**Experiment No. 8**

**Aim:** Implementation of anyone Hierarchical Clustering method.

**Theory:**

**Hierarchical Clustering : -**

There are two types of hierarchical clustering: Agglomerative and Divisive. In

the former, data points are clustered using a bottom-up approach starting with

individual data points, while in the latter top-down approach is followed where

all the data points are treated as one big cluster and the clustering process

involves dividing the one big cluster into several small clusters.

**Steps to Perform Hierarchical Clustering : -**

Following are the steps involved in agglomerative clustering:

● At the start, treat each data point as one cluster. Therefore, the number

of clusters at the start will be K, while K is an integer representing the

number of data points.

● Form a cluster by joining the two closest data points resulting in K-1

clusters.

● Form more clusters by joining the two closest clusters resulting in K-2

clusters.

● Repeat the above three steps until one big cluster is formed.

● Once the single cluster is formed, dendrograms are used to divide into

multiple clusters depending upon the problem. We will study the

concept of dendrogram in detail in an upcoming section.

There are different ways to find distance between the clusters. The distance

itself can be Euclidean or Manhattan distance. Following are some of the

options to measure distance between two clusters:

● Measure the distance between the closest points of two clusters.

● Measure the distance between the farthest points of two clusters.

● Measure the distance between the centroids of two clusters.

● Measure the distance between all possible combination of points

between the two clusters and take the mean.

Role of Dendrograms in Agglomerative Hierarchical Clustering

As we discussed in the last step, the role of dendrogram starts once the big cluster is

formed. Dendrogram will be used to split the clusters into multiple cluster of related data

points depending upon our problem.

**Code:**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

public class HierarchicalClusteringSimple {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of data points: ");

int numPoints = scanner.nextInt();

List<double[]> dataPoints = generateData(numPoints); // Generate userdefined

data

List<Cluster> clusters = initializeClusters(dataPoints);

int clusterNumber = 1;

while (clusters.size() > 1) {

int[] closestPair = findClosestClusters(clusters);

Cluster mergedCluster = mergeClusters(clusters.remove(closestPair[1]),

clusters.remove(closestPair[0]));

clusters.add(mergedCluster);

System.out.println("Cluster " + clusterNumber + ":");

printCluster(mergedCluster, 1);

System.out.println();

// Calculate and print the distance matrix

double[][] distanceMatrix = calculateDistanceMatrix(clusters);

System.out.println("Distance Matrix:");

printDistanceMatrix(distanceMatrix);

System.out.println();

clusterNumber++;

}

// Handle the case when there's only one cluster left

if (clusters.size() == 1) {

System.out.println("Final Cluster:");

printCluster(clusters.get(0), 1);

}

}

private static List<double[]> generateData(int numPoints) {

List<double[]> data = new ArrayList<>();

Scanner scanner = new Scanner(System.in);

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

System.out.print("Enter the number of dimensions (2 or 3): ");

int numDimensions = scanner.nextInt();

double[] point = new double[numDimensions];

System.out.println("Enter " + numDimensions + " values for data point

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

for (int j = 0; j < numDimensions; j++) {

point[j] = scanner.nextDouble();

}

data.add(point);

}

return data;

}

private static List<Cluster> initializeClusters(List<double[]> data) {

List<Cluster> clusters = new ArrayList<>();

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

clusters.add(new Cluster(data.get(i), i));

}

return clusters;

}

private static double calculateDistance(Cluster cluster1, Cluster cluster2) {

double[] centroid1 = cluster1.getCentroid();

double[] centroid2 = cluster2.getCentroid();

double sumSquaredDifferences = 0;

for (int i = 0; i < centroid1.length; i++) {

double difference = centroid1[i] - centroid2[i];

sumSquaredDifferences += difference \* difference;

}

return Math.sqrt(sumSquaredDifferences);

