Lecture 20

K-Means

K-Means is one of the most commonly used clustering algorithms, particularly in unsupervised machine learning. The goal of K-Means is to partition a dataset into K distinct, non-overlapping subsets (clusters) where each data point belongs to the cluster with the nearest mean (also called the cluster centroid).

Algorithm Steps

1. **Initialize Centroids**: Choose K initial centroids (cluster centers) randomly from the data points, or use some other method (e.g., k-means++ for better initialization).
2. **Assign Data Points to Clusters**: Assign each data point to the nearest centroid based on a chosen distance metric (typically Euclidean distance). This creates K clusters.
3. **Update Centroids**: Compute the new centroid of each cluster by taking the mean of all data points assigned to that cluster.
4. **Iterate**: Repeat the assignment and update steps until the centroids no longer change significantly, or until a predefined number of iterations is reached.
5. **Convergence**: The algorithm converges when the assignments no longer change, or the centroids do not move significantly between iterations.

Strengths of K-Means

- *Simplicity*: The algorithm is easy to implement and computationally efficient, especially with a small number of clusters.
- *Scalability*: It scales well to large datasets.
- *Interpretability*: Results are easy to interpret, with each cluster represented by its centroid.
- *Efficiency*: The time complexity is linear with respect to the number of data points, making it suitable for large datasets.

Weaknesses of K-Means

- *Predefined K*: The number of clusters K must be specified before running the algorithm, which might not be known in advance.
- *Sensitivity to Initialization*: The final clusters can vary depending on the initial selection of centroids. Poor initialization can lead to suboptimal clustering.
- *Sensitivity to Outliers*: Outliers can heavily influence the mean and thus distort the resulting clusters.
- *Assumes Spherical Clusters*: K-Means assumes clusters are spherical and equally sized, which may not be true for all datasets.
- *Convergence to Local Minima*: The algorithm can converge to a local minimum of the objective function, rather than the global minimum.

Choosing the Number of Clusters

- *Elbow Method*: Plot the within-cluster sum of squares (WCSS) against the number of clusters. The optimal KKK is often at the "elbow" point where the rate of decrease sharply slows down.
- *Silhouette Score*: Measures how similar a point is to its own cluster compared to other clusters. The score ranges from -1 to 1, with higher values indicating better-defined clusters.

Applications of K-Means

- *Market Segmentation*: Grouping customers based on purchasing behavior.

- *Image Compression*: Reducing the number of colors in an image by clustering pixel values.
- Document Clustering: Organizing a collection of documents into clusters based on similarity.
- *Anomaly Detection*: Identifying data points that do not belong to any cluster (outliers).

Visualizing K-Means Clustering
K-Means clustering results can be visualized using scatter plots (in 2D or 3D) where each cluster is represented by a different color, and centroids can be marked. Additionally, the convergence process can be illustrated by showing the movement of centroids over iterations.

K-Means Variants

- ***K-Means++****:* Improves the initialization step by spreading out the initial centroids, reducing the likelihood of poor clustering results.
- ***Mini-Batch K-Means****:* A variation of K-Means that processes the data in small batches, making it faster and more suitable for large datasets.

K-Means is a foundational technique in unsupervised learning, providing a simple yet powerful method for discovering patterns in data. Despite its limitations, it remains widely used in various domains due to its efficiency and ease of use.

In K-Means clustering, the choice of distance measure plays a critical role in determining the shape and structure of the clusters. By default, K-Means typically uses Euclidean distance, which measures the straight-line distance between two points in a multidimensional space. However, other distance measures can be used depending on the characteristics of the data and the desired clustering outcome.

