CSCE 874: Data Mining
Assignment 3

Name	Contribution %	Contributions	Signature & Date
Venkata Krishna Mohan Sunkara	50	Implemented the algorithm and helped in writing the report.	Venkata Krishna Mohan Sunkara 05/03/2018
Bill Mutabazi	50	Implemented the distance function and helped in writing the report.	Bill JeaaaMutabazi 05/03/2018

K-Means Algorithm

Introduction:

In this assignment, we were tasked to implement *k-means* algorithm to perform clustering and compare our results with the results of Weka.

Clustering is a task of finding groups of objects, such that the objects in a group will be similar to one another and different from the objects in other groups.

k-means algorithm is a partitional clustering algorithm which uses a similarity measure to group things together. It takes k as an input and finds k clusters that optimizes the chosen partition criterion. The algorithm is described as:

```
1. Select $K$ Points as the initial centroids.
2. Repeat
    a. Form $K$ Clusters by assigning all points to the closest centroid.
    b. Recompute the centroid of each cluster
3. Until The sum of distances between centroids falls below a threshold $\epsilon$.
```

Our work include : We implemented *k-means* algorithms using Java Programming language. We used a Weka data set "*iris.arff*" to test our implementation and below in this report, we will compare the differences of our results with the Weka results. Our program asks the user for the path to the dataset file, number of desired clusters (k) and the maximum number of allowed iterations along with the stopping criteria (ϵ). Our program then implements the K-means algorithm using Euclidean distance as a similarity measure.

General Design:

Our Program reads the file provided in the path and parses every line in the file to extract the data. The number of clusters, stopping criteria (ϵ, number of iterations) are taken as inputs from the user. The output of the program is displayed in the command window itself. The output of the program first displays the normalized data from the dataset and then displays the number of instances in each cluster. It also displays the index number of instances in each cluster. After the entire clustering process is finished, it displays the Sum Squared Error of the entire clustering process. This measure is used for comparing the goodness of clustering.

I. Input Extraction: A function is declared to handle reading the instances from an arff file. The extracted instance is stored as a double arraylist.
II. Normalization of the Instances: The instances obtained from the input file are then normalized by attributes using the min-max normalization.
III. Distance Computation: A function is declared to calculate the distance between two points using the euclidean metric. It generates the distances from every point in the dataset to every centroid in the clustering process.
IV. Assigning Points to the Cluster: Using the corresponding distances, every data point is assigned to a cluster to which it is closest to.
V. Calculation of Epsilon: A function is declared to calculate the sum of squared distances between the new centroids and the previous centroids.
VI. Calculation of SSE: A function is declared to calculate the Sum of Squared errors from every data point to its corresponding cluster center.

Data Structures used: The data structures used in the implementation are:

I. A List of Lists of Double which is used to store the instances of a dataset and also to store the centroids of a cluster.
II. A HashMap used to store the cluster name along with the points which belong to the corresponding cluster.

Algorithm Run time:

i) **As a function of number of clusters:**

In our program, as the number of clusters increases there is an increase in runtime for the k-means algorithms implementation because as more centroids being added the computation time will increase as well. We tested our program for different values of k ranging from 1 to 150 (max size of our dataset) and examined the run time of our algorithm. In the figure 01 we represent the plot of runtime against the k-values. We observed that there is a slight fluctuation in the graph at some k-values and these fluctuations are caused due to random initialization of our centroids. The runtime also depends on the hardware and other tasks being run on a system.

