**K – MEDOID CLUSTERING**

**Aim**

K-medoid is a clustering algorithm whose main goal is to partition a given dataset into K clusters. The primary goal is to find the medoids of each cluster. A medoid is a data point in a cluster that minimizes the distance to all other points in the same cluster. We are clustering the Iris dataset to K clusters.

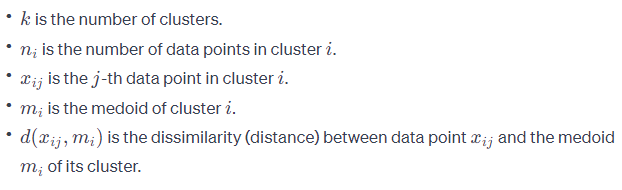
**Problem Description**

The iris dataset contains measurements of sepal length, sepal width, petal length, and petal width for three species of iris flowers, setosa, versicolor, and virginica. Our objective is to cluster the Iris dataset into k groups using the k-medoid algorithm, where k is the number of desired clusters.

**Implementation**

Start with choosing a database. Select the number of clusters to be identified. Initially, select k random points as the medoids from given n data points of dataset. Associate each data point to the closest medoid by using any of the most common distance metrics. Calculate the cost as the total sum of the distances of the data points from the assigned medoid.





Swap one medoid point with a non-medoid point and recalculate the cost. If the calculated cost with new medoid is more than the previous cost, we undo the swap, and the algorithm converges. Else we change one of the medoids to another data point and try the process upto allowed iterations and check if the respective cost is higher than the previous iteration or not.

**Algorithm**

1. Randomly select K data points from the dataset as initial cluster centroids.
2. Assign each data point to the nearest centroid by measuring the distance between each data point to the centroids.
3. For each non-medoid point in the cluster, calculate the total cost to all other points in the cluster. Select the data point with the minimum total dissimilarity as the new medoid for that cluster.
4. Convergence can be determined by checking whether the medoids remains unchanged between iterations.

**Code**

import pandas as pd

import numpy as np

import random as rn

import matplotlib.pyplot as plt

data = pd.read\_csv(r'C:\ardhra\Kmedoid\iris.csv')

print("Dataset size")

print("Rows {} Columns {}".format(data.shape[0], data.shape[1]))

df = data.copy()

x = data.iloc[:, :4].values

def distance(x, y):

return np.linalg.norm(x - y)

#calculating the Euclidean distance.

def initial\_seeds\_selection(x, k):

seeds = rn.sample(range(len(x)), k)

return [tuple(x[i]) for i in seeds]

#This is a python function for selecting initial seed points for the k-medoid clustering algorithm. The selected seed points are chosen randomly from the input dataset.

def make\_cluster(x, medoids):

cost = 0

clusters = {i: [] for i in range(len(medoids))}

for point in x:

distances = [distance(point, medoid) for medoid in medoids]

min\_distance = min(distances)

min\_index = distances.index(min\_distance)

cost += min\_distance

clusters[min\_index].append(tuple(point))

return clusters, cost

#assigning each data point in the datset to the nearest medoid and calculate the cost of the clustering.

def kmedoids(x, k, maxiter):

medoids = initial\_seeds\_selection(x, k)

clusters, cost = make\_cluster(x, medoids)

min\_cost = cost

for \_ in range(maxiter):

swap\_candidate = rn.choice(x)

if swap\_candidate not in medoids:

for i, medoid in enumerate(medoids):

temp = medoids.copy()

temp[i] = tuple(swap\_candidate)

new\_clusters, new\_cost = make\_cluster(x, temp)

if new\_cost < min\_cost:

min\_cost = new\_cost

clusters = new\_clusters

medoids = temp

else:

continue

return clusters, medoids, min\_cost

#implementing the k medoid algorithm with a specific number of iterations. That is, initialize the medoids, clusters, and cost using the initial seeds selection and cluster formation. Iterate for a maximum number of times to refine the medoids. Choose a random non-medoid point as a swap candidate. Check if the swap candidate is not already a medoid. Try swapping the candidate with each medoid and check if it reduces the cost. Make clusters and calculate the cost with the swapped medoid. If the new cost is lower, update the medoids, clusters and minimum cost. Return the final clusters , medoids and the minimum cost after the specified number of iterations.

k\_clusters = 3

max\_iterations = 100

final\_clusters, final\_medoids, final\_cost = kmedoids(x, k\_clusters, max\_iterations)

print("Clusters:")

for i, cluster in final\_clusters.items():

print(f"Cluster {i + 1}: {cluster}")

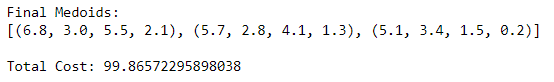
print("\nFinal Medoids:")

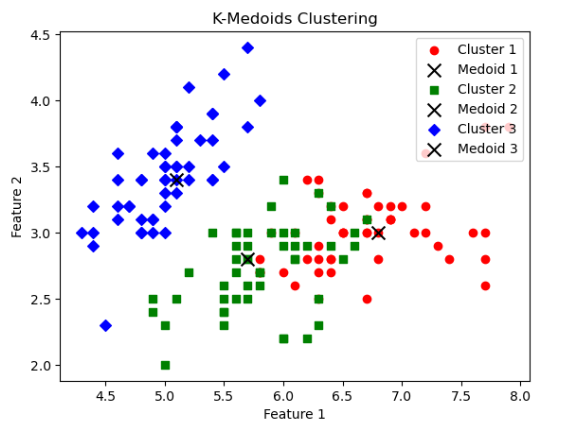
print(final\_medoids)

print("\nTotal Cost:", final\_cost)

plot\_clusters(x, final\_clusters, final\_medoids)

**Output**

****

****