Unsupervised Statistical Learning: Clustering Graphs

Siva Balakrishnan
Data Mining: 36-462/36-662

April 4th, 2019

Outline for Today

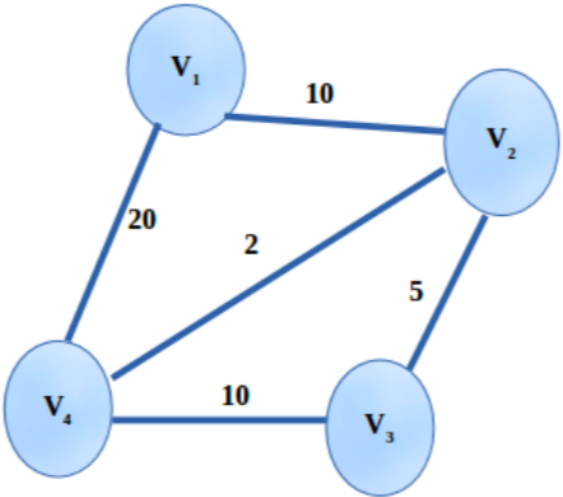
- Recap: Graphs and Clustering Graphs
- Spectral clustering
- Multi-dimensional scaling (MDS) (just an introduction)

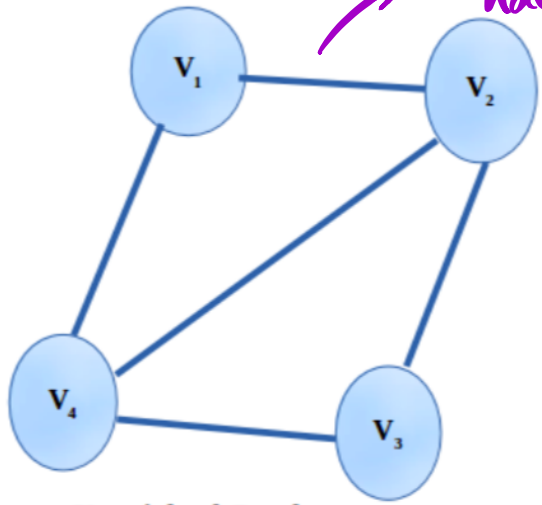
Recap: Graphs

It is often convenient and useful to think about data in terms of graphs.

- **(Unweighted) Graphs:** Just vertices and edges. Equivalent to every edge having weight 1.
- **Weighted Graphs:** Each edge, say between vertices i and j, has weight w_{ij}.

For us, graphs will usually be **undirected** (i.e. the edges do not have an orientation), and weights will usually be positive.

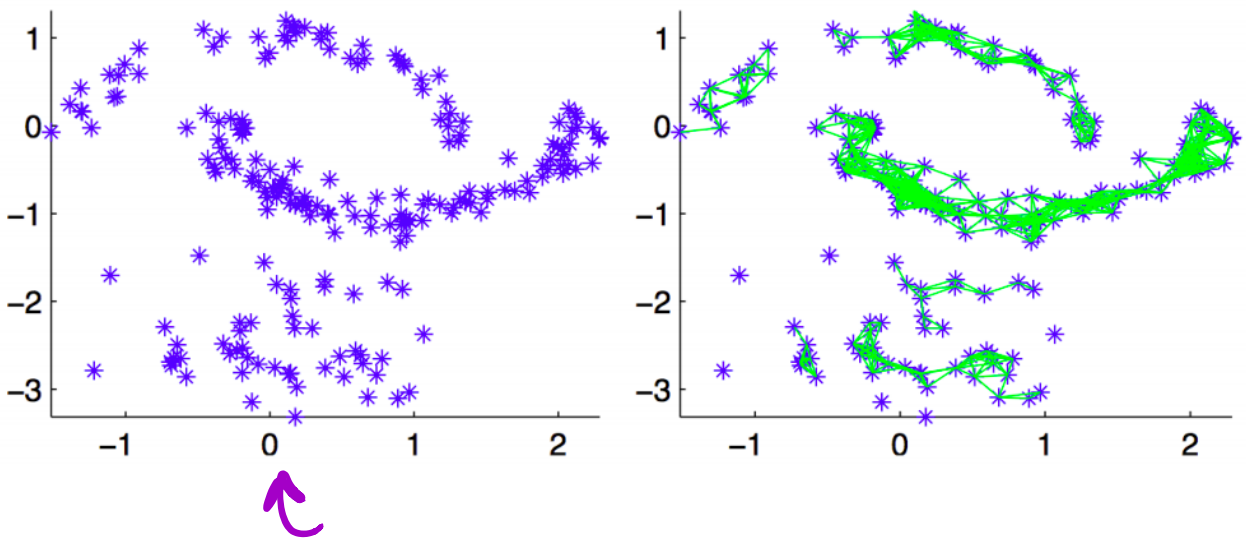




Recap: From Data to Graphs

We are given our usual collection of data points $\{X_1, \ldots, X_n\}$.
How do we build a graph from these?
Roughly:

1. **Nodes:** These are the data points.
2. **Edges/Weights:** We want to connect points that are similar. Weights will measure "similarity".



Recap: From Data to Graphs

Three canonical ways:

- **ϵ-neighborhood graph:** connect i & j if $\|x_i - x_j\|_2 \leq \epsilon$

unweighted

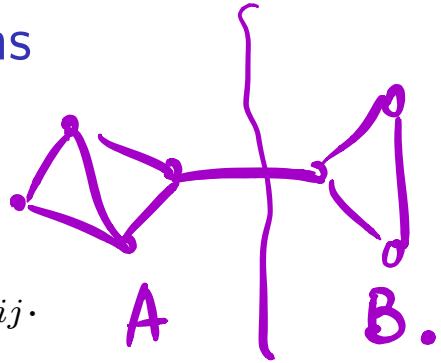
- **k-NN graph:** if either x_i is one of x_j's k-nearest neighbors or vice-versa.

- **Weighted similarity graph:** fully connected

$$W_{ij} = \exp\left\{ - \frac{\|x_i - x_j\|_2^2}{\sigma^2} \right\}.$$

$\sigma \rightarrow$

Recap: Clustering Graphs



First Attempt: Find best cut, i.e.:

min-cut.

$$\min_{A,B} \text{cut}(A,B) = \min_{A,B} \sum_{i \in A, j \in B} w_{ij}.$$

Does poorly in practice.

Second Attempt: Find best balanced cut:

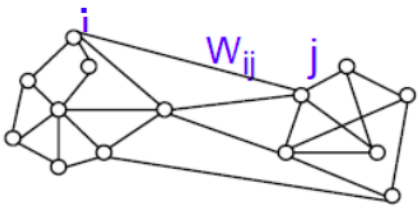
$$\min_{A,B \text{ of equal size}} \text{cut}(A,B).$$

Does very well in practice. However, hard to compute. Spectral clustering is a fast, approximate way to find a balanced cut.

Recap: Graphs as Matrices

- **Adjacency Matrix:** We can just collect the weights W_{ij} into a (symmetric) matrix W. This is called the adjacency matrix.

put all edge wts into a matrix.

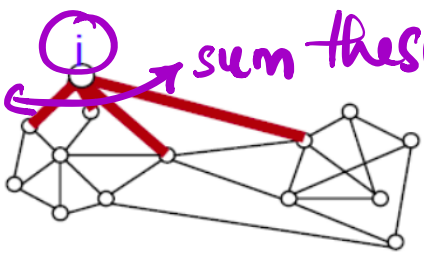


$W \in \mathbb{R}^{n \times n}$.

- **Degree Matrix:** The *degree* of a node is the sum of the weights of the edges connected to that node. We can collect the degrees in a diagonal matrix D, where

