An Implementation of Spectral Clustering in Julia

RIYA BHATIA

Section M

1 Introduction

A fundamental problem in computer science and machine learning is successfully drawing meaningful insights from data. Many times, the data that researchers aim to understand has also not been previously classified, and as a result, has a lack of labels associated with data points. In such cases, a particularly useful tool for understanding the structure of data is clustering, a technique that partitions data into groups such that two properties hold:

(a) Points within their clusters are especially similar to each other.

(b) Points within different clusters are especially dissimilar to each other.

Interestingly, clustering has been widely studied and utilized in a variety of real-world applications, playing a crucial role in data analysis. For example, clustering has been used in applications of document classification, where researchers place documents into categories based on their specific word frequencies and usages. The technique has been further used in bioinformatics, where gene expressions with similar DNA structures have been grouped together. As the clustering method groups data points into distinct clusters, the technique is also popularly used as a preprocessing step for supervised learning methods, which require data to be labeled (Aoullay, 2018).

Clustering itself is not an algorithm, but rather, constitutes a broad family of algorithms. This diversity in clustering stems from the fact that clusters are defined based on specific dataset properties and applications. Regardless of the application, a cluster should continue to hold the properties of similarity within clusters and dissimilarity between clusters; this measure of similarity, however, is dependent on the dataset and its features as well.

In this paper, we discuss spectral clustering in particular, which relies on foundations in graph theory and incorporates a more popular clustering algorithm, k-means clustering. We also compare spectral clustering with other clustering algorithms and find suitable uses for a variety of such techniques.

2 Mathematical Definitions and Notations

Before delving into spectral clustering, we want to first define specific terms that we will utilize in our discussion. Definitions concerning graphs inspired by those written on Wikipedia (Wikipedia contributors).

1. **Euclidean Norm**: The length of a vector, $\vec{v} = [v_1, v_2, ..., v_n]$, on an n-dimensional Euclidean space $\mathbb{R}^n$. The Euclidean norm, denoted $||\vec{v}||$, can be found using the formula,

$$||\vec{v}|| = \sqrt{\vec{v} \cdot \vec{v}} = \sqrt{v_1^2 + ... + v_n^2}$$

2. **Euclidean Distance**: The distance between two points in $\mathbb{R}^n$. In $\mathbb{R}^2$, if $x = (x_1, x_2)$ and $y = (y_1, y_2)$, then the Euclidean distance, denoted $d(x, y)$, is defined as

$$d(x, y) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2}$$

   In our first implementation of spectral clustering, we will utilize the Euclidean distance between points as a method to dictate similarity. Specifically, we'll use the Euclidean distance to compute the k nearest points to a specific point, where $k \in \mathbb{N}^+$. The similarity between the k nearest points will be the Euclidean distance, and the similarity between all other points will be 0.

3. **Undirected Similarity Graph**: In discrete mathematics, an undirected similarity graph G is defined as a structure that models pairs of elements that are in some way similar. G is defined as the pair $G = (V, E)$, where V is the set of elements denoted as vertices or nodes, and E is the set of vertices that are paired, denoted as edges. We also allow edges to be weighted based on the similarity between nodes.

4. **Connectivity**: In an undirected graph G, we define an unordered set of vertices, $\{a, b\}$, to be connected if there exists a path from a to b. Otherwise, the set of vertices is denoted as disconnected.

   In our second implementation of spectral clustering, we utilize the connectivity between points to dictate our measure of similarity.

5. **Weighted Adjacency Matrix**: An undirected similarity graph G with a set of vertices V can be represented by a weighted adjacency matrix W, an $n \times n$ square matrix, where $n = |V|$. In W, W_{ij} is represented by the similarity between vertex i and vertex j on G. As the graph is undirected, $W_{ij} = W_{ji}$ holds; thus, W is symmetric.

