Implementation of K Means

Ex No: 7

Date 5/10/22

AIM:

The aim of the experiment is to implement K means algorithm with the given points.

Cluster the following eight points into three groups where the distance function is Euclidean measure. State and use k-means algorithm with the assumption of initial clusters, A1, B1, C1. Give the final cluster result.
A1(2,10),
A2(2,5), A3(8,4), B1(5,8), B2(7,5),
B3(6,4), C1(1,2), C2(4,9).

CREATING DATASET:

1. Enter the given data into an arff formatted file.

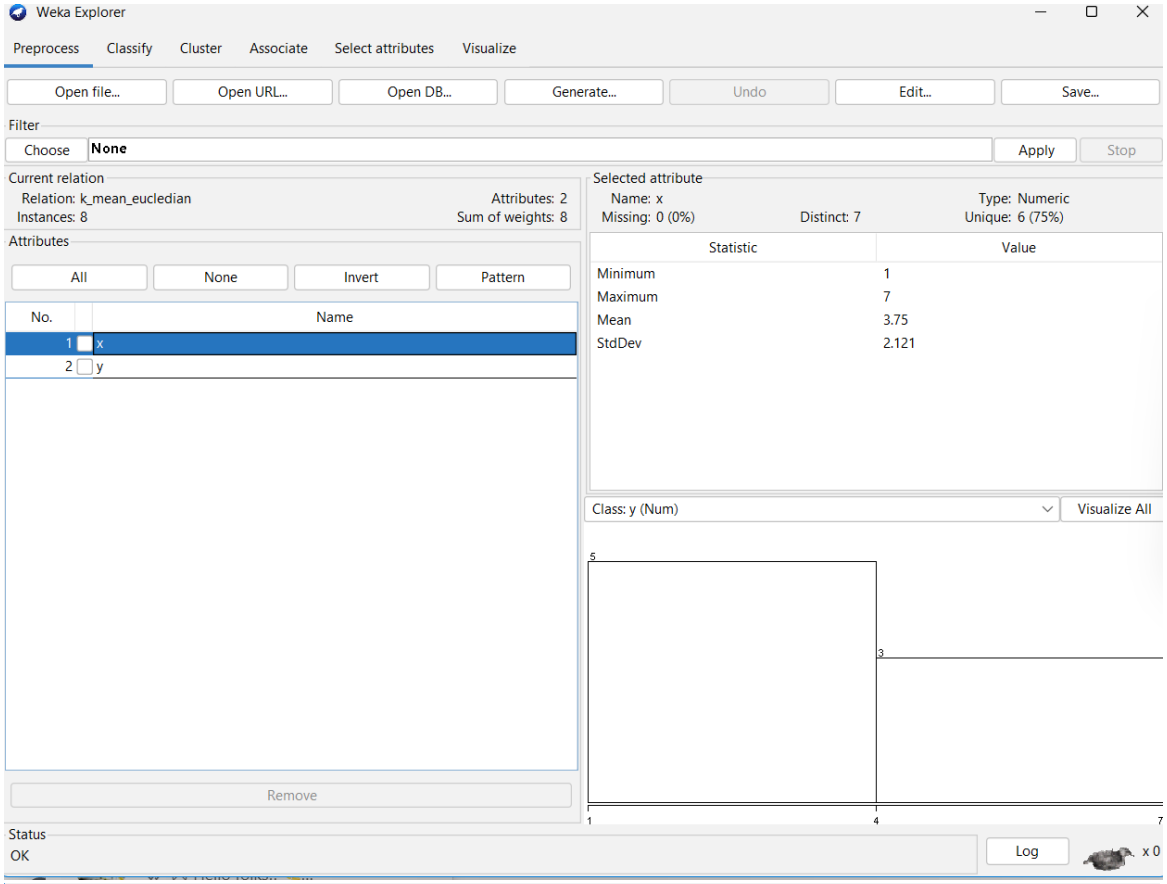
k_mean_eucledian.arff

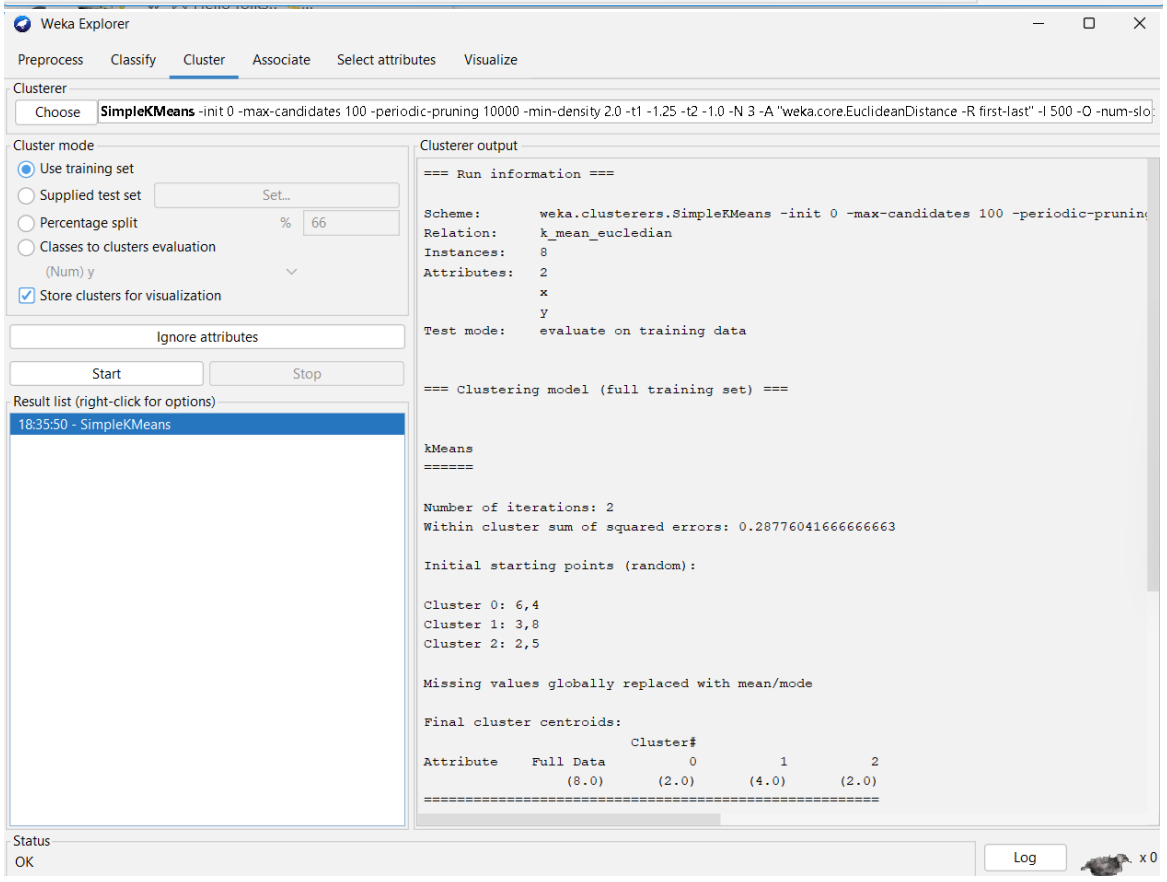
```
@RELATION k_mean_eucledian

@ATTRIBUTE x REAL
@ATTRIBUTE y REAL

@DATA
2,10
2,5
3,8
5,8
7,5
6,4
1,2
4,9
```