$$D_{ii} = \sum_{j=1}^{n} W_{ij}.$$

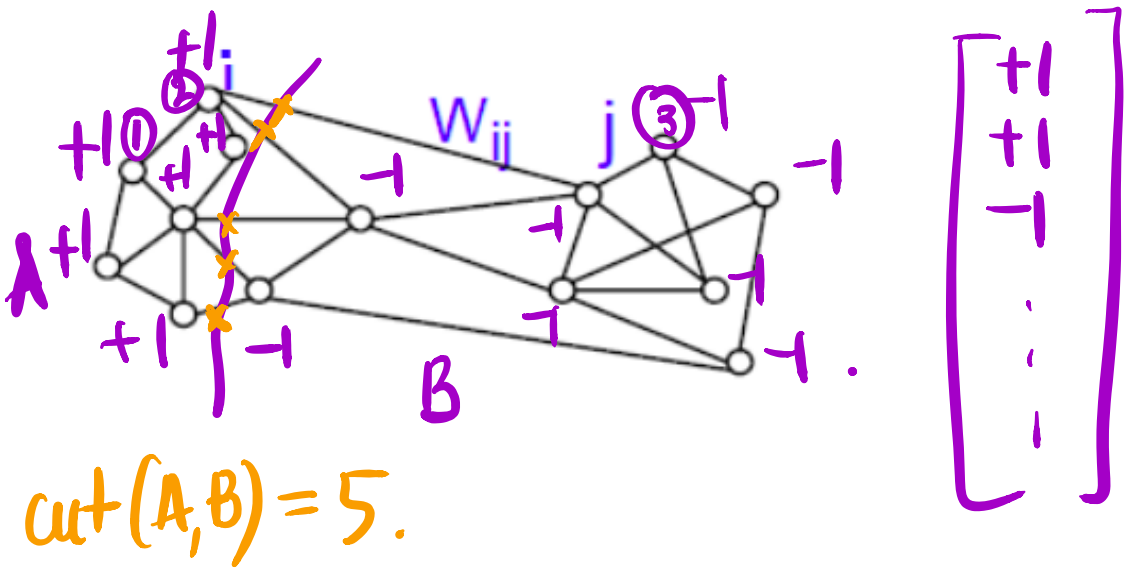
$\begin{bmatrix} \deg(1) & & 0 \\ & \deg(2) & \\ 0 & & \ddots \end{bmatrix}$



sum these wts deg(i).

Recap: Cuts as Vectors

- **Cut Vectors:** For every partition (A, B) of the vertices, we can associate a vector v_{AB}. The entries of v_{AB} will be $+1$ on A and -1 on B.



Recap: The Minimum Balanced Cut

Some ~~simple~~ tedious algebra shows the following:

$$\text{cut}(A, B) = \frac{1}{4} v_{AB}^T (D - W) v_{AB}.$$

cut vector corresponding to $\{A, B\}$.

So if we want to find the minimum balanced cut we can instead solve the following problem:

$$\arg\min_{v} v^T (D - W) v$$

v defines a cut

- Entries of v are all $\{+1, -1\}$ (so it is a cut vector).
- Entries of v sum to 0 (so it is balanced).

The Graph Laplacian

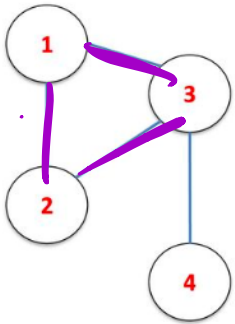
The matrix:

$$L = D - W,$$

is called the *Graph Laplacian*.

- The graph Laplacian is a very important matrix in understanding graphs (arises naturally in partitioning problems, understanding random walks on graphs, understanding flow and congestion in graphs...).

An example



Example graph

$W =$

	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	0	0	1	0

	1	2	3	4
1	2	0	0	0
2	0	2	0	0
3	0	0	2	0
4	0	0	0	1

	1	2	3	4
1	2	-1	-1	0
2	-1	2	-1	0
3	-1	-1	2	-1
4	0	0	-1	1

$L = D - W$

Properties of the Graph Laplacian

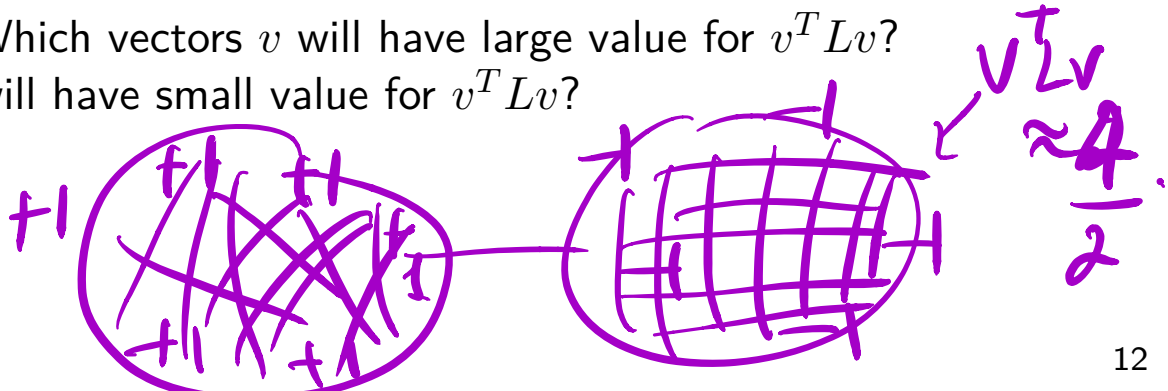
It is a symmetric, real-valued matrix, so it has an eigendecomposition. We have already seen that for any vector v:

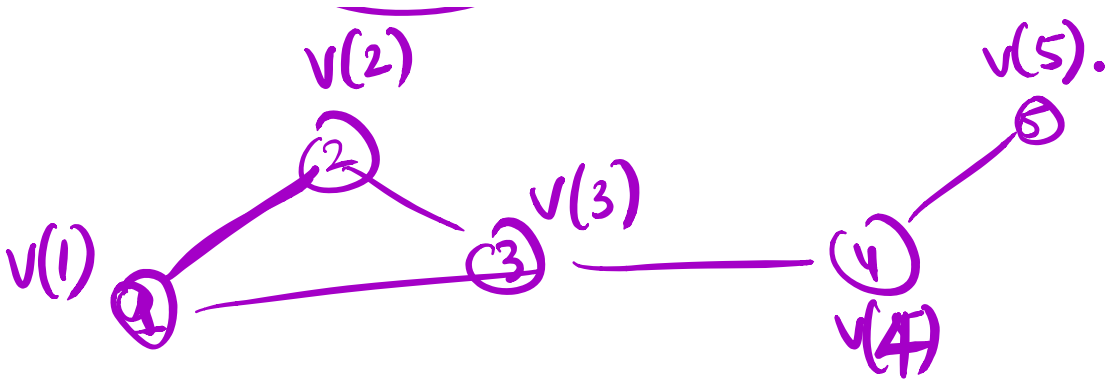
$$v^T L v = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} (v(i) - v(j))^2 \geq 0,$$

so all its eigenvalues are positive.

$$\frac{1}{2} \sum_{(i,j) \in E} (v(i) - v(j))^2$$

Intuition: Which vectors v will have large value for $v^T L v$? Which ones will have small value for $v^T L v$?



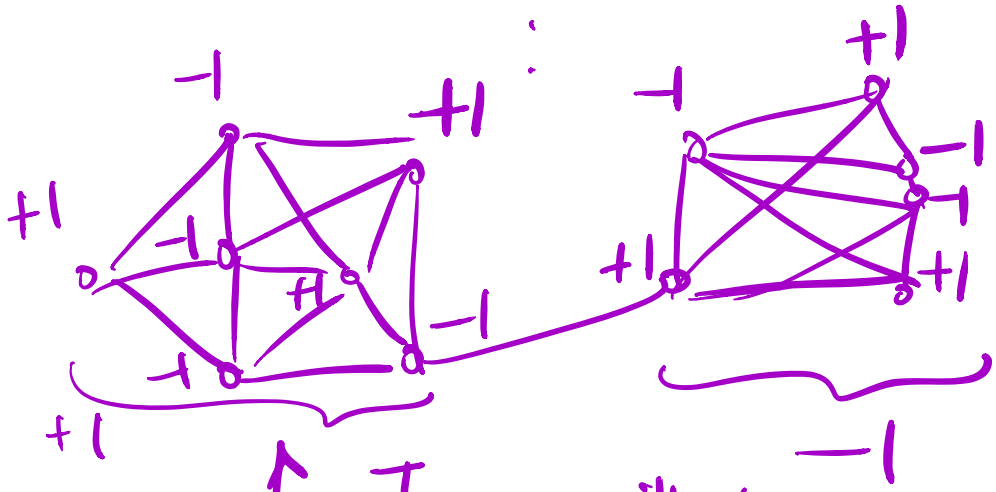


$$v^T L v = \sum_{\{i,j\} \in E} (v(i) - v(j))^2$$

$$+ (v(2) - v(3))^2$$

$$+ (v(1) - v(3))^2$$

$$\vdots$$



$v^T L v$ will be large

Spectrum of the Graph Laplacian

- All the eigenvalues of the Laplacian are positive.
- The vector $v = [1, 1, \ldots, 1]^T$ (you can normalize it if you prefer) is an eigenvector of the graph Laplacian, with eigenvalue 0. To see this we just have to check:

$$Lv = (D - W)\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0 \times \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

- This means that all other eigenvectors v_j must satisfy the condition that:

$$v_j^T \times \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = 0$$

→ eigenvectors are ⊥.

So every other eigenvector is "balanced".