6. **Degree Matrix**: The degree matrix D of an undirected similarity graph with a set of vertices V is defined as an $n \times n$ diagonal matrix, where $n = |V|$. D has diagonals $D_{ij} = d(v_i)$ if $i = j$, where $d(v_i)$ is the number of edges that are connected to vertex v_i. If $i \neq j$, then $D_{ij} = 0$.

7. **Graph Laplacian Matrix**: For an undirected similarity graph G represented by the weighted adjacency matrix W and degree matrix D, the graph Laplacian matrix L is defined as

$$L = D - W$$

3 The K-Means Clustering Algorithm

To motivate the usage of spectral clustering, we will first discuss the k-means clustering algorithm, a popular technique to cluster data based on how compact a set of points are. Unlike spectral clustering, which is a graph theoretic algorithm, k-means clustering is a centroid-based algorithm, as it is based on the idea of minimizing the distance between points and their relative cluster centroid. K-means clustering is especially useful for clustering data that has spherical boundaries between clusters.

In order to apply k-means clustering on a dataset, we utilize ScikitLearn.jl, an interface that brings scikit-learn tools in Python to Julia. We import the KMeans package from ScikitLearn to perform k-means clustering.

However, in order to perform this technique, we must pre-determine a specific number of clusters to partition the data set into; namely, we must find k. In order to find the optimal number of clusters, we utilize the Elbow Method, which follows the following algorithm (Umargono et al, 2019).

1. Apply k-means clustering on the dataset for varying values of k, where $k \in \mathbb{N}^+$. We utilize $k \in \{1, 2, ..., 10\}$.

2. For each k, we calculate the total within-cluster sum of squares (WSS). To calculate WSS, we calculate the Euclidean distance between each data point and the cluster centroid. The cluster centroids are updated with the mean of the total distances calculated as more points are taken in. This is demonstrated in the following pseudocode.

```
wss = empty array
k_values = values from 1 (inclusive) to 10 (inclusive)

for value in k_values
    k_means = KMeans(n_clusters=k)
    k_means.fit(input_data)
    add k_means.inertia_ to wss array
end
```

Listing 1: Algorithm to calculate total WSS in order to find the optimal number of k.

3. Plot the curve of the total WSS. The point where the "bend" in the curve occurs is denoted as the optimal number of clusters for the dataset.

Using the optimal number of clusters, we can then run k-means clustering on the input data. An example clustering on the ScikitLearn dataset, the make_moons dataset, is shown in Figure 1.

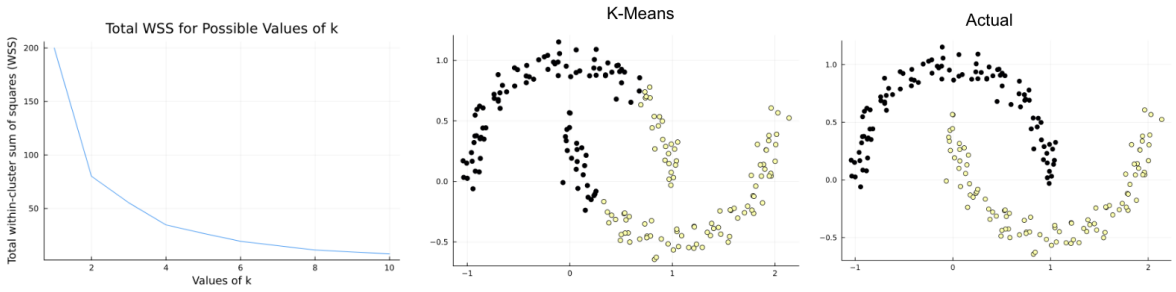

Figure 1: K-Means Clustering applied to the make_moons dataset from ScikitLearn. The leftmost line plot is generated using Listing 1, which depicts the bend being at $k = 2$; thus, we find that the optimal number of clusters is 2. The rightmost plot showcases the result of K-Means Clustering for this specific dataset.

