CSE 521: Design and Analysis of Algorithms I **Spring 2016**

Lecture 11: Clustering and the Spectral Partitioning Algorithm

Lecturer: Shayan Oveis Gharan *May 2nd* *Scribe: Yueqi Sheng*

11.1 A note on randomized algorithm, Unbiased estimates

In the design of randomized algorithms, we sometimes wish to estimate some quantity using an unbiased estimator. (Say we wish to estimate some quantity A, $X \sim \mu$ is an unbiased estimator if $\mathbb{E}[X] = A$.)

We would like those estimations to be close to its mean w.h.p. Given some distribution μ, let $Y_1, \cdots, Y_n$ be i.i.d samples from μ and $Y = \frac{1}{n}\sum_i Y_i$ be the empirical mean. Recall that in Problem 3, PS2, setting $n = 25$ or 50 doesn't make that much of a difference.

What does it take to get a decent answer? Crucially, how good the estimator is depends heavily on its variance. The usual concentration bounds gives

$$\mathbb{P}[|Y = \mathbb{E}[Y]| \geq \alpha\sigma(Y)] \leq \frac{1}{\alpha^2} \quad (Chebyshev)$$

If Y_i are Bernoulli random variables,

$$\mathbb{P}[|Y - \mathbb{E}[Y]| \geq \alpha\sigma(Y)\|] \leq \exp(-O(\alpha)) \quad (Chernoff)$$

Let $X \sim \mu$. It is easy to see $\mathbb{E}[Y] = \mathbb{E}[X]$, $\sigma(Y) = \frac{\sigma(X)}{\sqrt{n}}$. To get $\sigma(Y)$ small, we might need to sample many times.

For example, say $Y_i = \begin{cases} 0, & \frac{1}{2} \\ 4000, & \frac{1}{2} \end{cases}$, $\sigma(Y) = \frac{2000}{\sqrt{n}}$. To get $\sigma(Y) = 10$, we need to have $n \approx 10000$.

11.2 Spectral algorithms

Spectral method solves problems by looking at eigenvalues, eigenvectors, singular values etc. The first family of spectral algorithms uses the idea of low rank approximation. We saw an example in the last lecture where we designed an algorithm for the maximum cut problemj. Let us summarize the algorithm that we discussed.

11.2.1 Spectral Algorithms using Low Rank Approximation

Given a graph G, let A be the adjacency matrix of G and x_S be the indicator vector of a set S. The algorithm consists of roughly three steps,

- Formulate the problem as $\max_S x_S A(1 - x_S)$
- approximate A with $\tilde{A}_k$ where $\tilde{A}_k = \sum_{i=1}^k \sigma_i u_i v_i^T$ where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ are the singular values of A. Estimate $x^T \tilde{A}_k x = \sum_i \sigma_i \langle x, u_i\rangle \langle v_i, 1 - x\rangle$.

- For each $S \subset V$, there are $2k$ numbers to learn, $w(S) = (\langle u_i, x_S \rangle, \langle v_i, x_S \rangle)$. Approximate $w(S)$ using an $\epsilon - net$.

Remark: The idea of ϵ-net is to partition a space into small cells of diameter ϵ and represent each cells with a point. Intuitively, if ϵ is large, then the number of cells is much smaller than the number of points, which will make the algorithm faster. Meanwhile, by choosing ϵ small, we get better approximation since the representative point is closer to actual points in that cell. So, there is a trade-off between running time and the accuracy of the solution, We need to choose ϵ in a way depending on the amount of time we can spend to find the solution.

11.2.2 Graph Based Spectral Algorithms

In this lecture and the next one we discuss a second family of spectral algorithms. Here the highlevel idea is to model the problem input as a graph and use eigenvalues and eigenvectors of the matrices associated to the graph to solve the corresponding problem on the graph.

