Q1. Command to run code "python Q1.py jain.txt"

Algorithm (K-Means):

1. Input Data frame & classes
2. Randomly chosen **k** (Number of classes) data point from data frame and made it initial centroids
3. Assigned initial cluster to each data point using "Euclidean Distance"
4. Calculate mean for each cluster and updated centroids
5. Assigned new cluster to each data point using "Euclidean Distance" and updated centroids
6. Repeat step 4 until last assigned cluster become equals to new cluster for each data point
7. Return last assigned cluster & centroid

Algorithm (Spectral):

1. Input Data frame, classes & Sigma
2. Created Adjacency Matrix using weight formula

$$\mathbf{W}_{i,j} = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\sigma^2}}$$

3. Created Degree Matrix using formula

$$d_i = \sum_{j=1}^{n} w_{ij}$$

4. Created Laplacian Matrix Using L = D – W
5. Generated Eigen value & corresponding Eigen vector of Laplacian Matrix L
6. Taken **k** (Number of classes) Eigen vector corresponding to k smallest Eigen values
7. Normalised matrix generated from step 6
8. Applied K-means algorithm to assign cluster for each data point generated in step 7
9. Return result of step 8

Result (On jain dataset):

Class wise performance K Means

For class 2, estimated cluster label correct percentage is **100.00 %**
For class 1, estimated cluster label correct percentage is **80.43 %**

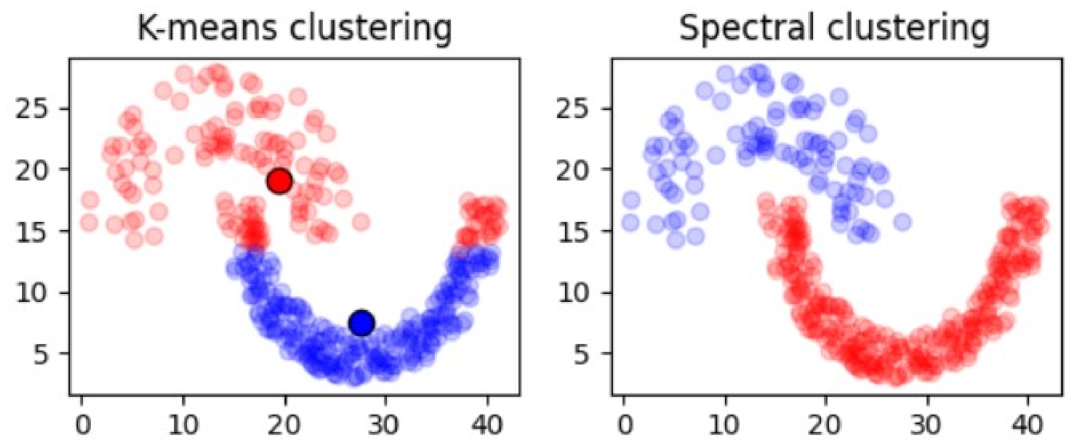
Class wise performance SPECTRAL

For class 2, estimated cluster label correct percentage is **100.00 %**
For class 1, estimated cluster label correct percentage is **100.00 %**

Over all

Estimated cluster label correct percentage (K MEANS) = **85.52 %**

Estimated cluster label correct percentage (Spectral) = **100.00 %**



Evaluation:

Seeing the performance of both algorithms for each data point I can say Spectral clustering is performing better than k-means.