From this, we notice that k-means clustering does not accurately identify the two moons, or clusters, in the dataset. Instead, observe that it clusters points around a specific centroid as one specific color, and as a result, is more useful for clusters with convex boundaries. For more complex geometries, such as the moons showcased in Figure 1, k-means continues to assume that clusters are spherical, creating a different set of clusters.

As we discuss spectral clustering, we will notice that this clustering algorithm allows us to project data points onto a higher dimension, from which k-means clustering can locate non-convex boundaries and effectively cluster the points.

4 The Spectral Clustering Algorithm

To begin implementing the spectral clustering algorithm given some input dataset, we first create a representation of the similarity, or relationship, between data points. In our implementation of spectral clustering, this is achieved using an undirected similarity graph given some specific measure of similarity. For this first implementation of the algorithm, we define similarity for a data point by the k-nearest neighbor algorithm using the Euclidean distance between two data points.

The k-nearest neighbor algorithm claims that for a vertex v_a of an undirected similarity graph G, if v_b is one such point that is closest according to Euclidean distance to v_a, then v_b is a nearest neighbor to v_a (Von Luxburgh, 2007). As there is no fixed value for k, we choose to use $k = 10$ for datasets of size greater than 100. However, if the dataset is extremely small, then we utilize the square of the size of the dataset as k. The k-nearest neighbors of the dataset are calculated utilizing the kneighbors_graph function from Scikit-Learn.

Using the k-nearest neighbors graph, we create the weighted adjacency matrix W utilizing the distances between neighbors v_i and v_j as the corresponding W_{ij} entry in the matrix W, or 0 if v_i and v_j are not neighbors using the following code.

```
data_size = number of rows in input dataset
if data_size > 100
    set n to 10
else
    set n to sqrt(data_size)
end

G = kneighbors_graph(X, n_neighbors=n, mode="distance")
W = convert G to array
```

Listing 2: Algorithm to calculate k-nearest neighbors and create weighted adjacency matrix W given input dataset.

Observe that we specifically use kneighbors_graph to create the similarity graph. This choice is because other library functions create directed graphs such that if v_j is a nearest neighbor to v_i, then W_{ij} is represented by the Euclidean distance between the two points. However, this does not directly imply that v_i is also a nearest neighbor to v_j. Therefore, in order to create an undirected graph and as a result, create a symmetric weighted adjacency matrix, we connect v_i and v_j with an edge if either

1. v_j is a nearest neighbor to v_i, or
2. v_i is a nearest neighbor to v_j

This is successfully accomplished using the kneighbors_graph function from ScikitLearn.

After this, we construct the diagonal degree matrix D by adding each row of W, denoted by r_i, and placing the sum at D_{ij} if $i = j$. Then, to compute the graph Laplacian matrix, we apply

$$L = D - W$$

Following this, we compute the eigenvalues and eigenvectors of the graph Laplacian matrix, L. This is because the graph Laplacian matrix is symmetric, so by the spectral theorem, it has real and non-negative eigenvalues. Thus, we extract the indices of the k smallest eigenvalues and find the corresponding k eigenvectors to those eigenvalues. We extract the k smallest eigenvectors as these eigenvectors as able to identify all possible clusters, as shown in the example Figure 2.