Effects of Modifying the Distance Measure

1. ***Euclidean Distance (Standard)****:*
    - *Behavior*: Euclidean distance tends to form spherical clusters because it measures the straight-line distance between points.
    - *Effect*: Clusters tend to be compact and of similar size, and the algorithm is sensitive to differences in scale across dimensions. It works well when clusters are well-separated and roughly circular in shape.
2. ***Manhattan Distance (L1 Norm, City Block Distance)****:*
    - *Behavior*: Manhattan distance measures the sum of absolute differences between coordinates of two points.
    - *Effect:* Clusters tend to be more box-like (aligned with the axes of the feature space) rather than spherical. This distance measure is more robust to outliers in individual dimensions, especially when dealing with high-dimensional data. It also emphasizes the structure of data when features are not necessarily on the same scale.
3. ***Minkowski Distance (Generalization of Euclidean and Manhattan)****:*
    - *Behavior*: Minkowski distance is a generalization of Euclidean and Manhattan distances, defined as $d(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$
    - *Effect*: When $p = 1$, it becomes Manhattan distance, and when $p = 2$, it is equivalent to Euclidean distance. Varying the parameter p allows a balance between the properties of Euclidean and Manhattan distances, giving more flexibility in shaping clusters.
4. ***Cosine Similarity***:
    - *Behavior:* Cosine similarity measures the cosine of the angle between two vectors, which essentially compares the orientation (direction) rather than the magnitude.

- *Effect:* This measure is useful when the magnitude of the vectors is less important than their direction. It is often used in text mining or when dealing with high-dimensional sparse data. Clusters formed using cosine similarity are less influenced by the magnitude of the features and are more about the directionality, leading to clusters that may be more aligned along certain axes.

5. ***Mahalanobis Distance***:
   - *Behavior*: Mahalanobis distance accounts for the correlation between variables and scales the distance according to the variance-covariance matrix of the data.
   - *Effect*: This measure is useful when the data has correlated features or different scales, as it considers the distribution of the data. Clusters are more elliptical, and this method works well when the clusters have different shapes and variances.
6. ***Chebyshev Distance (Maximum or L∞ Norm)****:*
   - *Behavior*: Chebyshev distance measures the maximum difference along any coordinate dimension.
   - *Effect*: Clusters tend to form hypercubes or bounding boxes. This distance is particularly useful when the data has sharp boundaries or when the largest single difference is more important than aggregate differences across dimensions.

Implications of Using Different Distance Measures

1. *Cluster Shape and Size*: Different distance measures can lead to clusters with different shapes. For example, Euclidean distance tends to form spherical clusters, while Manhattan distance can result in clusters aligned with the axes.
2. *Sensitivity to Outliers*: Some distance measures, like Euclidean distance, are sensitive to outliers because they are based on squared differences, which amplify large deviations. Others, like Manhattan distance, are more robust to outliers.
3. *Interpretation of Results:* The choice of distance measure affects the interpretability of the clusters. In some cases, different measures might reveal different aspects of the data, such as similarity in orientation (cosine similarity) versus similarity in magnitude (Euclidean distance).
4. *Suitability for Data Type*: Some distance measures are better suited for certain types of data. For example, cosine similarity is often used in text data or high-dimensional data, while Mahalanobis distance is suitable for data with correlated variables.
5. *Computational Complexity*: Different distance measures can also vary in computational complexity. For example, calculating Mahalanobis distance requires inversion of the covariance matrix, which can be computationally expensive for large datasets.

Modifying the distance measure in K-Means clustering can significantly impact the resulting clusters in terms of their shape, size, and overall structure. The choice of distance measure should be guided by the specific characteristics of the data, the goals of the analysis, and the desired properties of the clusters. Experimenting with different distance measures and evaluating the results using validation metrics can help determine the best approach for a given dataset.

K-Means clustering is known to be sensitive to the initialization of centroids, which can lead to suboptimal clustering results if the centroids are poorly initialized. Fortunately, there are several methods and strategies to reduce this sensitivity and improve the stability and quality of the clusters:

1. K-Means++ Initialization

- *Description*: K-Means++ is a smart centroid initialization technique designed to spread out the initial cluster centers. The first centroid is chosen randomly, but each subsequent centroid is

selected with a probability proportional to its squared distance from the closest already-chosen centroid.
- *Effect*: This method helps ensure that the centroids are well-separated initially, which typically leads to faster convergence and better clustering results compared to random initialization.
- *Implementation*: K-Means++ is often the default initialization method in many K-Means implementations in software libraries like Scikit-Learn in Python or kmeans in R.

