Experiment No. 7

**Aim: I**mplementation of Clustering algorithm (K-means/K-medoids).

**Theory:**

K-Means is a popular clustering algorithm used for partitioning a dataset into distinct groups or clusters based on similarity. The goal of K-Means is to assign each data point to one of K clusters in a way that minimizes the total within-cluster variance. It is an iterative algorithm that gradually refines the cluster assignments by optimizing a cost function.

Algorithm Steps:

1. Initialization: Choose the number of clusters, K, and initialize K cluster centroids randomly or using a specific strategy, such as randomly selecting K data points as initial centroids.

2. Assignment: Assign each data point to the nearest cluster centroid based on a distance metric, commonly using Euclidean distance.

3. Update: Recalculate the centroids of each cluster by taking the mean of all data points assigned to that cluster.

4. Convergence Check: Check if the centroids have changed significantly from the previous iteration. If they have not, or a predetermined number of iterations has been reached, the algorithm stops.

5. Repeat Assignment and Update: Repeat the assignment and update steps iteratively until convergence or until a maximum number of iterations is reached.

Key Concepts:

- \*\*Cluster Centroid:\*\* Each cluster is represented by its centroid, which is the mean of all data points assigned to that cluster.

- \*\*Within-Cluster Variance:\*\* The total within-cluster variance is the sum of squared distances between each data point and its cluster centroid. Minimizing this variance results in tighter, more coherent clusters.

- \*\*Choosing K:\*\* The choice of the number of clusters, K, is critical. One common approach is the "elbow method," where the within-cluster variance is plotted for different values of K. The "elbow point" in the plot signifies a good trade-off between fitting the data closely and avoiding overfitting.

- Initialization Impact: The algorithm's final solution can be sensitive to the initial centroid placement. Poor initialization can lead to suboptimal solutions or slow convergence. Strategies like K-Means++ aim to mitigate this issue by initializing centroids in a smart way.

Advantages:

- Efficient for large datasets.

- Simple to implement and understand.

- Scales well to high-dimensional data.

- Can be applied to various types of data.

Limitations:

- Requires the number of clusters K to be specified beforehand.

- Sensitive to initial centroid placement, which can lead to convergence to local optima.

- Assumes clusters are spherical and equally sized, which might not always hold true.

- Not suitable for non-linear or complex cluster shapes.

- Prone to outliers, as they can significantly affect centroid placement.

**Code:**

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

import java.util.Scanner;

class Point {

double x, y;

public Point(double x, double y) {

this.x = x;

this.y = y;

}

// Override equals and hashCode methods to compare

points by rounded values

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null || getClass() != obj.getClass())

return false;

Point other = (Point) obj;

return Double.compare(Math.round(x),

Math.round(other.x)) == 0 &&

Double.compare(Math.round(y),

Math.round(other.y)) == 0;

}

@Override

public int hashCode() {

return Double.hashCode(Math.round(x)) +

Double.hashCode(Math.round(y));

}

}

class Cluster {

Point centroid;

List<Point> points;

public Cluster(Point centroid) {

this.centroid = centroid;

this.points = new ArrayList<>();

}

public void clearPoints() {

points.clear();

}

public void addPoint(Point point) {

points.add(point);

}

}

public class KMeans {

private int k;

private int maxIterations;

private List<Point> data;

private List<Cluster> clusters;

public KMeans(int k, int maxIterations, List<Point>

data) {

this.k = k;

this.maxIterations = maxIterations;

this.data = data;

this.clusters = new ArrayList<>();

}

public void initializeClusters() {

Random rand = new Random();

if (data.size() < k) {

throw new IllegalArgumentException("Not enough

data points to initialize " + k + " clusters.");

}

for (int i = 0; i < k; i++) {

Point randomPoint =

data.get(rand.nextInt(data.size()));

Cluster cluster = new Cluster(randomPoint);

clusters.add(cluster);

}

}

public double distance(Point p1, Point p2) {

return Math.sqrt(Math.pow(p1.x - p2.x, 2) +

Math.pow(p1.y - p2.y, 2));

}

public void assignPointsToClusters() {

for (Cluster cluster : clusters) {

cluster.clearPoints();

