Data Mining - Assignment 4

Group 18
Gustavo Teodoro D. Beck - Fernando García Sanz

December 2, 2020

1 Task

The task of this assignment is to implement and to test the spectral graph clustering (K-eigenvector) algorithm described by Andrew Y. Ng, Michael I. Jordan, and Yair Weiss in the paper "*On Spectral Clustering: Analysis and an algorithm*" [1]. In order to test our solution two datasets where employed:

- `example1.txt` [2]: A collection prepared by Ron Burt of physicians in four towns in Illinois, Peoria, Bloomington, Quincy and Galesburg. The goal is to identify the 4 clusters.
- `example2.txt`: Synthetic data set with weights that define the relationship between the nodes.

2 Method

```python
def main():
    path = "data/example1.dat"
    graph = load_graph(path)
    print("Number of nodes: ", len(graph.nodes))
    pos = nx.spring_layout(graph)
    A = affinity_matrix(graph)
    L = L_matrix(A)
    X, k, eigen_values, fiedler_vec = eigenvector_matrix(L)
    Y = normalized_X(X)
    clustering = KMeans(n_clusters=k).fit(Y)
    labels = clustering.labels_
    plot_graph(graph, labels, pos)
    plot_eigen_values_diff(eigen_values)
    plot_fiedler(fiedler_vec)
    plot_sparse(A)


if __name__ == "__main__":
    main()
```

Figure 1: Our implementation.

The proposed method explains a different approach in order to perform graph clustering. The needed steps are the following:

1. Obtain the affinity matrix (A) of the graph. In order to do so, the adjacency matrix of the full graph has been computed and employed as the affinity one.

```python
def affinity_matrix(graph):
    """
    A (n x n)
    A_ii = 0
    A_ij = exp(-|s_i - s_j|^2 / (2 * sigma^2))
    """
    A = np.asarray(nx.adjacency_matrix(graph).todense())

    return A
```

Figure 2: Affinity matrix computation.

2. Compute the Laplacian matrix (L). This is done by first computing the D-matrix, a diagonal matrix where each element of the diagonal is equal to the sum of affinity matrix elements for that particular row - i.e. $D_{i,i} =$ sum of affinity matrix elements in row i. Then, the L-matrix is computed as $L = D^{-1/2}AD^{-1/2}$.

```python
def L_matrix(A):
    """
    L and D (n x n)
    D = diagonal matrix with D_ii = sum of affinity matrix i_th row
    L = D ^ (-1/2) * A * D ^ (-1/2)
    """
    D = np.diag(np.sum(A, axis=1))
    D_inv = np.linalg.inv(np.sqrt(D))
    L = D_inv @ A @ D_inv

    return L
```

Figure 3: L-Matrix computation.

3. Compute the eigenvectors of the L-matrix. The eigenvalues and eigenvectors are computed here, as well as the *Fiedler Vector*, which is the eigenvector corresponding to second smallest eigenvalue of the L-matrix. Given the eigenvalues, it is possible to calculate k, which denotes the position of the *eigengap*, the biggest difference between two consecutive eigenvalues. This suggests the appropriate number of clusters needed.

```python
def compute_k(eigen_values):
    eigen_values_diff = np.diff(eigen_values)
    index_largest_diff = np.argmax(eigen_values_diff) + 1
    k = len(eigen_values) - index_largest_diff
    return k


def eigenvector_matrix(L):
    """
    X (n x k)
    k largest eigenvectors of L orthogonal from each other
    """
    X_values, X_vectors = la.eigh(L)
    k = compute_k(X_values)
    fiedler_vec = sorted(X_vectors[:, -2])
    X = X_vectors[:, -k:]

    return X, k, X_values, fiedler_vec
```

Figure 4: Eigenvectors, eigengap, and Fiedler vector computation.