2. Multiple Runs and Averaging (Ensemble Clustering)
- *Description*: This method involves running the K-Means algorithm multiple times with different random initializations and then choosing the best clustering result based on a defined metric (such as the lowest within-cluster sum of squares, WCSS).
- *Effect*: By averaging the results over multiple runs, this approach reduces the likelihood of getting stuck in local minima due to poor initialization.
- *Implementation*: Many K-Means implementations allow you to specify the number of runs (often called n_init), and they automatically return the best result.

3. Hierarchical Agglomerative Clustering Initialization
- *Description*: Use the results of a hierarchical agglomerative clustering algorithm to initialize the centroids. After performing hierarchical clustering, the centroids of the resulting clusters can be used as the initial centroids for K-Means.
- *Effect*: This method leverages the hierarchical structure of the data to find a good set of initial centroids, reducing the dependency on random initialization.
- *Implementation*: Perform hierarchical clustering first, and then extract the centroids of the clusters as the starting points for K-Means.

4. Density-Based Initialization
- *Description*: This method involves initializing centroids in regions of high data density. A common approach is to divide the data space into a grid and select initial centroids from the densest regions.
- *Effect*: By starting in dense areas, this method increases the likelihood that the centroids are representative of the data distribution, leading to more stable clustering outcomes.
- *Implementation*: This method is less common in standard libraries but can be implemented manually by identifying high-density regions and using them for initialization.

5. Maximin Initialization
- *Description*: Start by selecting the first centroid randomly. Then, for each subsequent centroid, choose the point that is farthest from the previously selected centroids.
- *Effect*: Similar to K-Means++, this method helps in spreading out the initial centroids and ensures that they are well-separated, which typically leads to better convergence.
- *Implementation*: This method is a heuristic and can be implemented by iteratively computing distances and selecting the farthest points as centroids.

6. Use of Robust Clustering Metrics
- *Description*: Instead of relying solely on the sum of squared distances to measure the quality of clustering, consider using more robust metrics, such as silhouette score, which accounts for both cohesion (how close points in a cluster are) and separation (how distinct a cluster is from others).

- *Effect*: By choosing the initialization that maximizes a more robust metric, the sensitivity to initialization can be reduced.
- *Implementation*: After multiple K-Means runs, evaluate the clustering result using a metric like the silhouette score and choose the best initialization.

7. Global Optimization Techniques

- *Description*: Techniques such as simulated annealing or genetic algorithms can be used to optimize the placement of centroids in a more global sense.
- *Effect*: These methods can escape local minima by exploring a broader search space for centroid initialization, although they are computationally more expensive.
- *Implementation*: Global optimization methods are typically more advanced and may require custom implementation, although some clustering libraries might offer these as options.

Summary

- *K-Means++* is the most widely-used method due to its simplicity and effectiveness.
- *Multiple runs* with random initializations, combined with a robust metric for choosing the best run, is a common and straightforward approach. (This method is frequently a feature of built-in k-means packages.)
- Other techniques like **hierarchical clustering initialization** and **density-based initialization** can further improve results, especially when dealing with more complex data distributions.

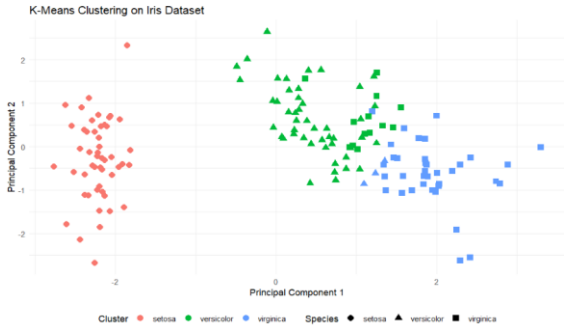
By using one or a combination of these methods, you can significantly reduce the sensitivity of K-Means to the initialization of centroids, leading to more consistent and reliable clustering results.

