K-MEANS

GANDHAR RAVINDRA PANSARE, SWARN GABA, VEDIKA HALWASIYA

STAT-S 610 INTRODUCTION TO STATISTICAL COMPUTING

INDIANA UNIVERSITY BLOOMINGTON

Project Objectives

What we want our code to do:

1. Implement a robust K-means clustering algorithm.
2. Provide interactive visualizations for understanding the algorithm's progress.
3. Evaluate the quality of clustering for different k values to guide users.

How we will know our code is working correctly ?

To ensure that the code is functioning correctly, we will validate it through a combination of **theoretical checks**, **empirical testing**, and **visual inspection.**

Few points are discussed below -

- Validation of output structure and results against expected outcomes.
- Verify that centroids stabilize after iterations, meeting the convergence criteria.
- Confirm that visualizations align with expected clustering patterns, showing distinct groups.

We will discuss additional validation checks, including unit tests and comparisons with known results, later in the presentation.

Design & Implementation

kmeans_custom

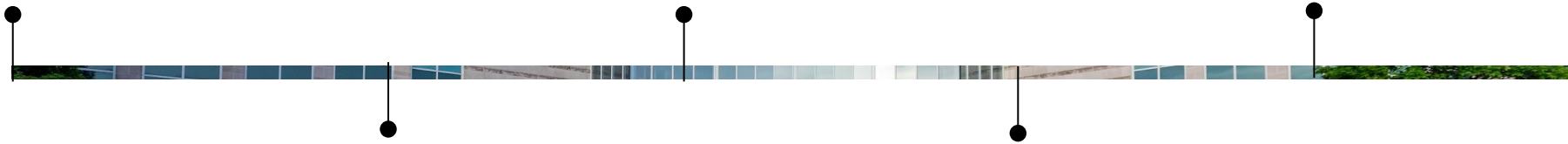
- Implements K-means clustering algorithm
- Inputs: Data, k, initial centroids (optional), max iteration and tolerance threshold (convergence criteria)
- Outputs: Final centroids, cluster assignments, iteration history

silhouette_custom

- Computes silhouette scores for clustering quality assessment
- Inputs: Data, labels (cluster assignments)
- Outputs: Silhouette score

Integration and Testing

Combines functions, generates synthetic data, and validates outputs



visualize_kmeans_progress

- Visualizes the progress of the K-means algorithm with animation and static plots
- Inputs: Data, history (centroids and labels from each iteration)
- Outputs: PNG images and MP4 animation

evaluate_k

- Evaluates clustering quality across k values
- Inputs: Data, k range
- Outputs: Silhouette scores and its plot against each k

Design Choices

Implementation of K-means

- Implements the algorithm in a simple and modular way.
- Breaks the process into clear steps: initialization, assignment, and update.
- Code is designed for reproducibility and easy debugging.

Visualization Choices

- Iterative progress visualized via scatter plots and animations.
- Helps users intuitively understand how clusters evolve.

Evaluation Metric

Silhouette scores provide an interpretable measure of cluster quality, balancing intra-cluster cohesion and inter-cluster separation.

Core Function

K-Means

kmeans_custom (K-means Algorithm)

Goal: Perform K-means clustering.

Key Steps:

1. Initialize centroids randomly (if not provided).
2. Assign data points to the nearest centroid.
3. Recompute centroids as the mean of assigned points.
4. Repeat until centroids stabilize or reach max iterations.

```r
kmeans_custom <- function(data, k, init_centroids = NULL, max_iter = 300, tol = 1e-4) {

  # Ensure the data is in matrix form
  data <- as.matrix(data)
  n <- nrow(data) # Number of data points

  # Initialize centroids randomly if not provided
  if (is.null(init_centroids)) {
    set.seed(42) # For reproducibility
    init_centroids <- data[sample(1:n, k), ]
  }

  centroids <- init_centroids # Initial centroids
  history <- list() # To store the centroids and labels for each iteration

  for (iter in 1:max_iter) {

    # Step 1: Assign each point to the nearest centroid
    distances <- as.matrix(dist(rbind(centroids, data)))[1:k, (k + 1):(k + n)] # Distance matrix
    labels <- apply(distances, 2, which.min) # Assign each point to the nearest centroid

    # Validation: Ensure the number of labels matches the number of rows in the data
    if (length(labels) != n) {
      stop("Mismatch between number of data points and cluster labels.")
    }

    # Save the current centroids and labels for visualization
    history[[iter]] <- list(centroids = centroids, labels = labels)

    # Step 2: Update centroids by computing the mean of assigned points
    new_centroids <- t(sapply(1:k, function(i) colMeans(data[labels == i, , drop = FALSE])))

    # Step 3: Check for convergence (stop if centroids don't change significantly)
    if (all(sqrt(rowSums((new_centroids - centroids)^2)) < tol)) {
      break
    }
    centroids <- new_centroids # Update centroids for the next iteration
  }

  # Return the final centroids, labels, and history of iterations
  return(list(centroids = centroids, labels = labels, history = history))
}
```

Visualizations

visualize_kmeans_progress (Visualization)

Goal: Show clustering progression over iterations with static plots and animations.

