Stat260/CS294: Spectral Graph Methods **Lecture 12 - 03/03/2015**

Lecture: Some Practical Considerations (1 of 4)

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

Warning: these notes are still very rough. They provide more details on what we discussed in class, but there may still be some errors, incomplete/imprecise statements, etc. in them.

12 How spectral clustering is typically done in practice

Today, we will shift gears. So far, we have gone over the theory of graph partitioning, including spectral (and non-spectral) methods, focusing on *why* and *when* they work. Now, we will describe a little about *how* and *where* these methods are used. In particular, for the next few classes, we will talk somewhat informally about some practical issues, e.g., how spectral clustering is done in practice, how people construct graphs to analyze their data, connections with linear and kernel dimensionality reduction methods. Rather than aiming to be comprehensive, the goal will be to provide illustrative examples (to place these results in a broader context and also to help people define the scope of their projects). Then, after that, we will get back to some theoretical questions making precise the how and where. In particular, we will then shift to talk about how diffusions and random walks provide a robust notion of an eigenvector and how they can be used to extend many of the vanilla spectral methods we have been discussing to very non-vanilla settings. This will then lead to how we can use spectral graph methods for other related problems like manifold modeling, stochastic blockmodeling, Laplacian solvers, etc.

Today, we will follow the von Luxburg review. This review was written from a machine learning perspective, and in many ways it is a very good overview of spectral clustering methods; but beware: it also makes some claims (e.g., about the quality-of-approximation guarantees that can be proven about the output of spectral graph methods) that—given what we have covered so far—you should immediately see are not correct.

12.1 Motivation and general approach

The motivation here is two-fold.

- Clustering is an extremely common method for what is often called *exploratory data analysis*. For example, it is very common for a person, when confronted with a new data set, to try to get a first view of the data by identifying subsets of it that have similar behavior or properties.
- Spectral clustering methods in particular are a very popular class of clustering methods. They are usually very simple to implement with standard linear algebra libraries; and they often outperform other methods such as k-means, hierarchical clustering, etc.

The first thing to note regarding general approaches is that Section 2 of the von Luxburg review starts by saying "Given a set of data points $x_1, \ldots, x_n$ and some notion of similarity $s_{ij} \geq 0$ between

all pairs of data points x_i and x_j ..." That is, the data are vectors. Thus, any graphs that might be constructed by algorithms are constructed from primary data that are vectors and are useful as intermediate steps only. This will have several obvious and non-obvious consequences. This is a very common way to view the data (and thus spectral graph methods), especially in areas such as statistics, machine learning, and areas that are not computer science algorithms. That perspective is not good or bad per se, but it is worth emphasizing that difference. In particular, the approach we will now discuss will be very different than what we have been discussing so far, which is more common in CS and TCS and where the data were a graph $G = (V, E)$, e.g., the single web graph out there, and thus in some sense a single data point. Many of the differences between more algorithmic and more machine learning or statistical approaches can be understood in terms of this difference. We will revisit it later when we talk about manifold modeling, stochastic blockmodeling, Laplacian solvers, and related topics.

12.2 Constructing graphs from data

If the data are vectors with associated similarity information, then an obvious thing to do is to represent that data as a graph $G = (V, E)$, where each vertex $v \in V$ is associated with a data point x_i an edge $e = (v_i v_j) \in E$ is defined if s_{ij} is larger than some threshold. Here, the threshold could perhaps equals zero, and the edges might be weighted by s_{ij}. In this case, an obvious idea to cluster the vector data is to cluster the nodes of the corresponding graph.

Now, let's consider how to specify the similarity information s_{ij}. There are many ways to construct a similarity graph, given the vectors $\{x_i\}_{i=1}^n$ data points as well as pairwise similarity (or distance) information s_{ij} (of d_{ij}). Here we describe several of the most popular.

- **ϵ-NN graphs.** Here, we connect all pairs of points with distance $d_{ij} \leq \epsilon$. Since the distance "scale" is set (by $\leq \epsilon$), it is common not to including the weights. The justification is that, in certainly idealized situations, including weights would not incorporate more information.
- **k-NN graphs.** Here, we connect vertex i with vertex j if v_i is among the k-NN of v_i, where NNs are given by the distance d_{ij}. Note that this is a directed graph. There are two common ways to make it undirected. First, ignore directions; and second, include an edge if (v_i connects to v_j AND v_j connects to v_i) or if (v_i connects to v_j OR v_j connects to v_i). In either of those cases, the number of edges doesn't equal k; sometimes people filter it back to exactly k edges per node and sometimes not. In either case, weights are typically included.
- **Fully-connected weighted graphs.** Here, we connect all points with a positive similarity to each other. Often, we want the similarity function to represent local neighborhoods, and so s_{ij} is either transformed into another form or constructed to represent this. A popular choice is the Gaussian similarity kernel

