IE-494: Big Data Processing

Implementation Project: Parallelized K-Medoids Clustering Algorithm (Customer Segmentation)

Project Group: T23

Name: Desai Aditya Veeral
ID: 202201451

Name: Parth Vadodaria
ID: 202201174

Assigned by: P.M. Jat

Problem Statement:

- Perform “Customer Segmentation” using K-Medoids, a variation of K-Means.

Solution:

- Our work centers around the K-Medoid clustering. *K-Medoids is a method that is an extension of K-Means, where K-Medoids is specifically designed to overcome the weakness of K-Means which is vulnerable to the influence of outlier data [2].* It uses an actual data point as a center of clusters, called a medoid, and not mean. It aims to apply K-Medoid clustering towards customer segmentation in possibly noise and outlier-prone real-world data.
- The goal is to classify customers based on purchasing behaviour through attributes, Invoice no. and Monetary value (Quantity * Unit Price) to establish well-defined customer groups for focused marketing and business planning.
- This can help in identification of several distinct customer groups, such as:
    - **Frequent, high-value customers**: Customers who purchase often and spend large amounts.
    - **Infrequent, low-value customers**: Customers who make only a few purchases but spend little.
    - **Frequent, low-value customers**: Customers who make regular but small-value transactions.
    - **Infrequent, high-value customers**: Customers who purchase occasionally but spend a lot when they do.

- Practical Application of this can be:
    - Businesses can allocate resources such as customer service or sales efforts to high-value segments, ensuring that customer acquisition and retention strategies are tailored to each group’s characteristics.

Approach/Algorithm Implemented:

- Our approach is mentioned below:
    - Randomly select k medoids from the dataset. Each medoid represents a cluster center.
    - Calculate the euclidean distance between each data point and each of the k medoids.
    - Assign each data point to the cluster whose medoid is closest to it. The closest medoid is determined by the smallest distance between the point and each of the k medoids.
    - For each cluster, update the medoid by selecting the point within that cluster that minimizes the total distance to all other points in the cluster.

This new point becomes the new center for that cluster. The process of finding the best medoid for each cluster is repeated for all k clusters.

- Repeat steps 2 to 4 iteratively until the positions of the medoids converges.
- After convergence, calculate the clustering cost, which is the sum of squared distances from each point to its assigned medoid.
- Plot the cost for different values of k to identify the optimal number of clusters using the Elbow Method.
- Plot the clusters using a scatter plot of the final clusters to all data points, and label them with the respective cluster number.

- Required Pseudo-Code

```
Begin function assign_cluster(point, medoids):
     Initialize an empty list to store distances
     For each medoid in medoids:
         Compute the distance from the point to the
medoid using the chosen distance metric (e.g.,
Euclidean)
         Append the computed distance to the list
     Assign the point to the cluster corresponding to
the medoid with the minimum distance
     Return the cluster ID

 End function
```

- **assign_cluster(point, medoids)**: function calculates the distance from the given point to all the medoids. The point is assigned to the cluster of the nearest medoid.

```
 Begin function update_medoids(data, medoids, k,
max_iterations):
    For each iteration in range(max_iterations):
        Initialize an empty dictionary to store
clusters
        For each data point in the dataset:
            Assign the point to the nearest medoid
using assign_cluster()
            Append the point to the corresponding
cluster in the dictionary
```

```
        Initialize an empty list for the new medoids
        For each cluster ID in range(k):
            For each candidate point in the cluster:
                Compute the total distance from the
candidate to all other points in the cluster
                Select the candidate with the minimum
total distance as the new medoid for that cluster
            Add the new medoid to the new_medoids
list

        If the new medoids are very close to the old
medoids (i.e., convergence is reached):
            Print "Converged after iteration X."
            Break the loop
        Else:
            Update medoids to the new_medoids

    Return the final medoids

 End function
```

- **update_medoids(data, medoids, k, max_iterations)**: function updates the medoids by calculating the total distance of each point in a cluster to every other point. The point with the minimum total distance is selected as the new medoid for that cluster. The process continues until convergence.

```
 Begin function compute_cost(data, medoids):
    Initialize total_cost to 0
    For each data point in the dataset:
       Assign the point to the nearest medoid using
assign_cluster()
        Compute the squared distance from the point
to its assigned medoid
        Add the squared distance to total_cost

    Return total_cost

 End function
```

- **compute_cost(data, medoids)**: function calculates the total cost of the clustering as the sum of squared distances between each data point and its assigned medoid.