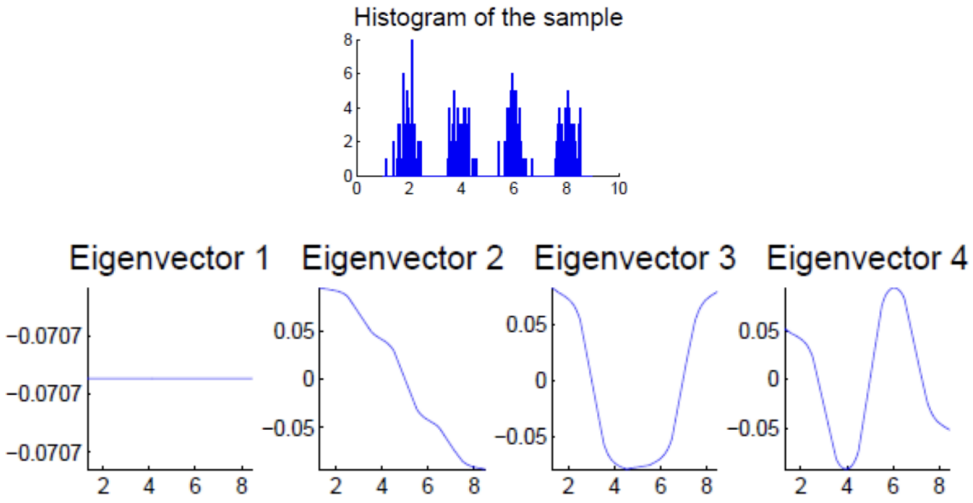

Figure 2: The first eigenvector constitutes the all ones vector; the second ultimately reaches zero, which separates the two categories of clusters unevenly. However, the four eigenvectors altogether correctly identify the four clusters (Singh, 2010).

An important aspect to note is that we choose the smallest non-negative to be the minimal eigenvalue we choose. This is completed according to the following algorithm.

```
using LinearAlgebra
np = pyimport("numpy")
k = number of clusters

eigvalues = np.real(eigvals(L))
eigvectors = np.real(eigvecs(L))[:,1:k]
```

Listing 3: Calculations of eigenvalues and eigenvectors and retrieval of the eigenvectors corresponding to the smallest k eigenvalues.

Note that the eigenvalues and eigenvectors of the graph Laplacian matrix will be complex numbers; to fix this, we simply take the real portion of each eigenvector and eigenvalue using the numpy library in Julia.

Lastly, we apply k-means clustering to project the data into a lower dimension by first finding the optimal k value according to the Elbow Method detailed in Section 3, and utilizing the KMeans function from ScikitLearn to produce assignments for the datasets. The following result is found.

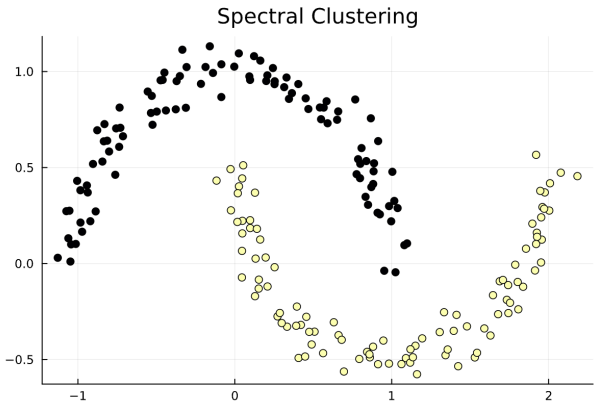

Figure 3: Result from spectral clustering algorithm. The algorithm correctly identifies both clusters.

We further implement a spectral clustering algorithm that utilizes connectivity, instead of Euclidean distance, to measure the similarity between data points. Here, if two vertices v_i and v_j are connected via an edge on the initial undirected similarity graph, then W_{ij} will be 1; otherwise, it will be 0. Instead of W_{ij} being the distance between nearest neighbors, the weighted adjacency matrix is filled with 1s and 0s to represent connections or disconnections between vertices.

In this algorithm, all other steps are the same onwards. We receive the same result as the first algorithm implemented, shown in Figure 3.

5 Evaluating Other Clustering Algorithms

We also evaluate other clustering algorithms available on different data shapes. Interestingly, another relatively popular clustering algorithm, affinity propagation, also relies on graph theory to create clusters, so we choose to investigate this in comparison to spectral clustering. We also investigate another group of clustering models, hierarchical clustering, where clusters are created with a three-step method ("2.3. Clustering"):

1. Find the two clusters whose centroids have the smallest Euclidean distance. This is the metric of similarity used.

2. Merge the clusters that have the highest metric of similarity.

3. Iterate until the specified number of clusters have been merged and created.

In hierarcial clustering, a tree data structure is used to represent all clusters that can be made as more clusters are merged together and created. The leaves of the tree are clusters with one data point each, while the root is a singular cluster that includes all data samples.