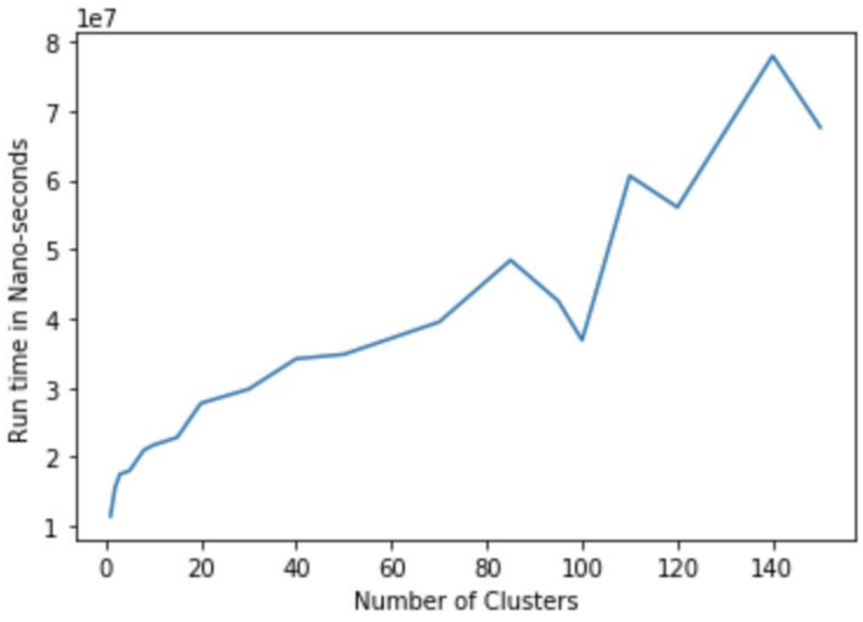


Figure 01: Plot of run time against the number of clusters.

ii) Number of dimension:
Our dataset consists of just 4 attributes, so we tried executing our program on the same dataset but with different number of features. We observed that as the number of features increases the run-time of our algorithm also increases. It is accounted by an increase in the computation time for calculating the distance between two data points. Here K is set to 8 with ϵ = 0.001 and the max number of iterations as 100.

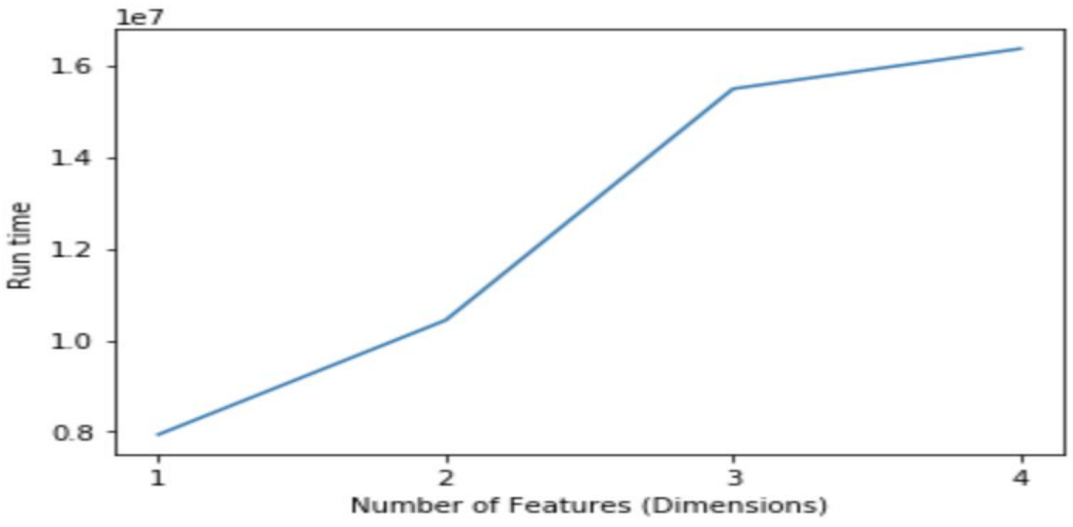


Figure 02: Plot of runtime against the number of features in the dataset.

iii) **Size of the dataset:**
Also as size of the dataset increases, the time taken for executing our program also increases. In the figure 03 we can observe that there is a general trend of increase in the run time as the dataset size increases. We can also observe fluctuations in the run time in some cases, this can be accounted due to random initializations of centroids at the start of the algorithm. The overall runtime of the algorithm is represented as O(n*t*k*p), where n is the number of instances, t is the number of iterations, k is the number of clusters and p is the number of features in the dataset. In this case, n is increased and the number of iterations which is correlated with random initializations of centroids by our algorithm, So we observe much variability in the run time of our algorithm as the size of the dataset is increased. Here K is set to 8, ϵ to 0.001 and the max number of iterations to 100.