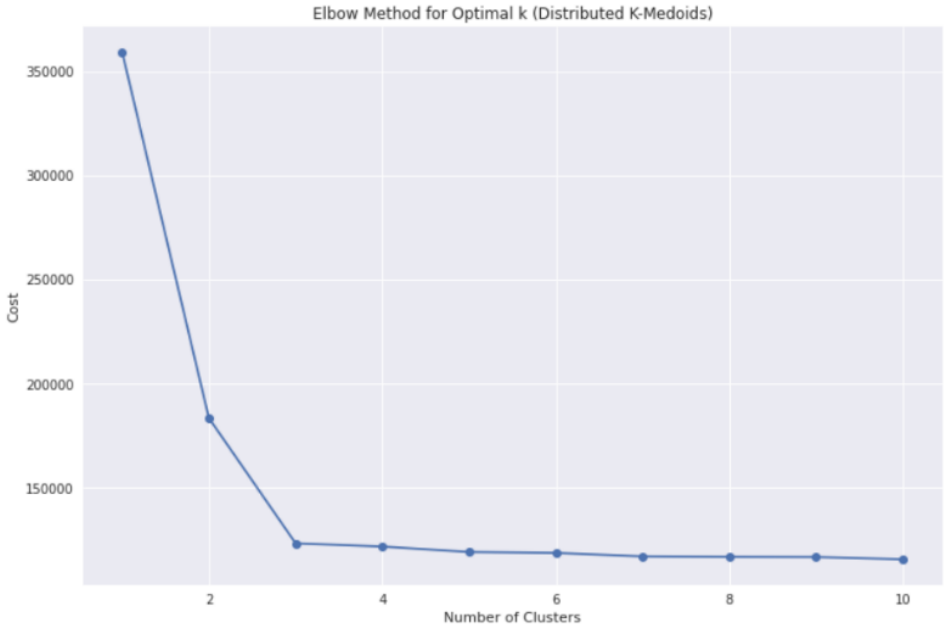
```
 Begin function k_medoids_clustering(data, k,
max_iterations):
    Initialize medoids randomly from the dataset
    For i in range(1, k + 1):  # Iterate for each k
        Run update_medoids() to update the medoids
        Compute the cost of the current clustering
using compute_cost()
        Store the cost for the current k

    Plot the cost vs. k (Elbow Method) to determine
the optimal number of clusters

    Assign each data point to the final cluster based
on the nearest medoid
    Plot the customer clusters and highlight the
medoids

 End function
```

- **k_medoids_clustering(data, k, max_iterations)**: The main function that runs the K-Medoids algorithm. It initializes the medoids randomly, updates the medoids iteratively, and computes the cost for each clustering step. Finally, it assigns clusters to data points and visualizes the results, including the elbow plot and customer clusters.

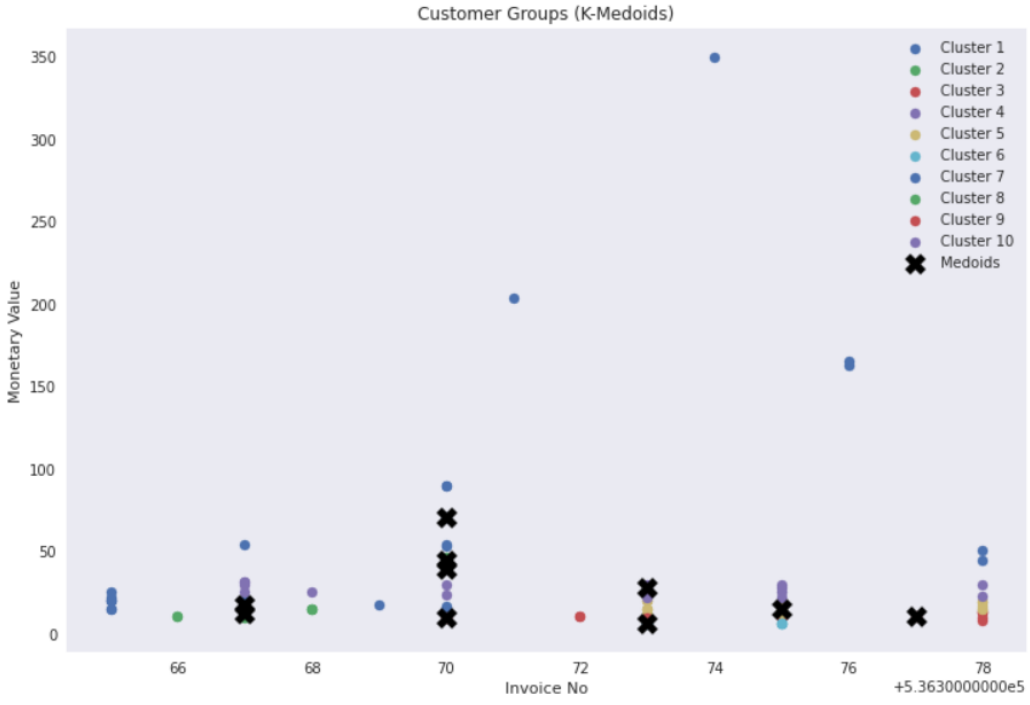
Dataset:

- The data used in this implementation comes from the Kaggle online repository. The data link: https://www.kaggle.com/code/hellbuoy/online-retail-k-means-hierarchical-clustering/input. The dataset contains online transaction data from online retail stores.