}

private static int[] findClosestClusters(List<Cluster> clusters) {

int[] closestPair = { 0, 1 };

double minDistance = calculateDistance(clusters.get(0), clusters.get(1));

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

for (int j = i + 1; j < clusters.size(); j++) {

double distance = calculateDistance(clusters.get(i),

clusters.get(j));

if (distance < minDistance) {

minDistance = distance;

closestPair[0] = i;

closestPair[1] = j;

}

}

}

return closestPair;

}

private static Cluster mergeClusters(Cluster cluster1, Cluster cluster2) {

double[] mergedCentroid = new double[cluster1.getCentroid().length];

for (int i = 0; i < mergedCentroid.length; i++) {

mergedCentroid[i] = (cluster1.getCentroid()[i] +

cluster2.getCentroid()[i]) / 2;

}

List<Integer> mergedDataPoints = new

ArrayList<>(cluster1.getDataPoints());

mergedDataPoints.addAll(cluster2.getDataPoints());

return new Cluster(mergedCentroid, mergedDataPoints);

}

private static void printCluster(Cluster cluster, int level) {

StringBuilder indentation = new StringBuilder();

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

indentation.append(" ");

}

if (cluster.getDataPoints().size() > 1) {

System.out.println(indentation.toString() + "Data Points: " +

cluster.getDataPoints());

for (Cluster child : cluster.getChildren()) {

printCluster(child, level + 1);

}

} else {

System.out.println(indentation.toString() + "Data Point: " +

cluster.getDataPoints().get(0));

}

}

private static double[][] calculateDistanceMatrix(List<Cluster> clusters) {

int numClusters = clusters.size();

double[][] distanceMatrix = new double[numClusters][numClusters];

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

for (int j = 0; j < numClusters; j++) {

if (i != j) {

distanceMatrix[i][j] = calculateDistance(clusters.get(i),

clusters.get(j));

} else {

distanceMatrix[i][j] = 0.0; // Diagonal elements are 0

}

}

}

return distanceMatrix;

}

private static void printDistanceMatrix(double[][] distanceMatrix) {

for (int i = 0; i < distanceMatrix.length; i++) {

for (int j = 0; j < distanceMatrix[0].length; j++) {

System.out.printf("%.2f ", distanceMatrix[i][j]);

}

System.out.println();

}

}

public static class Cluster {

private double[] centroid;

private List<Integer> dataPoints;

private List<Cluster> children;

public Cluster(double[] centroid, int dataPoint) {

this.centroid = centroid;

this.dataPoints = new ArrayList<>();

this.children = new ArrayList<>();

dataPoints.add(dataPoint);

}

public Cluster(double[] centroid, List<Integer> dataPoints) {

this.centroid = centroid;

this.dataPoints = dataPoints;

this.children = new ArrayList<>();

}

public double[] getCentroid() {

return centroid;

}

public List<Integer> getDataPoints() {

return dataPoints;

}

public List<Cluster> getChildren() {

return children;

}

}

}

Enter the number of data points: 10

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 1:

3604

502.54

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 2:

8435

159.42

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 3:

4848

364.69

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 4:

7225

364.69

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 5:

1975

117.11

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 6:

2542

364.69

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 7:

4398

502.54

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 8:

49

502.54

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 9:

4031

263.33

Enter the number of dimensions (2 or 3): 2

Enter 2 values for data point 10:

7911

263.33

Cluster 1:

Data Points: [8, 6]

Distance Matrix:

0.00 4843.17 1251.61 3623.62 1673.98 1070.91 3555.00 4313.64 622.11

4843.17 0.00 3592.87 1227.29 6460.14 5896.57 8393.02 534.20 4226.41

1251.61 3592.87 0.00 2377.00 2883.65 2306.00 4800.98 3064.68 633.76

3623.62 1227.29 2377.00 0.00 5255.83 4683.00 7177.32 693.45 3010.56

1673.98 6460.14 2883.65 5255.83 0.00 618.70 1964.19 5937.80 2255.22

1070.91 5896.57 2306.00 4683.00 618.70 0.00 2496.81 5369.96 1672.60

3555.00 8393.02 4800.98 7177.32 1964.19 2496.81 0.00 7865.64 4167.22

4313.64 534.20 3064.68 693.45 5937.80 5369.96 7865.64 0.00 3698.43

622.11 4226.41 633.76 3010.56 2255.22 1672.60 4167.22 3698.43 0.00

Cluster 2:

Data Points: [9, 1]

Distance Matrix:

0.00 1251.61 3623.62 1673.98 1070.91 3555.00 622.11 4578.27

1251.61 0.00 2377.00 2883.65 2306.00 4800.98 633.76 3328.53

3623.62 2377.00 0.00 5255.83 4683.00 7177.32 3010.56 960.32

1673.98 2883.65 5255.83 0.00 618.70 1964.19 2255.22 6198.72

1070.91 2306.00 4683.00 618.70 0.00 2496.81 1672.60 5633.09

3555.00 4800.98 7177.32 1964.19 2496.81 0.00 4167.22 8129.22

622.11 633.76 3010.56 2255.22 1672.60 4167.22 0.00 3962.22

4578.27 3328.53 960.32 6198.72 5633.09 8129.22 3962.22 0.00

Cluster 3:

Data Points: [5, 4]

Distance Matrix:

0.00 1251.61 3623.62 3555.00 622.11 4578.27 1370.70

1251.61 0.00 2377.00 4800.98 633.76 3328.53 2592.46

3623.62 2377.00 0.00 7177.32 3010.56 960.32 4968.04

3555.00 4800.98 7177.32 0.00 4167.22 8129.22 2224.94

622.11 633.76 3010.56 4167.22 0.00 3962.22 1961.15

4578.27 3328.53 960.32 8129.22 3962.22 0.00 5914.57

1370.70 2592.46 4968.04 2224.94 1961.15 5914.57 0.00

Cluster 4:

Data Points: [8, 6, 0]

Distance Matrix:

0.00 2377.00 4800.98 3328.53 2592.46 941.99

2377.00 0.00 7177.32 960.32 4968.04 3316.67

4800.98 7177.32 0.00 8129.22 2224.94 3860.71

3328.53 960.32 8129.22 0.00 5914.57 4270.02

2592.46 4968.04 2224.94 5914.57 0.00 1663.04

941.99 3316.67 3860.71 4270.02 1663.04 0.00

Cluster 5:

Data Points: [8, 6, 0, 2]

Distance Matrix:

0.00 7177.32 960.32 4968.04 2846.64

7177.32 0.00 8129.22 2224.94 4330.75

960.32 8129.22 0.00 5914.57 3799.25

4968.04 2224.94 5914.57 0.00 2126.37

2846.64 4330.75 3799.25 2126.37 0.00

Cluster 6:

Data Points: [9, 1, 3]

Distance Matrix:

0.00 2224.94 4330.75 7653.01

2224.94 0.00 2126.37 5440.70

4330.75 2126.37 0.00 3322.39

7653.01 5440.70 3322.39 0.00

Cluster 7:

Data Points: [8, 6, 0, 2, 5, 4]

Distance Matrix:

0.00 7653.01 3274.53

7653.01 0.00 4380.57

3274.53 4380.57 0.00

Cluster 8:

Data Points: [8, 6, 0, 2, 5, 4, 7]

Distance Matrix:

0.00 6016.50

6016.50 0.00

Cluster 9:

Data Points: [8, 6, 0, 2, 5, 4, 7, 9, 1, 3]

Distance Matrix:

0.00

Final Cluster:

Data Points: [8, 6, 0, 2, 5, 4, 7, 9, 1, 3]

**Conclusion:** Thus, in this experiment, we have implemented Hierarchical Clustering method using agglomerative algorithm.