = "-")
      sigma[, , k] = t(X_centered) %*% (X_centered * gamma[, k]) / Nk[k] + diag(1e-6, D)
    }

    new_logL = log_likelihood()
    log_likelihood_values[iter] = new_logL

    if (abs(new_logL - logL) < tol) {
      message("Converged at iteration ", iter)
      log_likelihood_values = log_likelihood_values[1:iter]
      break
    }
    logL = new_logL
  }

  return(list(mu = mu, sigma = sigma, pi_k = pi_k, gamma = gamma, log_likelihood = log_likelihood_values
}
```

Test for different dimensions

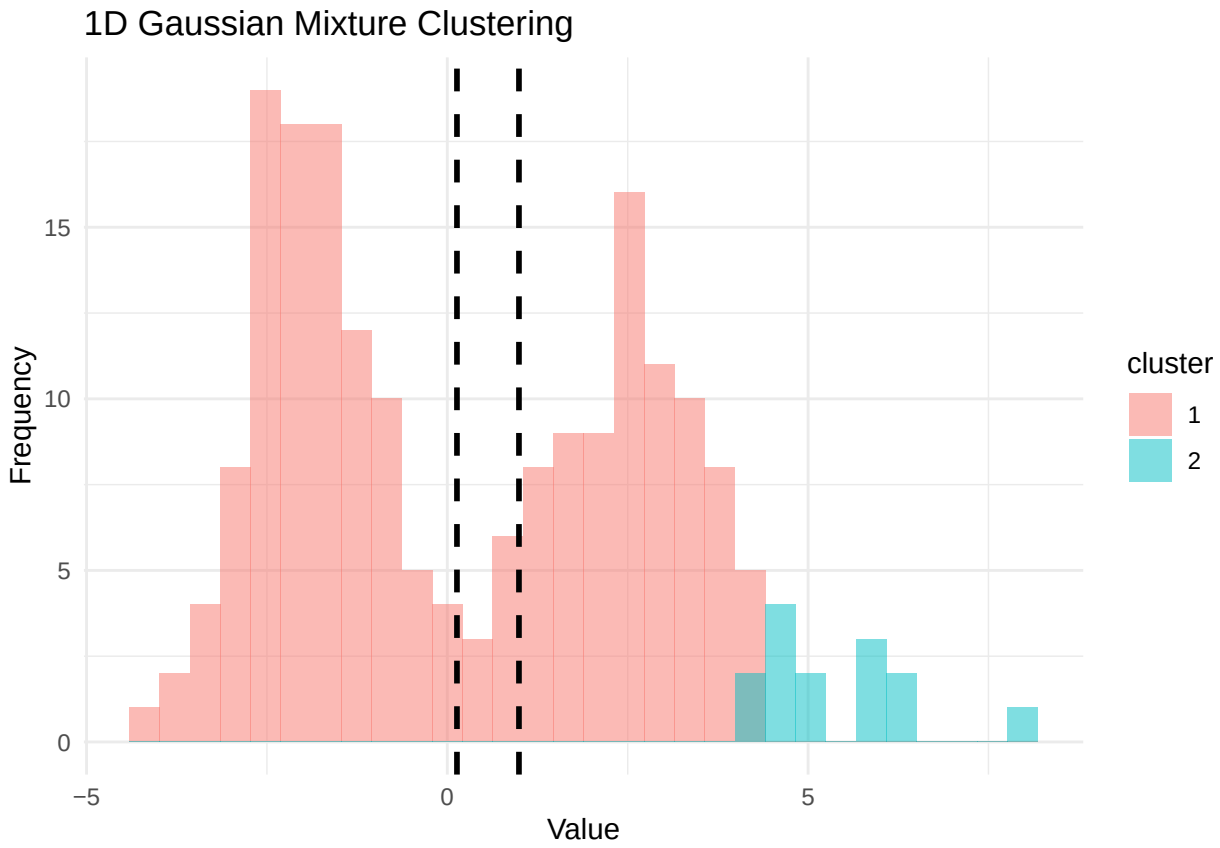
1D

```
set.seed(123)
X_1D = matrix(c(rnorm(100, mean = -2, sd = 1),
                rnorm(100, mean = 3, sd = 1.5)), ncol = 1)

