Assignment3

November 10, 2023

[1]:
```
from HW3 import kmeans_template
from HW3 import Hierarchical_template
from HW1 import pca_template
import numpy as np
```

0.1 1) The cluster centroids obtained on YeastGene dataset after the T iterations

[2]:
```
data_filename = 'HW3\YeastGene.csv'
centroid_filename = 'HW3\YeastGene_Initial_Centroids.csv'
k = 6
T = 7
```

Running the file i.e. whatever is in the main for kmeans_template

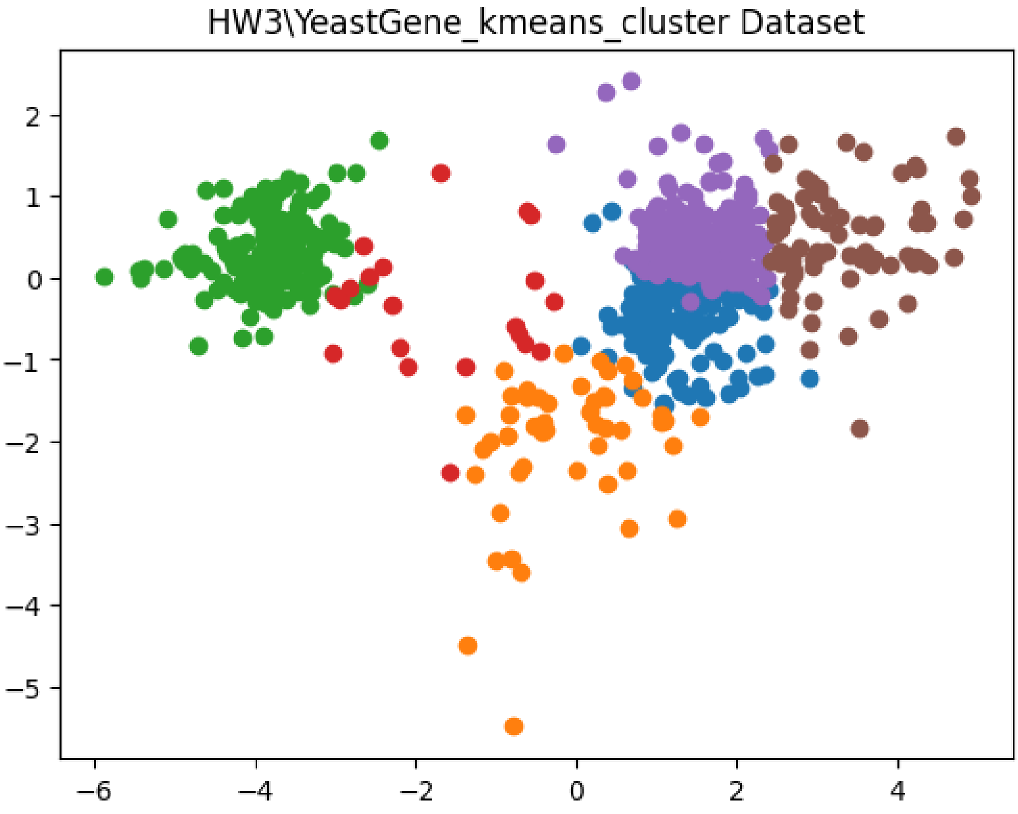
[3]:
```
save_filename = data_filename.replace('.csv', '_kmeans_cluster.csv')
data = kmeans_template.loadDataSet(data_filename)
centroids = kmeans_template.loadCenterSet(centroid_filename)
centroids, clusterAssment = kmeans_template.kMeans(data, T, k, centroids )
print("Centroids returned")
print(centroids)

print("\nCentroids rounded off to 4 decimal places to better read it.")
for centroid in centroids: print(np.round(centroid,4))
kmeans_template.saveData(save_filename, data, clusterAssment)
```

```
Centroids returned
[array([-0.24127143, -0.12875   ,  0.06225   ,  0.17335   ,  0.21791429,
        1.65169286,  1.90532143]), array([-0.9535098 , -1.47164706,  0.07752941,
-0.1794902 , -1.00482353,
        1.15121569,  0.96880392]), array([ 0.16510366,  0.09165244, -0.10388415,
-0.55258537, -0.63008537,
       -1.72318293, -1.75481098]), array([ 0.02328571,  0.25080952, -0.27695238,
-0.364     , -0.73542857,
       -0.89461905,  0.70014286]), array([-1.55555556e-03,  1.56506173e-01,
3.56253086e-01,  7.01580247e-01,
        1.00971605e+00,  1.84231481e+00,  1.64341975e+00]), array([-0.03932895,
0.15394737,  0.43607895,  1.10581579,  1.44871053,
        3.01634211,  2.82938158])]
```

```
Centroids rounded off to 4 decimal places to better read it.
[-0.2413 -0.1288  0.0622  0.1734  0.2179  1.6517  1.9053]
[-0.9535 -1.4716  0.0775 -0.1795 -1.0048  1.1512  0.9688]
[ 0.1651  0.0917 -0.1039 -0.5526 -0.6301 -1.7232 -1.7548]
[ 0.0233  0.2508 -0.277  -0.364  -0.7354 -0.8946  0.7001]
[-1.6000e-03  1.5650e-01  3.5630e-01  7.0160e-01  1.0097e+00  1.8423e+00
  1.6434e+00]
[-0.0393  0.1539  0.4361  1.1058  1.4487  3.0163  2.8294]
```

0.2 2) The scatter plot obtained on YeastGene dataset after applying PCA and plotting points using different colors for different clusters.

