Unsupervised Statistical Learning: Clustering Graphs

Siva Balakrishnan

Data Mining: 36-462/36-662

April 2nd, 2019

Outline for Today

- Solving HW6, Problem 1 (quickly)
- Recap: Mixture Models
- Representing datasets as graphs
- Clustering graphs
- Spectral clustering

HW6, Problem 1

In class: $v^T \hat{\Sigma} v$ for any unit vector v.

$\hookrightarrow$ = variance along v.

$$\arg\max_{\|v\|_2} v^T \hat{\Sigma} v = v_1$$

$$\cancel{v_1^T \hat{\Sigma} v_1 > v^T \hat{\Sigma} v \text{ for any other dim } v.}$$

$$v_1^T \hat{\Sigma} v_1 = v_1^T \left[\sum_{i=1}^{d} \lambda_i v_i v_i^T \right] v_1$$

$$\hat{\Sigma} = V \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \ddots \end{bmatrix} V^T$$

$$= \sum_{i=1}^{d} \lambda_i (v_i^T v_1)^2$$

$$= \lambda_1.$$

$v_1^T v_1 = 1$

$v_1^T v_j = 0 \quad \forall j \neq 1.$

$$v^T \hat{\Sigma} v$$

what if $v = v_2$? $\quad v_2^T \hat{\Sigma} v_2 = \lambda_2 < \lambda_1$

$\vdots$

$$v = \sum_{i=1}^{d} \alpha_i v_i$$

$$v^T \hat{\Sigma} v = \underbrace{\sum_{i=1}^{d} \alpha_i^2 \lambda_i}_{\leq \lambda_1 \times \underbrace{\sum_{i=1}^{d} \alpha_i^2}_{1 \text{ since } v \text{ is a unit vector}}}.$$

$$\leq \lambda_1.$$

Recap: Mixture Models Motivation

- We wanted to fix two significant problems with K-means clustering:
  - It is a "hard" clustering method, i.e. each point gets assigned to a single cluster and so deals badly with overlapping clusters.
  - It can also do poorly in cases where the clusters have non-spherical shapes.

- **Bonus:** Perhaps incorporate a bit more "statistical modeling" into clustering.

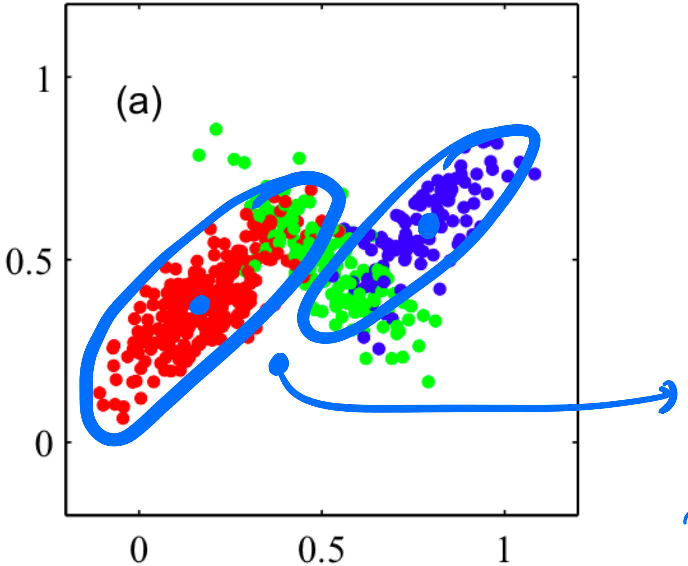
Recap: Mixture Models

- Want to roughly imagine the case, where each cluster has a different distribution.
- The generative model we are imagining is:
  - We first choose a cluster by drawing $Z \sim \{1, \ldots, K\}$.
  - We then draw a sample from the distribution corresponding to cluster Z.

  However, we are not shown the Z values (the cluster labels).
- This is called a *mixture model*:

$$f(x) = \sum_{k=1}^{K} \mathbb{P}(Z = k) p(x | Z = k) = \sum_{k=1}^{K} \lambda_k f_Z(x).$$

Recap: Gaussian Mixture Models



prob blue
prob red
prob green.

weights.