$$s(x_i, x_j) = \exp\left(\frac{1}{2\sigma^2}\|x_i - x_j\|_2^2\right),$$

  where σ is a parameter that, informally, acts like a width. This gives a matrix that has a number of nice properties, e.g., it is positive and it is SPSD, and so it is good for MLers who like kernel-based methods. Moreover, it has a strong mathematical basis, e.g., in scientific computing. (Of course, people sometimes use this $s_{ij} = s(x_i, x_j)$ information to construct ϵ-NN or k-NN graphs.)

Note that in describing those various ways to construct a graph from the vector data, we are already starting to see a bunch of knobs that can be played with, and this is typical of these graph construction methods.

Here are some comments about that graph construction approach.

- Choosing the similarity function is basically an art. One of the criteria is that typically one is not interested in resolving differences that are large, i.e., between moderately large and very large distances, since the goal is simply to ensure that those points are not close and/or since (for domain-specific reasons) that is the least reliable similarity information.

- Sometimes this approach is of interest in semi-supervised and transductive learning. In this case, one often has a lot of unlabeled data and only a little bit of labeled data; and one wants to use the unlabeled data to help define some sort of geometry to act as a prior to maximize the usefulness of the labeled data in making predictions for the unlabeled data. Although this is often thought of as defining a non-linear manifold, you should think of it at using unlabeled data to specify a data-dependent model class to learn with respect to. (That makes sense especially if the labeled and unlabeled data come from the same distribution, since in that case looking at the unlabeled data is a kin to looking at more training data.) As we will see, these methods often have an interpretation in terms of a kernel, and so they are used to learn linear functions in implicitly-defined feature spaces anyway.

- k-NN, ϵ-NN, and fully-connected weighted graphs are all the same in certain very idealized situations, but they can be very different in practice. k-NN often homogenizes more, which people often like, and/or it connects points of different "size scales," which people often find useful.

- Choosing k, ϵ, and σ large can easily "short circuit" nice local structure, unless (and sometimes even if) the local structure is extremely nice (e.g., one-dimensional). This essentially injects large-scale noise and expander-like structure; and in that case one should expect very different properties of the constructed graph (and thus very different results when one runs algorithms).

- The fully-connected weighted graph case goes from being a rank-one complete graph to being a diagonal matrix, as one varies σ. An important question (that is rarely studied) is how does that graph look like as one does a "filtration" from no edges to a complete graph.

- Informally, it is often thought that mutual-k-NN is between ϵ-NN and k-NN: it connects points within regions of constant density, but it doesn't connect regions of very different density. (For ϵ-NN, points on different scales don't get connected.) In particular, this means that it is good for connecting clusters of different densities.

- If one uses a fully-connected graph and then sparsifies it, it is often hoped that the "fundamental structure" is revealed and is nontrivial. This is true in some case, some of which we will return to later, but it is also very non-robust.

- As a rule of thumb, people often choose parameters s.t. ϵ-NN and k-NN graphs are at least "connected." While this seems reasonable, there is an important question of whether it homogenizes too much, in particular if there are interesting heterogeneities in the graph.

12.3 Connections between different Laplacian and random walk matrices

Recall the combinatorial or non-normalized Laplacian

$$L = D - W,$$

and the normalized Laplacian

$$L_{sym} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} W D^{-1/2}.$$

There is also a random walk matrix that we will get to more detail on in a few classes and that for today we will call the (somewhat non-standard name) random walk Laplacian

$$L_{rw} = D^{-1} L = I - D^{-1} W = D^{-1/2} L_{sym} D^{1/2}.$$

Here is a lemma connecting them.

Lemma 1. *Given the above definitions of L, L_{sym}, and L_{rw}, we have the following.*

1. *For all $x \in \mathbb{R}^n$,*
$$x^T L_{sym} x = \frac{1}{2} \sum_{ij} W_{ij} \left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \right)^2.$$
2. *λ is an eigenvalue of L_{rw} with eigenvector u iff λ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2} u$*
3. *λ is an eigenvalue of L_{rw} with eigenvector u iff λ and u solve the generalized eigenvalue problem $Lu = \lambda D u$.*
4. *0 is an eigenvalue of L_{rw} with eigenvector $\vec{1}$ iff 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}\vec{1}$*
5. *L_{sym} and L_{rw} are PSD and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \cdots \leq \lambda_n$.*

Hopefully none of these claims are surprising by now, but they do make explicit some of the connections between different vectors and different things that could be computed, e.g., one might solve the generalized eigenvalue problem $Lu = \lambda D u$ or run a random walk to approximate u and then from that rescale it to get a vector for L_{sym}.

