Eren Barış Bostancı
68770

ENGR 421 – HW8

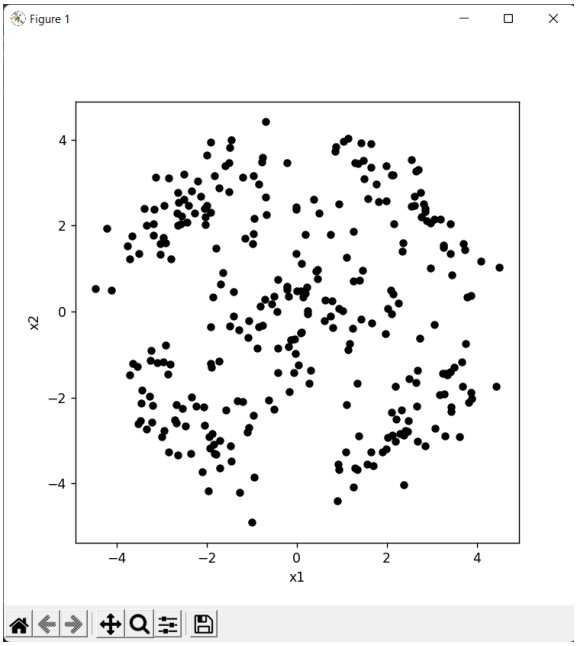
Data Importing:

I imported the data.

```
X = np.genfromtxt("hw08_data_set.csv", delimiter=",")

# sample size
N = X.shape[0]
# cluster count
K = 5


plt.figure(figsize=(6, 6))
plt.plot(X[:, 0], X[:, 1], "k.", markersize=10)
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



B, D, L, Lsymmetric, Z Matrixes

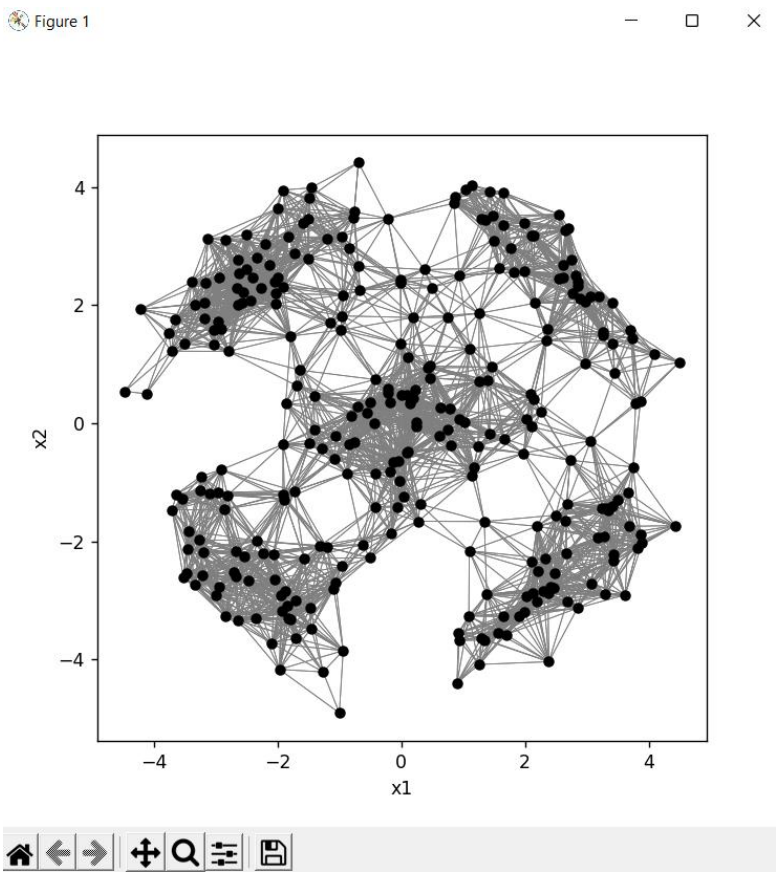
I constructed B matrix according to the rule given in pdf and plot connectivity matrix. I set threshold to 1.25 and used Euclidean distance formula for comparing with threshold.

$$b_{ij} = \begin{cases} 1, & \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 < \delta \\ 0, & \text{otherwise.} \end{cases}$$
$$b_{ii} = 0$$

