University of Skövde
School of Informatics
Master of Data science
Haftamu Hailu Tefera
al17hafte@student.his.e

Data Mining - IT726A

Assignment 2: K-means Clustering

The objective of this assignment is to use k-mean clustering algorithm to cluster iris data set into different clusters.

The whole problem is solved using the following 8 small functions.

1. Load the data
   This load function returns iris data set as data matrix

2. Calculate the Euclidean distance between two data points

```
    def distance(x, y):
        return np.sqrt(np.sum((x - y)**2))
```

   returns the minimum distance between x and y.

3. Assign labels to the data points

```
def assign_labels(X, centers):
    Dist=[[distance(x, c) for c in centers] for x in X]
    labels= np.argmin(Dist, axis=1)
    return labels
```

   The **np.argmin** function returns the indices of the minimum values along the specified axis which is column in this case.
   The Assign_labels function assigns labels to each data points into different clusters and at the last returns array of labels.

4. Calculating the new centroids after data points are labelled

   After we get the data points labelled with the assign_label function we need to calculate the new position of the centroids.

```
def calculate_centers(X, labels):
    newcenter = np.array([np.mean(X[labels == c], axis=0) for c in range(len(X))])
    return newcenter
```

This calculate_center function compute the new cluster centers by taking the mean of the set of data points that belongs to a particular cluster based on the outcome of the assign_ labels function. In this case some data points can move to different cluster based on their mean distance from their previous clusters
X[labels == c] numpy select all of the data points which have c as their closest cluster and finally the function returns position of the centroids.

5. **Test the algorithm Convergence**

```
def test_convergence(old_centers, new_centers):
   iter = 100
 return np.all(old_centers - new_centers )<iter if old_centers.shape[0] ==
new_centers.shape[0] else False
```

Test whether the previously computed centroids are stable or not. The convergence test is attained when the value of the old centroid and new centroid is equal.

6. **Evaluate Performance of each cluster**

```
def evaluate_performance(X, labels, centers):
   size=len(X)
   return np.mean([distance(X[i], centers[labels[i]]) for i in range(size)])
```

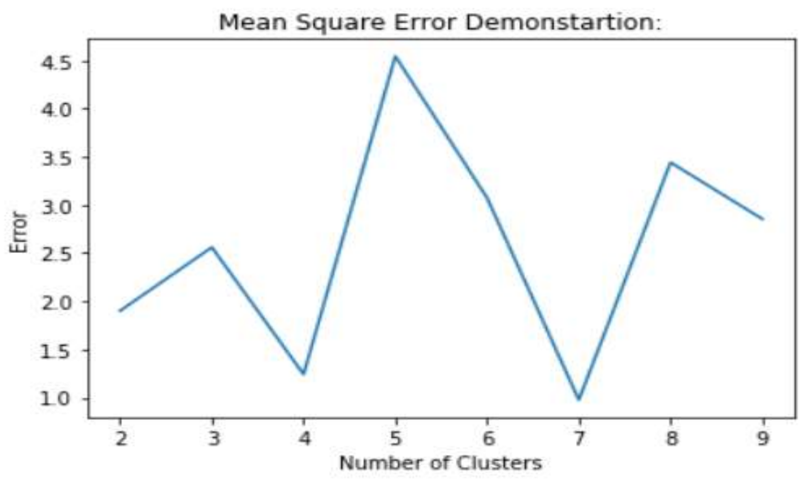
This function calculates the mean distance of a given cluster using the distance function with data points and each label as parameter.

It returns the mean distance of a given cluster.

