Graph Spectral Clustering

Catherine Huang, Tynan Sigg, Clark Wang, Jonathan Wang

February 22, 2021

1 Motivation

Cluster analysis is an unsupervised learning problem, where the goal is to partition a dataset into subsets, or **clusters**, such that data within a cluster are similar (high intra-cluster similarity), and data from different clusters are dissimilar (low inter-cluster similarity). Admittedly, this definition is fairly open-ended and allows for a large number of possible formalizations. That said, the general concept of dividing datasets into smaller groups has been successfully applied to a wide variety of tasks. In this note, we will introduce one such technique for clustering, **graph spectral clustering** (GSC), that makes use of the eigenvectors of a **similarity matrix** calculated from the data, allowing one to cluster in lower dimensions through dimensionality reduction. GSC has many applications in machine learning, ranging from computer vision and speech processing, to exploratory data analysis. Though this note does not delve deeply into all the math behind GSC, the goal is that after having read this note, one should feel comfortable enough to understand how to use GSC, as well as be able to identify appropriate scenarios to do so. If interested, the references section offers content that will further understanding and appreciation of this wonderful intersection of graph theory and machine learning.

2 K-means

In CS 61A, you may have been introduced to a simple clustering algorithm called **k-means** to group restaurants for recommendation purposes. The standard k-means algorithm is as follows:

Algorithm 1: k-means

Result: Return clusters $A_1, \cdots, A_k$

Given data $X \in \mathbb{R}^{n \times d}$, number of desired clusters k

1. Initialize k points $c_i \in \mathbb{R}^d$ with $i = 1, \ldots, k$ to be the centroids of each cluster. Initializing with k-means++ (**Section 2.1**) picks these centroids to be as distant as possible from each other, to help ensure good clusters.
2. Assign each data point to the closest centroid. The points that correspond to the same centroid form the k clusters.
3. Re-assign each centroid to be the center of its current cluster (using the updated assignments of data).
4. Continue to iterate steps 2 and 3 until updating the centroids in step 3 no longer changes the cluster assignments in step 2.

K-means is one of the most popularly used unsupervised learning algorithms, partly due to its simplicity. However, there are some scenarios where it may fail. For example, consider the "two moons" dataset pictured below:

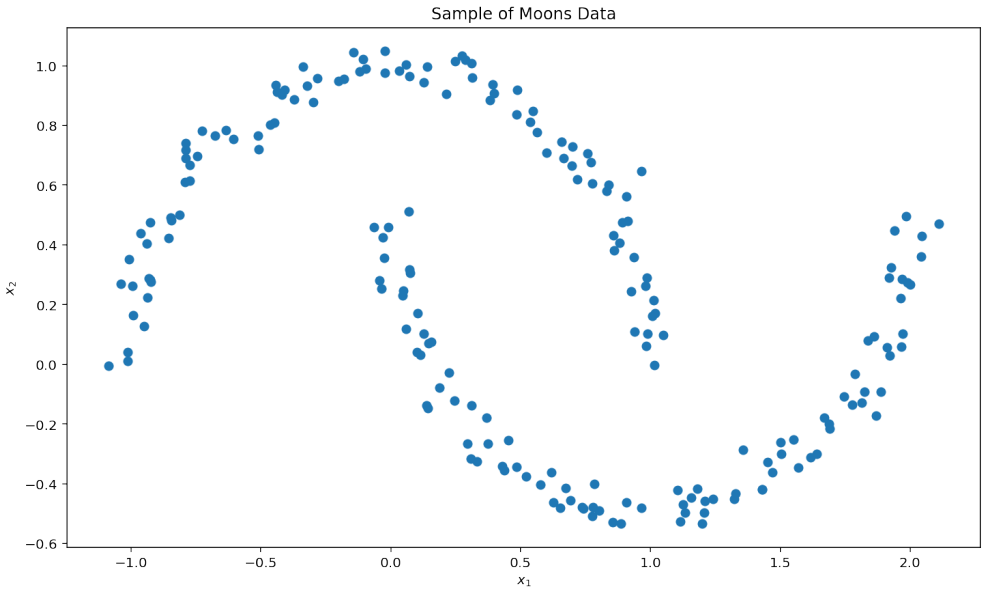
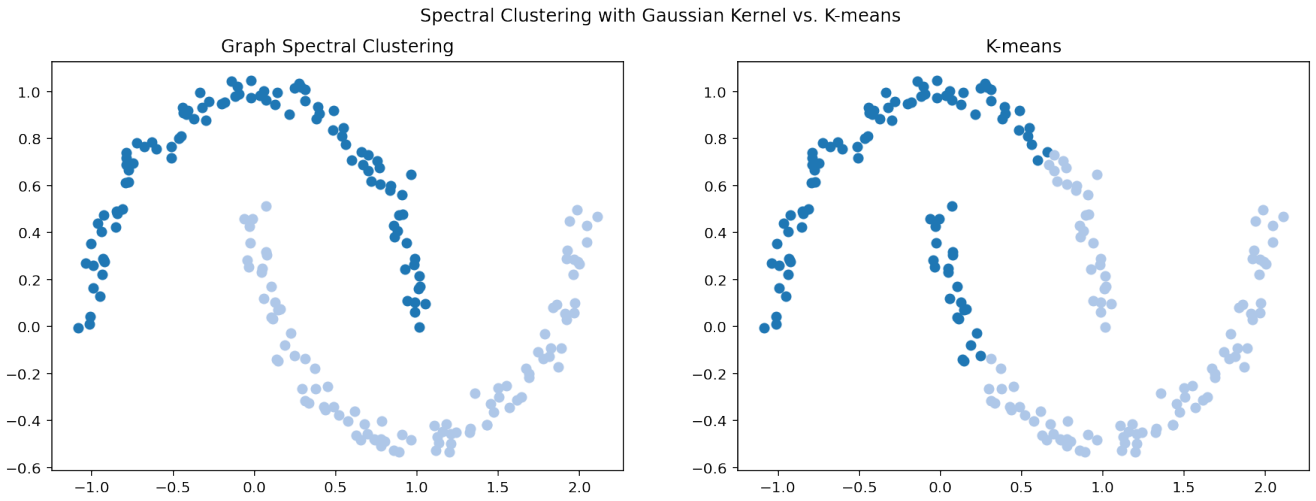


Figure 1: Two moons dataset

k-means is a **centroid-based** clustering algorithm, because it selects a typical example from each cluster and classifies other points based on their proximity to each centroid. As you saw in CS61A, this creates clusters in the form of Voronoi cells, which are convex and bounded by straight lines. However, this won't work in the case of the dataset above, where the intuitive clustering is non-convex. Thus in this case, one might have to turn to more involved clustering techniques, such as GSC. GSC actually builds on k-means, and drawing from graph theory, allows for more complex cluster boundaries, as shown in the comparison below for this dataset.



2.1 K-means++

Before moving on, we will discuss an addition to the vanilla k-means algorithm, which greatly improves the performance of the basic algorithm in the context of GSC. Specifically, k-means++ improves the process of initializing the cluster centroids at the beginning, before iterating through the regular k-means algorithm. The k-means++ initialization algorithm is as follows:

Algorithm 2: k-means++

Result: Return clusters $A_1, \cdots, A_k$
Given data $X \in \mathbb{R}^{n \times d}$, number of desired clusters k

1. Choose one centroid c_1, chosen uniformly at random from X.
2. Choose a new centroid c_i, chosen with new probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$, where $D(x)$ is the shortest distance from data point x to its closest centroid.
3. Repeat the previous step, until k centroids have been chosen.
4. Continue with the original k-means algorithm.

