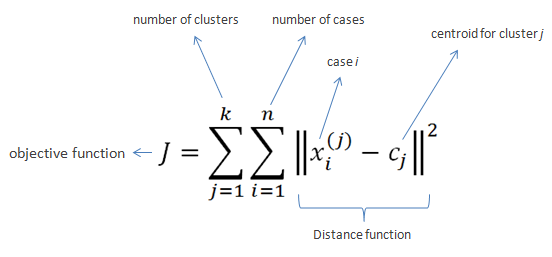
Experiment No: 5

**Aim:** Implementation of K-MEANS algorithm (Java, C, C++).

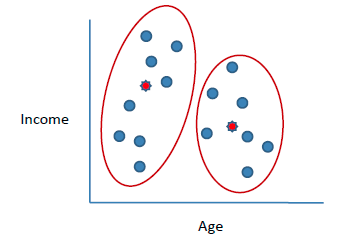
**Theory:**

K-Means clustering intends to partition *n* objects into *k* clusters in which each object belongs to the cluster with the nearest mean. This method produces exactly *k* different clusters of greatest possible distinction. The best number of clusters *k* leading to the greatest separation (distance) is not known as a priori and must be computed from the data. The objective of K-Means clustering is to minimize total intra-cluster variance, or, the squared error function:



**Algorithm:**

1. Clusters the data into *k* groups where *k* is predefined.
2. Select *k* points at random as cluster centers.
3. Assign objects to their closest cluster center according to the *Euclidean distance* function.
4. Calculate the centroid or mean of all objects in each cluster.
5. Repeat steps 2, 3 and 4 until the same points are assigned to each cluster in consecutive rounds.



K-Means is relatively an efficient method. However, we need to specify the number of clusters, in advance and the final results are sensitive to initialization and often terminates at a local optimum. Unfortunately there is no global theoretical method to find the optimal number of clusters. A practical approach is to compare the outcomes of multiple runs with different *k* and choose the best one based on a predefined criterion. In general, a large *k* probably decreases the error but increases the risk of overfitting.

***Example*:**

Suppose we want to group the visitors to a website using just their age (a one-dimensional space) as follows:

15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65

**Initial clusters:**

Centroid (C1) = 16 [16]

Centroid (C2) = 22 [22]

Iteration **1**:

C1 = 15.33 [15,15,16]

C2 = 36.25 [19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65]

Iteration **2**:

C1 = 18.56 [15,15,16,19,19,20,20,21,22]

C2 = 45.90 [28,35,40,41,42,43,44,60,61,65]

Iteration **3**:

C1 = 19.50 [15,15,16,19,19,20,20,21,22,28]

C2 = 47.89 [35,40,41,42,43,44,60,61,65]

Iteration **4**:

C1 = 19.50 [15,15,16,19,19,20,20,21,22,28]

C2 = 47.89 [35,40,41,42,43,44,60,61,65]

No change between iterations 3 and 4 has been noted. By using clustering, 2 groups have been identified 15-28 and 35-65. The initial choice of centroids can affect the output clusters, so the algorithm is often run multiple times with different starting conditions in order to get a fair view of what the clusters should be.

**Application of K-means**

### **Vector quantization**

Vector quantization of colors present in the image above into Voronoi cells using *k*-means.

*k*-means originates from signal processing, and still finds use in this domain. For example, incomputer graphics, color quantization is the task of reducing the color palette of an image to a fixed number of colors *k*. The *k*-means algorithm can easily be used for this task and produces competitive results. A use case for this approach is image segmentation. Other uses of vector quantization include non-random sampling, as *k*-means can easily be used to choose *k* different but prototypical objects from a large data set for further analysis.

### **Cluster analysis**

In cluster analysis, the *k*-means algorithm can be used to partition the input data set into *k* partitions (clusters).

However, the pure *k*-means algorithm is not very flexible, and as such is of limited use (except for when vector quantization as above is actually the desired use case). In particular, the parameter *k* is known to be hard to choose (as discussed above) when not given by external constraints. Another limitation of the algorithm is that it cannot be used with arbitrary distance functions or on non-numerical data. For these use cases, many other algorithms have been developed since.

### **Feature learning**

*k*-means clustering has been used as afeature learning (or dictionary learning) step, in either (semi-)supervised learning or unsupervised learning. The basic approach is first to train a *k*-means clustering representation, using the input training data (which need not be labelled). Then, to project any input datum into the new feature space, we have a choice of "encoding" functions, but we can use for example the thresholded matrix-product of the datum with the centroid locations, the distance from the datum to each centroid, or simply an indicator function for the nearest centroid,or some smooth transformation of the distance. Alternatively, by transforming the sample-cluster distance through a Gaussian RBF, one effectively obtains the hidden layer of a radial basis function network.