4. All eigenvectors in descending value order until reaching the eigengap are retrieved and normalized. These normalized vectors will be used in order to split the graph into k clusters by means of the *K-means* algorithm.

```python
def normalized_X(X):
    """
    Y (n x k)
    Y = X_ij / (Sum_j (X_ij ^ 2)) ^(1/2)
    """
    Y = X / np.sqrt(np.sum(pow(X, 2), axis=1)).reshape((-1, 1))

    return Y
```

Figure 5: Retrieved eigenvectors normalization.

3 Results

Once the K-eigenvector algorithm is implemented we tested it in both datasets, and the outcomes are the followings:

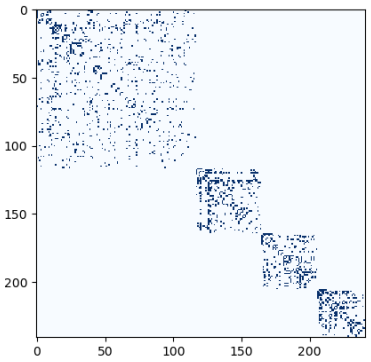

Figure 6: Affinity matrix sparsity representation.

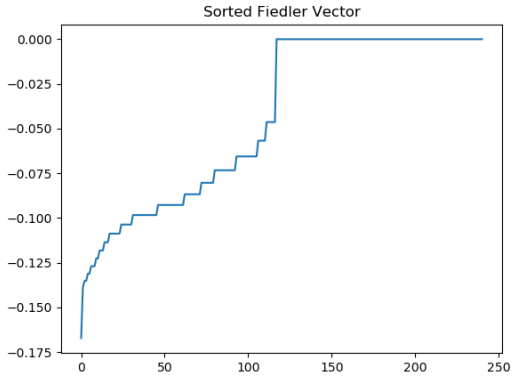

Figure 7: Fiedler vector of `example1.txt`.

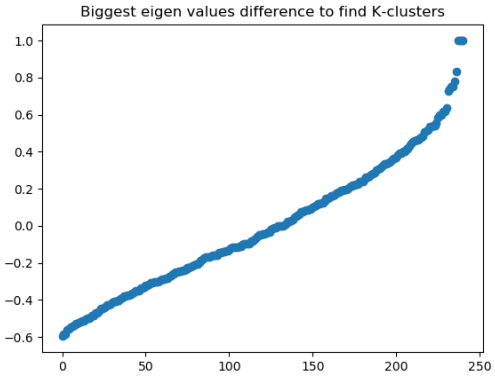

Figure 8: Eigengap representation.

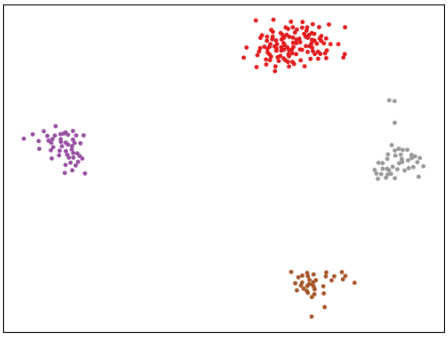

Figure 9: Clusters nodes representation of `example1.txt`.

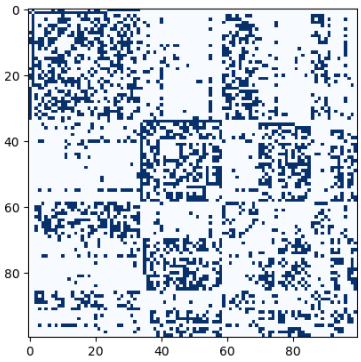

Figure 10: Affinity matrix sparsity representation.

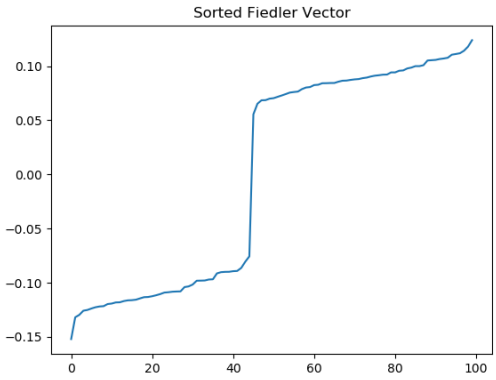

Figure 11: Fiedler vector of `example2.txt`.