- Model: $\lambda_1, \lambda_2, \lambda_3$ and

$$f_{\text{red}}(x) = N(\mu_{\text{red}}, \Sigma_{\text{red}}),$$
$$f_{\text{blue}}(x) = N(\mu_{\text{blue}}, \Sigma_{\text{blue}}),$$
$$f_{\text{green}}(x) = N(\mu_{\text{green}}, \Sigma_{\text{green}}),$$

Recap: Clustering with a Mixture Model

"soft" clustering.

- Suppose someone handed us a mixture model. How would we "soft" cluster our data?
- For a point x we would compute for $i \in \{1, \ldots, K\}$:

$$P(Z = i | X = x) = \frac{\lambda_i f_i(x)}{\sum_{j=1} \lambda_j f_j(x)}.$$

just use Bayes rule.

Main question: given data how do we estimate the mixture parameters?

Recap: Estimating a Mixture Model – Expectation-Maximization

- EM is a general method for (approximately) maximizing the (marginal) likelihood when you have missing data. We won't get too much into the details but describe the EM algorithm for GMMs directly.

- Roughly, we want to first "guess" the latent variables Z_i and then if we knew those we could just maximize the (usual/complete) likelihood.
- It resembles k-means. Except instead of assigning each point to a single cluster we "softly" assign them so they contribute fractionally to each cluster.

Recap: Estimating a Mixture Model – Expectation-Maximization

- We initialize the parameters $(\lambda_k, \mu_k, \Sigma_k)_{k=1}^K$ randomly, and then alternate the following two steps:
  1. **E-step:** We compute the cluster memberships for each point i:

     soft assignments of pts to clusters.

     $$P(Z_i = k|X_i) = \frac{\lambda_k \phi_k(X_i; \mu_k, \Sigma_k)}{\sum_{j=1}^K \lambda_j \phi_j(X_i; \mu_j, \Sigma_j)}$$

     as before.
  2. **M-step:** Recompute the parameters:

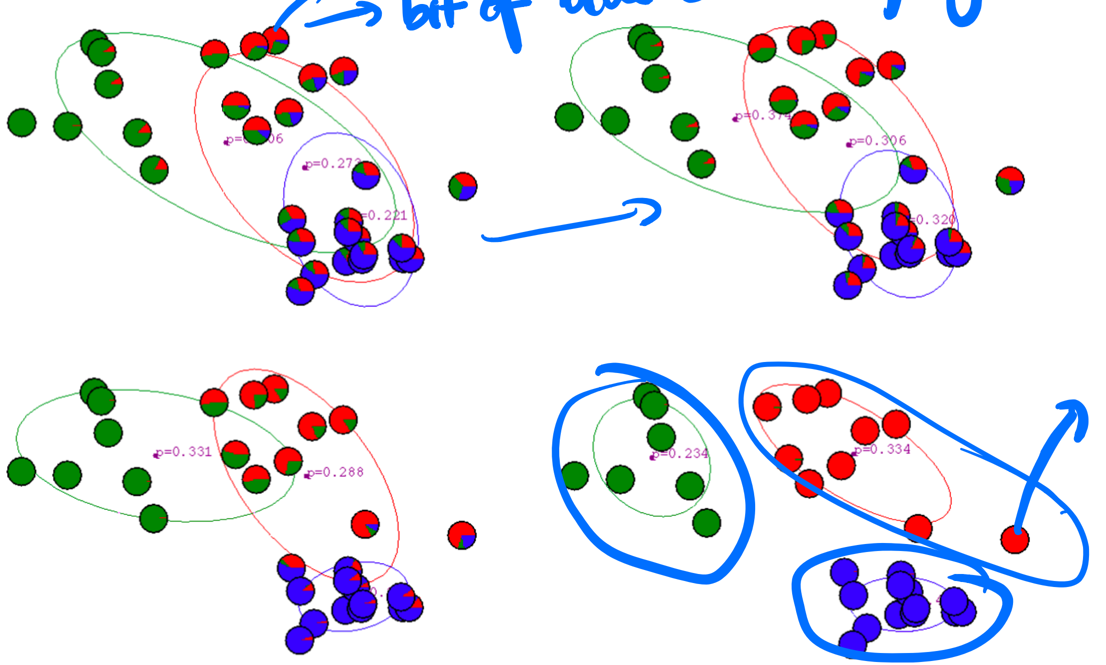
     $$\lambda_k = \frac{\sum_{i=1}^n P(Z_i = k|X_i)}{n},$$

     $$\mu_k = \frac{\sum_{i=1}^n P(Z_i = k|X_i) X_i}{\sum_{i=1}^n P(Z_i = k|X_i)},$$

     and similarly update the covariance matrix.