12.4 Using constructed data graphs

Spectral clustering, as it is often used in practice, often involves first computing several eigenvectors (or running some sort of procedures that compute some sort of approximate eigenvectors) and then performing k-means in a low-dimensional space defined by them. Here are several things to note.

- This is harder to analyze than the vanilla spectral clustering we have so far been considering. The reason is that one must analyze the k means algorithm also. In this context, k-means is essentially used as a rounding algorithm.

- A partial justification of this is provided by the theoretical result on using the leading k eigenvectors that you considered on the first homework.
- A partial justification is also given by a result we will get to below that shows that it works in very idealized situations.

We can use different Laplacians in different ways, as well as different clustering, k-means, etc. algorithms in different ways to get spectral-like clustering algorithms. Here, we describe 3 canonical algorithms (that use L, L_{rw}, and L_{sym}) to give an example of several related approaches.

Assume that we have n points, $x_1, \ldots, x_n$, that we measure pairwise similarities $s_{ij} = s(x_i, x_j)$ with symmetric nonnegative similarity function, and that we denote the similarity matrix by $S = (S_{ij})_{i,j=1,\ldots,n}$. The following algorithm, let's call it PopularSpectralClustering, takes as input a similarity matrix $S \in \mathbb{R}^{n \times n}$ and a positive integer $k \in \mathbb{Z}^+$ which is the number of clusters; and it returns k clusters. It does the following steps.

1. Construct a similarity graph (e.g., with ϵ-NN, k-NN, fully-connected graph, etc.)
2. Compute the unnormalized Laplacian $L = D - A$.
3. - If (use L)
     then compute the first k eigenvectors $u_1, \ldots, u_k$ of L,
   - else if (use L_{rw})
     then compute first k generalized eigenvectors $u_1, \ldots, u_k$ of the generalized eigenvalue problem $Lu = \lambda Du$. (Note by the above that these are eigenvectors of L_{rw}.)
   - else if (use L_{sym})
     then compute the first k eigenvectors $u_1, \ldots, u_k$ of L_{sym}.
4. Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vevtors $u_1, \ldots, u_k$ as columns.
5. - If (use L_{sym})
     then $u_{ij} \leftarrow u_{ij} / \left(\sum_k u_{ik}^2\right)^{1/2}$, i.e., normalize U row-wise.
6. For $i = \{1, \ldots, n\}$, let $y_i \in \mathbb{R}^k$ be a vector containing the i^{th} row of U.
7. Cluster points $(y_i)_{i \in [n]}$ in $\mathbb{R}^k$ with a k-means algorithm into clusters, call them $C_1, \ldots, C_k$.
8. Return: clusters $A_1, \ldots, A_k$, with $A_i = \{j : y_j \in C_i\}$.

Here are some comments about the PopularSpectralClustering algorithm.

- The first step is to construct a graph, and we discussed above that there are a lot of knobs. In particular, the PopularSpectralClustering algorithm is not "well-specified" or "well-defined," in the sense that the algorithms we have been talking about thus far are. It might be better to think of this as an algorithmic approach, with several knobs that can be played with, that comes with suggestive but weaker theory than what we have been describing so far.
- k-means is often used in the last step, but it is not necessary, and it is not particularly principled (although it is often reasonable if the data tend to cluster well in the space defined by U). Other methods have been used but are less popular, presumably since k-means is

good enough and there are enough knobs earlier in the pipeline that the last step isn't the bottleneck to getting good results. Ultimately, to get quality-of-approximation guarantees for an algorithm like this, you need to resort to a Cheeger-like bound or a heuristic justification or weaker theory that provides justification in idealized cases.

- In this context, k-means is essentially a rounding step to take a continuous embedding, provided by the continuous vectors $\{y_i\}$, where $y_i \in \mathbb{R}^k$, and put them into one of k discrete values. This is analogous to what the sweep cut did. But we will also see that this embedding, given by $\{y_i\}_{i=1}^n$ can be used for all sorts of other things.