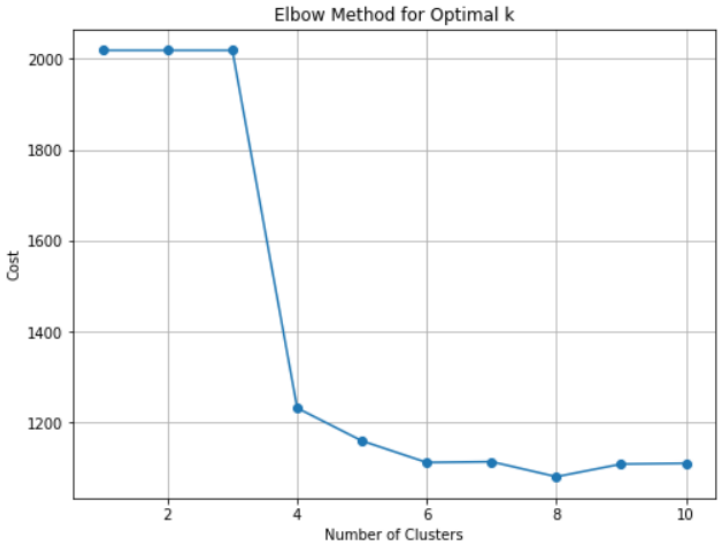
Results:



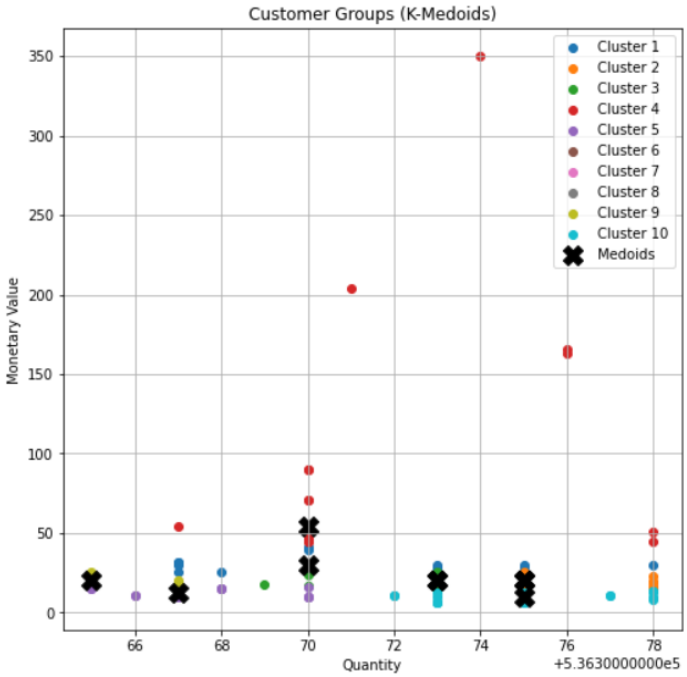
a. **Elbow graph of Distributed K-medoid cluster algorithm**



b. **Scatter Plot of Distributed K-medoid cluster algorithm**



c. <u>Elbow graph of Non-distributed K-medoid clustering using inbuilt function:</u> *KMedoids from sklearn_extra.cluster*



b. <u>Elbow graph of Non-distributed K-medoid clustering using inbuilt function:</u> *KMedoids from sklearn_extra.cluster*

Observations:

- From (a) and (c),
  - As k increases, the cost decreases because having more clusters allows each cluster to capture smaller and more specific groups of points, reducing the overall distance from points to their medoids.
  - The "elbow" point on the graph is the point where the rate of cost reduction diminishes significantly. Before the elbow, adding more clusters results in a substantial decrease in cost. After the elbow, the decrease in cost becomes smaller and less significant. This point suggests the optimal number of clusters.
  - We observe that the elbow occurs at k = 3 and k = 4 for (a) and (c) respectively, that is likely the optimal number of clusters for segmenting customers based on their Invoice No. and Monetary Value.

- From (b) and (d),
  - **Scatter plot represents**
    - frequent and high-value customers → top-right
    - infrequent and low-value customers → bottom-left
    - frequent and low-value customers → bottom-right
    - infrequent, high-value customers → top-left
  - Most clusters are concentrated near lower values of Monetary Value, indicating that many customers have low spending patterns. There are a few scattered points at higher Monetary Value values, representing outliers or unique customer behaviours.
  - Outliers may represent high-value customers or rare purchasing behaviours.

- Practical Implications which may be interpreted from above are:
  - Clusters with distinct spending patterns can be targeted with tailored marketing strategies.
    - Low-spending customers: Offer promotions or upsell strategies to increase spending.
    - High-spending customers: Prioritize loyalty programs or exclusive offers to retain these valuable customers.
  - High Monetary Value customers (outliers) may represent a significant portion of revenue. Special attention can be given to these customers for retention and engagement.

References