}

for (Point point : data) {

Cluster nearestCluster = null;

double minDistance = Double.MAX\_VALUE;

for (Cluster cluster : clusters) {

double d = distance(point,

cluster.centroid);

if (d < minDistance) {

minDistance = d;

nearestCluster = cluster;

}

}

if (nearestCluster != null) {

nearestCluster.addPoint(point);

}

}

// Reassign unassigned points

for (Point point : data) {

if (!isAssigned(point)) {

Cluster nearestCluster = null;

double minDistance = Double.MAX\_VALUE;

for (Cluster cluster : clusters) {

double d = distance(point,

cluster.centroid);

if (d < minDistance) {

minDistance = d;

nearestCluster = cluster;

}

}

if (nearestCluster != null) {

nearestCluster.addPoint(point);

}

}

}

}

public boolean isAssigned(Point point) {

for (Cluster cluster : clusters) {

if (cluster.points.contains(point)) {

return true;

}

}

return false;

}

public void updateClusterCentroids() {

for (Cluster cluster : clusters) {

double sumX = 0;

double sumY = 0;

List<Point> points = cluster.points;

int numPoints = points.size();

for (Point point : points) {

sumX += point.x;

sumY += point.y;

}

if (numPoints > 0) {

cluster.centroid.x = sumX / numPoints;

cluster.centroid.y = sumY / numPoints;

}

}

}

public void runKMeans() {

initializeClusters();

for (int iteration = 0; iteration < maxIterations;

iteration++) {

assignPointsToClusters();

updateClusterCentroids();

}

}

public List<Cluster> getClusters() {

return clusters;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

List<Point> data = new ArrayList<>();

List<Point> originalData = new ArrayList<>();

System.out.println("Enter the number of data

points:");

int numPoints = scanner.nextInt();

for (int i = 0; i < numPoints; i++) {

System.out.println("Enter data point " + (i +

1) + " (x y):");

double x = scanner.nextDouble();

double y = scanner.nextDouble();

Point point = new Point(x, y);

data.add(point);

originalData.add(point);

}

System.out.println("\nEnter the number of clusters

(k):");

int k = scanner.nextInt();

System.out.println("\nEnter the maximum number of

iterations:");

int maxIterations = scanner.nextInt();

KMeans kMeans = new KMeans(k, maxIterations,

data);

kMeans.runKMeans();

List<Cluster> clusters = kMeans.getClusters();

for (int i = 0; i < clusters.size(); i++) {

Cluster cluster = clusters.get(i);

System.out.println("\nCluster " + (i + 1) + "

Centroid: (" + cluster.centroid.x + ", " +

cluster.centroid.y + ")");

System.out.println("Points in Cluster " + (i +

1) + ": " + cluster.points.size());

System.out.println("Original Points in Cluster

" + (i + 1) + ":");

for (Point originalPoint : originalData) {

if

(cluster.points.contains(originalPoint)) {

System.out.println("(" +

originalPoint.x + ", " + originalPoint.y + ")");

}

}

}

scanner.close();

}

}

**OUTPUT:**

Enter the number of data points:

6

Enter data point 1 (x y):

1 2

Enter data point 2 (x y):

2 3

Enter data point 3 (x y):

3 4

Enter data point 4 (x y):

10 12

Enter data point 5 (x y):

12 14

Enter data point 6 (x y):

11 13

Enter the number of clusters (k):

2

Enter the maximum number of iterations:

100

Cluster 1 Centroid: (11.5, 13.5)

Points in Cluster 1: 3

Original Points in Cluster 1:

(11.5, 13.5)

(12.0, 14.0)

(11.0, 13.0)

Cluster 2 Centroid: (1.5, 2.5)

Points in Cluster 2: 3

Original Points in Cluster 2:

(1.0, 2.0)

(2.0, 3.0)

(1.5, 2.5)

// K- means algorithm

**Conclusion:** Thus, we have implemented K-means algorithm (Clustering algorithm) using python programming language. K-Means is a widely used clustering algorithm due to its simplicity and efficiency. While it has its limitations, it serves as a foundation for more advanced clustering methods and provides valuable insights into data grouping and pattern discovery. Understanding its theoretical underpinnings helps in making informed decisions about its application and potential modifications.