Let's look at the code for the basic algorithm using Euclidean distance. K-Means is a clustering algorithm and can be used in an unsupervised fashion to discover clusters in the data, but it can also be used in a semi-supervised way as a classification method.

```
# Load the iris dataset
data(iris)

# Extract the feature matrix and the true labels
iris_data <- iris[, 1:4]  # Only the numeric data
true_labels <- iris[, 5]  # The species (true labels)

set.seed(123)  # Set seed for reproducibility

# Euclidean distance function
euclidean_distance <- function(x, y) {
  sqrt(sum((x - y) ^ 2))
}

# K-Means algorithm
k_means <- function(data, k, max_iter = 100) {
  # Randomly initialize the centroids by selecting k points from the dataset
  centroids <- data[sample(1:nrow(data), k), ]

  # Create a vector to store the cluster assignments
```

```
  clusters <- rep(0, nrow(data))

  # Iterative optimization
  for (iter in 1:max_iter) {
    # Step 1: Assign clusters
    for (i in 1:nrow(data)) {
      distances <- apply(centroids, 1, function(centroid) euclidean_distance(data[i, ], centroid))
      clusters[i] <- which.min(distances)
    }

    # Step 2: Update centroids
    new_centroids <- matrix(NA, nrow = k, ncol = ncol(data))
    for (j in 1:k) {
      if (sum(clusters == j) == 0) {
        # In case a cluster is empty, reinitialize that centroid randomly
        new_centroids[j, ] <- data[sample(1:nrow(data), 1), ]
      } else {
        new_centroids[j, ] <- colMeans(data[clusters == j, , drop = FALSE])
      }
    }

    # Check for convergence (if centroids don't change)
    if (all(new_centroids == centroids)) {
      break
    }

    centroids <- new_centroids
  }

  list(centroids = centroids, clusters = clusters)
}

# Run the K-Means algorithm
k = 3
kmeans_result <- k_means(iris_data, k)

# Extract the cluster assignments
predicted_clusters <- kmeans_result$clusters

# Convert predicted_clusters to factor with levels matching true_labels
predicted_clusters <- as.factor(predicted_clusters)

# Assign appropriate labels to the predicted clusters
# This step requires a bit of manual inspection to determine which cluster corresponds to which species.
# Assuming the clusters 1, 2, 3 correspond to species "setosa", "versicolor", "virginica"
levels(predicted_clusters) <- levels(true_labels)
```

```r
# Now create the confusion matrix
conf_matrix <- confusionMatrix(predicted_clusters, true_labels)

# Print the confusion matrix
print(conf_matrix)

# Calculate overall accuracy
accuracy <- sum(diag(conf_matrix$table)) / sum(conf_matrix$table)
cat("Overall Accuracy: ", accuracy, "\n")

# Assign species names based on which cluster has the majority of each species
library(dplyr)

# Create a mapping of clusters to species based on the majority vote
mapping <- data.frame(Cluster = predicted_clusters, Species = true_labels) %>%
  group_by(Cluster) %>%
  summarize(Majority_Species = names(which.max(table(Species))))

# Apply the mapping to the predicted clusters
predicted_labels <- mapping$Majority_Species[match(predicted_clusters, mapping$Cluster)]

# Convert predicted labels to a factor with the same levels as true labels
predicted_labels <- factor(predicted_labels, levels = levels(true_labels))

# Confusion matrix with updated labels
conf_matrix <- confusionMatrix(predicted_labels, true_labels)

# Print the confusion matrix
print(conf_matrix)

# Calculate overall accuracy
accuracy <- sum(diag(conf_matrix$table)) / sum(conf_matrix$table)
cat("Overall Accuracy: ", accuracy, "\n")

# PCA for visualization
pca_result <- prcomp(iris_data, scale. = TRUE)
iris_pca <- data.frame(pca_result$x[, 1:2], Cluster = as.factor(predicted_clusters), Species =
true_labels)

# Plot the clusters
library(ggplot2)
ggplot(iris_pca, aes(x = PC1, y = PC2, color = Cluster, shape = Species)) +
  geom_point(size = 3) +
  labs(title = "K-Means Clustering on Iris Dataset",
       x = "Principal Component 1", y = "Principal Component 2") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



In this code example, we've used PCA to create a plot, however, you can also just select two of the original variables for this purpose. This code also includes two methods for creating the confusion matrix.

Running K-Means multiple times with different random initializations of centroids is a common technique to reduce sensitivity to the initial starting conditions. This technique is often referred to as the **"Multiple Runs"** or **"Random Restarts"** method. The goal is to choose the best solution (clustering) from these multiple runs based on a criterion like the total within-cluster sum of squares (WCSS).