Recap: Estimating a Mixture Model – EM in Action

mostly red
bit of blue & bit of green.



Graphs and Weighted Graphs

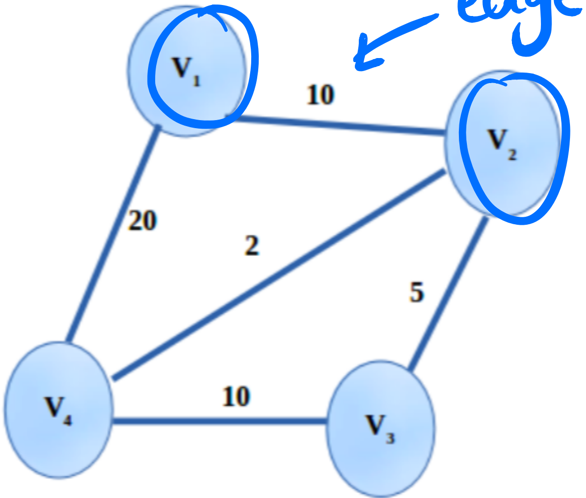
It is often convenient and useful to think about data in terms of graphs.

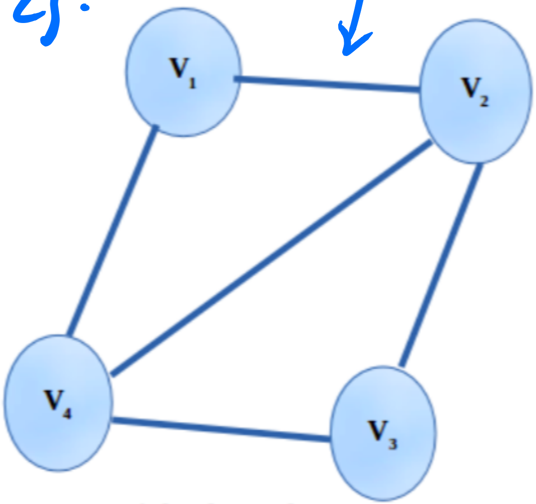
- **(Unweighted) Graphs:** Just vertices and edges. Equivalent to every edge having weight 1.
- **Weighted Graphs:** Each edge, say between vertices i and j, has weight w_{ij}. $\longrightarrow$ n - vertices.

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

edge $\{1,2\}$.

all wts are 1.





Graphs from Data

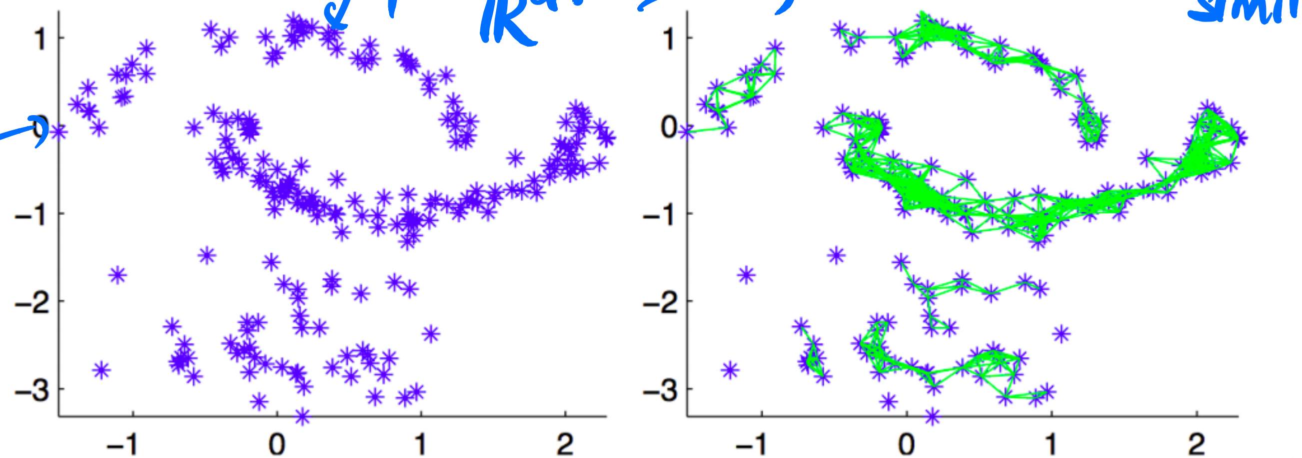
We are given our usual collection of data points $\{X_1, \ldots, X_n\}$.
How do we build a graph from these?
Roughly:

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

✓ pts in $\mathbb{R}^d$.

connect pts that are "similar".

data



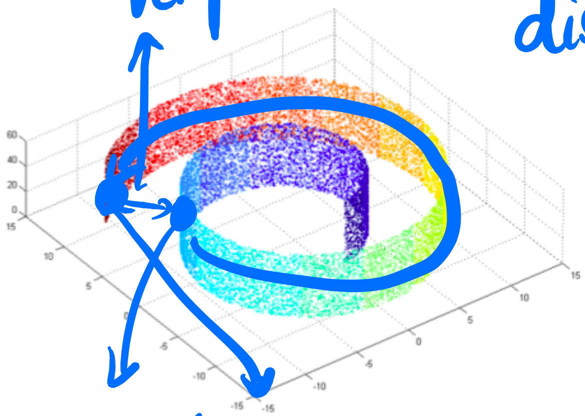
Data Surfaces and Similarities

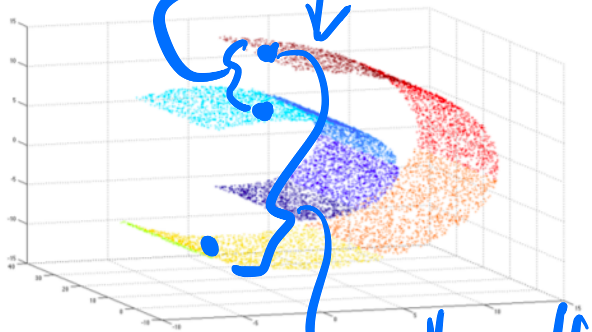
Why are we building graphs? Two answers:

1. Gives us a new way to think about data, and come up with algorithms (we'll see a few examples).
2. We don't trust the Euclidean distance. We want the geometry of our data to inform our notion of similarity.

very close in Euclidean dist.

are they more similar



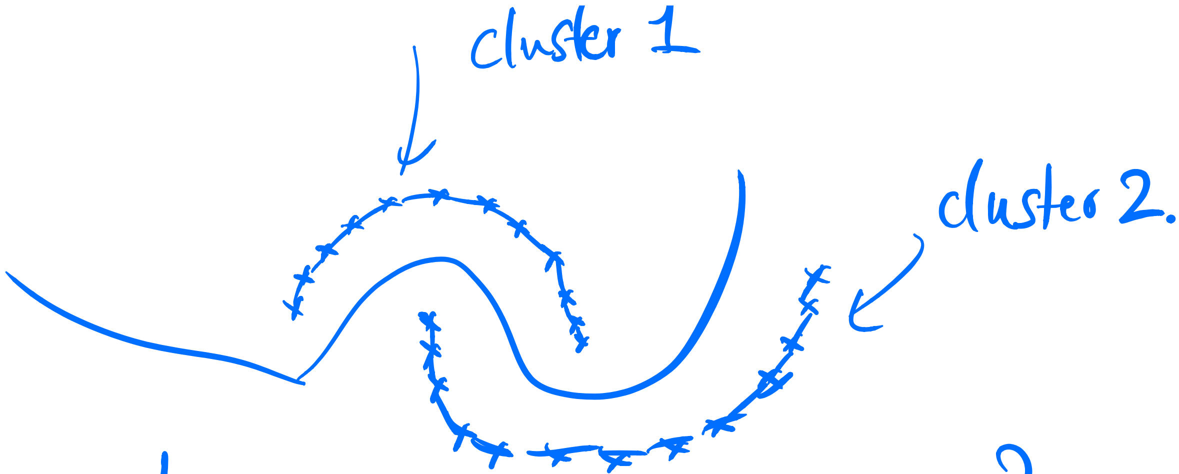


than these two?