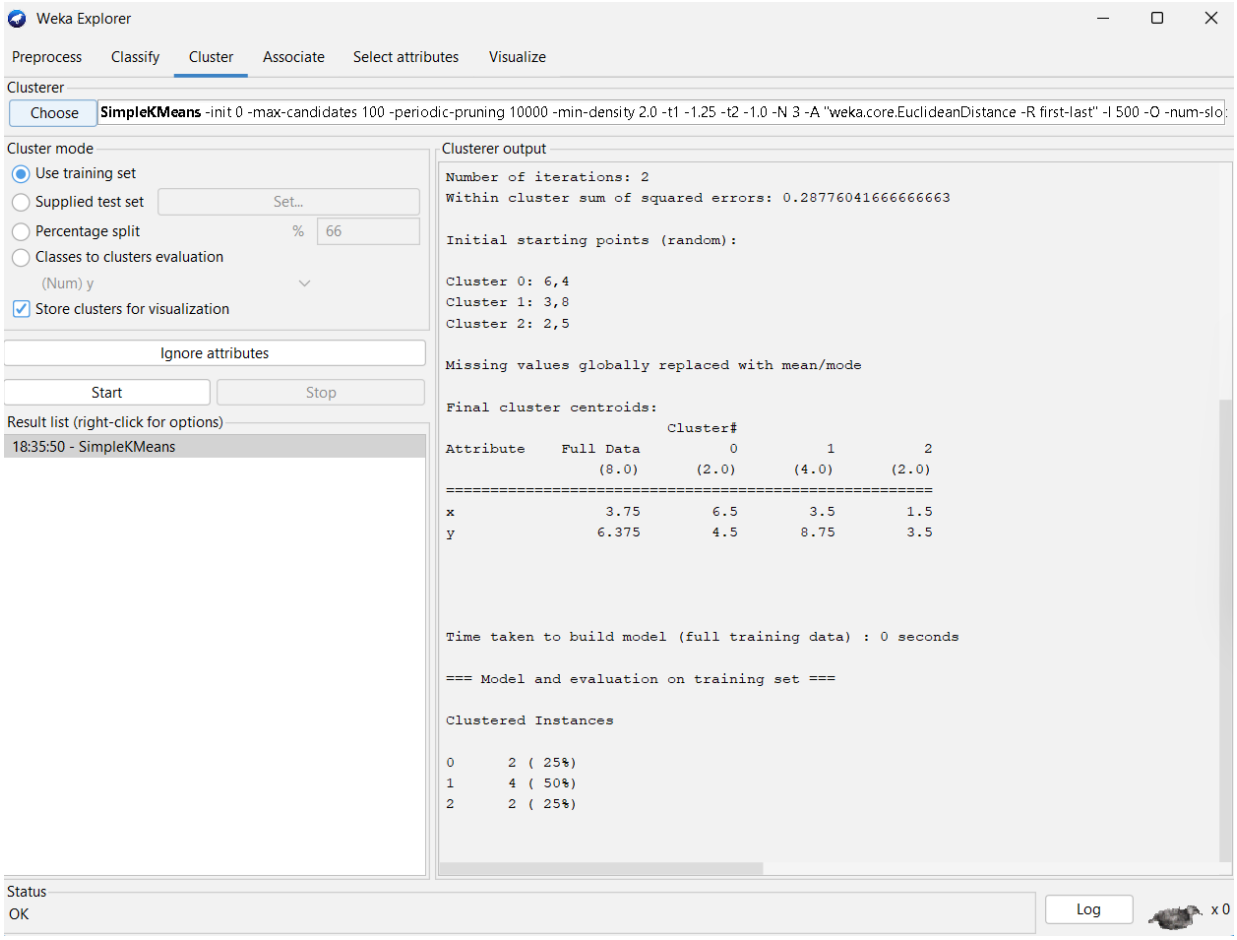
2. Now use the dataset to cluster.

K MEANS IN WEKA:

1. Open weka and click explore.
2. In preprocessing tab click open file button.
3. Select the dateset file you going to perform decision tree.
4. Then click cluster tab.
5. Select the cluster by clicking the choose button.
6. Select the SimpleKMeans which is present under the options.
7. Finally click start. Then it automatically built the model


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: k_mean_eucledian
Instances: 8
Attributes: 2
Sum of weights: 8
Selected attribute
Name: x
Missing: 0 (0%)
Distinct: 7
Type: Numeric
Unique: 6 (75%)
Attributes
All
None
Invert
Pattern
No.
Name
1
x
2
y
Statistic
Value
Minimum
1
Maximum
7
Mean
3.75
StdDev
2.121
Class: y (Num)
Visualize All
Remove
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance -R first-last" -I 500 -O -num-slo
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Num) y
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
18:35:50 - SimpleKMeans
Clusterer output
=== Run information ===
Scheme: weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-prunin
Relation: k_mean_eucledian
Instances: 8
Attributes: 2
x
y
Test mode: evaluate on training data
=== Clustering model (full training set) ===
kMeans
======
Number of iterations: 2
Within cluster sum of squared errors: 0.28776041666666663
Initial starting points (random):
Cluster 0: 6,4
Cluster 1: 3,8
Cluster 2: 2,5
Missing values globally replaced with mean/mode
Final cluster centroids:
Cluster#
Attribute Full Data 0 1 2
(8.0) (2.0) (4.0) (2.0)
Status
OK
Log
x 0



DATA

```
@RELATION k_mean_eucledian
@ATTRIBUTE x REAL
@ATTRIBUTE y REAL
@DATA
2,10
2,5
3,8
5,8
7,5
6,4
1,2
4,9
```