The point so far

We want to find a good balanced cut. We have seen that this is the same as finding a vector v which minimizes:

$$\min_v v^T(D-W)v,$$

where v satisfies two conditions:

- Its entries are $+1$ and -1 (so it defines a cut).
- Its entries sum to 0 (so that the cut is balanced):

$$\sum_{i=1}^{n} v(i) = 0.$$

Basic Spectral Clustering

We want to solve the (computationally difficult) problem:

$$\min_v v^T(D-W)v,$$

where v satisfies two conditions:

- Its entries are $+1$ and -1 (so it defines a partition).
- Its entries sum to 0 (so that the partition is balanced):

$$\sum_{i=1}^{n} v(i) = 0.$$

Instead we will solve the relaxation:

$$\min_v v^T(D-W)v,$$

where v satisfies **one** condition:

- Its entries sum to 0 (so that the partition is balanced):

$$\sum_{i=1}^{n} v(i) = 0.$$

Basic Spectral Clustering

Instead we will solve the relaxation:

$$\min_v v^T(D - W)v,$$

where v satisfies **one** condition:

- Its entries sum to 0 (so that the partition is balanced):

$$\sum_{i=1}^{n} v(i) = 0.$$

The solution is just the second smallest eigenvector of the Laplacian (easy to compute). However, we now have a problem.

Entries of v_2 are not +1 or −1.

And a solution: $\{A : v_2(i) > 0\}$

$\{B : v_2(i) \leq 0\}$.

