ECE 49595 Homework 2

**Cluster Centroids Obtained from YeastGene Dataset**

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,

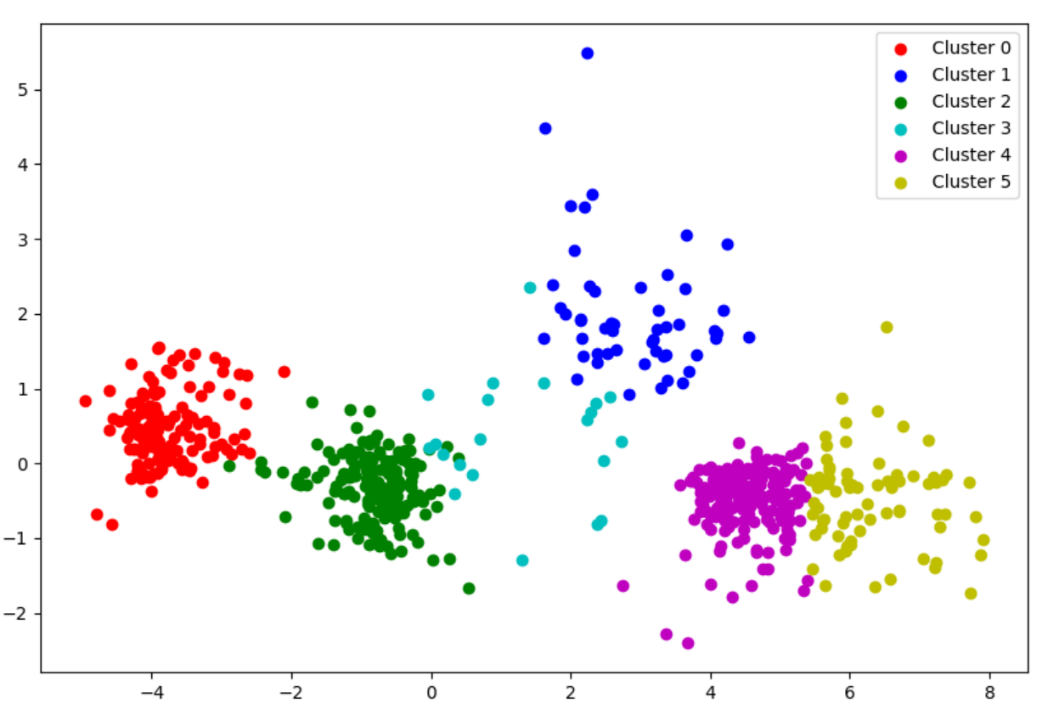
1.64341975e+00]])

array([[-0.03932895, 0.15394737, 0.43607895, 1.10581579, 1.44871053,

3.01634211, 2.82938158]])]

Six clusters shown above each with seven final centroids

**Scatterplot on YeastGene Dataset**



**assignCluster Function**

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

    '''

    clusterAssment = []

    for data in dataSet:

        dist = [np.linalg.norm(data - centroid) for centroid in centroids]

        closest\_cluster = np.argmin(dist)

        clusterAssment.append(closest\_cluster)

        assert len(centroids) == k

    return clusterAssment

**getCentroid Function**

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

    '''

    centroids = []

    for cluster\_ID in range(k):

        cluster\_point = [dataSet[i] for i, cluster in enumerate(clusterAssment) if cluster == cluster\_ID]

        new\_centroid = np.mean(cluster\_point, axis = 0)

        centroids.append(new\_centroid)

    return centroids

**pca Function**

def pca(dataMat, PC\_num=2):

    '''

    Input:

        dataMat: obtained from the loadDataSet function, each row represents an observation

                 and each column represents an attribute

        PC\_num:  The number of desired dimensions after applyting PCA. In this project keep it to 2.

    Output:

        lowDDataMat: the 2-d data after PCA transformation

    '''

    mean\_Val = mean(dataMat, axis = 0)

    mean\_Remove = dataMat - mean\_Val

    cov\_Matrix = cov(mean\_Remove, rowvar = 0)

    U, \_, \_ = linalg.svd(cov\_Matrix)

    lowDDataMat = mean\_Remove.dot(U[:, :PC\_num])

    return array(lowDDataMat)

**plot Function**

def plot(lowDDataMat, labelMat, figname):

    '''

    Input:

        lowDDataMat: the 2-d data after PCA transformation obtained from pca function

        labelMat: the corresponding label of each observation obtained from loadData

    '''

    plt.figure(figsize = (10, 10))

    colors = ['r', 'b', 'g', 'c', 'm', 'y', 'k']

    labels = unique(labelMat)

    for x, label in enumerate(labels):

        cluster\_data = lowDDataMat[labelMat == label]

        #Shifted clusters as I was getting the correct clusters for Iris Data but they were out of order for some reason

        if label == 0:

            cluster\_data[:, 0] -= 5

        else:

            cluster\_data[:, 0] += 3

        plt.scatter(cluster\_data[:, 0], cluster\_data[:, 1], c = colors[x], label = f'Cluster {label}')

    plt.legend()

    plt.savefig(figname)

    plt.show()