$7 \to f \to 1$

How similar are points?

→ close in pixels.



↳ Euclidean dist not too meaningful.

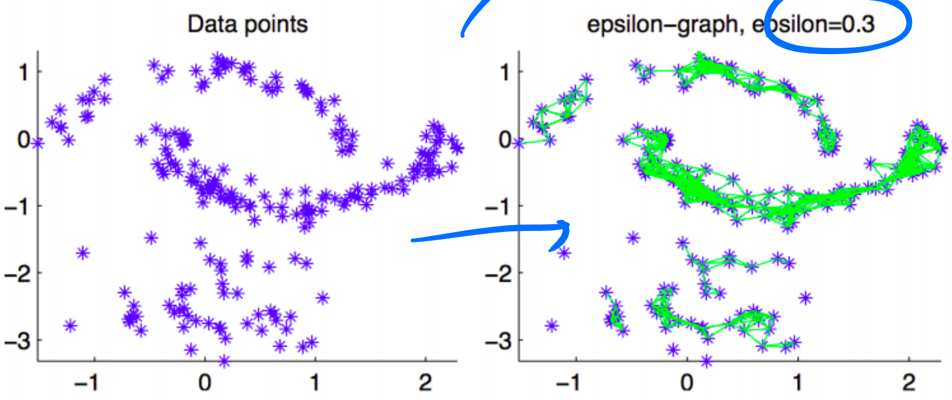
How do we build graphs?

Three canonical ways:

tuning parameter.

- **ϵ-neighborhood graph:** Connect every pair of points (i, j) for which $\|X_i - X_j\|_2 \le \epsilon$.

connect two pts if $\|X_i - X_j\| \le 0.3$.

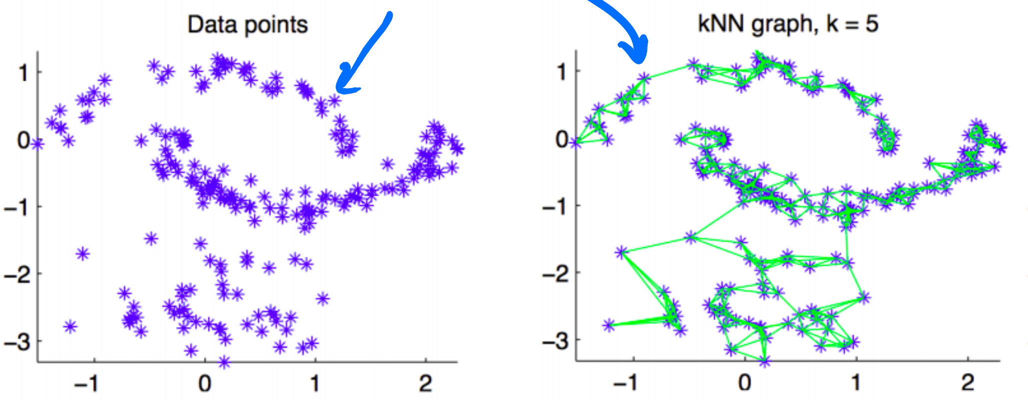


How do we build graphs?

tuning parameter.

Three canonical ways:

- k-**nearest neighbor graph:** Connect (i, j) if either X_i is one of X_j's k-nearest neighbors or if X_j is one of X_i's k-nearest neighbors.



How do we build graphs?

Three canonical ways:

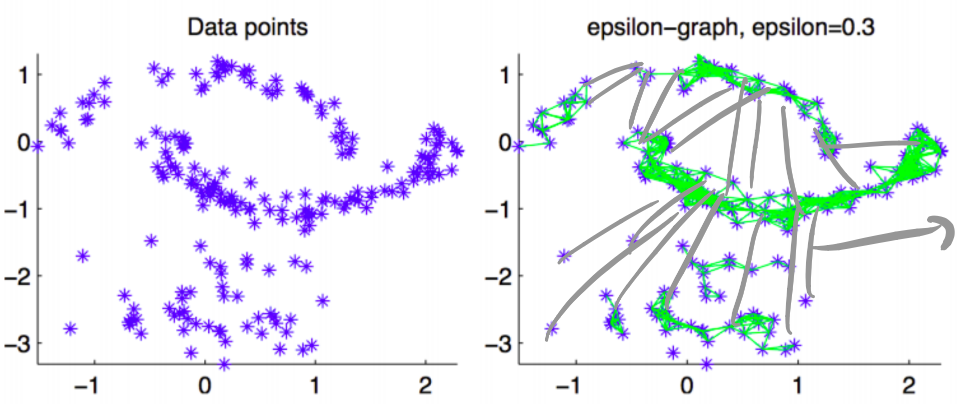
- **Weighted Euclidean Distance graph:** Connect (i, j) with weight:

far away
ess. wt = 0

$$w_{ij} = \exp(-\|X_i - X_j\|_2^2/\sigma^2),$$

tuning parameter

for some bandwidth σ.



very light
↳ low wt.

Back to Clustering

Now that we have built a graph from our data – we can solve many statistical learning problems (classification, regression, clustering) using the graph. Suppose we wanted to cluster our data (for now, into 2 clusters).

- We want to *partition* our graph into two pieces.
- Hopefully, cut as few edges as possible (or minimize the weight of the edges we cut).

Formally, if we have a graph G, we partition the vertices into two sets A and B. The cost of the partition is:

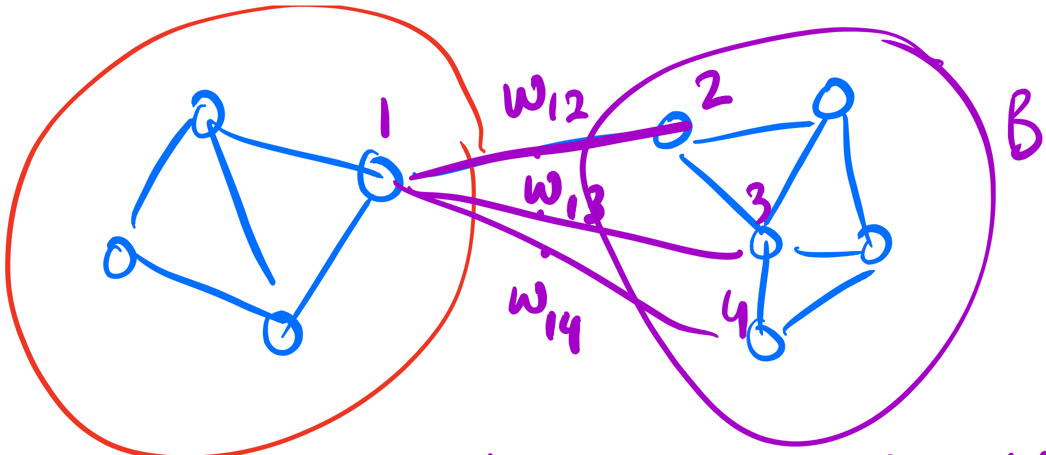
G → n vertices

A B.

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

↓ wt from A to B.

For an unweighted graph, the cost is just the number of edges we cut. Just like in k-means - we can try to find the best partition, i.e. the one that cuts the fewest edges.



$$\text{price} = w_{12} + w_{13} + w_{14}.$$

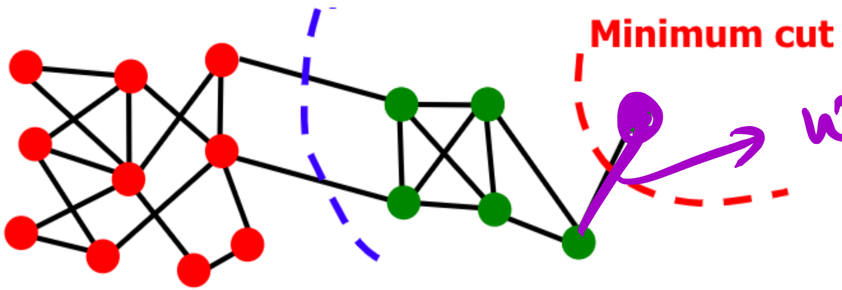
Graph Partitioning

Find best cut:

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

Good news: There is a fast algorithm that solves this problem and finds the best cut.
Bad news: Usually does terribly in practice. Often just splits off "whiskers".

want to look for balanced cut.



whisker

Want to encourage our clustering algorithm to find *big clusters*.