# Run EM Algorithm for 1D data
result_1D = EM_GMM(X_1D, K = 2)
```

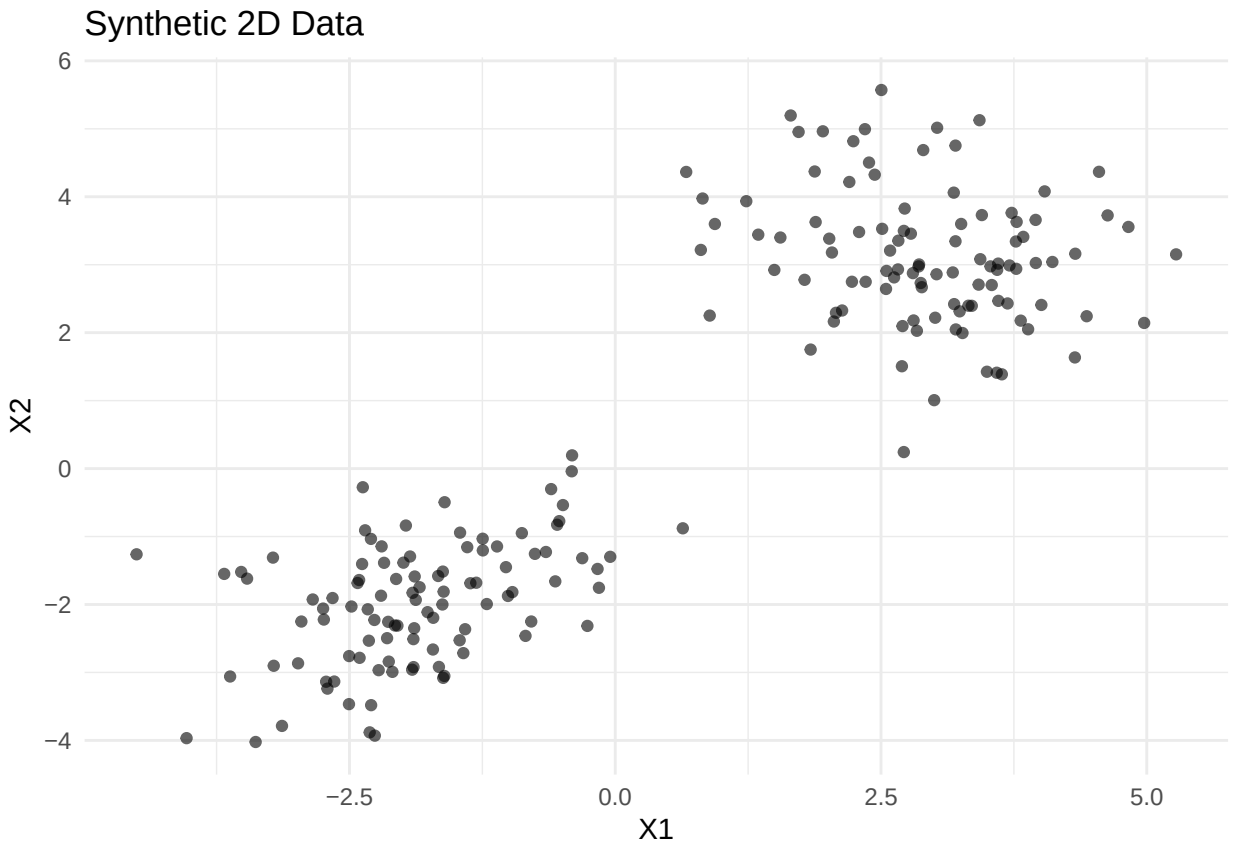
```
## [1] "1D detected, calling EM_GMM_1D"
```

```
# Corrected function call
plot_1D_2D(X_1D, result_1D$gamma, result_1D$mu, result_1D$sigma)
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