Therefore, here, clusters begin as distinct groups with one sample each and recursively merge together to form larger clusters.

We apply the above two clustering models and compare their performance with spectral clustering and k-means clustering.

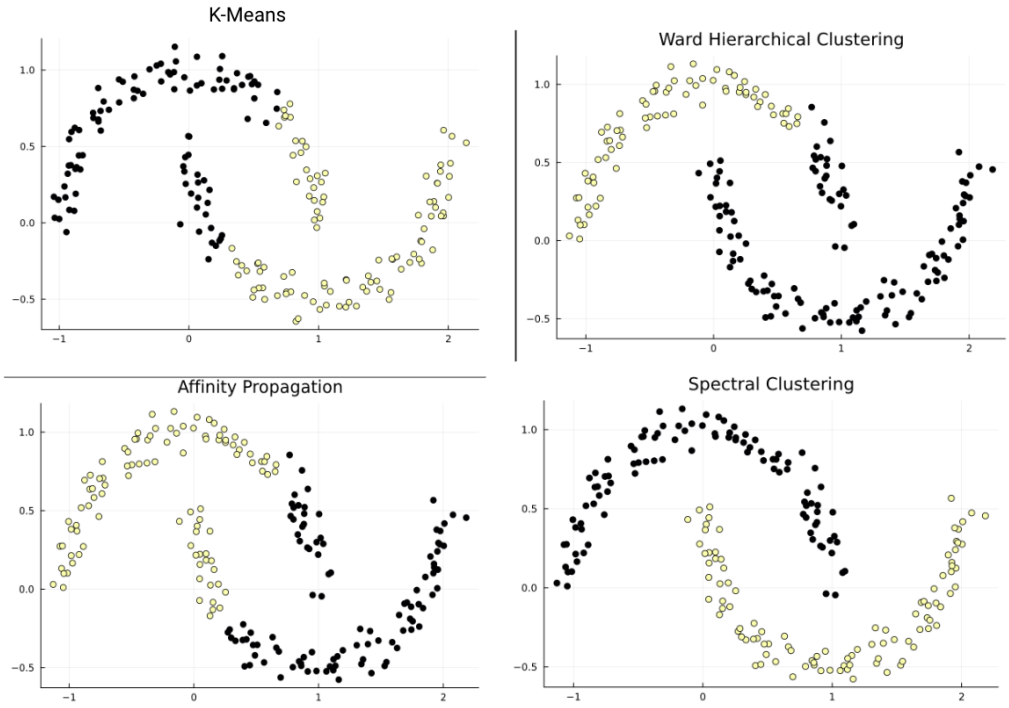

Figure 4: A comparison of four clustering algorithms: K-Means clustering, Ward Hierarchical Clustering, Affinity Propagation, and Spectral Clustering on the ScikitLearn data set, make_moons, is depicted. Spectral clustering is the only algorithm that produces the correct result. Parameters for affinity propagation algorithm: damping=0.9, preference=−100.

This is an especially interesting spread since as noted, spectral clustering is the mere algorithm that produces the correct result. However, affinity propagation also utilizes a graph structure to calculate a point's k-nearest neighbors; unlike spectral clustering (and the other clustering algorithms observed above), affinity propagation does not intake a set number of clusters. Rather, it sends soft messages between points and considers exemplar points, while also taking the damping and preference parameters, to understand the similarity between points ("2.3. Clustering"). As a result, affinity propagation is highly accurate for data with a large number of clusters; with the make_moons data set having two clusters, affinity propagation was not well-suited for this scenario.