This initialization algorithm fixes a common problem with k-means, which occurs when multiple centroids end up very close to one another and split data along lines that do not correspond to underlying clusters. In general, a centroid must end up near a cluster in order for that cluster to be picked out by the algorithm. k-means++ addresses this by spreading out the initial placement of the centroids, making it more likely to select data points that are farther from the existing centroids.

3 Graphs

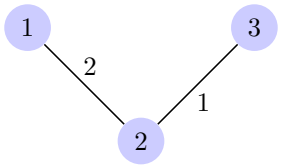


Figure 2: Example of an undirected graph

This section will briefly introduce the relevant bits of graph theory one needs to know for GSC. All clustering algorithms are based on a notion of similarity between datapoints. In GSC, similarity data is represented by a **weighted graph**. A **graph** is defined by a set of vertices and a set of edges where each edge connects exactly two vertices. For example, in Figure 2, the set of vertices is $V = \{1, 2, 3\}$, and the set of edges is $E = \{(1, 2), (2, 3)\}$. There is also a **weight** associated with each edge, which can be thought of as a function $w : E \to \mathbb{R}$. Here, the weight of edge $(1, 2)$ is 2, so $w((1, 2)) = 2$. We will write $w_{ij} := w((i, j))$ for simplicity. This graph is also **undirected**, where the weights of edges in both directions are equal. In GSC, we represent data points as vertices, and the weight of the edge between two vertices represents the similarity between their corresponding data (higher weights represent more similar values). We will consider undirected similarity graphs, i.e. those where the similarity metric is symmetric.

4 Vertex Similarity

The closer data points x_i and x_j are, the higher the weight w_{ij} for the edge (i, j) should be in the similarity graph. Typically, we use the Gaussian similarity function, also known as the RBF kernel (introduced in Week 4), $s(x_i, x_j) = \exp(-||x_i - x_j||^2/2\sigma^2)$, where σ is a hyperparameter determining the scaling of the metric. When σ is high, this function drops off quickly meaning everything is "far apart," i.e. only data that are very close will have high similarity scores. When σ is lower, the opposite is true. This is shown in Figure 3.

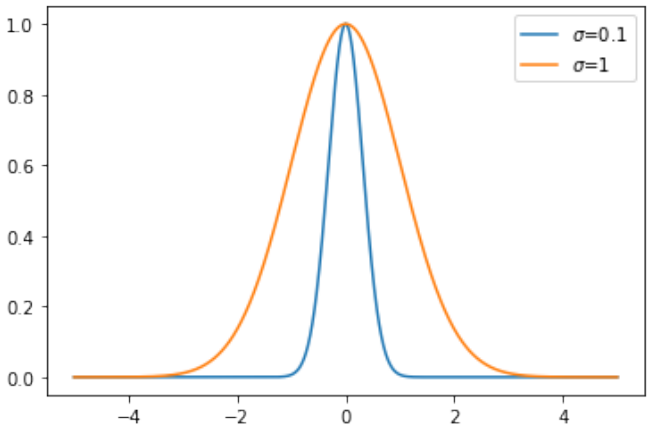


Figure 3: A comparison of kernels with different σ values

5 Matrix Representations of Graphs

Various matrices can be used to represent weighted graphs and their features. The following ones are useful for understanding the GSC algorithm.

5.1 Weighted Adjacency Matrix

Definition 5.1 (Weighted Adjacency Matrix)**.** *The adjacency matrix of a weighted graph $G = (V, E)$ is the matrix $W = [w_{ij}]_{i,j \in V}$, where each w_{ij} is the weight of edge (i, j).*

Since we are working with undirected graphs, our weighted adjacency matrix is symmetric (i.e. $w_{ij} = w_{ji}$). Note that an adjacency matrix completely and uniquely characterizes a weighted graph. However, it turns out that other representations will be more useful for the desired application of clustering.

