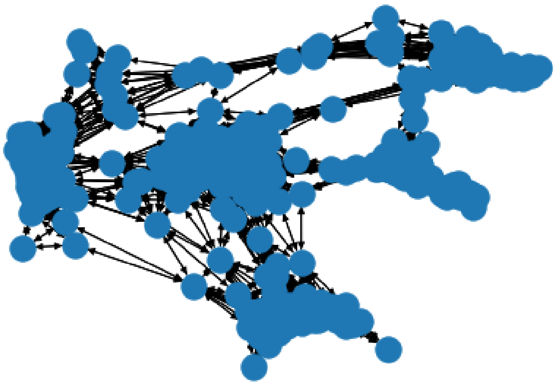
ENGR 421 / DASC 521: Introduction to Machine Learning
Homework 06: Spectral Clustering

İzel Yazıcı - Jun 11, 2021

I calculated the Euclidean distances between the pairs of data points. The data point pairs with distance less than or equal to $\delta = 1.25$ are considered as connected. Then I constructed B connectivity matrix.
I visualized the connectivity matrix by drawing a line between two data points if they are connected. Your figure should be similar to the following figure.



I normalized the Laplacian matrix using the following formula:

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

```
# Laplacian Matrix
def compute_laplacian(W):

    # Lsymetric = I-D^{-1/2} B D^{-1/2}

    # calculate row sums
    d = W.sum(axis=1)

    #create degree matrix
    D = np.diag(d)


    #H = np.sum(G[0])
    for id, x in enumerate(W):
        D[id][id] = np.sum(x)

    I = np.identity(len(W))

    # Lsymetric = I-D^{-1/2} B D^{-1/2}

    D_inv_sqrt = np.linalg.inv(np.sqrt(D))

    L = I - np.dot(D_inv_sqrt, W).dot(D_inv_sqrt)

    return L
```

Then I found the eigenvectors of the normalized Laplacian matrix and pick $R = 5$ eigenvectors that corresponds to R smallest eigenvectors. Using these eigen vectors I have constructed the matrix $\mathbf{Z}$

```python
# Eigen Vectors
R = 5
def get_eigvecs(L, R):

    eigvals, eigvecs = np.linalg.eig(L)
    # sort eigenvalues and select R smallest values - get their indices
    ix_sorted_eig = np.argsort(eigvals)[:R]

    #select k eigenvectors corresponding to R-smallest eigenvalues
    return eigvecs[:,ix_sorted_eig]

```

I runned k-means clustering algorithm on $\mathbf{Z}$ matrix to find $K = 5$ clusters. When I was initializing the algorithm, I have used the following rows of $\mathbf{Z}$ matrix for initial centroids: 85, 129, 167,187, and 270

```python
# K=5
def k_means_pass(X, K, n_iters):

    #initial centroids
    centers = [85,129,167,187,270]

    for iteration in range(n_iters):
        #calculate distances for every point in X to each of the k centers
        distance_pairs = pairwise_distances(X, centers)

        #assign label to each point - index of the centroid with smallest distance
        labels = np.argmin(distance_pairs, axis=1)
        new_centers = [np.nan_to_num(X[labels == i].mean(axis=0)) for i in range(K)]
        new_centers = np.array(new_centers)

        #check for convergence of the centers
        if np.allclose(centers, new_centers):
            break

        #update centers for next iteration
        centers = new_centers


    return centers, labels
```

Finaly I called the spectral_clustering function that I defined as following:

```python

def spectral_clustering(X, K):

    #create weighted adjacency matrix
    W = nearest_neighbor_graph(X)

    #create unnormalized graph Laplacian matrix
    L = compute_laplacian(W)

    #create projection matrix with first k eigenvectors of L
    E = get_eigvecs(L, K)

    #return clusters using k-means on rows of projection matrix
    y = k_means_clustering(E, K)
    return np.ndarray.tolist(y)

spec_clusters= spectral_clustering(X, 5)
```

But unfortunately I couldn’t get similar solution as in the HW documentation spectral clustering figure.