ECE 59500 IDM: PROGRAMMING ASSIGNMENT 3 REPORT

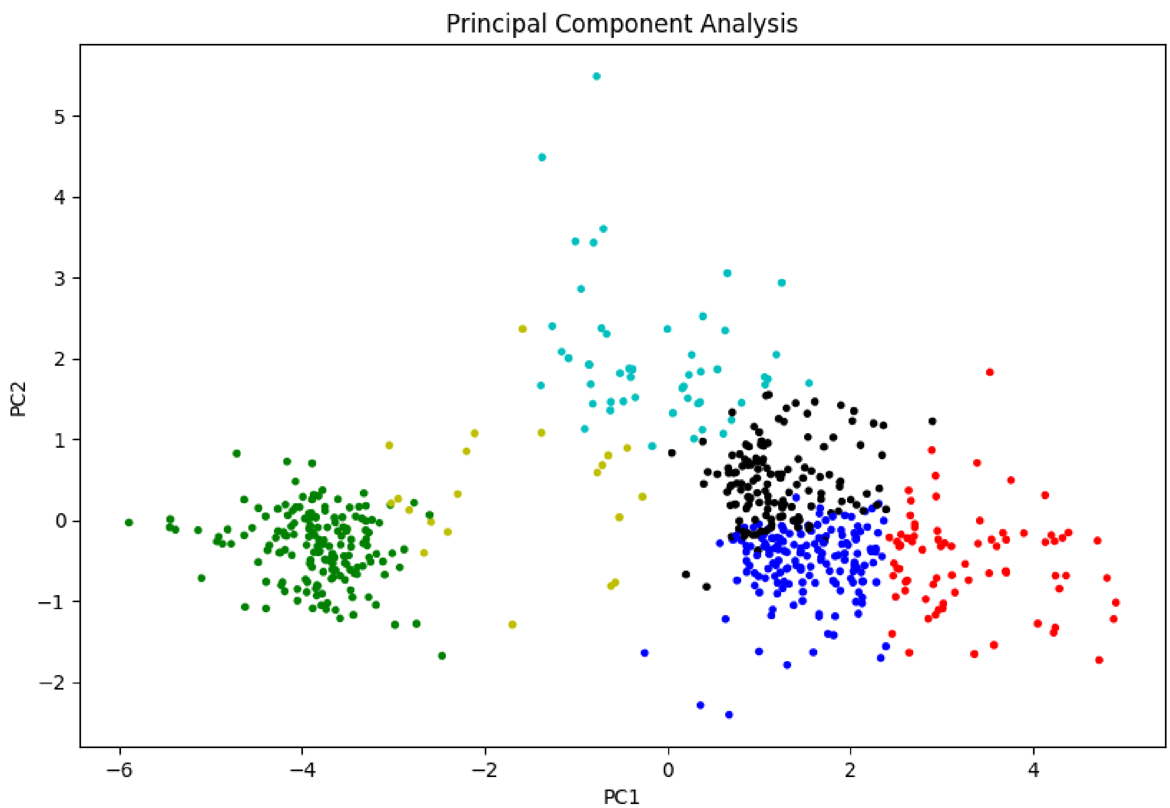
(sramagir@purdue.edu)

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

Answer:

[[[-0.24127142857142866, -0.12875, 0.062249999999999986, 0.1733500000000001, 0.21791428571428573, 1.6516928571428573, 1.9053214285714295]],
[[-0.9535098039215685, -1.4716470588235298, 0.07752941176470592, -0.1794901960784313, -1.0048235294117647, 1.1512156862745098, 0.9688039215686274]],
[[0.1651036585365854, 0.0916524390243902, -0.10388414634146342, -0.5525853658536584, -0.6300853658536584, -1.7231829268292682, -1.7548109756097554]],
[[0.023285714285714295, 0.2508095238095238, -0.2769523809523809, -0.36399999999999993, -0.7354285714285714, -0.8946190476190475, 0.7001428571428571]],
[[-0.0015555555555555737, 0.15650617283950607, 0.35625308641975295, 0.7015802469135799, 1.0097160493827158, 1.8423148148148147, 1.6434197530864196]],
[[-0.039328947368421054, 0.15394736842105264, 0.436078947368421, 1.1058157894736842, 1.4487105263157893, 3.016342105263157, 2.8293815789473675]]]

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



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

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

(4.) The codes of your K-means and hierarchical clustering algorithm implementation.

