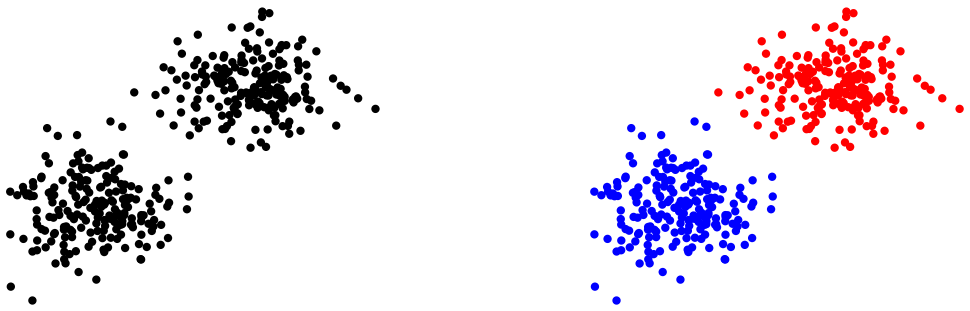
EE 510: Mathematical Foundations of Machine Learning **Winter 2021**

Demo: Spectral Clustering

Instructor Name: John Lipor

1 Introduction

In this demo, you will implement and experiment with the Spectral Clustering algorithm, which is a method for clustering data points based on their pairwise similarity. Clustering is one of the most common problems solved in the domain of *unsupervised learning*. While there is no single agreed-upon definition of clustering, for our purposes, we can think of it as unsupervised classification. You are given a set of data points that you believe lie in K groups, and your goal is to automatically determine these groups using only the data themselves. A before-and-after of an ideal clustering is shown in Fig. 1.



(a) (b)

Figure 1: (a) Set of unlabeled data points that we wish to cluster without any explicit knowledge of which points belong together. (b) Data point with correct labels shown.

The simplest and most widely-used clustering algorithm is the K-means algorithm, which finds a set of K *centroids* and clusters points according to their nearest centroid. However, K-means has many known flaws, one being that it is incapable of clustering data in some "common sense" configuration (more later). In this demo, we'll look at an alternative to K-means known as *spectral clustering*, which is based heavily in the eigenvalue decomposition.

2 Spectral Clustering

The basic idea behind spectral clustering is that we can take some notion of similarity between points (e.g., pairwise distance) and use this to embed the points in a different space, where K-means does perform well. Spectral clustering proceeds as follows.

1. Given a set of unlabeled points $\{x_1, x_2, \ldots, x_N\}$, construct a graph where the nodes are the data points and the edges denote the similarity between points
2. Form the $N \times N$ *Laplacian* matrix L from the graph
3. Infer a partition of the graph (clusters) from the eigenvalue decomposition of L

We break these steps down below.

2.1 Similarity Graphs

Similarity graphs are defined by a graph structure and edge weights. We won't get too deep into how a graph is formally defined, but what is important for spectral clustering is that we maintain a *weighted adjacency matrix* $W \in \mathbb{R}^{N\times N}$. This matrix is defined such that $w_{ij} \geq 0$ for all i, j, and $w_{ij} > 0$ if and only if the points x_i and x_j are *connected* to one another (a.k.a., *adjacent*). Note that we define W to be symmetric. As an example, you could create a graph where every "point" is a user on facebook, and two points are connected if people are friends, resulting in a value of $w_{ij} = 1$. There are a large number of ways to define whether two points are connected. A few examples are

- *k-nearest neighbor graph*: Every x_i is connected to its k nearest neighbors (based on some notion of distance, such as the 2-norm)
- *ε-ball graph*: Every x_i is connected to every x_j having $\|x_i - x_j\| \leq \varepsilon$
- *complete graph*: All points are connected

Aside from determing the answer to the binary question of whether two points are connected, there are also a variety of ways to determine the *strength* of the connection between two points. We will consider

- *Constant*: Set
$$w_{ij} = \begin{cases} 1, & x_i, x_j \text{ connected} \\ 0, & \text{otherwise} \end{cases}$$
- *Gaussian*: Set
$$w_{ij} = \begin{cases} e^{-\|x_i - x_j\|^2/2\sigma^2}, & x_i, x_j \text{ connected} \\ 0, & \text{otherwise} \end{cases},$$
where $\sigma > 0$ is a tuning parameter that you can play with.