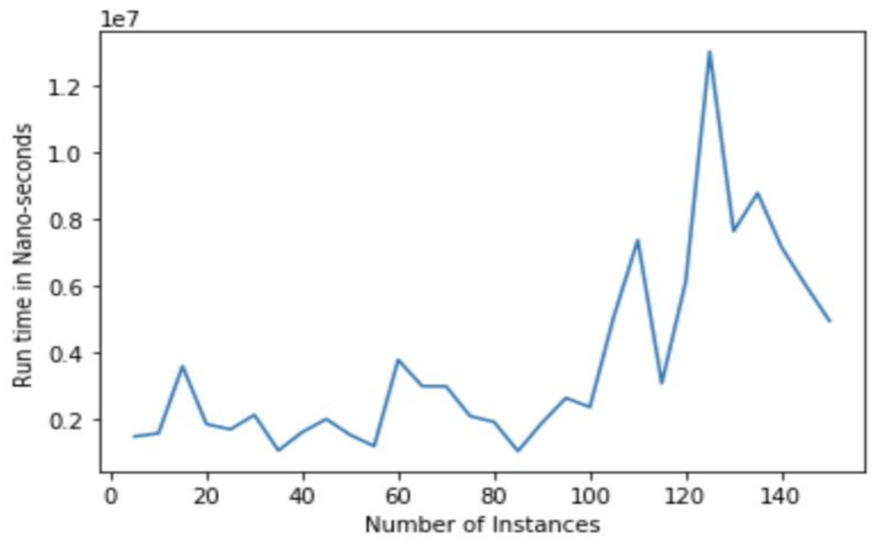


Figure 03: Plot of runtime against the number of instances in the dataset

Goodness of clustering determined by SSE as a function of the number of clusters:

While implementing k-Means algorithm to the number of clusters, we observed that the sum squared error (SSE) reduces as the number of clusters increase because with the increase number of clusters the probability of having the centroid close to each will be increased as well, in that the SSE will be decreased along the increase of the number of clusters.

We observed the same trend in our algorithm implementation. Figure 04 represents the plot of Sum Squared error against the number of clusters. We tested our implementation on the iris.arff dataset for different values of k ranging from 1 to 50, with ϵ = 0.001 and the max number of iterations to 100. Figure 04 indicates that the SSE value goes on decreasing by increasing the number of clusters. If the k value is set to 150, we observed an SSE of 0 as every point has its own cluster. To determine the optimal number of clusters, we can use SSE as a cluster goodness metric. If we consider the below graph then we can estimate a good value of clustering to be around 8-25. By observing the graph we can come to a conclusion that there is not much difference in the SSE value after a certain k, in this case 22. After k = 22, the SSE values goes on decreasing but by just a small inappropriate amount. So, even if we select larger k value it doesn't improve much than k=22.

If we would also like to consider runtime in to account, then we are more inclined towards lower k-value. For k=8, the SSE is low and much better than the previous k-values. Even if we go further, we observe only a small decrease in the SSE values. Finally, it comes to a point to decide between the runtime and the SSE value. One can decide to select k=8, if we want low number of clusters but with better runtime or select k=22, if we want more accurate clustering by compromising on the runtime.

