ENGR 421 - Machine Learning

Mehmet Ustek

HW8 Report

Completeness of the project:
All requirements are done with correct outputs regarding the homework description.

Initially, I gathered the data points from the "hw08_data_set.csv" file. The resulting data points are visualized below:

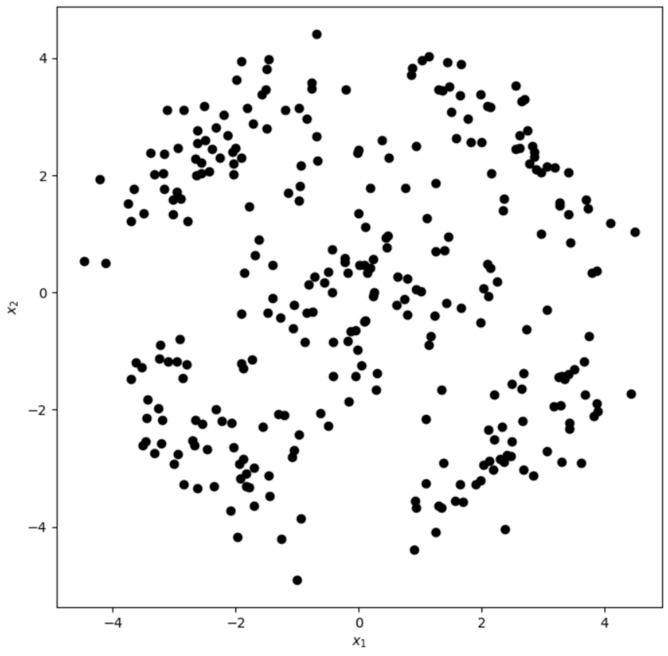


Figure 1: Data points

The coding part:
First, I created functions bij(X1, X2, threshold), L_symmetric(D,B), get_D_matrix(B).
The first function bij returns the adjacency matrix based on the threshold 1.25. The diagonals are set to 0 and the other values are assigned whether the point is connected with the corresponding point in the matrix. If two points' distance is smaller than 1.25, then they are connected, otherwise they are not connected.
The second function get_D_matrix returns the D matrix given the adjacency matrix. The D matrix consists of the sum of the rows, which makes up the diagonals of the D matrix for that corresponding row.
The third function L_symmetric(D,B) implements the laplacian symmetric matrix notation calculating with equation $L = I - (D^{-½} * B * D^{-½})$, and it returns the normalized L matrix. For experimenting with other algorithms, I also created L_not_normalized and L_random_walk algorithms too.
Now that we have B, D, L matrices, we can continue with other steps.
To visualize the connectivity matrix, I created a function draw(B, X) that takes B matrix and X data points as parameters and draws the below connectivity matrix visualization:

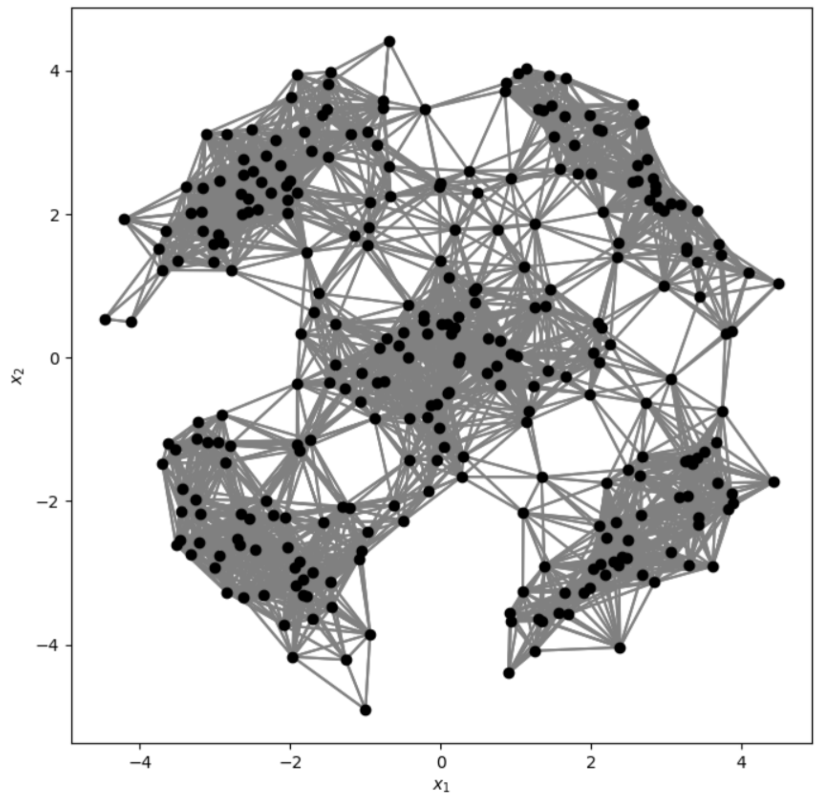


Figure 2: Connectivity matrix Visualization

To display this visual, I draw a line between each connected point using the pyplot module.

Inside the function spectral_clustering(X, R), I call the functions bij, get_D_matrix, and L_symmetric to gather the values B, D, L. Then using the np.linalg.eig function I gathered the eigenvalues and eigenvectors. Since these eigenvalues are not ordered, I ordered them with the argsort function on eigenvalues. Since the smallest eigenvalue is 0, we have to skip it. Now we have R=5 smallest eigenvectors ranging from 1 to 6, which makes up the Z matrix. Using this Z matrix, we have to call the K-means algorithm as such: K_means(Z). The initial centroids for K-means are given at homework description as rows 28, 142, 203, 270 and 276 of the Z matrix. (with 0 included, python representation) The K_means algorithm will return membership values for each data point. Next, we should find appropriate centroids, or center of masses to indicate on our visuals. To do that, call:
Centroids = update_centroids(memberships, X) with our original data points X.
The centroids and memberships from the K-means algorithm are as follows:

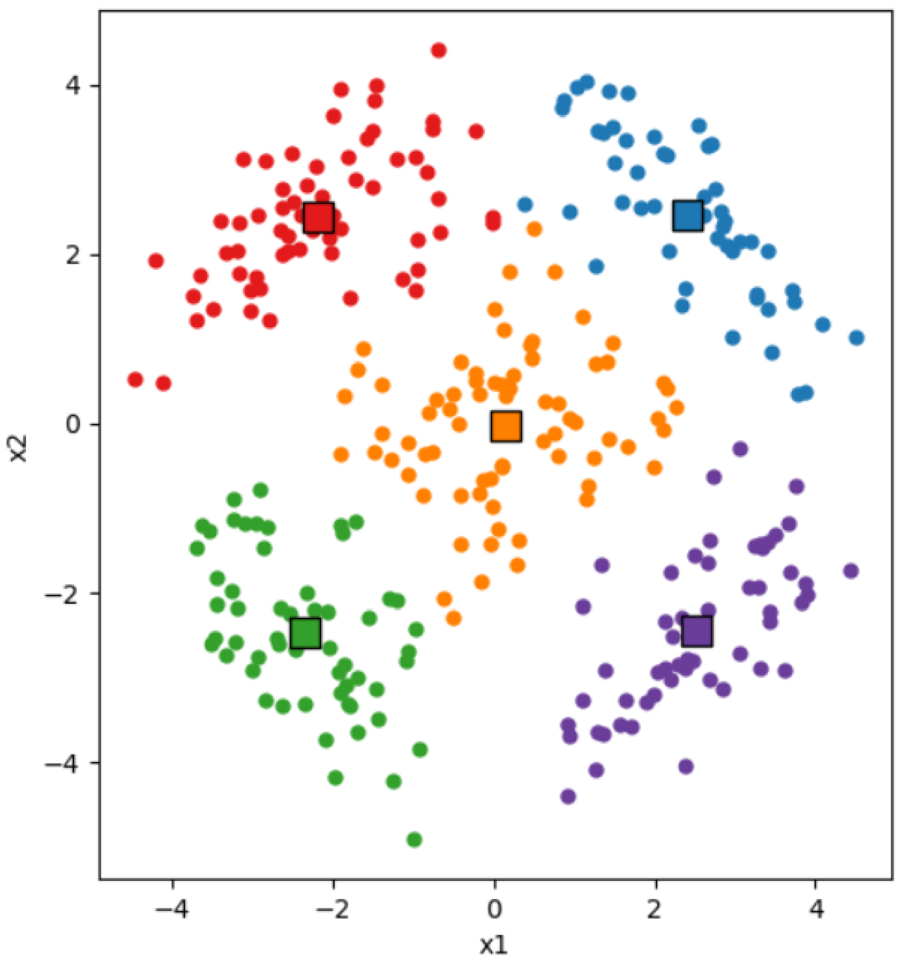
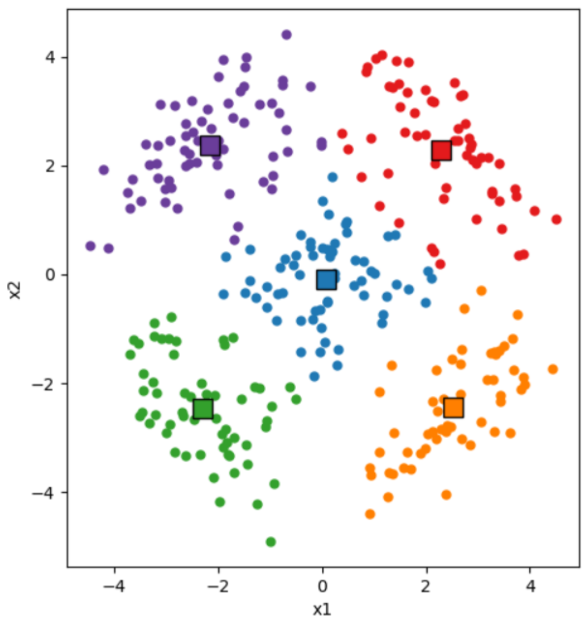
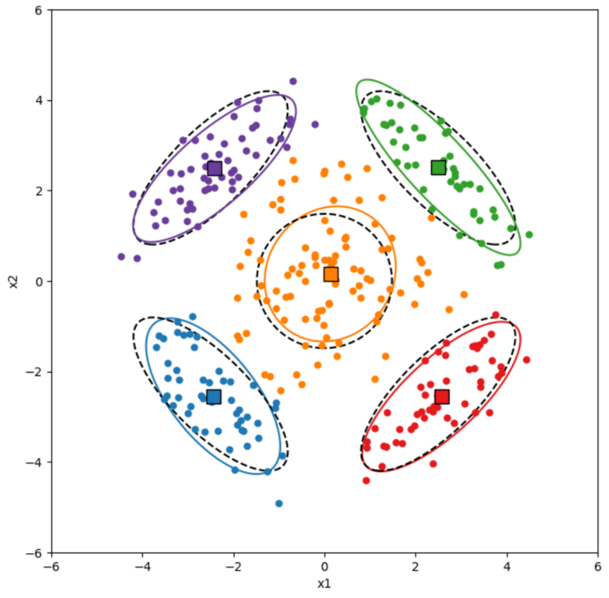


Figure 3: Centroids and Memberships of **Spectral Clustering**

The resulting memberships displays the correctness of the spectral clustering. All clusters are separated quite effectively. For this small number of data points of length 300, the 3 clustering algorithms gives totally different results. Below, we can see K-means clustering and Expectation Maximization Clustering algorithms for comparison. Especially the edge values in the upper middle region, varies for each algorithm. I genuinely believe that Spectral Clustering does a better job for these data points, but it is hard even for humans to classify these points into clusters. But I believe with normalizations and connectivities, the spectral clustering does a slightly better job.





Figures 4-5: K-means Clustering and Expectation Maximization Clustering (from left to right)

Acknowledgements:
I have never taken or given a code, or a snippet of code to/from anyone. Thus, all the work completely belongs to me.
I have also pushed all my code into github during the development of this project. In any case of suspicion of plagiarism, I can provide that repository.