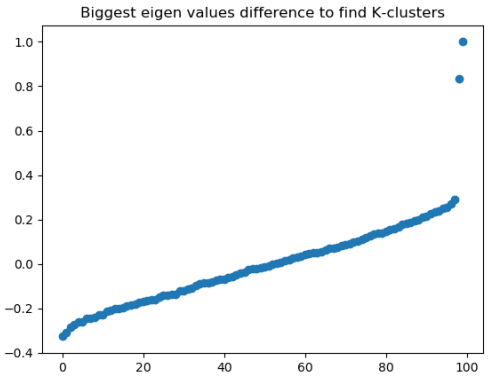

Figure 12: Eigengap representation.

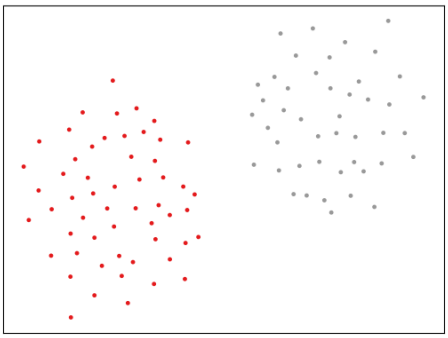

Figure 13: Clusters nodes representation of `example2.txt`.

After analysing the outcomes it is clear to see that `example1.txt` has clearer clusters. This can be identified before performing the K-means algorithm in figure 6. The sparse representation of the affinity matrix shows that 4 clusters have a strong connection within their clusters but not with each other. This is confirmed in figure 8 when 4 eigenvalues are more far apart from the others. Finally, figure 9 represents the nodes and the clusters classified by different colors based on the K-means algorithm.

On the other hand, for the synthetic data, the sparse representation of the affinity matrix (figure 10) shows many relationships between the nodes. However, the eigengap method ((figure 12) managed to identify that 2 eigenvalues could be distinguished from the others, generating the representation of the nodes in figure 13. In order to comprehend the differences between the affinity matrices representations, we decided to plot the connections between the nodes, which confirmed that the `example2.txt` had strong connections within its clusters, but fewer connections between the clusters.

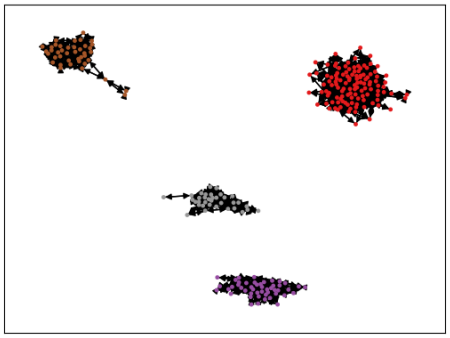

Figure 14: Clusters nodes' connections representation of `example1.txt`.

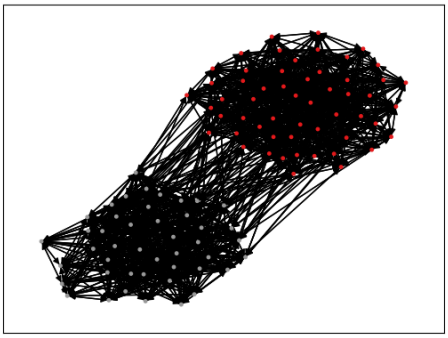

Figure 15: Clusters nodes' connections representation of `example2.txt`.

4 How to Run

In order to execute the code, having the datasets inside a `data` folder, it is just needed to execute the `SpectralClustering.py` class via `python3 SpectralClustering.py` command. It is possible to change the analyzed dataset by just changing the variable `path` inside the code.

The expected outputs are: the representation of the nodes and its clusters, the eigengap analysis, the Fiedler Vector plot, and the affinity matrix visualization.

References

[1] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in neural information processing systems*, pp. 849–856, 2002.

[2] R. Burt, "Innovation among physicians." `http://moreno.ss.uci.edu/data.html#ckm`.