```
# Load necessary library
library(dplyr)
library(caret)
library(ggplot2)

# Define the Euclidean distance function
euclidean_distance <- function(x1, x2) {
  return(sqrt(sum((x1 - x2)^2)))
}

# Define the K-Means function
kmeans_custom <- function(data, k, max_iters = 100) {
  # Randomly select k initial centroids
  set.seed(Sys.time())  # Change the seed based on current time for randomness
  centroids <- data[sample(1:nrow(data), k), ]

  for (i in 1:max_iters) {
    # Assign clusters based on closest centroid
    clusters <- apply(data, 1, function(row) {
      distances <- apply(centroids, 1, euclidean_distance, x2 = row)
      return(which.min(distances))
    })

    # Calculate new centroids
    new_centroids <- sapply(1:k, function(cluster) {
      return(colMeans(data[clusters == cluster, ]))
    })

    new_centroids <- t(new_centroids)
```

```
    # Check for convergence (if centroids do not change)
    if (all(centroids == new_centroids)) {
      break
    }

    centroids <- new_centroids
  }

  return(list(clusters = clusters, centroids = centroids))
}

# Function to calculate the Within-Cluster Sum of Squares (WCSS)
calculate_wcss <- function(data, clusters, centroids) {
  wcss <- 0
  for (i in 1:nrow(data)) {
    cluster <- clusters[i]
    centroid <- centroids[cluster, ]
    wcss <- wcss + sum((data[i, ] - centroid)^2)
  }
  return(wcss)
}

# Multiple runs K-Means
multiple_runs_kmeans <- function(data, k, num_runs = 10, max_iters = 100) {
  best_wcss <- Inf
  best_clusters <- NULL
  best_centroids <- NULL

  for (run in 1:num_runs) {
    set.seed(run)  # Set a seed for reproducibility
    kmeans_result <- kmeans_custom(data, k, max_iters)
    wcss <- calculate_wcss(data, kmeans_result$clusters, kmeans_result$centroids)

    if (wcss < best_wcss) {
      best_wcss <- wcss
      best_clusters <- kmeans_result$clusters
      best_centroids <- kmeans_result$centroids
    }
  }

  return(list(clusters = best_clusters, centroids = best_centroids, wcss = best_wcss))
}

# Prepare the iris dataset (excluding the species "setosa" and using only numeric variables)
iris_data <- iris[iris$Species != "setosa", ]
iris_data <- iris_data %>% select(-Species)
true_labels <- factor(iris[iris$Species != "setosa", ]$Species)
```

```
# Apply K-Means with multiple runs
k <- 2  # Setting k=2 for simplicity
num_runs <- 10
kmeans_result <- multiple_runs_kmeans(iris_data, k, num_runs)

# Manually adjust predicted cluster labels to match true labels
predicted_clusters <- factor(kmeans_result$clusters, levels = 1:k)
levels(predicted_clusters) <- levels(true_labels)

# Confusion matrix comparison
conf_matrix <- confusionMatrix(predicted_clusters, true_labels)
print(conf_matrix)

# Visualize the best clustering result
iris_plot_data <- iris[iris$Species != "setosa", ]
iris_plot_data$Cluster <- predicted_clusters

ggplot(iris_plot_data, aes(x = Petal.Length, y = Petal.Width, color = Cluster)) +
  geom_point(size = 3) +
  labs(title = "K-Means Clustering Result", x = "Petal Length", y = "Petal Width") +
  theme_minimal()

# Output the best WCSS
cat("Best WCSS from multiple runs: ", kmeans_result$wcss, "\n")
```

We would need more time to go over the other improvements made to k-means that are already out there, but this should give you some sense of the power of k-means, and why it's so popular, as well as the options that may be available to you in the built-in packages.

Resources:

1. https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/
2. https://www.datacamp.com/tutorial/k-means-clustering-r
3. https://www.geeksforgeeks.org/k-means-clustering-in-r-programming/
4. https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/kmeans

5. https://towardsdatascience.com/k-means-clustering-concepts-and-implementation-in-r-for-data-science-32cae6a3ceba
6. https://medium.com/@ahmadbintangarif/k-means-clustering-with-r-abdb10448cc1