K MEANS USING JAVA

Code:

```
package Javatree;
import java.io.BufferedReader;
import java.io.FileReader;
import weka.clusterers.SimpleKMeans;
import weka.core.Instances;
class Main {
  public static void main(String[] args) throws Exception {
        BufferedReader breader= new BufferedReader(new FileReader(
            "C:\\Users\\kaush\\Desktop\\k_mean_eucledian.arff"));
         Instances Train = new Instances(breader);
         //Train.setClassIndex(Train.numAttributes() - 1); // comment out this line
         SimpleKMeans kMeans = new SimpleKMeans();
         kMeans.setSeed(0);
         kMeans.setPreserveInstancesOrder(true);
         kMeans.setNumClusters(3);
         kMeans.buildClusterer(Train);
         int[] assignments = kMeans.getAssignments();
         System.out.print(kMeans);
                int i = 0;
         for (int clusterNum : assignments) {
          System.out.printf("Instance %d -> Cluster %d\n", i, clusterNum);
          i++;
         }
         breader.close();
        }
}
```

Output:

```
Instance 0 -> Cluster 2
Instance 1 -> Cluster 1
Instance 2 -> Cluster 2
Instance 3 -> Cluster 2
Instance 4 -> Cluster 0
Instance 5 -> Cluster 0
Instance 6 -> Cluster 1
Instance 7 -> Cluster 2
```

Each instance matched with its cluster

```
kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 0.28776041666666663

Initial starting points (random):

Cluster 0: 6,4
Cluster 1: 2,5
Cluster 2: 5,8

Missing values globally replaced with mean/mode

Final cluster centroids:
                          Cluster#
Attribute    Full Data          0          1          2
                 (8.0)      (2.0)      (2.0)      (4.0)
=========================================================
x                 3.75        6.5        1.5        3.5
y                6.375        4.5        3.5       8.75
```

K MEANS USING PYTHON

```
points=((2,10,'a1'),(2,5,'a2'),(8,4,'a3'),(5,8,'b1'),(7,5,'b2'),(6,4,'b3'),(1,2,'c1'),(4,9,'c2'))
c1=[(2,10)]
c2=[(5,8)]
c3=[(1,2)]
clu1=[c1[0]]
clu2=[c2[0]]
clu3=[c3[0]]
def k_means(clu1,clu2,clu3,c1,c2,c3,c=1):
  while True:
    t_c_1=[]
    t_c_2=[]
    t_c_3=[]
    for i in points:
      d1=((c1[0][0]-i[0])**2+(c1[0][1]-i[1])**2)**0.5
      d2=((c2[0][0]-i[0])**2+(c2[0][1]-i[1])**2)**0.5
      d3=((c3[0][0]-i[0])**2+(c3[0][1]-i[1])**2)**0.5
      get_min=min(d1,d2,d3)
      if get_min==d1:
        t_c_1.append(i)
      if get_min==d2:
        t_c_2.append(i)
      if get_min==d3:
        t_c_3.append(i)
    if t_c_1==clu1 and t_c_2==clu2 and t_c_3==clu3:
      break
    else:
      clu1[:]=t_c_1
      clu2[:]=t_c_2
      clu3[:]=t_c_3
      print('Iteration '+str(c))
      print(*clu1,'center',*c1)
      print(*clu2,'center',*c2)
```

```
            print(*clu3,'center',*c3)
            print('----------------')
            print()
            new_mean_c1_x=sum([i[0] for i in c1])/len(c1)
            new_mean_c1_y=sum([i[1] for i in c1])/len(c1)
            new_mean_c2_x=sum([i[0] for i in c2])/len(c2)
            new_mean_c2_y=sum([i[1] for i in c2])/len(c2)
            new_mean_c3_x=sum([i[0] for i in c3])/len(c3)
            new_mean_c3_y=sum([i[1] for i in c3])/len(c3)
            c1=[(new_mean_c1_x,new_mean_c1_y)]
            c2=[(new_mean_c2_x,new_mean_c2_y)]
            c3=[(new_mean_c3_x,new_mean_c3_y)]
        c+=1
    k_means(clu1,clu2,clu3,c1,c2,c3)
    print('Answer')
    print(*clu1,'center',*c1)
    print(*clu2,'center',*c2)
    print(*clu3,'center',*c3)
```

Output:

```
Iteration 1
(2, 10, 'a1') center (2, 10)
(8, 4, 'a3') (5, 8, 'b1') (7, 5, 'b2') (6, 4, 'b3') (4, 9, 'c2') center (5, 8)
(2, 5, 'a2') (1, 2, 'c1') center (1, 2)
----------------

Answer
(2, 10, 'a1') center (2, 10)
(8, 4, 'a3') (5, 8, 'b1') (7, 5, 'b2') (6, 4, 'b3') (4, 9, 'c2') center (5, 8)
(2, 5, 'a2') (1, 2, 'c1') center (1, 2)
```

RESULT:

Successfully we implemented K means clustering algorithm in weka, java with weka library and using python.