For example, given a data set $P \in \mathbb{R}^n$, one can construct a graph with each node being a point $x \in P$ and edge (x, y) between each nodes with weight $w(x, y) = \|x - y\|_2$. Another common example would be a social network graph.

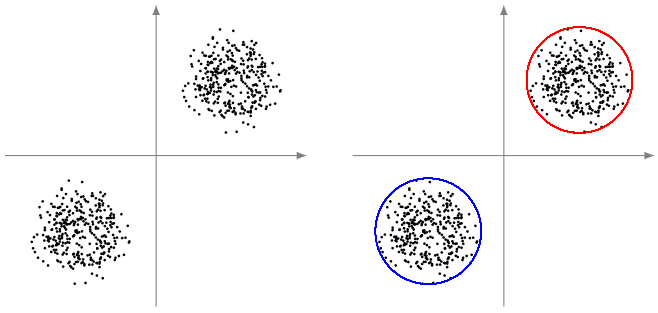
This implies the following general scheme for designing spectral algorithms.

- Construct a weighted graph G.
- Compute the eigenvalues and eigenvectors of the adjacency matrix (A) or the Laplacian matrix L_G.
- Search the graph to find a solution to the original problem using the eigenvectors that was found in the previous step.

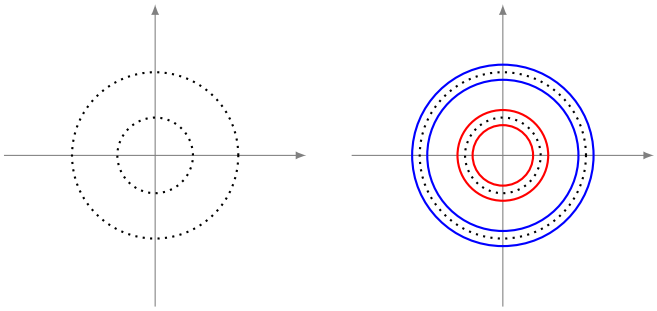
11.3 Clustering

Clustering refers to group points that are "close" to each other together. One well known algorithm for clustering based on Euclidean distance is the k-means algorithm. Given $P \in \mathbb{R}^n$, it finds k cluster centers and assign all points in P to the nearest center.

The following example demonstrates an instance of the algorithm for $k = 2$



However, k-means is only good when the points are naturally separated since it requires all points in a cluster are close to each other. The following instance shows an example where k-means algorithm would fail.



Ideally, we wish to cluster points on a circle (as shown on the right) but they are both centered at $(0, 0)$. It is not clear what k-means will do in this case.

Another idea is to use spectral methods: Given the same set of points P, construct a weighted complete graph $G = (V, E)$ with $V = P$ and $w(x, y) = \exp(-\frac{\|p-q\|_2}{\sigma^2})$ for all $x, y \in P$ for some carefully chosen σ. Use a graph partitioning algorithm to partition the graph. That partitioning gives a clustering of the original data points. Intuitively, this will give us the right cluster showing on the right; even though there are points which are very far in the same cluster it is possible to go from any point in a cluster to any other point in that cluster while taking only small steps.

11.3.1 Quality of cluster

The above idea reduces data clustering to graph partitioning. So, from now on we talk about graph partitioning. The first question that we need to answer is how to measure the quality of a a graph partitioning. While the exact answer depends on specific problems, here are some standard ways of measuring this quality.

Given unweighted graph G and $S \subseteq V$, one can look at

Diameter. $diam(S) = \max_{u,v \in S} d(u, v)$ where $d(u, v)$ is the length of shortest path from u to v in G

Clustering Coefficient. This is a measure of the density of triangles in S with respect to the vertex degrees. See https://en.wikipedia.org/wiki/Clustering_coefficient for more information.

Conductance. $\phi(S) = \frac{E[S,\bar{S}]}{\text{vol}(S)}$ where $E[S, \bar{S}] = |\{(u, v) : u \in S, v \in \bar{S}\}|$ and $\text{vol}(S) = \sum_{v \in S} d(v)$

(Note that the above definition could be easily extended to weighted graph as we will discuss later.)