K-Means Clustering Implementation - Python Code:

```
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


    Implement K-means algorithm as follows:
            Repeat T times:
              • For each object xi
                 ▪ Calculate Euclidean distance between xi and each of the K centroids
                 ▪ Assign xi to the cluster whose centroid is the closest to xi
              • For each cluster
                 ▪ Calculate its centroid as the mean of all the objects in that cluster
    '''
    #TODO

    clusterAssment = []    # List for assigned cluster id for each data point

    for each_object_X in dataSet:

        # Initialization of required variables
        min_dist = 1000000        # A very high value
        index_counter = 0

        Xi = -1        # Index of the datapoint Xi which is closest to the cluster (whose
centroid is known)

        for each_centroid in centroids:

            dist = distance.euclidean(each_object_X, each_centroid)
```

```
            if dist < min_dist:
                Xi = index_counter
                min_dist = dist

            index_counter = index_counter + 1

        clusterAssment.append(Xi)

    return clusterAssment


def getCentroid(dataSet, k, clusterAssment):
    '''Recalculating the 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
    '''
    #TODO
    centroids = []      # List Initialization

    for each_cluster in range(0, k):

        centr = []

        for j in range(0, len(clusterAssment)):
            if clusterAssment[j] == each_cluster:
                centr.append(dataSet[j])

        centr = np.array(centr)

        cluster_centroid = centr.mean(axis = 0)  # The mean of all the objects in that cluster

        centroids.append(cluster_centroid.tolist())

    return centroids
```

Hierarchical Clustering Implementation – Python Code:

```
def merge_cluster(distance_matrix, cluster_candidate, T):

    ''' Comments: 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)]


    Implement Hierarchical clustering algorithm (with Min as inter-cluster distance
definition):
        • Obtain the distance matrix by computing Euclidean distance between each pair of
objects
        • Let each object be a cluster (Assign the cluster index as 1 to N, where N is the
number of objects)
        • Set the current index as T = N+1
        • Repeat
          ▪ Find the smallest entry in the distance matrix—suppose the entry is i-th
            row and j-th column
          ▪ 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
          ▪ Remove the rows and columns of the two old clusters and add new row
```

and column for the new cluster to the distance matrix by computing the distance between the new cluster and each of the remaining clusters

  ▪ *T=T+1*

- *Until only one cluster remains*

```
'''
#TO DO

merge_list = []

# Finding Minimum Value (Smallest entry) in the Distance Matrix (currently)
min_value = np.inf
for i in range(len(distance_matrix)):
  for j in range(i):
    if( distance_matrix[i][j] < min_value and i != j):
      min_value = distance_matrix[i][j]
      min_i = i
      min_j = j


# min_i => which cluster in cluster_candidate? min_j => which cluster in
cluster_candidate?
# print(min_i,min_j)
cluster_i = 0
cluster_j = 0

for k in cluster_candidate:
  values = cluster_candidate[k]
  # number of times element exists in list
  exist_count = values.count(min_i)
  # checking if it is more than 0
  if exist_count > 0:
    cluster_i = k

for l in cluster_candidate:
  values2 = cluster_candidate[l]
  # number of times element exists in list
  exist_count2 = values2.count(min_j)
  # checking if it is more than 0
  if exist_count2 > 0:
    cluster_j = l

#-------------------------------------------------------------------
# Finding merge_list
```

```
    # Extracting specific keys from dictionary
    res = dict((k, cluster_candidate[k]) for k in [cluster_i, cluster_j]
          if k in cluster_candidate)


    # Converting into list of tuples
    merge_list = [(k, v) for k, v in res.items()]

    # Printing list of tuple
    # print(merge_list)

    #--------------------------------------------------------------------
    # print(cluster_i, cluster_j)

    # print(merge_list)

    # Pop Clusters having i and j from cluster_candidate dict
    cluster_candidate.pop(cluster_i)
    cluster_candidate.pop(cluster_j)

    # Add a new entry with cluster index T which has value: list of indexes of data points
in merged cluster
    merge_list_temp = merge_list

    l1 = merge_list_temp[0][1]
    l2 = merge_list_temp[1][1]
    new_list = l2 + l1              # Concatenates l1 to l2

    # Now we need to add the new_list to cluster_candidate dict with cluster index as T

    cluster_candidate[T] = new_list

    # print(cluster_candidate)

    return cluster_candidate, merge_list
```