Moreover, ward hierarchical clustering is also quite similar to k-means clustering in the sense that it utilizes the Euclidean distance between centroids to create clusters. The addition of the ward allows the Euclidean distance between centroids to be effectively minimized ("2.3. Clustering"). However, such an approach was seen to be ineffective with k-means clustering, and it was further noticed that ward hierarchical clustering was ineffective for the make_moons data set as well.

6 Conclusion

In the end, the spectral clustering algorithm implemented in this paper is just one of many methods to implement the algorithm. While the algorithm can be written in a variety of ways, it follows the concepts of creating a similarity graph, creating the corresponding degree matrix and graph Laplacian matrix, finding the first k smallest eigenvalues and their corresponding eigenvectors, and projecting the data to a lower dimension (k-dimension) space, and clustering the data. In our approach, we utilize k-means clustering to do this; however, there are other methods to cluster the data. Moreover, we do not choose to normalize the graph Laplacian matrix, but other studies may choose to normalize the matrix according to the dataset inputted. Finally, other methods to create a similarity graph may include the Gaussian Similarity function, which has been utilized by other studies.

In addition to creating the spectral clustering algorithm, we also expanded our work to other clustering algorithms. We found that k-means clustering does not work well on datasets that have clusters with non-spherical boundaries and that affinity propagation, while it also relies on a graph structure, is more well-suited for datasets that have a larger amount of clusters, as it does not intake an input k number of clusters. While the clustering algorithms focused on in this paper are only a few of those that exist, there are far more, suitable for a variety of different tasks and objectives.

References

[1] 2.3. Clustering. (n.d.). Retrieved December 9, 2022, from https://scikit-learn.org/stable/modules/clustering.html.

[2] Aoullay, A. (2018, June 3). Spectral clustering for beginners. *Medium*. Retrieved December 9, 2022, from https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8

[3] Fleshman, W. (2019, February 21). Spectral clustering. *Medium*. Retrieved December 9, 2022, from https://towardsdatascience.com/spectral-clustering-aba2640c0d5b.

[4] Singh, A. (2010). Spectral Clustering. Retrieved December 9, 2022, from https://www.cs.cmu.edu/ aarti/ Class/10701/slides/Lecture21_2.pdf.

[5] Umargono, E; Endro Suseno, J; Gunawan S.K., Vincensius (2019). K-Means Clustering Optimization using the Elbow Method and Early Centroid Determination Based-on Mean and Median. *ScitePress*. Retrieved December 9, 2022, from https://www.scitepress.org/Papers/2019/99084/99084.pdf.

[6] Von Luxburgh, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4), 395-416. Retrieved December 9, 2022.

[7] Wikipedia contributors (2022, December 9). Graph. Retrieved December 9, 2022, from https://en.wikipedia.org/wiki/Graph.

