Pattern Recognition Assignment No:3
Code
18MCMI09
Krishnan.S

All the algorithms were implemented in Python3. The libraries that were used were Numpy, Matplotlib and scikit-learn. Outputs of plots are shown in the analysis report.

1.
The **KMeans()** function takes the data set and the value for k as inputs. The data set should be a numpy array. It returns the cluster labels for all the data points as an numpy array and the k centroids. This is the same Euclidean function I wrote for the first assignment. The plotting will work up to 10 clusters. It plots three features. A copy of the previous cluster labels is kept after each iteration in order to figure out when to stop I.e the previous and current labels would be the same. The centroids are found using the **np.mean()** function. The **np.where()** function returns positions of values in that array that satisfy the given predicate. It was used in this case to locate where the points of each cluster are in the data set array.

Code:

```
def Euclidean( x, y ):
    return sqrt(sum((x-y)**2))


#The Algorithm
def KMeans( D, k ):
    centroids = np.array(D[np.random.randint(0, len(D)-1, size = k)])
    iterations = 0
    clusters = np.array([-1]*len(D))
    prevClusters = copy.deepcopy(clusters)

    while iterations < 50:
        for i in range(len(D)):
            distances = [Euclidean(D[i], x) for x in centroids]
            clusters[i] = np.argmin(distances)

        if np.array_equal(clusters, prevClusters):
            break
        prevClusters = copy.deepcopy(clusters)
        centroids = np.array([ np.mean(D[np.where(clusters == x)], axis = 0) for x in range(k)])
        iterations += 1
    return clusters, centroids


#Driver code
if __name__ == '__main__':

    #Reading the data
    allofthem = []
    reader = open("Iris.data", "r")
    lines = reader.read().split("\n")
    for line in lines:
        line = line.split(",")
        allofthem.append(list(map(float, line[:-1])))
    allofthem = np.array(allofthem)

    #The call
    k = 3
    clusters, centroids = KMeans( allofthem, k )
```

```
#The plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter( centroids[:,0], centroids[:,1], centroids[:,2], c = 'C3' )

for x,c in zip(range(k), ['C0', 'C1', 'C2', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9']):
    A = allofthem[np.where(clusters == x)]
    ax.scatter( A[:, 0], A[:, 1], A[:, 2], c = c, label = 'Cluster '+ str(x), alpha = 0.4)
plt.legend()
plt.show()
```

2.

The **Measures()** function takes the data set, the actual class labels (if any), the cluster labels obtained through K-Means and a flag that checks whether external measures can be calculated or not. If there are no class labels, an empty list can be passed. The **mode()** and numpy predicate indexing were used to figure out the purity of each cluster. The other measures were found using the **silhouette_score(), davies_bouldin_score()** and **f1_score()** functions from the **sklearn.metrics** module. The confusion matrix is obtained using the **confusion_matrix()** function from the same module.

Code:

```
def Measures( D, actualLabels, clusters, Ex ):

    print("Internal Measures")
    print("Silhouette Coefficient : ", silhouette_score(D, clusters, metric="euclidean"))
    print("Davies Bouldin Score : ", davies_bouldin_score(D, clusters))

    if Ex == 0 :
        print("No class labels present, so no external measures can be computed")
        Return

    P = 0
    for x in set(actualLabels):
        A = actualLabels[actualLabels == x]
        B = clusters[clusters == x]
        P += min( (A == mode(A)[0][0]).sum(), (B == mode(B)[0][0]).sum())
    P = P/len(allofthem)

    print("\nExternal Measures")
    print("Purity: ", P)
    print("F Measure : ", f1_score(actualLabels, clusters, average = 'macro'))

    print("Confusion Matrix:")
    print(confusion_matrix(actualLabels, clusters))
```

Output:

```
Internal Measures
Silhouette Coefficient :  0.5525919445213676
C:\Users\Krishnan Meep\AppData\Local\Programs\Python\
  score = (intra_dists[:, None] + intra_dists) / cent
Davies Bouldin Score :  0.6623228649898758

External Measures
Purity:  0.8933333333333333
F Measure :  0.8917748917748917
Confusion Matrix:
[[50  0  0]
 [ 0 48  2]
 [ 0 14 36]]
```

With the iris data set

```
Internal Measures
Silhouette Coefficient : 0.5698116547095559
C:\Users\Krishnan Meep\AppData\Local\Programs\Python\Python36\lib\
  score = (intra_dists[:, None] + intra_dists) / centroid_distance
Davies Bouldin Score : 0.5342620136848031
No class labels present, so no external measures can be computed
```

With the 3D Road Network dataset, k = 4

```
Internal Measures
Silhouette Coefficient : 0.5356452115090792
C:\Users\Krishnan Meep\AppData\Local\Programs\Python\Python36\lib\site-p
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
Davies Bouldin Score : 0.5342189177841222
No class labels present, so no external measures can be computed
```

With the 3D Road Network dataset, k = 7

3.

The **SpectralClustering()** function from the **sklearn.cluster** module was used to run Kernel K-Means on the dataset. It returns a clustering object, and the labels are stored in the labels_ member.

Code:

```
if __name__ == '__main__':
    allofthem = []
    reader = open("3D_spatial_network.txt", "r")
    lines = reader.read().split("\n")
    lines.pop()
    for line in lines:
        line = line.split(",")
        allofthem.append(list(map(float, line[1:])))
    allofthem = np.array(allofthem)

    #Where the clustering happens
    k = 4
    clusters = SpectralClustering(n_clusters = k).fit(allofthem[:10000])
    clusters = clusters.labels_

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    for x,c in zip(range(k), ['C0', 'C1', 'C2', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9']):
        A = allofthem[np.where(clusters == x)]
        ax.scatter( A[:, 0], A[:, 1], A[:, 2], c = c, label = 'Cluster '+ str(x), alpha = 0.4)
    plt.legend()
    plt.show()

    Measures( allofthem[:10000], [], clusters, 0 )

    #WIth K Means just to compare
    k = 4
    clusters, centroids = KMeans( allofthem[:10000], k )
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    #ax.scatter( centroids[:,0], centroids[:,1], centroids[:,2], c = 'C3' )

    for x,c in zip(range(k), ['C0', 'C1', 'C2', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9']):
        A = allofthem[np.where(clusters == x)]
        ax.scatter( A[:, 0], A[:, 1], A[:, 2], c = c, label = 'Cluster '+ str(x), alpha = 0.4)
    plt.legend()
```

```
    plt.show()
    Measures( allofthem[:10000], [], clusters, 0)
```

Output:

```
Internal Measures
Silhouette Coefficient :  0.5796023121592779
C:\Users\Krishnan Meep\AppData\Local\Programs\Python\Python36\lib\site-pa
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
Davies Bouldin Score :  0.51096886160606
No class labels present, so no external measures can be computed
```

With K Means, k = 4

```
Internal Measures
Silhouette Coefficient :  0.5499834898652997
C:\Users\Krishnan Meep\AppData\Local\Programs\Python\Python36\lib\site-
  score = (intra_dists[:, None] + intra_dists) / centroid_distances
Davies Bouldin Score :  0.45180371963683935
No class labels present, so no external measures can be computed
```

With Kernel K Means, k = 4