ST. FRANCIS INSTITUTE OF TECHNOLOGY

MT. POINSUR, BORIVALI (W), MUMBAI

Lab Manual of Data Warehouse and Mining

Name: Keegan Vaz
Sec: TE CMPN-B
Roll No.: 28
PID: 192120

Experiment 5: Implementation of Clustering Algorithm K Means

Aim: Implementation of clustering algorithms like K-means.

Theory:

1. Partition clustering: Introduction and its Applications Ans:

   This clustering method classifies the information into multiple groups based on the characteristics and similarity of the data. It's the data analysts to specify the number of clusters that has to be generated for the clustering methods. In the partitioning method when database(D) contains multiple(N) objects then the partitioning method constructs user-specified(K) partitions of the data in which each partition represents a cluster and a particular region. There are many algorithms that come under partitioning methods, some of the popular ones are K-Mean, PAM(K-Medoids), CLARA algorithm (Clustering Large Applications) etc.

   Applications:

   - Group related documents for browsing.
   - Group genes and proteins that have similar functionality.
   - Group stocks with similar price fluctuations.
   - Reduce the size of large data sets.
   - Group users with similar buying mentalities.

2. K-means Algorithm

   Ans:

   The K means algorithm takes the input parameter K from the user and partitions the dataset containing N objects into K clusters so that resulting similarity among the data objects inside the group (intracluster) is high but the similarity of data objects with the data objects from outside the cluster is low (intercluster). The similarity of the cluster is determined with respect to the mean value of the cluster. It is a type of square error algorithm. At the start randomly k objects from the dataset are chosen in which each of the objects represents a cluster mean(centre). For the rest of the data objects, they are assigned to the nearest cluster based on their distance from the cluster mean. The new mean of each of the clusters is then calculated with the added data objects.

   Pseudocode -

```
Step 1: Randomly select k samples & mark them as initial clusters.
Step 2: Repeat.
    ● Assign/ reassign in sample to any given cluster to which it is most similar depending
      upon the mean of the cluster.
    ● Update the cluster's mean until No Change.
```

Implementation:

Code - (kmean.py)

-

```
inputFile = open("input-data.txt", "r")
outputFile = open("output-data.txt", "w+") k =
int(input("Enter the number of clusters: "))
print()

data = [float(x.rstrip()) for x in inputFile] centroids =

dict(zip(range(k), data[0:k])) clusters = dict(zip(range(k), [[]

for i in range(k)])) point_assignments = dict(zip(range(k), [[]

for i in range(k)])) old_point_assignments = dict()

real_point_assignments = dict(zip(data, [0 for i in range(len(data))]))

def assign_to_clusters(data, clusters, centroids, point_assignments):
    for i in data:
        closest_index = 0
        dist_to_centroid = 100
        for j in centroids:
            if abs(i - centroids[j]) < dist_to_centroid:
                closest_index = j
                dist_to_centroid = abs(i - centroids[j])
        clusters[closest_index].append(i)
        real_point_assignments[i] = closest_index
```

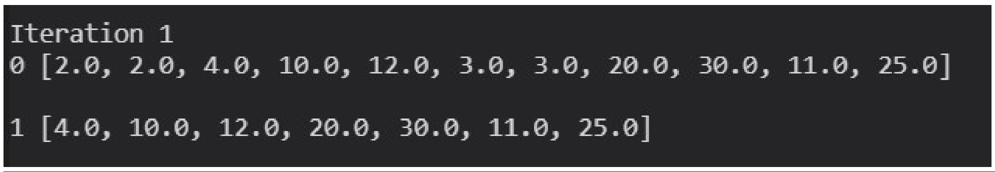
```
def assign_to_centroids(data, clusters, centroids):
   for x in clusters:
      clust_sum = sum(clusters[x]) avg =
      float(clust_sum / len(clusters[x]))
      centroids[x] = min(clusters[x], key=lambda y:abs(y-avg))

counter = 0 while point_assignments !=
old_point_assignments:

   old_point_assignments = point_assignments

   assign_to_clusters(data, clusters, centroids, point_assignments)

   assign_to_centroids(data, clusters, centroids)

   point_assignments = clusters
   clusters = dict(zip(range(k), [[] for i in range(k)])) counter +=
   1 print("Iteration %d" % counter) for x in point_assignments:
   print("%d [%s]" % (x, str(point_assignments[x]).strip("[]")))
   print()

newline_counter = 0 for x in real_point_assignments: print("Point %s in cluster %d" %
(str(x).rstrip("0"), real_point_assignments[x])) newline_counter += 1 if newline_counter <
len(data): outputFile.write("Point %s in cluster %d\n" % (str(x).rstrip("0"),
real_point_assignments[x]))
   else:
      outputFile.write("Point %s in cluster %d" % (str(x).rstrip("0"), real_point_assignments[x]))

inputFile.close()
outputFile.close()
```