Balanced Partitions

One alternative is to try to find a balanced cut, i.e.:

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

- You can also imagine variants where you force both clusters to be big (but not necessarily half the vertices) and so on.
- Turns out that this problem is difficult to solve computationally.
- *Spectral clustering* algorithms will give us a way to approximately solve such "balanced partitioning" problems.

Spectral clustering methods are basically "eigenvector-based" methods for clustering. How do cuts and eigenvectors relate?

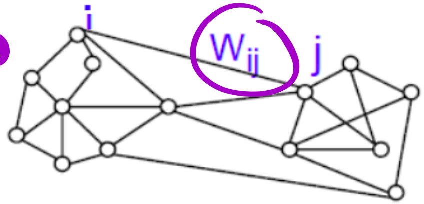
Graphs as Matrices

if unweighted, =1 if connected, 0 o/w.

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

$W \to n$ times n matrix.

W_{ij} = edge wt b/w pts i & j.

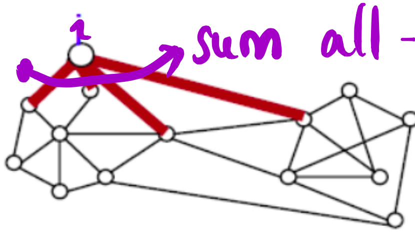


$$\begin{array}{cc} & \begin{array}{ccc} & & j \end{array} \\ i & = \begin{bmatrix} 1 & & \\ 0 & 0 & w_{ij} \\ & 0 & \end{bmatrix} \end{array}$$

- **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

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

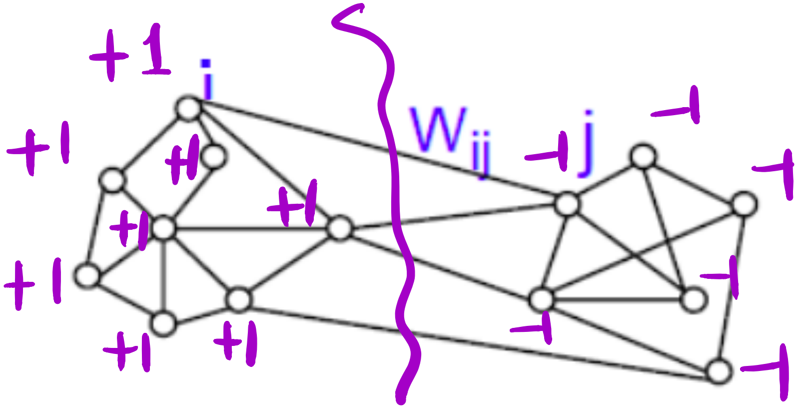
$$D_{ii} = \sum_{j=1}^{n} W_{ij}.$$



sum all these weights.

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.



$v \in \mathbb{R}^n$.
↳ one entry for every node in G.
+1 if node $\in A$
-1 if node $\in B$.

Our goal for the next few slides is to understand the following relations:

✓ ✓

$$\text{cut}(A, B) = \sum_{i\in A, j\in B} W_{ij} = \frac{1}{8}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}(v_{AB}(i) - v_{AB}(j))^2$$

:-(

$$= \frac{1}{4} v_{AB}^T (D - W) v_{AB}.$$

The second equality:

Suppose i & j both in A.

$v_{AB}(i) = v_{AB}(j) = +1.$

0

if i & j in B same thing.

if $i \in A$, $j \in B$.

W_{ij}

$v_{AB}(i) = +1$

$v_{AB}(j) = -1$

$\cancel{8}\, W_{ij}$

Any cut : (A, B).