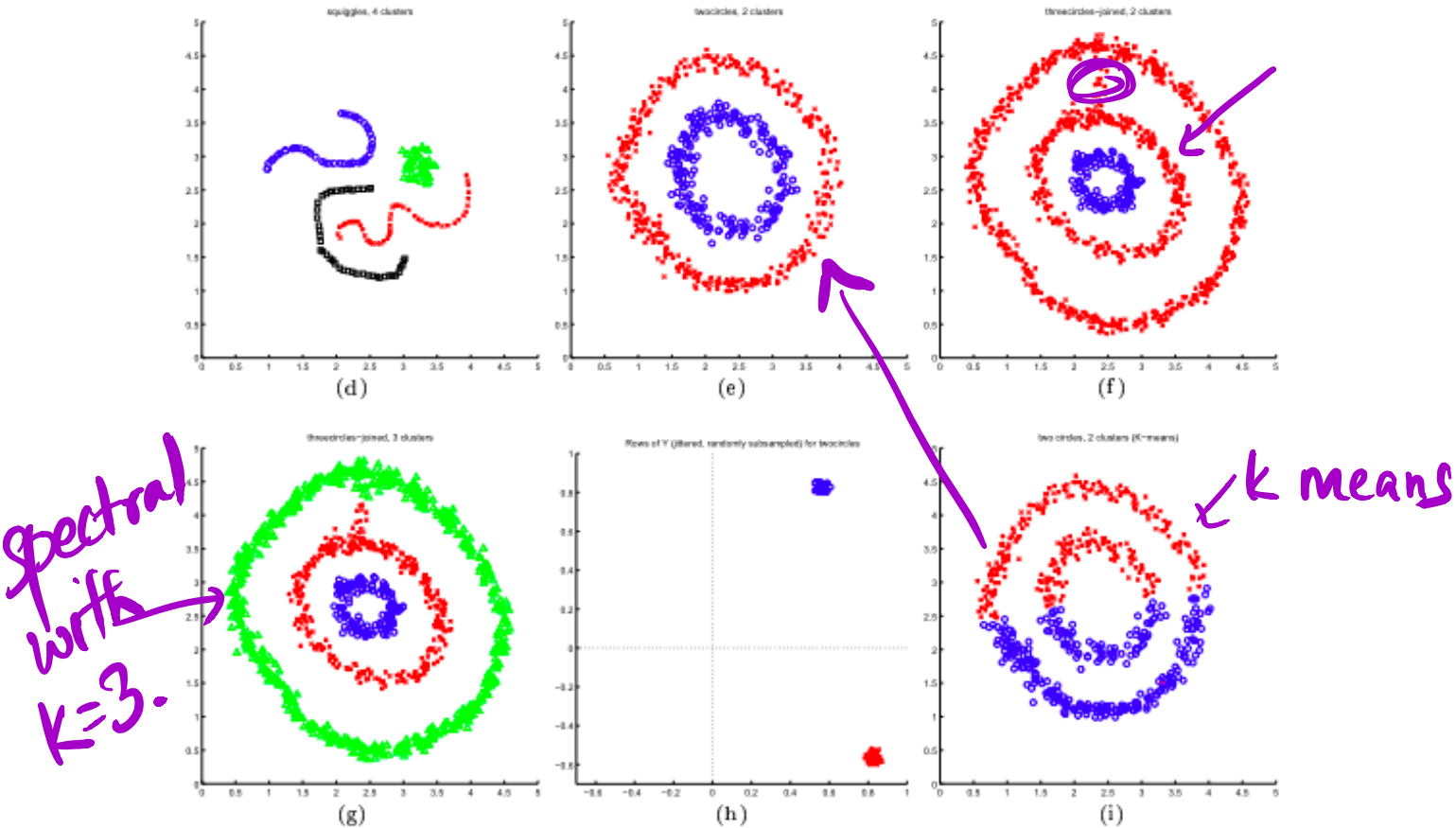
Algorithm

If we want to cluster our data into two clusters we will follow these steps:

- Build a (weighted) graph on the data points (in one of three ways).
- Construct the graph Laplacian matrix, i.e. compute the matrix $D - W$.
- Find its second-smallest eigenvector v_2.
- Threshold its entries to find the clusters, i.e. take $A = \{i : v_2(i) > 0\}$, and $B = \{i : v_2(i) \leq 0\}$.

The second smallest eigenvector of the Laplacian has its own name (Fiedler vector).

Some Examples



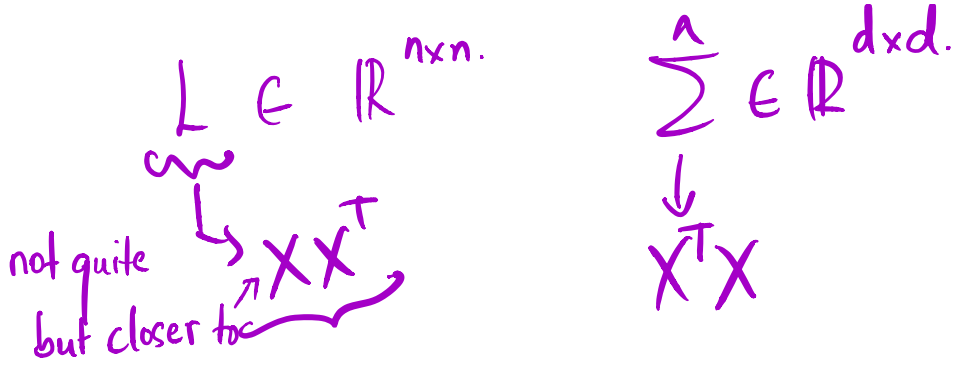
How do we cluster into more than 2 clusters?

Algorithm for clustering into k-clusters

If we want to cluster our data into k-clusters we will follow these steps:

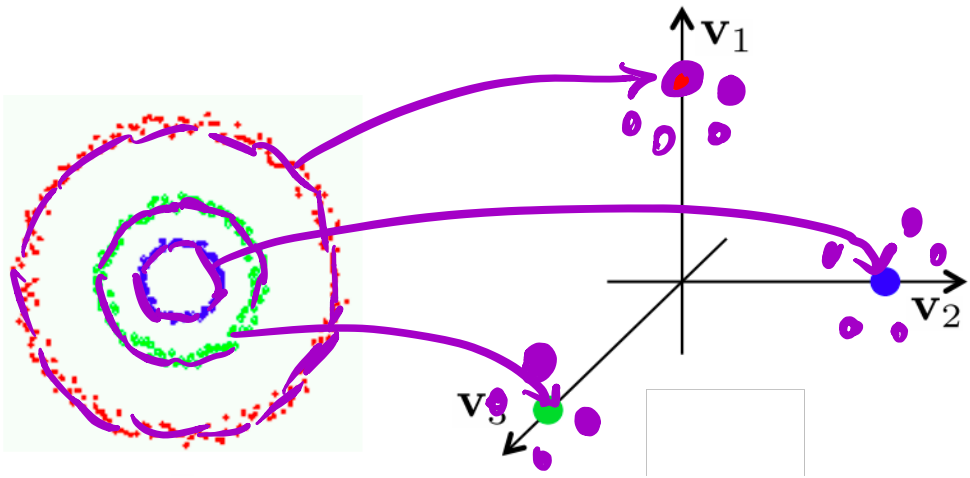
- Build a (weighted) graph on the data points (in one of three ways).
- Construct the graph Laplacian matrix, i.e. compute the matrix $D - W$.
- Find its smallest k eigenvectors $\{v_1, v_2, \ldots, v_k\}$, put them in a matrix $V \in \mathbb{R}^{n \times k}$.
- Interpret the rows of V as our data points. Run k-means on this data to find k-clusters.