(input-data.txt) -

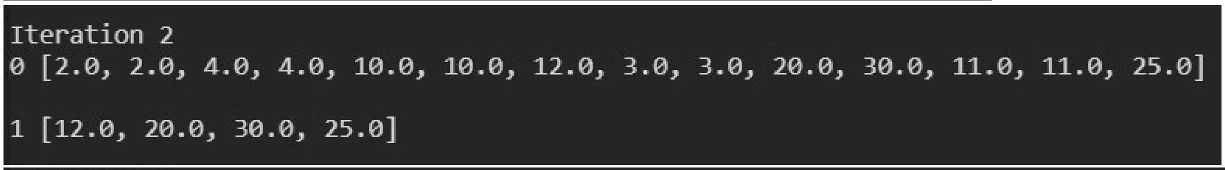
```
2
4
10
12
3
20
30
11
25
```

Output –

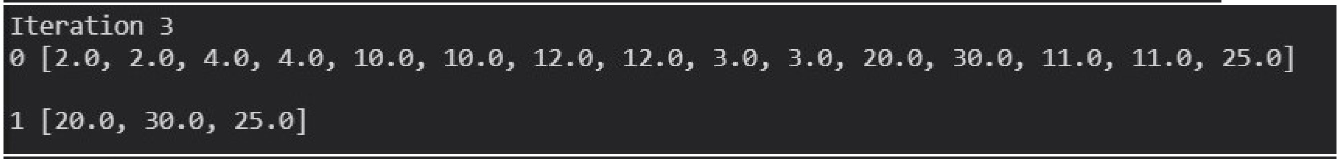
Enter the number of clusters = 2.



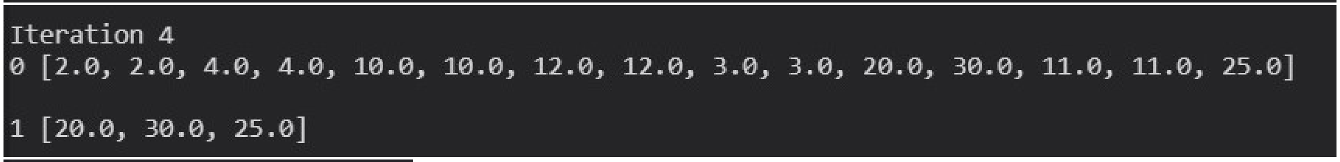
```
Iteration 1
0 [2.0, 2.0, 4.0, 10.0, 12.0, 3.0, 3.0, 20.0, 30.0, 11.0, 25.0]

1 [4.0, 10.0, 12.0, 20.0, 30.0, 11.0, 25.0]
```



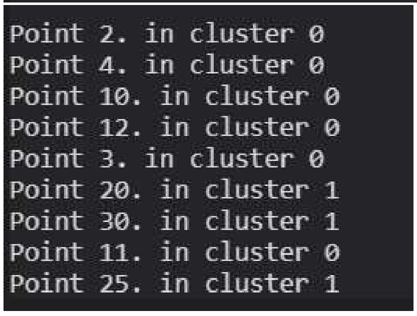
```
Iteration 2
0 [2.0, 2.0, 4.0, 4.0, 10.0, 10.0, 12.0, 3.0, 3.0, 20.0, 30.0, 11.0, 11.0, 25.0]

1 [12.0, 20.0, 30.0, 25.0]
```



```
Iteration 3
0 [2.0, 2.0, 4.0, 4.0, 10.0, 10.0, 12.0, 12.0, 3.0, 3.0, 20.0, 30.0, 11.0, 11.0, 25.0]

1 [20.0, 30.0, 25.0]
```



```
Iteration 4
0 [2.0, 2.0, 4.0, 4.0, 10.0, 10.0, 12.0, 12.0, 3.0, 3.0, 20.0, 30.0, 11.0, 11.0, 25.0]

1 [20.0, 30.0, 25.0]
```



```
Point 2. in cluster 0
Point 4. in cluster 0
Point 10. in cluster 0
Point 12. in cluster 0
Point 3. in cluster 0
Point 20. in cluster 1
Point 30. in cluster 1
Point 11. in cluster 0
Point 25. in cluster 1
```

Conclusion: From the above experiment we are able to learn how the k-means algorithm in partition clustering works. The program stops when the means of the items in the cluster matches with the iteration. It basically segregates on the basis of where the point lies, taking the midpoint as the mean for that cluster and the items will be added to the cluster which is near to it. This method can be applied in many places like segregating large data sets in simpler time.