Q2. Command to run code "python Q2.py iris.data"

Algorithm (PCA):

1. Input Data frame(X) & dimensional (K)

   -----Compute & normalize Eigen Vector-----

2. Compute dot product Y = X & X^t
3. Compute Eigen value and Eigen vector Y
4. Normalised each Eigen vector and stored in a list in decreasing order of Eigen values

   ------ Reduce Dimension of data point-----

5. Compute dot product of first K Eigen vector with data set to K dimensional data
6. Return output of step 5 and Normalised Eigen vector

Result (On iris dataset):

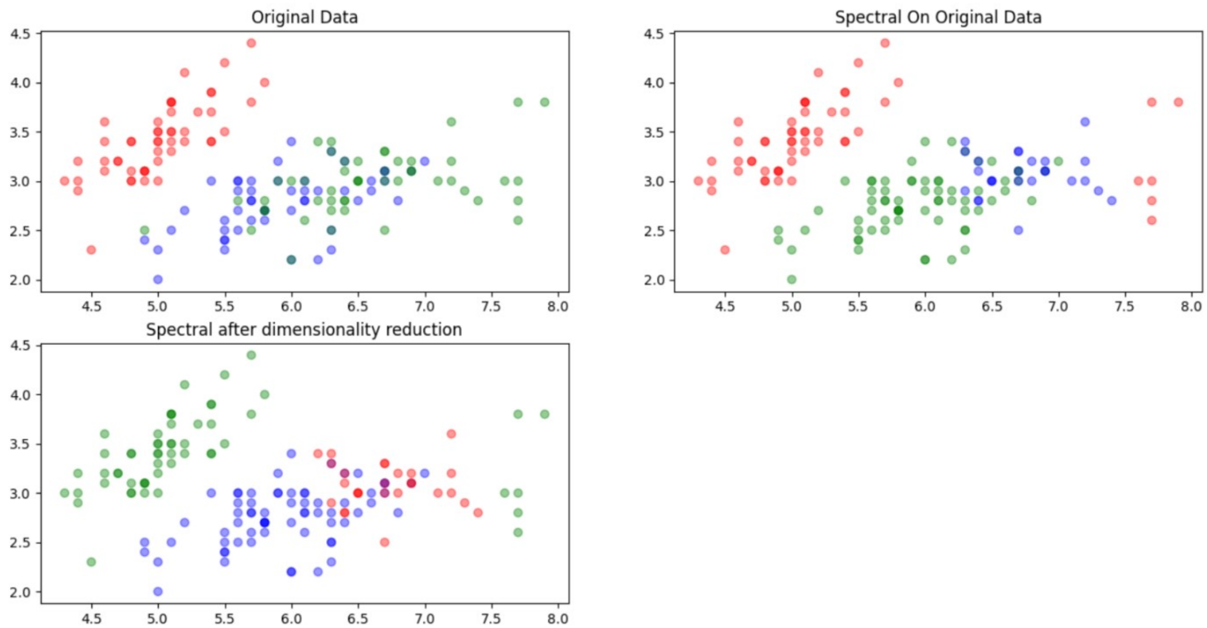
Input:

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa

4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa

Output for K = 2

5.912204	2.303442
5.572076	1.973831
5.446485	2.096533
5.436019	1.871681
5.875066	2.329348



Reconstruction Error For K=1 is **191.132976631595**
Reconstruction Error For K=2 is **41.687072308635**
Reconstruction Error For K=3 is **17.360507255270**
Reconstruction Error For K=4 is **0.000000000001**

Class wise performance on original data

For class Iris-setosa, estimated cluster label correct percentage is **100.00 %**
For class Iris-versicolor, estimated cluster label correct percentage is **100.00 %**
For class Iris-virginica, estimated cluster label correct percentage is **52.00 %**
Over all performances: 84.000 %

Class wise performance on 2-dimensional data

For class Iris-setosa, estimated cluster label correct percentage is **100.00 %**

For class Iris-versicolor, estimated cluster label correct percentage is **100.00 %**
For class Iris-virginica, estimated cluster label correct percentage is **54.00 %**
Over all performances: **84.667 %**

Spectral on original vs Reduced dimensionality data

For class Iris-setosa, estimated cluster label correct percentage is **100.00 %**
For class Iris-versicolor, estimated cluster label correct percentage is **100.00 %**
For class Iris-virginica, estimated cluster label correct percentage is **98.53 %**
Over all performances: **99.333 %**