5.2 Degree Matrix

Definition 5.2 (Degree of a Vertex)**.** *The degree of a vertex $v_i \in V$ is $d_i = \sum_{j=1}^{n} w_{ij}$, the sum of the weights of all edges incident to vertex v_i. This allows us to define the diagonal degree matrix D, where the degrees $d_1, ..., d_n$ are the diagonal entries.*

5.3 Laplacian Matrix

The main similarity graph representation used in GSC is the Laplacian matrix.

Definition 5.3 (Standard Graph Laplacian)**.** *For a graph with degree matrix D and weighted adjacency matrix W, we define the **standard Laplacian** matrix $L_s := D - W$.*

Definition 5.4 (Symmetric Normalized Laplacian)**.** *For a graph with degree matrix D and weighted adjacency matrix W, we define the **symmetric normalized Laplacian** matrix $L_n := I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = D^{-\frac{1}{2}} L_s D^{-\frac{1}{2}}$.*

The version of GSC we present here will use the symmetric normalized Laplacian as the primary representation for the data. The high level motivation for this choice is that its eigenvalue/eigenvector structure relates to a problem called **ratio cut**, which also involves dividing data into partitions based on a similarity metric. Although the exact correspondence is beyond the scope of this course and you are not responsible for knowing it, more details can be found in Appendix A.1.

6 Graph Spectral Clustering

With our matrix representations of the similarity graph, we can finally put together the entire GSC algorithm. We will begin this section by first showing the pseudocode of the algorithm, then we will walk through GSC being run on a toy dataset, so you can witness the clustering in action from beginning to end. For additional intuition behind the algorithm, refer to Appendix A.1.

Algorithm 3: Normalized Spectral Clustering

Result: Return clusters $A_1, \cdots, A_k$

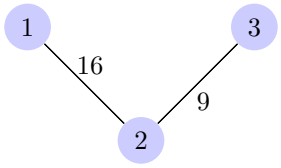
Given data $X \in \mathbb{R}^{n \times d}$, number of desired clusters k

1. Construct a similarity graph $S \in \mathbb{R}^{n \times n}$ according to some similarity metric (e.g. the Gaussian or exponential kernels)
2. Compute $L_n = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$, the symmetric normalized graph Laplacian of S.
3. Compute $v_{1 \cdots k}$, the eigenvectors of L_n corresponding to its k smallest eigenvalues. [a]
4. Construct the matrix $H = [v_1; ...; v_k] \in \mathbb{R}^{n \times k}$ having the first k eigenvectors of L_n as the columns. Then normalize the rows of the matrix to have norm 1.
5. Run k-means on the rows of H to cluster the data. Each data point is assigned to the cluster of its corresponding row in H.

[a] This is always possible by the spectral theorem, which states that symmetric matrices are always orthogonally diagonalizable.

6.1 Working through an example

The following is a simple, concrete example to help you better understand the normalized version of the graph spectral clustering algorithm. Let's say the following graph is the similarity graph of our dataset of size 3.



The following matrices are derived from this similarity graph:

$$D = \begin{bmatrix} 16 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 9 \end{bmatrix} \quad W = \begin{bmatrix} 0 & 16 & 0 \\ 16 & 0 & 9 \\ 0 & 9 & 0 \end{bmatrix} \quad L \coloneqq D - W = \begin{bmatrix} 16 & -16 & 0 \\ -16 & 25 & -9 \\ 0 & -9 & 9 \end{bmatrix}$$

D is the degree matrix of the graph. Notice that each diagonal entry are equal to the sum of the edge weights of edges adjacent to the vertex corresponding to that entry. For example, $D_{2,2} = 16 + 9 = 25$, for the two adjacent edges $(1, 2)$ and $(2, 3)$.

W is the weighted adjacency matrix of the graph. Each non-diagonal entry (i, j) is equal to the weight of the edge from i to j if such an edge exists, and is 0 otherwise. For example, $W_{1,2} = 16$, since the edge from vertex 1 to vertex 2 has weight 16. Notice that this matrix is symmetric (why?), and the diagonal entries must be 0.