Might seem a bit mysterious: Why is this better than running k-means on the original data?


$L \in \mathbb{R}^{n \times n}$.
not quite
but closer to
XX^T
$\hat{\Sigma} \in \mathbb{R}^{d \times d}$.
$X^T X$

The Spectral Embedding

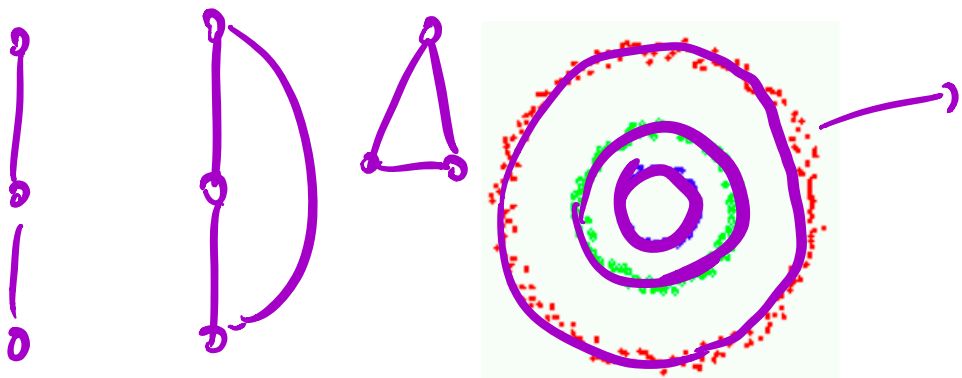
Just some intuition through pictures:



Key point: The spectral embedding (i.e. using the eigenvectors of the Laplacian as the data points) tends to separate clusters very well. k-means on the embedding performs much better than k-means on the original data.

Why does this happen?

Let us first consider a simple case for clustering graphs. Suppose our graph has three separate connected components:



What does the adjacency matrix look like? What about the Laplacian?

$$W = \begin{bmatrix} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} & 0 & 0 \\ 0 & \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} & 0 \\ 0 & 0 & \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \end{bmatrix} \qquad L = D - W = \begin{bmatrix} L_1 & 0 & \\ 0 & L_2 & \\ & & L_3 \end{bmatrix}$$

Clustering disconnected graphs

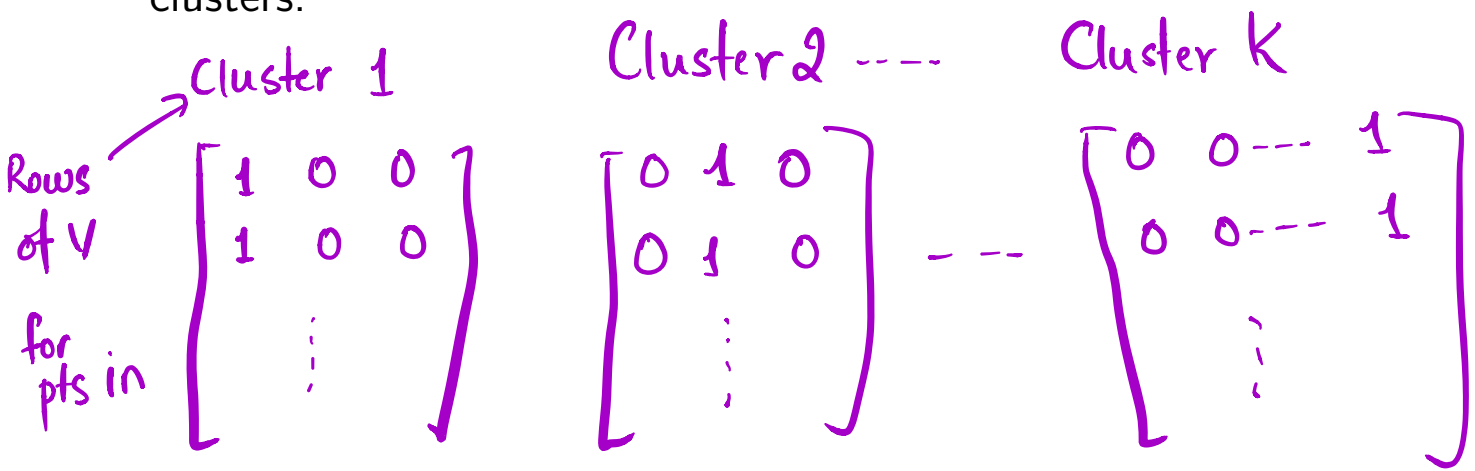
So we have just argued that the Laplacian of our disconnected graph looks like this:

$$L = \begin{bmatrix} L_1 & & 0 \\ & L_2 & \\ 0 & & L_3 \end{bmatrix}$$

What are the bottom k eigenvectors of this matrix?