**Complexity of K-Means Algorithm is O( n \* K \* I \* d )**

**Where,**

n = number of points, K = number of clusters,

I = number of iterations,

d = number of attributes

**Limitations of K-means**

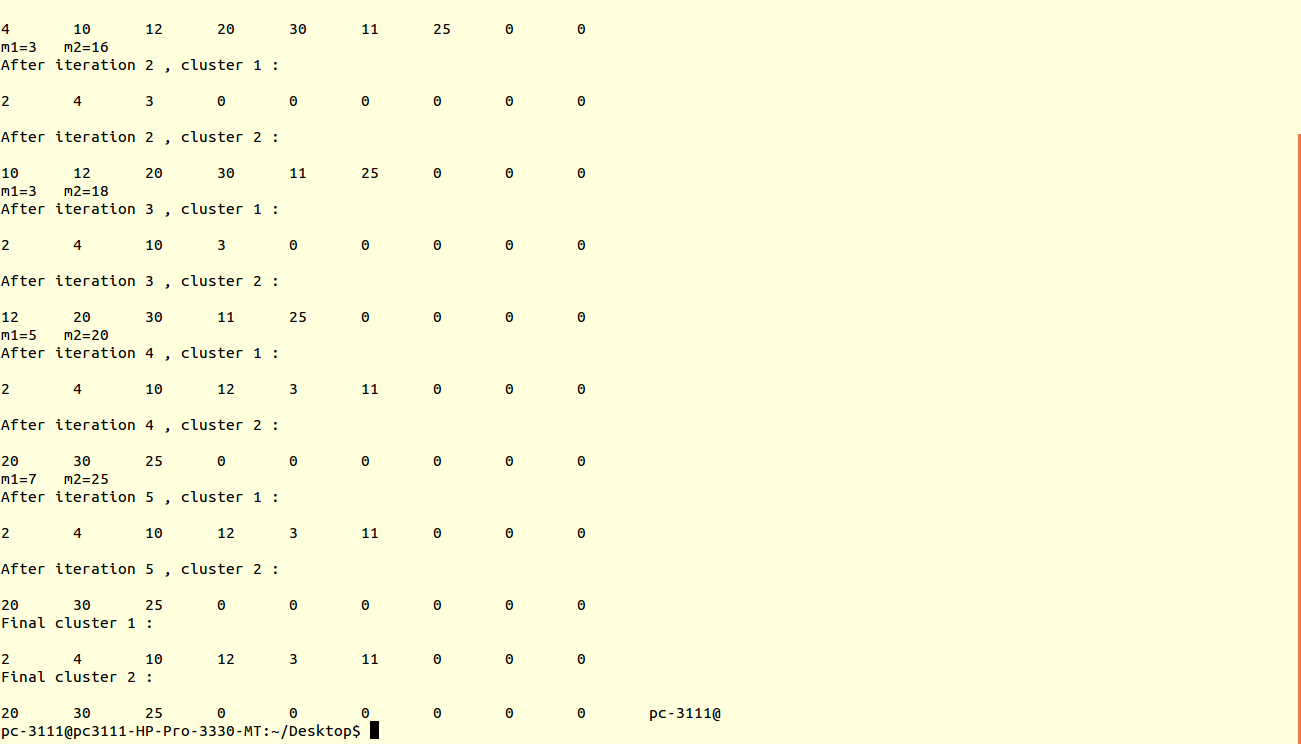
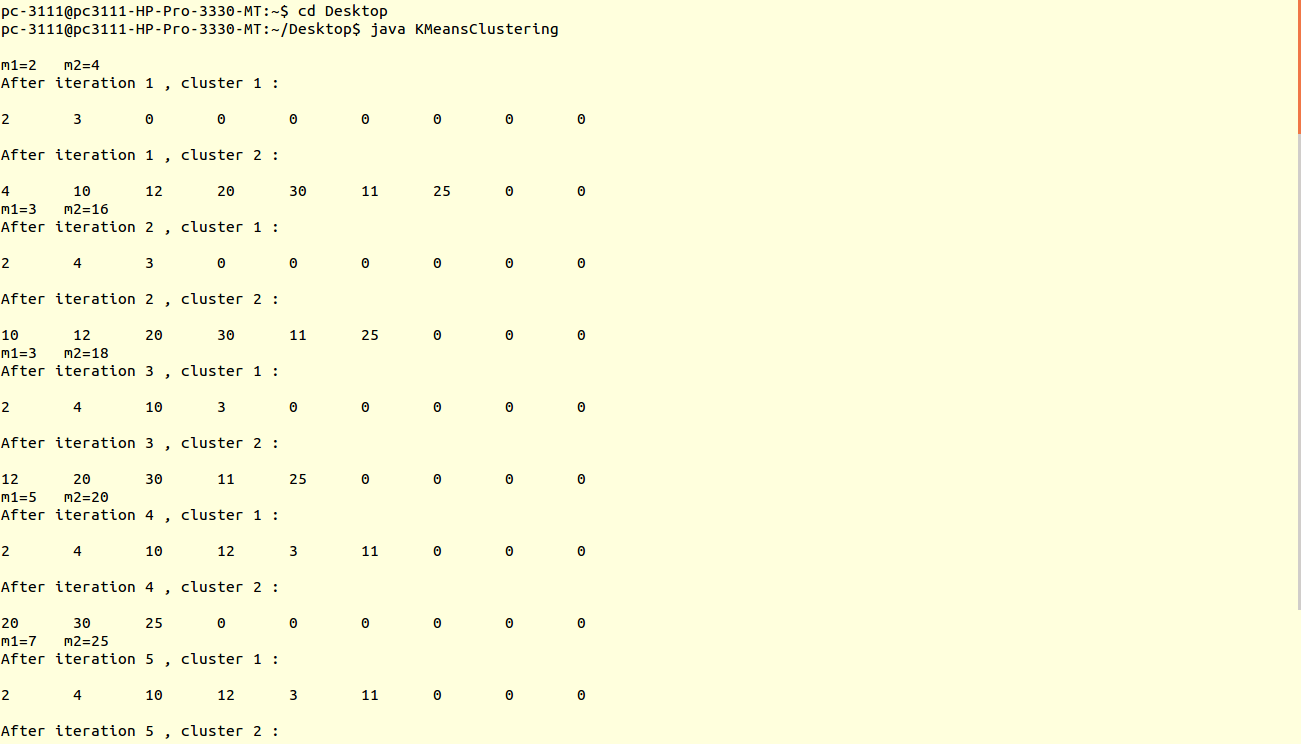
* K-means has problems when clusters are of differing
  + Sizes
  + Densities
  + Non-globular shapes
* Problems with outliers
* Empty clusters