- Remark: If one considers the k-means objective function, written as an IP and then relaxes it (from having the constraint that each data point goes into one of k clusters, written as an orthogonal matrix with one nonzero per column, to being a general orthogonal matrix), then you get an objective function, the solution to which can be computed by computing a truncated SVD, i.e., the top k singular vectors. This provides a 2-approximation to the k-means objective. There are better approximation algorithms for the k-means objective, when measured by the quality-of-approximation, but this does provide an interesting connection.

- The rescaling done in the "If (use L_{sym}) then" is typical of many spectral algorithms, and it can be the source of confusion. (Note that the rescaling is done with respect to $(P_U)_{ii} = \left(UU^T\right)_{ii}$, i.e., the statistical leverage scores of U, and this means that more "outlying" points get down-weighted.) From what we have discussed before, it should not be surprising that we need to do this to get the "right" vector to work with, e.g., for the Cheeger theory we talked about before to be as tight as possible. On the other hand, if you are approaching this from the perspective of engineering an algorithm that returns clusters when you expect them, it can seem somewhat ad hoc. There are many other similar ad hoc and seemingly ad hoc decisions that are made when engineering implementations of spectral graph methods, and this lead to a large proliferation of spectral-based methods, many of which are very similar "under the hood."

All of these algorithms take the input data $x_i \in \mathbb{R}^n$ and change the representation in a lossy way to get data points $y_i \in \mathbb{R}^k$. Because of the properties of the Laplacian (some of which we have been discussing, and some of which we will get back to), this *often* enhances the cluster properties of the data.

In idealized cases, this approach works as expected, as the following example provides. Say that we sample data points from $\mathbb{R}$ from four equally-spaced Gaussians, and from that we construct a NN graph. (Depending on the rbf width of that graph, we might have an essentially complete graph or an essentially disconnected graph, but let's say we choose parameters as the pedagogical example suggests.) Then $\lambda_1 = 0$; λ_2, λ_3, and λ_4 are small; and λ_5 and up are larger. In addition, v_1 is flat; and higher eigenfunctions are sinusoids of increasing frequency. The first few eigenvectors can be used to split the data into the four natural clusters (they can be chosen to be worse linear combinations, but they can be chosen to split the clusters as the pedagogical example suggests). But this idealized case is chosen to be "almost disconnected," and so it shouldn't be surprising that the eigenvectors can be chosen to be almost cluster indicator vectors. Two things: the situation gets much messier for real data, if you consider more eigenvectors; and the situation gets much messier for real data, if the clusters are, say, 2D or 3D with realistic noise.

12.5 Connections with graph cuts and other objectives

Here, we will briefly relate what we have been discussing today with what we discussed over the last month. In particular, we describe the graph cut point of view to this spectral clustering algorithm. I'll follow the notation of the von Luxburg review, so you can go back to that, even though this is very different than what we used before. The point here is not to be detailed/precise, but instead to remind you what we have been covering in another notation that is common, especially in ML, and also to derive an objective that we haven't covered but that is a popular one to which to add constraints.

To make connections with the PopularSpectralClustering algorithm and MinCut, RatioCut, and NormalizedCut, recall that

$$\text{RatioCut}(A_1, \ldots, A_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{W\left(A_i, \bar{A}_i\right)}{|A_i|} = \sum_{i=1}^{k} \frac{\text{cut}\left(A_i, \bar{A}_i\right)}{|A_i|},$$

where $\text{cut}(A_1, \ldots, A_k) = \frac{1}{2}\sum_{i=1}^{k}\left(A_i, \bar{A}_i\right)$.

First, let's consider the case $k = 2$ (which is what we discussed before). In this case, we want to solve the following problem:

$$\min_{A \subset V} \text{RatioCut}\left(A, \bar{A}\right). \tag{1}$$

Given $A \subset V$, we can define a function $f = (f_1, \ldots, f_n)^T \in \mathbb{R}^n$ s.t.

$$f_i = \begin{cases} \sqrt{|\bar{A}|/|A|} & \text{if } v_i \in A \\ -\sqrt{|A|/|\bar{A}|} & \text{if } v_i \in \bar{A} \end{cases}. \tag{2}$$

In this case, we can write Eqn. (1) as follows:

$$\begin{aligned} &\min_{A \subset V} f^T L f \\ &\text{s.t. } \; f \perp \vec{1} \\ &\qquad f \text{ defined as in Eqn. (2)} \\ &\qquad \|f\| = \sqrt{n}. \end{aligned}$$

In this case, we can relax this objective to obtain

$$\begin{aligned} &\min_{A \subset V} f^T L f \\ &\text{s.t. } \; f \perp \vec{1} \\ &\qquad \|f\| = \sqrt{n}, \end{aligned}$$

which can then be solved by computing the leading eigenvectors of L.