[4]:
```
filename = "HW3\YeastGene_kmeans_cluster.csv"
figname = filename
figname = figname.replace('csv','jpg')
dataMat, labelMat = pca_template.loadDataSet(filename)
lowDDataMat = pca_template.pca(dataMat)
pca_template.plot(lowDDataMat, labelMat, figname)
```



0.3 3) The order of merging in the hierarchical clustering on the Utilities dataset.

[5]:

```
data_filename = 'HW3/Utilities.csv'
cluster_number = 1
save_filename = data_filename.replace('.csv', '_hc_cluster.csv')
data = Hierarchical_template.loadDataSet(data_filename)
clusterAssment = Hierarchical_template.agglomerative_with_min(data,␣
 ↪cluster_number)
Hierarchical_template.saveData(save_filename, data, clusterAssment)
```

```
0-th merging: 12, 21, 23
1-th merging: 10, 13, 24
2-th merging: 4, 24, 25
3-th merging: 7, 23, 26
4-th merging: 25, 20, 27
5-th merging: 14, 19, 28
6-th merging: 1, 18, 29
7-th merging: 15, 26, 30
8-th merging: 29, 28, 31
9-th merging: 2, 27, 32
10-th merging: 8, 16, 33
11-th merging: 32, 30, 34
12-th merging: 34, 22, 35
13-th merging: 9, 31, 36
14-th merging: 35, 36, 37
15-th merging: 6, 37, 38
16-th merging: 3, 38, 39
17-th merging: 39, 33, 40
18-th merging: 17, 40, 41
19-th merging: 11, 41, 42
20-th merging: 5, 42, 43
```

0.4 4) The codes of your K-means and hierarchical clustering algorithm implementation (i.e., the four functions: assignCluster, getCentroid, merge_cluster and update_distance).

0.4.1 K-Means functions

[6]:

```
def assignCluster(dataSet, k, centroids):
    '''For each data point, assign it to the closest centroid
    Inputs:
        dataSet: each row represents an observation and
                 each column represents an attribute
        k:  number of clusters
        centroids: initial centroids or centroids of last iteration
    Output:
        clusterAssment: list
            assigned cluster id for each data point
    '''
    #TODO
    clusterAssment = []
    for data in dataSet:
        dist = []
        for centroid in centroids:
            dist.append(np.linalg.norm(data-centroid))
        clusterAssment.append(np.argmin(dist))

    return clusterAssment


def getCentroid(dataSet, k, clusterAssment):
    '''recalculate centroids
    Input:
        dataSet: each row represents an observation and
            each column represents an attribute
        k:  number of clusters
        clusterAssment: list
            assigned cluster id for each data point
    Output:
        centroids: cluster centroids

    #take call the same values from cluster assesment and mean them to find␣
 ↪centroid
    '''
    dataSet = np.array(dataSet)
    clusterAssment = np.array(clusterAssment)
    centroids = []
    for i in range(k):
        centroids.append(np.mean(dataSet[clusterAssment == i],axis=0))
    return centroids
```

0.4.2 Hierarchical clustering functions

[7]:

```
def merge_cluster(distance_matrix, cluster_candidate, T):
    ''' Merge two closest clusters according to min distances
    1. Find the smallest entry in the distance matrix-suppose the entry
        is i-th row and j-th column
    2. Merge the clusters that correspond to the i-th row and j-th column
        of the distance matrix as a new cluster with index T

    Parameters:
    ------------
    distance_matrix : 2-D array
        distance matrix
    cluster_candidate : dictionary
```

```
        key is the cluster id, value is point ids in the cluster
    T: int
        current cluster index

    Returns:
    ------------
    cluster_candidate: dictionary
        upadted cluster dictionary after merging two clusters
        key is the cluster id, value is point ids in the cluster
    merge_list : list of tuples
        records the two old clusters' id and points that have just been merged.
        [(cluster_one_id, point_ids_in_cluster_one),
         (cluster_two_id, point_ids_in_cluster_two)]
    '''
    merge_list = []
    min_val_index = np.argmin(distance_matrix.flatten())
    i , j = np.unravel_index([min_val_index], distance_matrix.shape)

    #cluster_i = []
    #cluster_j = []
    for k , v in cluster_candidate.items():
        if i in v:
            cluster_i = v
            k_i = k
        if j in v:
            cluster_j = v
            k_j = k

    del cluster_candidate[k_i] #removing the original clusters
    del cluster_candidate[k_j]
    merge_list.append((k_i , cluster_i))
    merge_list.append((k_j , cluster_j))
    #cluster_candidate[T] = cluster_i.extend(cluster_j) # sort of joining the clusters
    cluster_candidate[T] = cluster_i + cluster_j

    return cluster_candidate, merge_list


def update_distance(distance_matrix, cluster_candidate, merge_list):
    ''' Update the distantce matrix

    Parameters:
    ------------
    distance_matrix : 2-D array
        distance matrix
    cluster_candidate : dictionary
```

```
        key is the updated cluster id, value is a list of point ids in the␣
↪cluster
    merge_list : list of tuples
        records the two old clusters' id and points that have just been merged.
        [(cluster_one_id, point_ids_in_cluster_one),
         (cluster_two_id, point_ids_in_cluster_two)]

    Returns:
    ------------
    distance_matrix: 2-D array
        updated distance matrix
    '''
    l_1 = merge_list[0][1]
    l_2 = merge_list[1][1]
    for i in l_1:
        for j in l_2:
            distance_matrix[i][j] = 1e7
            distance_matrix[j][i] = 1e7

    return distance_matrix
```