**Program:**

public class KMeansClustering {  
  
public static void main(String args[]) {  
 int arr[] = {2, 4, 10, 12, 3, 20, 30, 11, 25}; // initial data  
 int i, m1, m2, a, b, n = 0;  
 boolean flag;  
 float sum1, sum2;  
 a = arr[0];  
 b = arr[1];  
 m1 = a;  
 m2 = b;  
 int cluster1[] = new int[arr.length], cluster2[] = new int[arr.length];  
 do {  
 sum1 = 0;  
 sum2 = 0;  
 cluster1 = new int[arr.length];  
 cluster2 = new int[arr.length];  
 n++;  
 int k = 0, j = 0;  
 for (i = 0; i < arr.length; i++) {  
 if (Math.abs(arr[i] - m1) <= Math.abs(arr[i] - m2)) {  
 cluster1[k] = arr[i];  
 k++;  
 } else {  
 cluster2[j] = arr[i];  
 j++;  
 }  
 }  
 System.out.println();  
 for (i = 0; i < k; i++) {  
 sum1 = sum1 + cluster1[i];  
 }  
 for (i = 0; i < j; i++) {  
 sum2 = sum2 + cluster2[i];  
 }  
 //printing Centroids/Means\  
 System.out.println("m1=" + m1 + " m2=" + m2);  
 a = m1;  
 b = m2;  
 m1 = Math.round(sum1 / k);  
 m2 = Math.round(sum2 / j);  
 flag = !(m1 == a && m2 == b);  
  
 System.out.println("After iteration " + n + " , cluster 1 :\n"); //printing the clusters of each iteration  
 for (i = 0; i < cluster1.length; i++) {  
 System.out.print(cluster1[i] + "\t");  
 }  
  
 System.out.println("\n");  
 System.out.println("After iteration " + n + " , cluster 2 :\n");  
 for (i = 0; i < cluster2.length; i++) {  
 System.out.print(cluster2[i] + "\t");  
 }  
 } while (flag);  
  
 System.out.println("Final cluster 1 :\n"); // final clusters  
 for (i = 0; i < cluster1.length; i++) {  
 System.out.print(cluster1[i] + "\t");  
 }  
 System.out.println();  
 System.out.println("Final cluster 2 :\n");  
 for (i = 0; i < cluster2.length; i++) {  
 System.out.print(cluster2[i] + "\t");  
 }  
}

}

**Output:**

****

**Conclusion:** Thus, java program for K-means Algorithm has been successfully studied and implemented.