L is the unnormalized Laplacian matrix obtained by subtracting the weighted adjacency matrix from the degree matrix.

The normalized Laplacian L_n is defined as $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$. Below, we transform the unnormalized Laplacian matrix into its normalized form through simple matrix multiplications. Remember that the inverse of a diagonal matrix is simply the diagonal entries replaced with their reciprocals.

$$
\begin{aligned}
D^{-\frac{1}{2}} &= \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{5} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} \\
L_n &= D^{-\frac{1}{2}} L_s D^{-\frac{1}{2}} \\
&= D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}} \\
&= D^{-\frac{1}{2}} D D^{-\frac{1}{2}} - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \\
&= I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \\
&= I - \begin{bmatrix} 0 & \frac{4}{5} & 0 \\ \frac{4}{5} & 0 & \frac{3}{5} \\ 0 & \frac{3}{5} & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & -\frac{4}{5} & 0 \\ -\frac{4}{5} & 1 & -\frac{3}{5} \\ 0 & -\frac{3}{5} & 1 \end{bmatrix}
\end{aligned}
$$

After we've found the normalized Laplacian, we can find the eigendecomposition of L_n. Q is the eigenvector matrix of L_n, where each column i in Q is eigenvector v_i of L_n. D is a diagonal matrix where each diagonal entry $D_{i,i}$ is the eigenvalue λ_i of L_n corresponding to eigenvector v_i. The intermediate calculations in this eigendecomposition are left to the reader as an exercise.

$$
L_n = QDQ^{-1} = \begin{bmatrix} \frac{4}{3} & -\frac{3}{4} & \frac{4}{3} \\ \frac{5}{3} & 0 & -\frac{5}{3} \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} \frac{4}{3} & -\frac{3}{4} & \frac{4}{3} \\ \frac{5}{3} & 0 & -\frac{5}{3} \\ 1 & 1 & 1 \end{bmatrix}^{-1}
$$

If we want to cluster this into two clusters, we take the two eigenvectors corresponding to the lowest two eigenvalues to form our H_{temp} (where the rows of the matrix are not yet normalized). Thus, we let $k = 2$.

$$
\begin{aligned}
H_{temp} &= \begin{bmatrix} | & | \\ v_1 & v_2 \\ | & | \end{bmatrix} = \begin{bmatrix} \frac{4}{3} & -\frac{3}{4} \\ \frac{5}{3} & 0 \\ 1 & 1 \end{bmatrix} \\
H &= \begin{bmatrix} \frac{16}{\sqrt{337}} & -\frac{9}{\sqrt{337}} \\ 1 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}
\end{aligned}
$$

Once we have our H matrix, we finally perform k-means on the rows of H, where each row corresponds to a data point in the original data set. Remember that we also need to normalize the rows of H_{temp} to obtain matrix H.

In this example, $\begin{bmatrix} \frac{16}{\sqrt{337}} & -\frac{9}{\sqrt{337}} \end{bmatrix}$ represents point 1, $\begin{bmatrix} 1 & 0 \end{bmatrix}$ represents point 2, and $\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$ represents point 3. After k-means is complete, our two clusters are $\{(\frac{16}{\sqrt{337}}, -\frac{9}{\sqrt{337}}), (1,0)\}$ and$\{(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})\}$. This corresponds to cluster assignments $\{1,2\}$ and $\{3\}$. Looking back at our original diagram, if we had to divide these groups into two clusters, the natural division would indeed be $\{1,2\}$ and $\{3\}$, because the two most similar points are 1 and 2 with a similarity value of 16.