Approach:

- Save plots of cluster assignments and centroids at each iteration.
- Combine into an animated MP4 file.

```r
visualize_kmeans_progress <- function(data, history, save_png = TRUE) {

  # Prepare data for animation by combining all iterations into a single data frame
  animation_data <- do.call(rbind, lapply(1:length(history), function(iter) {

    # Extract centroids and labels from the current iteration
    centroids <- history[[iter]]$centroids
    labels <- history[[iter]]$labels

    # Validation: Ensure labels match the number of rows in the data
    if (length(labels) != nrow(data)) {
      stop("Mismatch between number of labels and number of rows in the data.")
    }

  # Save each iteration as a static PNG file
  if (save_png) {
    for (iter in 1:length(history)) {

      iteration_data <- animation_data[animation_data$iteration == iter, ]

      # Create a scatter plot for the current iteration
      plot <- ggplot(iteration_data, aes(x = x, y = y, color = cluster)) +
        geom_point(aes(shape = ifelse(is_centroid, "Centroid", "Point")), size = 2) +
        scale_shape_manual(values = c("Point" = 16, "Centroid" = 4)) +
        labs(title = paste("K-means Progress: Iteration", iter), color = "Cluster", shape = "")

      # Save the plot as a PNG file
      ggsave(filename = paste0("kmeans_iteration_", iter, ".png"), plot = plot)
    }
  }

  # Create an animated plot showing the progression of K-means
  plot <- ggplot(animation_data, aes(x = x, y = y, color = cluster)) +
    geom_point(aes(shape = ifelse(is_centroid, "Centroid", "Point")), size = 2) +
    scale_shape_manual(values = c("Point" = 16, "Centroid" = 4)) +
    labs(title = "K-means Progress: Iteration {closest_state}", color = "Cluster", shape = "") +
    transition_states(iteration, transition_length = 2, state_length = 1) +
    ease_aes("linear")

  # Save the animation as an MP4 video
  animate(plot, nframes = length(history), fps = 1, renderer = av_renderer("kmeans_animation.mp4"))
}
```

Silhouette Scores

silhouette_custom (Clustering Quality Metric)

- **Goal**: Computes silhouette scores to evaluate clustering quality.
- **Approach**:
  1. Compute intra-cluster distances (a_i).
  2. Compute inter-cluster distances (b_i).
  3. Calculate silhouette score for each point: $s_i = (b_i - a_i) / \max(a_i, b_i)$.

```r
silhouette_custom <- function(data, labels) {

  data <- as.matrix(data) # Ensure the data is a matrix
  dist_matrix <- as.matrix(dist(data)) # Compute pairwise distances

  n <- nrow(data) # Number of data points
  sil_values <- matrix(0, nrow = n, ncol = 3) # Initialize silhouette values matrix

  for (i in 1:n) {

    current_cluster <- labels[i] # Current cluster assignment

    # Intra-cluster distance: Mean distance to points in the same cluster
    same_cluster_indices <- which(labels == current_cluster)

    if (length(same_cluster_indices) > 1) {
      a_i <- mean(dist_matrix[i, same_cluster_indices[-which(same_cluster_indices == i)]])
    } else {
      a_i <- 0
    }

    # Inter-cluster distance: Minimum mean distance to other clusters
    other_cluster_distances <- sapply(unique(labels[labels != current_cluster]), function(cluster) {
      other_indices <- which(labels == cluster)
      mean(dist_matrix[i, other_indices])
    })

    b_i <- min(other_cluster_distances)

    # Silhouette score for the point
    s_i <- (b_i - a_i) / max(a_i, b_i)

    # Store the values
    sil_values[i, ] <- c(a_i, b_i, s_i)
  }

  colnames(sil_values) <- c("a", "b", "s") # Assign column names
  return(sil_values)
}
```

Evaluation

evaluate_k (Optimal k)

Goal: Determine the optimal number of clusters (k) using silhouette scores.