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

$\downarrow$

find good cut

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

8

Cuts and Matrices

The other equality is a bit more difficult, but just algebra (we are going to skip this). For any vector v:

$$
\begin{aligned}
v^T(D-W)v = v^T D v - v^T W v &= \sum_{i=1}^{n} v(i)^2 d_{ii} - \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} v(i) v(j) \\
&= \sum_{i=1}^{n} v(i)^2 \sum_{j=1}^{n} W_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} v(i) v(j) \\
&= \frac{1}{2}\Bigg[\sum_{i=1}^{n}\sum_{j=1}^{n} v(i)^2 W_{ij} - 2\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} v(i) v(j) \\
&\qquad + \sum_{i=1}^{n}\sum_{j=1}^{n} v(j)^2 W_{ij}\Bigg] \\
&= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}(v(i)-v(j))^2.
\end{aligned}
$$

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

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

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...).

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.

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:

$$Lv =$$

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

$$v_j \times \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \qquad .$$

So every other eigenvector is "balanced".

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.

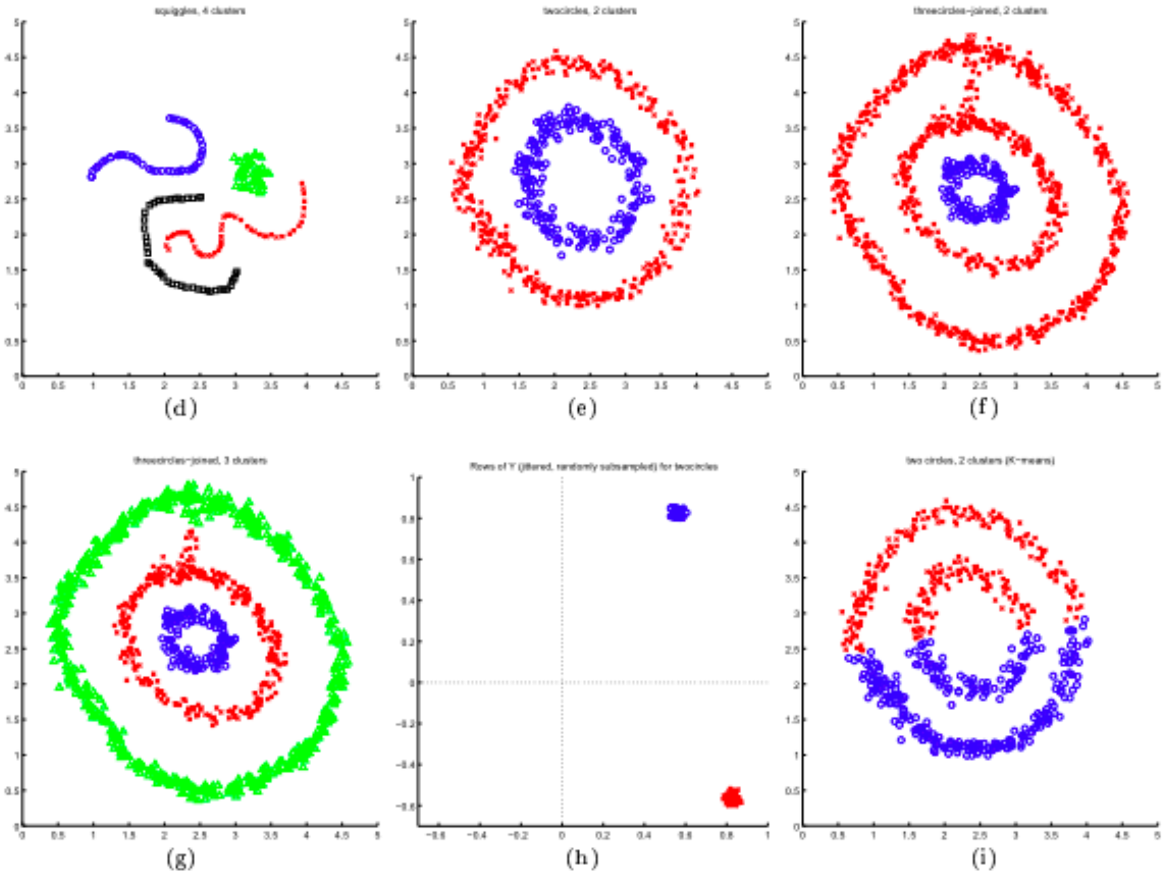
And a solution:

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 $A = \{i : v(i) \geq 0\}$, and $B = \{i : v(i) < 0\}$.

Some Examples



How do we cluster into more than 2 clusters?