In this notes we will focus on conductance.

11.3.2 Conductance

Conductance of S is a measure of how "connected" S is to the rest of the graph. Let us first state some cute observations of conductance.

Claim 11.1. *For any G and $S \subseteq V$,*

$$0 \leq \phi(S) \leq 1$$

Proof. Since $E[S, \bar{S}] = |\{(u, v) : u \in S, v \in \bar{S}\}| \geq 0$. we have $\phi(S) = \frac{E[S,\bar{S}]}{vol(S)} \geq 0$. In particular, observe that if, $\phi(S) = 0$, then S is a connected component of G.

Let $E(S) = \{(u, v) : u, v \in S\}$ be the set of edges in S. For the other direction, observe that

$$\text{vol}(S) = |E(S, \overline{S})| + 2|E(S)| \geq |E(S, \overline{S})|.$$

Thus $\text{vol}(S) \geq |E(S, \bar{S})|$. In the extreme case, we have $\phi(S) = 1$ when no edge lies inside S. □

If $\phi(S) \approx 0$, then vertices of S are more connected to each other as opposed to the rest of the graph, so one can think of S as a community or a cluster. In the other extreme, if $\phi(S) \approx 1$, it means that most of the edges incident to the vertices of S are leaving S. So, vertices of S do not make up a separate community.

We can define conductance of a graph in a similar manner,

Definition 11.2. *The conductance of G is*

$$\phi(G) = \min_{\text{vol}(S) \leq \frac{\text{vol}(V)}{2}} \phi(S)$$

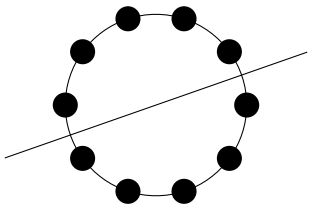
By a similar argument, if $\phi(G) \approx 0$ then G has an ideal 2-partitioning and if $\phi(G) \approx 1$, the whole graph is just one cluster. To minimize conductance, one wish to group as many edges in a cluster as possible while try not to cut too many edges.

Fact 11.3. *If G is an n cycle, $\phi(G) = \frac{2}{n}$.*

Proof. Given any S, $|E(S, \overline{S})| \geq 2$. Since $d(v) = 2$ for all $v \in V$, $\text{vol}(S) \leq \frac{\text{vol}(V)}{2} = n$. This implies

$$\phi(S) \geq \frac{2}{n} \quad \forall S \subseteq V$$

The equality is by letting S be a path of length $n/2$. □



Similarly, for complete graph K_n, split the graph into two equal sized pieces gives

$$\phi(K_n) = \frac{(\frac{n}{2})^2}{\frac{n}{2}(n-1)} \approx \frac{1}{2}$$

This says complete graph is very well connected.

The definition could be easily generalized to weighted graph.

Definition 11.4. *If G is weighted,*

$$\phi(S) = \frac{w(S, \overline{S})}{\sum_i d_w(i)}$$

where $w(S, \overline{S})$ is the sum of the weight all edges in the cut $(S, \overline{S})$ and $d_w(.)$ is the weighted degree of a vertex.

Let us describe an application of conductance in planar graphs.

Theorem 11.5 (Spielman, Teng [ST96]). *If G is planar graph with constant degree, then $\phi(G) \leq O(\frac{1}{\sqrt{n}})$.*

The above theorem gives an amazing opportunity for employing divide and conquor approach on planar graphs. Roughly speaking, it says that by cutting only a few edges of a planar graph it can be decomposes into two separate pieces each of size roughly $n/2$. So, we can solve the problem at hand on each piece recursively and then merge the solutions considering the few edges that were cut.

Observe that the above bound is indeed tight. It is easy to see that for a $\sqrt{n} \times \sqrt{n}$ grid, we have $\phi(G) = O(1/\sqrt{n})$.