Ideally, we only want to connect points in the same cluster, but in practice we don't know the clusters. Hence, choosing parameters such as k, ε, σ can be a difficult problem that has a dramatic impact on the performance of clustering.

2.2 Graph Laplacians

As stated above, we use the graph Laplacian L to perform clustering. This is a matrix that is formed from W that is known to have useful properties for clustering. Define the (weighted) *degree* of a node x_i to be

$$d_i = \sum_{j=1}^{N} w_{ij} \tag{1}$$

and the *degree matrix* to be the diagonal matrix whose diagonal entries are the degrees, i.e.,

$$D = \begin{bmatrix} d_1 & 0 & 0 & \dots & 0 \\ 0 & d_2 & 0 & \dots & 0 \\ 0 & 0 & d_3 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & d_N \end{bmatrix}. \tag{2}$$

We then define the (unnormalized) *graph Laplacian* as

$$L = D - W. \tag{3}$$

Note that by definition L is independent of the self-similarity weights w_{ii} since

$$L_{ii} = d_i - w_{ii} = \sum_{j \neq i} w_{ij}.$$

There are also some empirical benefits to using the *normalized graph Laplacian*

$$\tilde{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}. \tag{4}$$

2.3 Theory

Our motivation for spectral clustering comes from some useful properties of L, some of which you may be asked to prove at a later date.

1. For every $f \in \mathbb{R}^N$
$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{N} w_{ij}(f_i - f_j)^2.$$
2. L is symmetric positive semidefinite.
3. The smallest eigenvalue of L is $\lambda_{min}(L) = 0$ with corresponding eigenvector 1_N.

To prove the third statement above, we simply need to show that the proposed are a valid eigenvalue-eigenvector pair, i.e., to show that $L1_N = 0$. To see this, note that

$$\begin{aligned} L1_N &= D1_N - W1_N \\ &= \begin{bmatrix} d_1 - \sum_{j=1}^{N} w_{1j} \\ d_2 - \sum_{j=1}^{N} w_{2j} \\ \vdots \\ d_N - \sum_{j=1}^{N} w_{Nj} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \end{aligned}$$

So how do these help us perform clustering? Let $A \subset \{x_1, \ldots, x_N\}$, e.g., let A be some points in a cluster. Define the indicator vector of size $N \times 1$ as

$$\mathbb{1}\{A\} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{bmatrix} \tag{5}$$

where

$$f_i = \begin{cases} 1, & x_i \in A \\ 0, & x_i \notin A \end{cases}.$$

Now note that the nullspace of L is the span of all eigenvectors associated with the eigenvalue 0 (there may be more than one!). We are now in a place to start understanding why the graph Laplacian is important.

Proposition 1. *If the graph has connected components (clusters) $A_1, A_2, \ldots, A_K$, then the nullspace of L has dimension K and is spanned by the vectors $\mathbb{1}\{A_1\}, \mathbb{1}\{A_2\}, \ldots, \mathbb{1}\{A_K\}$.*

Proof. Recall that the nullspace of L is $\mathcal{N}(L) = \{f \in \mathbb{R}^N : Lf = 0\}$. It suffices to show that

- $\mathbb{1}\{A_k\} \in \mathcal{N}(L)$ for each $k = 1, \ldots, K$
- $f \in \mathcal{N}(L)$ implies that $f = \sum_{k=1}^{K} \alpha_k \mathbb{1}\{A_k\}$ for some $\alpha_1, \ldots, \alpha_K \in \mathbb{R}$.