7 References

1. Ulrike von Luxburg. *A Tutorial on Spectral Clustering.* Max Planck Institute for Biological Cybernetics, 2007.
2. Jianbo Shi, Jitendra Malik. *Normalized Cuts and Image Segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.
3. Sheldon Axler, *Linear Algebra Done Right*, Springer Verlag, 1997
4. Arthur, D.; Vassilvitski, S., *k-means++: The Advantages of Careful Seeding*, ACM_SIAM, 2007

A Appendix

A.1 Motivation for the algorithm

Here we give some intuition for why the rows of the truncated normalized Laplacian are a reasonable representation for the data during the k-means step of the algorithm. This material is not in scope for the course but interested students can explore it and the given references to better understand the algorithm.

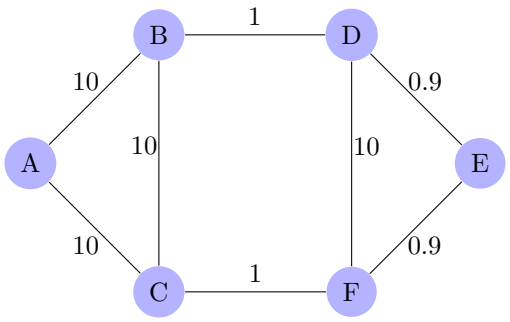
Consider a simple form of clustering where we only want to divide data into two groups A and B. Given a similarity graphs $S = (V, E, w)$, a reasonable metric for our goal of minimizing the similarity between these two groups is to minimize the sum of edge weights which "cross the cut," i.e. those of edges which have an end in both A and B. This can be expressed by the minimization problem

$$\min_{A,B} \sum_{i \in A, j \in B} w_{ij} \quad \text{s.t.} \quad V \setminus A = B$$

This problem is known as **min cut**. However, this metric might give a very unbalanced cut with one side much larger than the other. One way to fix this is to add a correction term to the cost function that penalizes small clusters. Choosing it to be the following gives a problem known as **ratio cut**.

$$\min_{A,B} \left(\sum_{i \in A, j \in B} w_{ij} \right) \left(\frac{1}{|A|} + \frac{1}{|B|} \right) \quad \text{s.t.} \quad V \setminus A = B$$

As an example of how these problems differ, consider the following similarity graph.



Min cut gives the partition {{A,B,C,D,F}, {E}}, whereas ratio cut gives the much more balanced partition {{A,B,C}, {D,E,F}}. This is a good start, but it only does 2-way clustering and ratio cut is very hard to compute (NP-hard, in fact). However, we can make further progress by noting that this problem has a continuous analogue which is computationally tractable. Ratio cut partitions as we have seen them so far can be represented as vectors f where f_i corresponds to the assignment of the i^{th} data point. Specifically, if we let

$$f_i = \begin{cases} \sqrt{\frac{|A|}{|B|}} & \text{if } i \in A \\ -\sqrt{\frac{|B|}{|A|}} & \text{if } i \in B \end{cases}$$

then it can be shown that $f^\intercal L f$ gives a value proportional to the ratio cut loss, so optimizing f over vectors of the form above is equivalent to ratio cut[1]! However, if we allow f to take other forms (and impose the constraint that $\|f\| = 1$ so we can't just take $f = \vec{0}$) this becomes an eigenvalue problem. This is because f being the eigenvector of L with the smallest eigenvalue minimizes the value of $f^\intercal L f$, which can be seen by eigendecomposing L.

Eigenvectors corresponding to small eigenvalues then represent fuzzy assignments of vertices to different clusters, with positive entries being in one cluster and negative entries in the other. As you will see in the associated project, graphs with k mostly disconnected clusters will have about k small eigenvalues corresponding to good

[1] This is most easily seen for the standard Laplacian L_s, but an analogous statement holds for L_n. See reference 1 for details.

continuous ratio cuts. We use the k associated eigenvectors as the columns of our H, on whose rows we do k-means clustering. Points corresponding to similar rows in this matrix are then often on the same side of the k best continuous ratio cuts. This cut-based similarity is the metric which k-means "sees" in the final clustering of the data.