7 Code

```
using PyCall
using ScikitLearn
using Plots
using LinearAlgebra

# imports numpy as well as scikit-learn dataset and models
np = pyimport("numpy")
@sk_import datasets: make_moons
@sk_import neighbors: kneighbors_graph
@sk_import cluster: KMeans
@sk_import cluster: AffinityPropagation
@sk_import cluster: AgglomerativeClustering

# for the code used in the paper:
# X, y = make_moons(n_samples=200, noise=0.07)

# Finds optimal k-value for k-means clustering
function find_optimal_k(X)
    wss = []

    # choose possible k values from 1 (inclusive) to 10 (inclusive)
    k_values = range(1, 10)

    for k in k_values
        k_means = KMeans(n_clusters=k)
        k_means.fit(X)
        append!(wss, k_means.inertia_)
    end
    return k_values, wss
end

# Plots the Elbow Method Curve to find optimal k - "bend" of graph is the k value
function plot_elbow(k_values, wss)
    plot(k_values, wss, legend=false)
    title!("Total WSS for Possible Values of k")
    xlabel!("Values of k")
    ylabel!("Total within-cluster sum of squares (WSS)")
end

# Applies k-means clustering and plots the result
function apply_and_plot_clustering(X, k)
    # assign number of clusters based on where "bend" in above graph is
    clusters = k

    # run k-means clustering
    kmeans = KMeans(n_clusters=clusters)
    kmeans.fit(X)
    y_kmeans = kmeans.predict(X)
```

```
# plot result of k-means clustering
    plot(X[:, 1], X[:, 2], seriestype=:scatter, marker_z=y_kmeans, legend=false)
    title!("K-Means")
end

# implements k-means clustering
function k_means_clustering(X)
    k = 2
    k_values, wss = find_optimal_k(X)
    plot_elbow(k_values, wss)
    apply_and_plot_clustering(X, k)
end

# calculates the optimal number of near neighbors
function find_num_neighbors(X)
    """
    Returns number of nearest neighbors that point should connect with
    depending on size of dataset. If dataset is especially small, then
    we use the square root of the number of rows in the datset.
    """
    if size(X)[1] > 100
        n = 10
    else
        n = sqrt(size(X)[1])
    end
    return n
end

# computes the kneighbors_graph and creates weighted adjacency matrix
function get_kneighbors_graph(X, mode)
    # finds the number of neighbors to look for
    n = find_num_neighbors(X)
    # creates weighted adjacency matrix
    W = kneighbors_graph(X, n_neighbors=n, mode=mode).toarray()
    return W
end

# computes the diagonal degree matrix, D
function compute_degree_matrix(W)
    D = np.diag(np.sum(A, axis=1))
    return D
end

# computes the graph Laplacian matrix, L
function compute_laplacian(W, D)
    L = D - W
    return L
end
```

```
# retrieves first k eigenvectors from L
function find_first_k_eigvectors(k, L)
    # the eigenvectors and eigenvalues of L will be complex.
    # thus, we extract the real portion of them.
    eigvalues = np.real(eigvals(L))
    eigvectors = np.real(eigvecs(L))

    # find first k eigenvectors
    first_k_eigvectors = np.real(eigvecs(L))[:,1:k]
    return first_k_eigvectors
end

# apply K-means clustering to cluster data points
function apply_k_means_clustering(first_k_eigvectors, k)
    k_means = KMeans(n_clusters=k)
    k_means.fit(first_k_eigvectors)
    return k_means
end

# plots spectral clustering result
function plot_spectral_clustering(X, k_means)
    plot(X[:, 1], X[:, 2], seriestype=:scatter, marker_z = k_means.labels_, legend=false)
    title!("Spectral Clustering [distance method]")
end

# main function to implement spectral clustering
function spectral_clustering(X)
    n = find_num_neighbors(X)
    W = get_kneighbors_graph(X, mode)
    D = compute_degree_matrix(W)
    L = compute_laplacian(W, D)
    first_k_eigvectors = find_first_k_eigvectors(k, L)
    k_means = apply_k_means_clustering(first_k_eigvectors, k)
    plot_spectral_clustering(X, k_means)
end

# parameter values inspired from
# https://scikit-learn.org/stable/modules/clustering.html
# creating Ward Hierarchical Clustering and Affinity Propagation models
function other_clustering(X)
    n = find_num_neighbors(X)

    # computes kneighbors_graph
    connectivity = kneighbors_graph(
                X, n_neighbors=n, include_self=false)

    ward = AgglomerativeClustering(
                n_clusters=2, linkage="ward", connectivity=connectivity).fit(X)
```

```
    affinity_propagation = AffinityPropagation(
                damping=0.9, preference=-100, random_state=0).fit(X)

        # creates plot for Ward Hierarchical Clustering
    plot(X[:, 1], X[:, 2], seriestype=:scatter, marker_z = ward.labels_, legend=false)
    title!("Ward Hierarchical Clustering")

        # creates plot for Affinity Propagation
    plot(X[:, 1], X[:, 2], seriestype=:scatter, marker_z = affinity_propagation.labels_,

         legend=false)
    title!("Affinity Propagation")
end
```