11.4 Spectral Partitioning Algorithm

In the rest of this lecture and the next one we describe a spectral algorithm to approximate $\phi(G)$.

The problem of approximating $\phi(G)$ is an NP-hard problem, so we can only hope to approximately find the value of $\phi(G)$. Spectral partitioning algorithm always returns an S with $\phi(S) \leq \sqrt{\phi(G)}$. Such an approximation may is ideal if $\phi(G)$ is large, but for smaller value of $\phi(G)$ the returned solution may be very far from being optimal. We will see some examples in the next lecture.

One can use linear programming to obtain an $O(\log n)$ approximation algorithm for $\phi(G)$ [LR99] or semidefinite programming to obtain a $\sqrt{\log n}$ approximation [ARV09].

Spectral algorithm are very appealing in this context because they are very easy to implement and they run in near linear time.

Now, we are ready to describe the spectral algorithm. In the next lecture we discuss some ideas of the analysis.

Given G, let A be the weighted adjacency matrix of G s.t.

$$A_{i,j} = w(i,j)$$

and D be the diagonal matrix where

$$D_{i,i} = d_w(i).$$

1. Compute the second largest eigenvector, v^2, of $D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$.
2. Sort the vertices of G according to $D^{-1/2}v^2$; so say we have a permutation $\pi : [n] \to [n]$ such that

$$\frac{v^2_{\pi(1)}}{\sqrt{d(\pi(1))}} \leq \frac{v^2_{\pi(2)}}{\sqrt{d(\pi(2))}} \leq \cdots \leq \frac{v^2_{\pi(n)}}{\sqrt{d(\pi(n))}}.$$

3. Sweep these vertices from left to right and return the least conductance set that you find, i.e.,

$$\min_{1 \leq i \leq n} \frac{|E(S_i, \overline{S_i})|}{\min\{\mathrm{vol}(S_i), \mathrm{vol}(\overline{S_i})\}} \text{ where } S_i = \{\pi(1), \ldots, \pi(i)\}.$$

11.5 Laplacian

Recall that besides adjacency matrix, another common object to look at is the Laplacian.

Definition 11.6. *Given a graph G, let A and D as defined above, Laplacian of G, L_G is defined as*

$$L_G = D - A$$

This matrix is very nice.

Fact 11.7. *G is unweighted, the quadratice form of L_G is*

$$x^T L_G x = \sum_i d(i)x_i^2 + 2 \sum_{(i,j)\in G; i<j} x_i x_j = \sum_{(i,j)\in G} (x_i - x_j)^2 \geq 0$$

Similarly, for a weighted graph G

$$x^T L_G x = \sum_{(i,j)\in G} w(i,j)(x_i - x_j)^2$$

This directly implies the following fact:

Fact 11.8. $L_G \succeq 0$.

Lemma 11.9. $\lambda_{min}(L_G) = 0$.

Proof. It is easy to see from the quadratic form of L_G, $1^T L_G 1 = 0$. Since $L_G \succeq 0$, we have $\lambda_{min}(L_G) = 0$. □

Lemma 11.10. *G is not connected iff the second smallest eigenvalue of L_G is 0.*

We will prove this lemma in the next lecture.

References

[ARV09] S. Arora, S. Rao, and U. Vazirani. "Expander flows, Geometric Embeddings and Graph Partitioning". In: *J. ACM* 56 (2 Apr. 2009), 5:1–5:37 (cit. on p. 11-5).

[LR99] T. Leighton and S. Rao. "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms". In: *Journal of The ACM* 46 (6 Nov. 1999), pp. 787–832. URL: http://doi.acm.org/10.1145/331524.331526 (cit. on p. 11-5).

[ST96] D. A. Spielman and S.-H. Teng. "Spectral Partitioning Works: Planar Graphs and Finite Element Meshes". In: *FOCS*. 1996, pp. 96–105 (cit. on p. 11-5).