Next, let's consider the case $k > 2$ (which is more common in practice). In this case, given a partition of the vertex set V into k sets $A_i, \ldots, A_k$, we can define k indicator vectors $h_j = (h_{ij}, \ldots, h_{nj})^T$ by

$$h_{ij} = \begin{cases} 1/\sqrt{|A_j|} & v_i \in A_j, \quad i \in [n], j \in [k] \\ 0 & \text{otherwise} \end{cases}. \tag{3}$$

Then, we can set the matrix $H \in \mathbb{R}^{n \times k}$ as the matrix containing those k indicator vectors as columns, and observe that $H^T H = I$, i.e., H is an orthogonal matrix (but a rather special one, since it has only one nonzero per row).

We note the following observation; this is a particular way to write the RatioCut problem as a Trace problem that appears in many places.

Claim 1. *RatioCut*$(A_1, \ldots, A_k) =$ *Tr*$(H^T L H)$

Proof. Observe that

$$h_i^T L h_i = \frac{\text{cut}\left(A_i, \bar{A}_i\right)}{|A_i|}$$

and also that

$$h_i^T L h_i = \left(H^T L H\right)_{ii}.$$

Thus, we can write

$$\text{RatioCut}\left(A_1, \ldots, A_k\right) = \sum_{i=1}^{k} h_i L h_i = \sum_{i=1}^{k} \left(H^T L H\right)_{ii} = \text{Tr}\left(H^T L H\right).$$

□

So, we can write the problem of

$$\min \text{RatioCut}\left(A_1, \ldots, A_k\right)$$

as follows:

$$\begin{aligned} \min_{A_1, \ldots, A_k} \ & \text{Tr}\left(H^T L H\right) \\ \text{s.t.} \ & H^T H = I \\ & H \text{ defined as in Eqn. (3).} \end{aligned}$$

We can relax this by letting the entries of H be arbitrary elements of $\mathbb{R}$ (stil subject to the overall orthogonality constraint on H) to get

$$\begin{aligned} \min_{H \in \mathbb{R}^{n \times k}} \ & \text{Tr}\left(H^T L H\right) \\ \text{s.t.} \ & H^T H = I, \end{aligned}$$

and the solution to this is obtained by computing the first k eigenvectors of L.

Of course, similar derivations could be provided for the NormalizedCut objective, in which case we get similar results, except that we deal with degree weights, degree-weighted constraints, etc. In particular, for $k > 2$, if we define indicator vectors $h_j = (h_{ij}, \ldots, h_{nj})^T$ by

$$h_{ij} = \begin{cases} 1/\sqrt{\text{Vol}(A_j)} & v_i \in A_j, \quad i \in [n], j \in [k] \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

then the problem of minimizing NormalizedCut is

$$\begin{aligned} \min_{A_1,\ldots,A_k} \ & \mathrm{Tr}\left(H^T L H\right) \\ \text{s.t.} \ & H^T D H = I \\ & H \text{ defined as in Eqn. (4)}, \end{aligned}$$

and if we let $T = D^{1/2}H$, then the spectral relaxation is

$$\begin{aligned} \min_{T \in \mathbb{R}^{n \times k}} \ & \mathrm{Tr}\left(T^T D^{-1/2} L D^{-1/2} T\right) \\ \text{s.t.} \ & T^T T = I, \end{aligned}$$

and the solution T to this trace minimization problem is given by the leading eigenvectors of L_{sym}. Then $H = D^{-1/2}T$, in which case H consists of the first k eigenvectors of L_{rw}, or the first k generalized eigenvectors of $Lu = \lambda Du$.

Trace optimization problems of this general for arise in many related applications. For example:

- One often uses this objective as a starting point, e.g., to add sparsity or other constraints, as in one variation of "sparse PCA."
- Some of the methods we will discuss next time, i.e., LE/LLE/etc. do something very similar but from a different motivation, and this provides other ways to model the data.
- As noted above, the k-means objective can actually be written as an objective with a similar constraint matrix, i.e., if H is the cluster indicator vector for the points, then $H^T H = I$ and H has one non-zero per row. If we relax that constraint to be any othogonal matrix such that $H^T H = I$, then we get an objective function, the solution to which is the truncated SVD; and this provides a 2 approximation to the k-means problem.