Clustering disconnected graphs

If our eigenvectors are just indicator vectors for the different clusters:



Then k-means will work well! ←

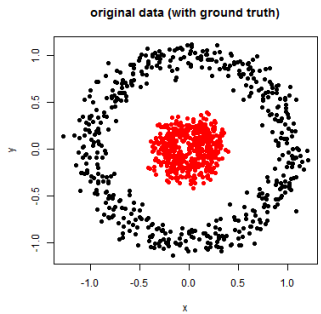
(this is like picture on slide 20).

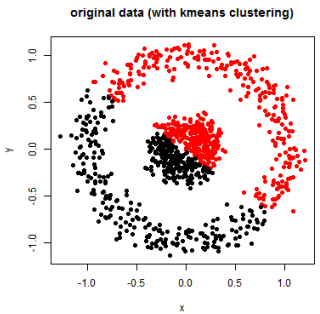
Clustering in more realistic cases

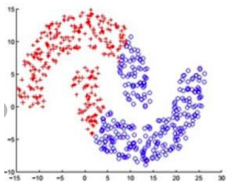
In most real data analysis the graph we build will not neatly separate into the k-clusters that we want.

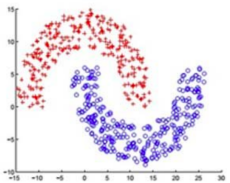
- The eigenvectors of a matrix (under some natural conditions) do not change much when you change the matrix by a small amount.
- So if our graph approximately looks like a k-piece disconnected graph (i.e. few edges between the pieces, and hopefully lots of edges within the pieces) then spectral clustering will work well.

Spectral clustering v/s k-means









- If clusters are (well-separated) blobs: k-means will do well but so will spectral clustering.
- If clusters are dense but have strange shapes then spectral clustering will typically do much better.

Spectral Clustering in Practice

- People often use the *normalized Laplacian* instead of the graph Laplacian.
- The normalized Laplacian divides each entry of the Laplacian by the square root of the degrees of the two corresponding nodes, i.e.:

$$L_{\text{normalized}} = D^{-1/2} L D^{-1/2}.$$

important – will use in HW!

So,

$$L_{\text{normalized}}(i,j) = \frac{L(i,j)}{\sqrt{d_i d_j}}.$$

The normalized Laplacian has many of the same properties as the graph Laplacian but tends to give better clusters in practice.

- The rest of the algorithm is identical (compute the bottom k eigenvectors and run k-means on them).

Spectral Clustering in Practice

- The algorithm now has more tuning parameters than k-means.

1. **The number of clusters k:** In the ideal case, we know we should have k eigenvalues close to 0, and the $k+1$ eigenvalue should be large. So in practice we often look for the first large gap in the eigenvalues. Roughly,

$$k^* = \arg\max_k |\lambda_k - \lambda_{k+1}|.$$

2. **The choice of the similarity graph:** Need to choose between k-NN similarity, ϵ-neighborhood and weighted. In practice, the k-NN graph often works well (and is a good starting point). It also gives us a sparse graph (which is useful computationally).
3. **The choice of k in the k-NN graph:** Again, a very hard question to answer. Often the heuristic is to make sure k is large enough so that the resulting graph has very few disconnected components. (If the graph has many disconnected components then spectral clustering just returns some subset of those components.)

Back to data visualization and PCA

$$X = U \times D \times V^T.$$

A curious fact:

- Recall, that in order to visualize data using PCA, all we needed we the principal component scores (i.e. the things in the matrix $U \times D$). Why?
- Suppose instead of giving you the matrix X (i.e. the data) I only gave you the matrix $X^T X$. Could you still "visualize" the data?
- How about if I gave you the matrix XX^T?

The matrix XX^T

- The matrix XX^T is called the *inner-product matrix*. Why?

- Our curious fact in words: We can visualize the data even without having the original data. Using PCA (or an eigendecomposition) we can go from similarities to a meaningful point cloud.