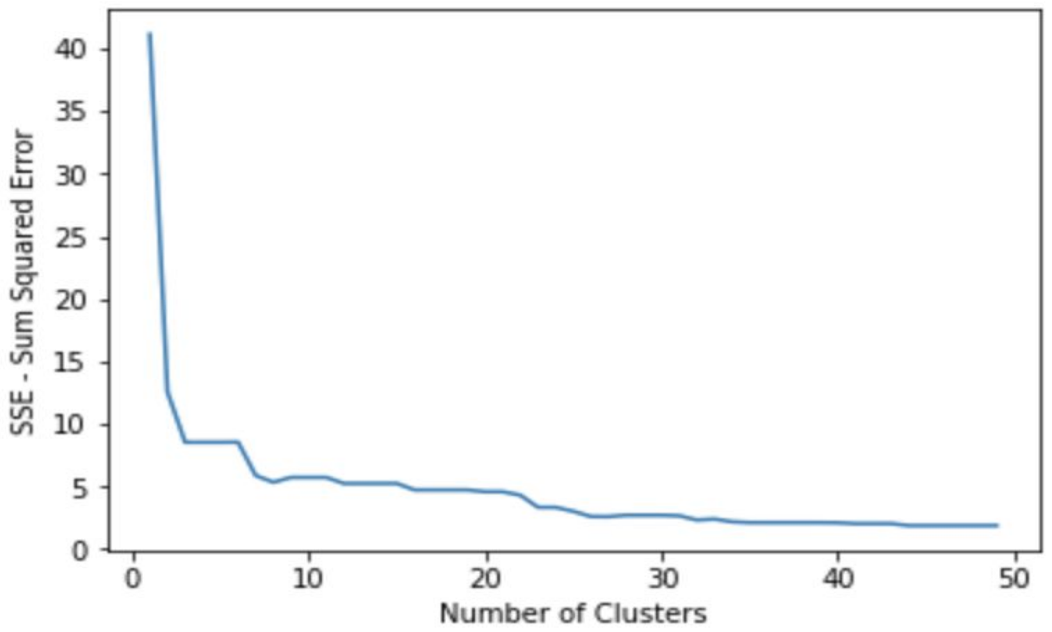


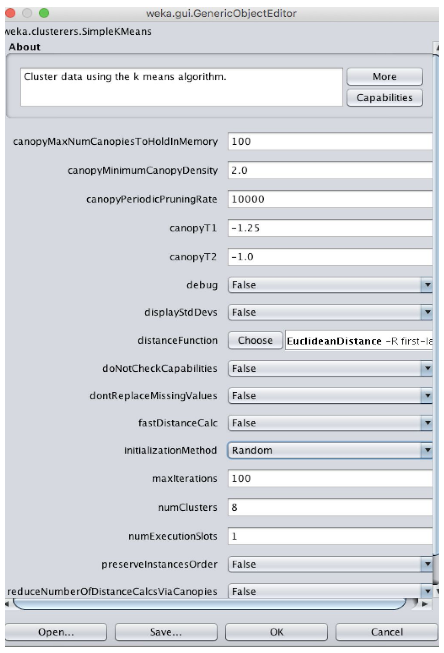
Figure 04: Plot of SSE against the number of clusters generated.

Clustering using Weka:

The Figure 05 below shows the results of k-means algorithm in Weka.
To run Clustering on Weka:

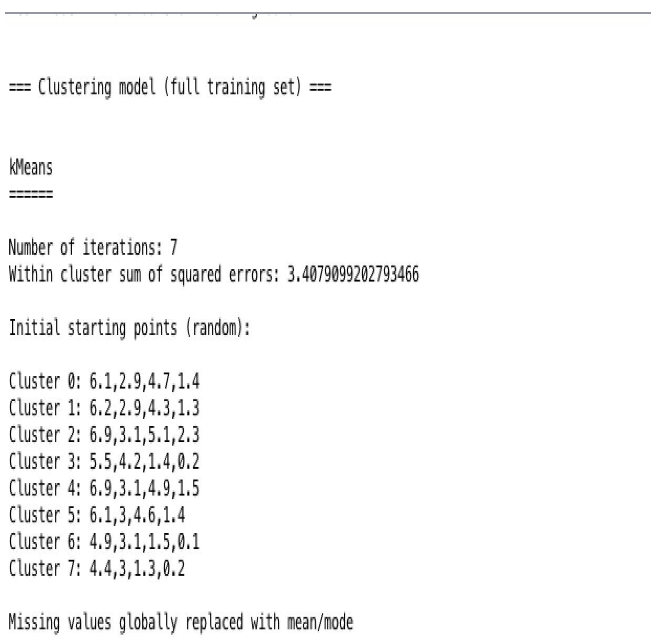
1. Open the iris.arff dataset in weka.
2. Go to the clustering tab and select simple k-means algorithm from the choose tab.
3. Now a window similar to the one shown in figure 05 appears. Change the number of iterations to 100 and the number of clusters to 8.
4. Now run the algorithm and the output is displayed as shown in figure 05.

Input: ***Output;***



weka.gui.GenericObjectEditor
weka.clusterers.SimpleKMeans
About
Cluster data using the k means algorithm.

canopyMaxNumCanopiesToHoldInMemory	100
canopyMinimumCanopyDensity	2.0
canopyPeriodicPruningRate	10000
canopyT1	-1.25
canopyT2	-1.0
debug	False
displayStdDevs	False
distanceFunction	EuclideanDistance -R first-la
doNotCheckCapabilities	False
dontReplaceMissingValues	False
fastDistanceCalc	False
initializationMethod	Random
maxIterations	100
numClusters	8
numExecutionSlots	1
preserveInstancesOrder	False
reduceNumberOfDistanceCalcsViaCanopies	False