First consider the case of $K = 1$. In this case, we know that $1_N \in \mathcal{N}(L)$ (since it corresponds to the zero eigenvalue). To show the second property, take some $f \in \mathcal{N}(L)$. Then $Lf = 0$, so

$$0 = f^T L f = \frac{1}{2} \sum_{i,j=1}^{N} w_{ij}(f_i - f_j)^2.$$

If x_i and x_j are adjacent, then $w_{ij} > 0$, which implies $f_i = f_j$. Since $K = 1$, all points are connected, which implies that $f_i = f_j$ for all i, j. Thus f is a multiple of 1_N.

Now consider $K > 1$ and suppose that the data are arranged such that

$$L = \begin{bmatrix} L_1 & 0 & \dots & 0 \\ 0 & L_2 & \dots & 0 \\ & & & \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & L_K \end{bmatrix}.$$

Notice that L_k is the graph Laplacian on A_k so we can use the $K = 1$ case above to see that

- $L\mathbb{1}\{A_k\} = 0$ for each k
- If $Lf = 0$, then f is piecewise constant on A_k, i.e.,

$$f = \sum_{k=1}^{K} \alpha_k \mathbb{1}\{A_k\}.$$

□

To connect the above proposition with a clustering algorithm, consider the following corollary.

Corollary 1. *If $\{u_1, u_2, \dots, u_K\} \subset \mathbb{R}^N$ is a basis for $\mathcal{N}(L)$ and*

$$y_i = \begin{bmatrix} u_1(i) \\ u_2(i) \\ \vdots \\ u_K(i) \end{bmatrix} \in \mathbb{R}^K$$

then $y_i = y_j$ if and only if x_i and x_j are in the same connected component (cluster).

2.4 Algorithm

The above theory tells us that if our clusters all consist of points that are connected to each other (possibly indirectly) and not connected to points in other clusters, then the *transformed* vectors $y_1, \dots, y_N \in \mathbb{R}^K$ will match exactly for points in the same cluster. In practice, we of course break these assumptions, but the general principle works well as long as there are not too many false or missing connections. Our approach is therefore to take the eigenvectors corresponding to the K smallest eigenvalues of L and work with those as our "basis" for the nullspace. The spectral clustering algorithm that you need to code is given below in Alg. 1.

Algorithm 1 Spectral Clustering

1: **Input:** Data points $x_1, \ldots, x_N$; number of clusters K
2: **Output:** Cluster assignment for each point
3: Construct similarity graph as described in Section 2.1
4: Construct the (possibly normalized) graph Laplacian as described in Section 2.2
5: Determine K smallest eigenvalues $\lambda_1, \ldots, \lambda_K$ and corresponding eigenvectors $u_1, \ldots, u_K$
6: Set $y_i = [u_1(i), \ldots, u_K(i)]^T$ for $i = 1, \ldots, N$
7: Cluster $\{y_i\}_{i=1}^N$ using K-means

3 Tasks

- Open each of the provided files and have a look at what they do. In particular, `syntheticTest` creates synthetic data lying on two concentric spheres that you will use to test your implementation of Spectral Clustering. I have already created the similarity matrix for you, which you can view using `imshow` in Python or `imagesc` in Matlab.
- Implement the Spectral Clustering algorithm with an option to use either the normalized or unnormalized Laplacian. See the file `mySpectralClustering`.
- Test your algorithm on synthetic data. Vary the parameter that controls the noise in the data, as well as the parameters for forming the weighted adjacency matrix. See `syntheticTest`.
- Test your algorithm on the Iris dataset. Vary the parameter that controls the noise in the data, as well as the parameters for forming the weighted adjacency matrix. See `irisTest`.
- If time permits, implement your own formation of a k-nearest neighbors (KNN) graph with the option to set the number of neighbors, the weight type (constant or Gaussian), and the parameter σ for Gaussian weights. See `myKNN.m` or `helperFunctions.py` for the included implementation.