7. Calculate Means Square Error for the all clusters

```
def mse():
   clusters=range(2,10)
   for k in clusters:
      centers, labels = k_means(X, k)
      error.append(evaluate_performance(X, labels, centers))
```

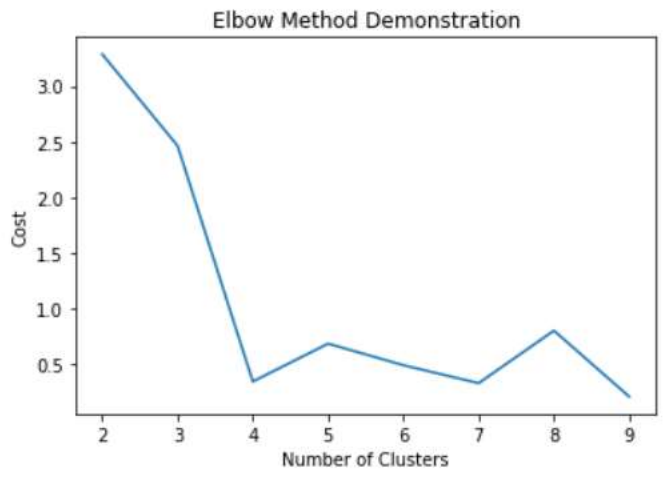
By calling the k-means (x, cluster) and evaluate performance functions 10 times mse() function calculates the mean square errors of all the clusters by using some helper function called append which add all the mean square errors values into an array called error.
Finally, it draws a graph means square error vs number of clusters. The graph looks like



8. Elbow Method Calculation

```
Def elbow():
for k in clusters:
    centers, labels = k_means(X, k)
    cost.append(np.sqrt(np.sum((X[k] - centers[labels[k]])**2)))
```

The elbow method calculates the required number of clusters used to classify the iris data set into different clusters. When I run elbow method, I got different k-values(elbow) ranges from 3 to 6 but most of the time the elbow is determined at k=4. As we can see from the chunk of code it iteratively calls the k-means and calculates the distance.



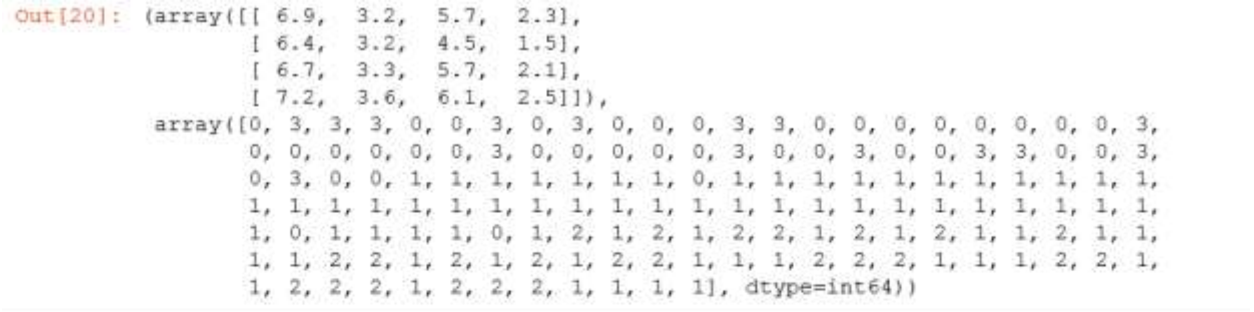
Therefore, we can classify the Iris data set using 4 clusters. We can use more clusters but 4 clusters are enough for this data set.

9. Testing the overall work using the k-means(x, cluster) function.

```
def  k_means():
while not test_convergence(new_centers, old_centers):
    old_centers = X[np.random.choice(X.shape[0], K)]
    labels = assign_labels(X, old_centers)
    new_centers =old_centers
  return new_centers, labels
```

Testing the k_means using four clusters

```
X= load_data()
k_means(X,4)
```



```
Out[20]: (array([[ 6.9,  3.2,  5.7,  2.3],
        [ 6.4,  3.2,  4.5,  1.5],
        [ 6.7,  3.3,  5.7,  2.1],
        [ 7.2,  3.6,  6.1,  2.5]]),
 array([0, 3, 3, 3, 0, 0, 3, 0, 3, 0, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 3,
        0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 3, 0, 0, 3, 0, 0, 3, 3, 0, 0, 3,
        0, 3, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 1, 1, 0, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1,
        1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1,
        1, 2, 2, 2, 1, 2, 2, 2, 1, 1, 1, 1], dtype=int64))
```

As we can from the output, the function returns list of centroids and labels for each data points in four different clusters.