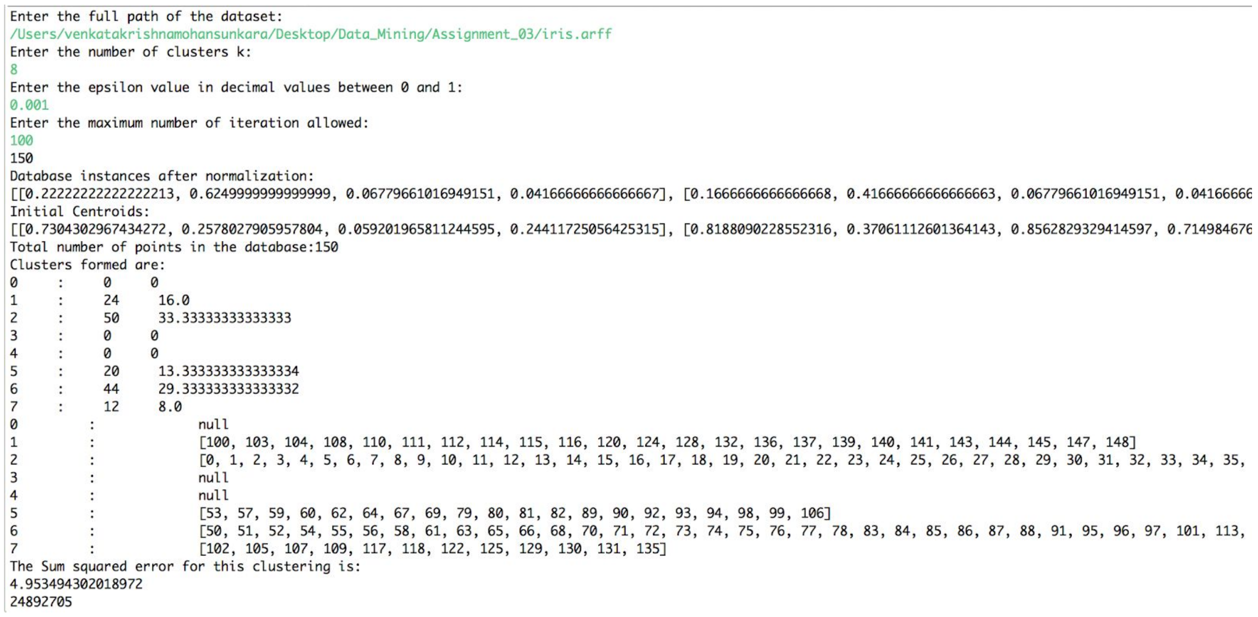
```
=== Clustering model (full training set) ===


kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 3.4079099202793466

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4
Cluster 1: 6.2,2.9,4.3,1.3
Cluster 2: 6.9,3.1,5.1,2.3
Cluster 3: 5.5,4.2,1.4,0.2
Cluster 4: 6.9,3.1,4.9,1.5
Cluster 5: 6.1,3,4.6,1.4
Cluster 6: 4.9,3.1,1.5,0.1
Cluster 7: 4.4,3,1.3,0.2

Missing values globally replaced with mean/mode
```

Figure 05: Simple K-means algorithm in Weka with the number of clusters: 8.

We can observe that the initial seeds are generated at random and every cluster consists of at least one datapoint. The SSE measure is also displayed in the output window as 3.407.

Clustering using our program:

The user is has to provide the path to the dataset file and the number of desired clusters (k) as input to the program. The user should also provide the stopping criteria ϵ and the maximum number of allowed iterations as input. Now if we run the program an output similar to the figure 06 is displayed.



```
Enter the full path of the dataset:
/Users/venkatakrishnamohansunkara/Desktop/Data_Mining/Assignment_03/iris.arff
Enter the number of clusters k:
8
Enter the epsilon value in decimal values between 0 and 1:
0.001
Enter the maximum number of iteration allowed:
100
150
Database instances after normalization:
[[0.22222222222222213, 0.6249999999999999, 0.06779661016949151, 0.04166666666666667], [0.1666666666666668, 0.41666666666666663, 0.06779661016949151, 0.0416666
Initial Centroids:
[[0.7304302967434272, 0.2578027905957804, 0.059201965811244595, 0.24411725056425315], [0.8188090228552316, 0.37061112601364143, 0.8562829329414597, 0.714984676
Total number of points in the database:150
Clusters formed are:
0     :     0     0
1     :     24     16.0
2     :     50     33.33333333333333
3     :     0     0
4     :     0     0
5     :     20     13.333333333333334
6     :     44     29.333333333333332
7     :     12     8.0
0         :             null
1         :             [100, 103, 104, 108, 110, 111, 112, 114, 115, 116, 120, 124, 128, 132, 136, 137, 139, 140, 141, 143, 144, 145, 147, 148]
2         :             [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
3         :             null
4         :             null
5         :             [53, 57, 59, 60, 62, 64, 67, 69, 79, 80, 81, 82, 89, 90, 92, 93, 94, 98, 99, 106]
6         :             [50, 51, 52, 54, 55, 56, 58, 61, 63, 65, 66, 68, 70, 71, 72, 73, 74, 75, 76, 77, 78, 83, 84, 85, 86, 87, 88, 91, 95, 96, 97, 101, 113,
7         :             [102, 105, 107, 109, 117, 118, 122, 125, 129, 130, 131, 135]
The Sum squared error for this clustering is:
4.953494302018972
24892705
```

Figure 06: Output of our program for k=8, ϵ = 0.001, max-iterations = 100

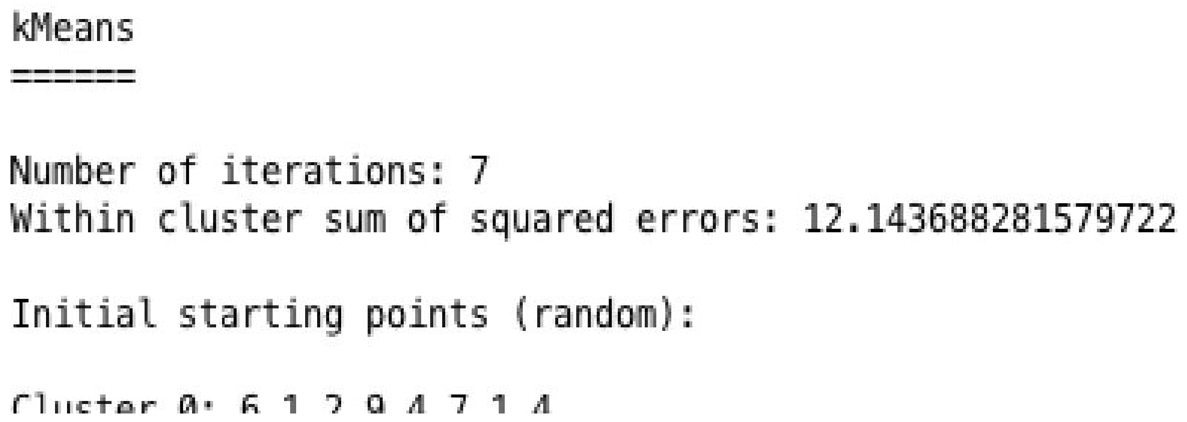
Comparison:

The output of our program differs from that of weka because our program uses random initialization of centroids and it doesn't guarantee that every cluster has at least one point. We can observe that some of the cluster generated by our program are null, this is because the random initialization of our centroids are so far from the original points that none of them are clustered to it. This accounts for a different value for SSE and the number of points in each cluster.

Also, weka doesn't contain any input for considering the ϵ value as a stopping criterion. The Stopping criterion used by weka is just the number of iterations. However, our program considers the number of iterations along with ϵ as a stopping criterion.