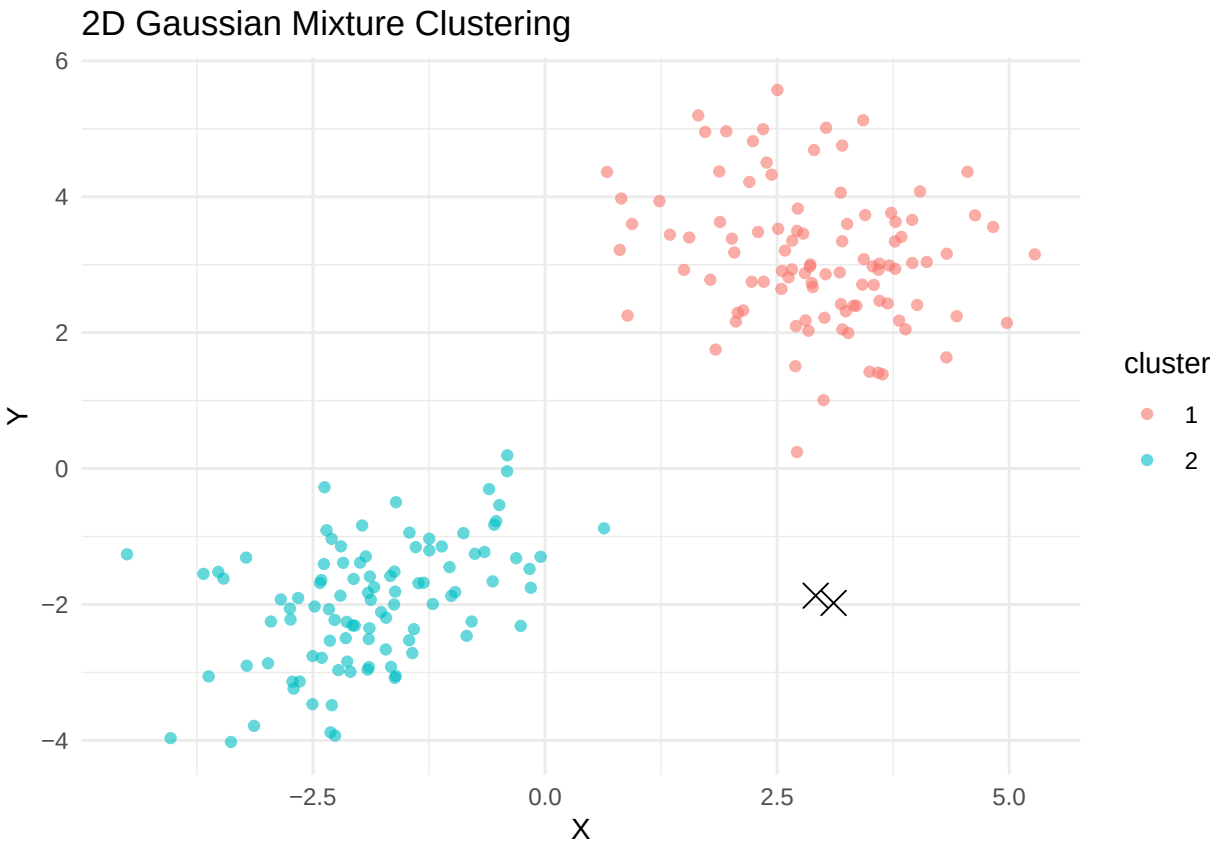

```r
# Generate synthetic 2D data
library(MASS)
set.seed(123)
X_2D = rbind(
  mvrnorm(n = 100, mu = c(-2, -2), Sigma = matrix(c(1, 0.5, 0.5, 1), ncol = 2)),
  mvrnorm(n = 100, mu = c(3, 3), Sigma = matrix(c(1, -0.3, -0.3, 1), ncol = 2))
)

# Visualize the generated data
library(ggplot2)
data_plot = as.data.frame(X_2D)
colnames(data_plot) = c("V1", "V2")
ggplot(data_plot, aes(x = V1, y = V2)) +
  geom_point(alpha = 0.6) +
  labs(title = "Synthetic 2D Data", x = "X1", y = "X2") +
  theme_minimal()
```




```
# Run EM Algorithm for 2D data
result_2D = EM_GMM(X_2D, K = 2)
```

```
## Converged at iteration 17
```

```
# Plot the clustering results
plot_1D_2D(X_2D, result_2D$gamma, result_2D$mu, result_2D$sigma)
```




```
library(jpeg)
library(grid)
# Load and reshape the image
img = readJPEG("C:/Users/flore/Desktop/git/Statistical_learning/Test_alpha/20BT.jpg")
dim_img = dim(img)
img_reshaped = matrix(img, nrow = dim_img[1] * dim_img[2], ncol = dim_img[3])

# Apply EM algorithm
result = EM_GMM(img_reshaped, K = 5)

# Reconstruct the segmented image
segmented_img = array(result$mu[apply(result$gamma, 1, which.max), ], dim = dim_img)

# Save the segmented image
writeJPEG(segmented_img, "C:/Users/flore/Desktop/git/Statistical_learning/segmented_image.jpg")

# Plot the segmented image
grid.raster(segmented_img, interpolate = FALSE)
```