Approach:

- Run K-means for a range of k values.
- Compute average silhouette scores.
- Identify the k with the highest score.

```r
evaluate_k <- function(data, k_range) {

  scores <- sapply(k_range, function(k) {
    # Run K-means clustering
    result <- kmeans_custom(data, k)

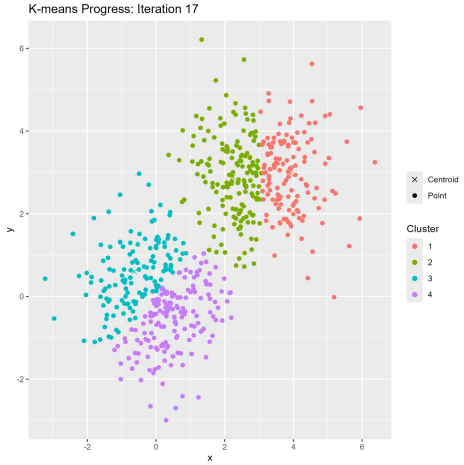
    # Compute silhouette scores
    sil_scores <- silhouette_custom(data, result$labels)

    # Return the mean silhouette score
    silhouette_score <- mean(sil_scores[, 3])
    return(silhouette_score)
  })

  # Plot silhouette scores against k
  plot(k_range, scores, type = "b", pch = 19, col = "blue",
       xlab = "Number of Clusters (k)", ylab = "Silhouette Score",
       main = "Silhouette Score vs. Number of Clusters")
  return(scores)
}
```

Demonstration !!!

Evidence of Functionality




```
### Purpose: This script integrates the custom K-means functions, runs clustering on a sample dataset,
visualizes the clustering process, and evaluates cluster quality.

#### Data Generation:

Generates a dataset of 600 points (300 points each from two Gaussian distributions) for testing the
clustering algorithm.

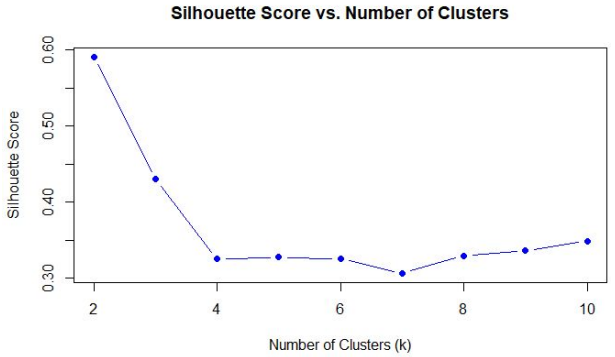
```{r}
# Generate sample data
# Create two clusters of 300 points each from a multivariate normal distribution

# Cluster 1: Centered at (0, 0) with a diagonal covariance matrix
set.seed(42) # Set seed for reproducibility
data <- as.data.frame(mvrnorm(300, mu = c(0, 0), Sigma = diag(2)))
```

```{r}
# Cluster 2: Centered at (3, 3) with the same covariance matrix
data <- rbind(data, as.data.frame(mvrnorm(300, mu = c(3, 3), Sigma = diag(2))))
```

Clustering Execution:
Uses kmeans_custom to find 4 clusters in the data.

```{r}
# Run K-means clustering on the data
# k = 4 specifies that we are looking for 4 clusters
result <- kmeans_custom(data, k = 4)
```
```



Empirical Testing

- Using testthat for systematic testing and automation.
- Created **unit tests** to confirm that individual functions behave as expected.

Test Cases for Key Functions

kmeans_custom:

- Validate the output structure: The result should include centroids, labels, and history.
- Ensure that the number of centroids in the output equals k.
- Check that the length of labels matches the number of data points.
- Confirm that centroids stabilize after convergence.

visualize_kmeans_progress:

- Confirm that the function generates:

❖ PNG files for each iteration.
❖ An MP4 animation summarizing the clustering process.

silhouette_custom:

- Check that the silhouette matrix has n rows (one per point) and 3 columns (a, b, and s).
- Validate that s(i) values are within the range [−1,1].

evaluate_k:

- Ensure that the length of the output matches the length of k_range.
- Check that the silhouette scores decrease or plateau as k increases beyond the optimal value (indicative of overfitting).

Empirical Testing Results

```
==> Testing R file using 'testthat'

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 0 ]
[ FAIL 0 | WARN 1 | SKIP 0 | PASS 0 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 0 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 1 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 2 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 3 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 4 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 5 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 6 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 7 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 8 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 9 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 10 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 11 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 12 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 13 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 14 ]
[ FAIL 0 | WARN 2 | SKIP 0 | PASS 15 ]
```

testthat::test_dir(".")

```
== Results =====================================
Duration: 2.3 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 15 ]
```

Conclusion

1. K-means algorithm implemented and tested
2. Visualizations successfully created for progression analysis
3. Silhouette scores confirm clustering quality

THANK YOU!

INDIANA UNIVERSITY
BLOOMINGTON