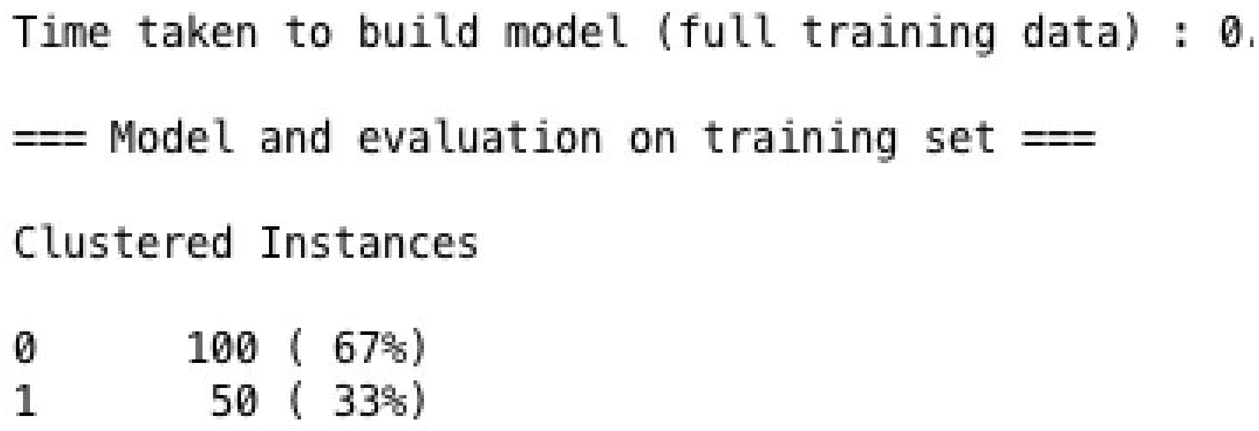
If we observe the figure 7a, 7b and 8. We obtain the same results as that of weka as the number of clusters is low and there isn't much random initialization involved.

Number of Clusters: 2



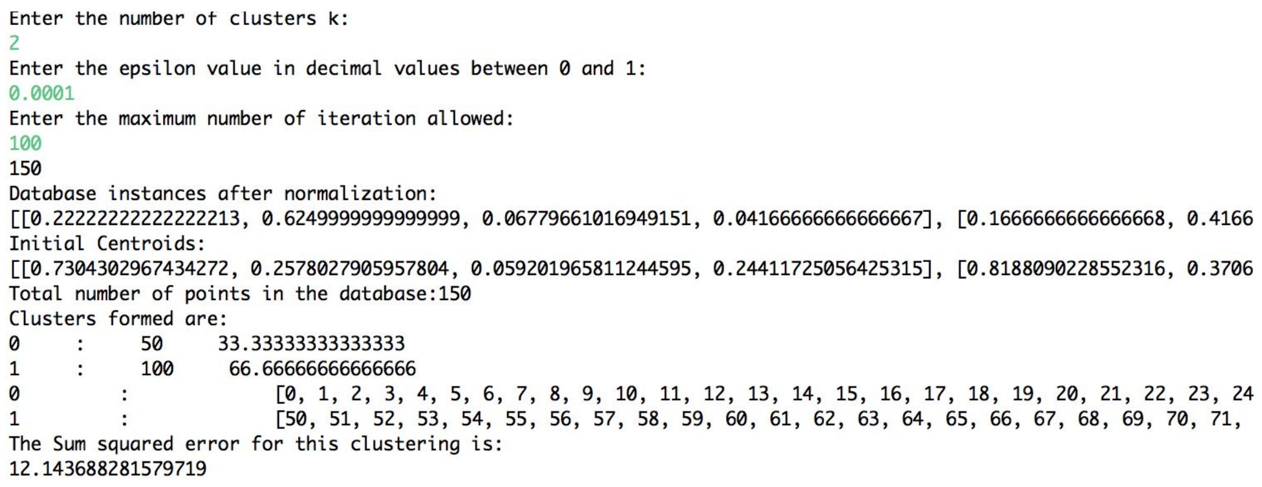
```
kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 12.143688281579722

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4
```

Figure 07a: Simple K-means algorithm in weka with k=2



```
Time taken to build model (full training data) : 0.

=== Model and evaluation on training set ===

Clustered Instances

0       100 ( 67%)
1        50 ( 33%)
```

Figure 07b: Simple K-means algorithm in weka with k=2



```
Enter the number of clusters k:
2
Enter the epsilon value in decimal values between 0 and 1:
0.0001
Enter the maximum number of iteration allowed:
100
150
Database instances after normalization:
[[0.22222222222222213, 0.6249999999999999, 0.06779661016949151, 0.04166666666666667], [0.1666666666666668, 0.4166
Initial Centroids:
[[0.7304302967434272, 0.2578027905957804, 0.059201965811244595, 0.24411725056425315], [0.8188090228552316, 0.3706
Total number of points in the database:150
Clusters formed are:
0     :     50     33.33333333333333
1     :     100     66.66666666666666
0        :             [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
1        :             [50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,
The Sum squared error for this clustering is:
12.143688281579719
```

Figure 08: Output of our program for k=2, ϵ = 0.001, max-iterations = 100