```
S = 1.25

B = np.zeros((N, N))
for i in range(N):
    for j in range(N):
        if i == j:
            B[i][j] = 0
        else:
            # euclidean distance
            distance = np.sqrt((X[j][0] - X[i][0]) ** 2 + (X[j][1] - X[i][1]) ** 2)
            if distance < S:
                B[i][j] = 1
            else:
                B[i][j] = 0


plt.figure(figsize=(6, 6))
for i in range(N):
    for j in range(N):
        if B[i][j] == 1:
            plt.plot([X[i][0], X[j][0]], [X[i][1], X[j][1]], "0.5", linewidth=0.5)
plt.plot(X[:, 0], X[:, 1], '.', markersize=10, color="black")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()
```



Then I calculated D, L and Lsymmetric matrixes according to the formula given in the lecture notes. However, I never used the L matrix since I need to use Lsymmetric matrix. I add it since it wants in pdf.

$$d_{ii} = \sum_{j \neq i} b_{ij} \quad \forall i$$

↳ # of neighbors of datapoint i

$$d_{ij} = 0 \quad \forall (i, j \neq i)$$

$$L_{N \times N} = D_{N \times N} - B_{N \times N}$$

$$\mathbf{L}_{symmetric} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{B}\mathbf{D}^{-1/2}$$

```
D = np.zeros((N, N))
for i in range(N):
    total = 0
    for j in range(N):
        total = total + B[i][j]
    D[i][i] = total

I = np.eye(N)
L = np.subtract(D, B)
sqrt_inv_D = np.sqrt(np.linalg.inv(D))
L_sym = np.subtract(I, np.matmul(sqrt_inv_D, np.matmul(B, sqrt_inv_D)))
```

I take the eigen values and vectors than take the 2nd smallest, 3rd smallest, 4th smallest, 5th smallest and 6th smallest eigenvalues and form Z matrix with eigen vectors.

```
R = 5
eigen_vals, eigen_vecs = np.linalg.eig(L_sym)
sorted_eigen_vals = eigen_vals.argsort()
Z = eigen_vecs[:, sorted_eigen_vals[1:R + 1]]
```

K-Means Clustering Algorithm

I take the update_centroids, update_memberships, plot_current_centroids, and algorithm iteration part from lab11. I set the initial centroids to 29th, 143rd, 204th, 271st, and 277th rows of Z matrix.

```
def update_centroids(memberships, X):
    if memberships is None:
        # initialize centroids
        centroids = np.vstack([Z[28], Z[142], Z[203], Z[270], Z[276]])
    else:
        # update centroids
        centroids = np.vstack([np.mean(X[memberships == k, :], axis=0) for k in range(K)])
    return centroids


def update_memberships(centroids, X):
    # calculate distances between centroids and data points
    D = spa.distance_matrix(centroids, X)
    # find the nearest centroid for each data point
    memberships = np.argmin(D, axis=0)
    return memberships
```

Eren Barış Bostancı
68770

```python
def plot_current_state(centroids, memberships, X):
    cluster_colors = np.array(["#1f78b4", "#33a02c", "#e31a1c", "#ff7f00", "#6a3d9a", "#b15928",
                               "#a6cee3", "#b2df8a", "#fb9a99", "#fdbf6f", "#cab2d6", "#ffff99"])
    if memberships is None:
        plt.plot(X[:, 0], X[:, 1], ".", markersize=10, color="black")
    else:
        for c in range(K):
            plt.plot(X[memberships == c, 0], X[memberships == c, 1], ".", markersize=10,
                     color=cluster_colors[c])
    for c in range(K):
        plt.plot(centroids[c, 0], centroids[c, 1], "s", markersize=12,
                 markerfacecolor=cluster_colors[c], markeredgecolor="black")
    plt.xlabel("x1")
    plt.ylabel("x2")
```

```python
centroids = None
memberships = None
iteration = 1
while True:
    print("Iteration#{}:".format(iteration))

    old_centroids = centroids
    centroids = update_centroids(memberships, Z)
    if np.alltrue(centroids == old_centroids):
        break

    old_memberships = memberships
    memberships = update_memberships(centroids, Z)
    if np.alltrue(memberships == old_memberships):
        break

    iteration = iteration + 1

centroids = update_centroids(memberships, X)
plot_current_state(centroids, memberships, X)
plt.show()
```