Spectral Clustering

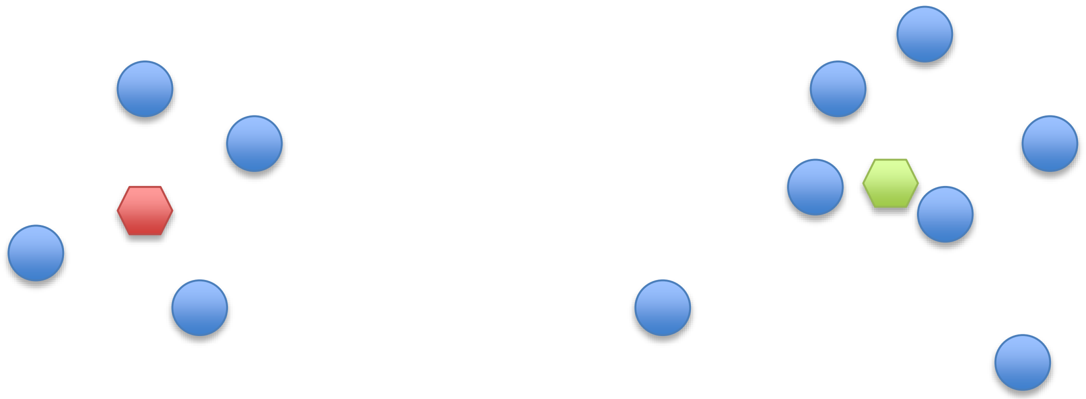
Based on slides

(with thanks to Shannon Quinn,

William Cohen of Carnegie Mellon University, and J. Leskovec, A. Rajaraman, and J. Ullman of Stanford University)

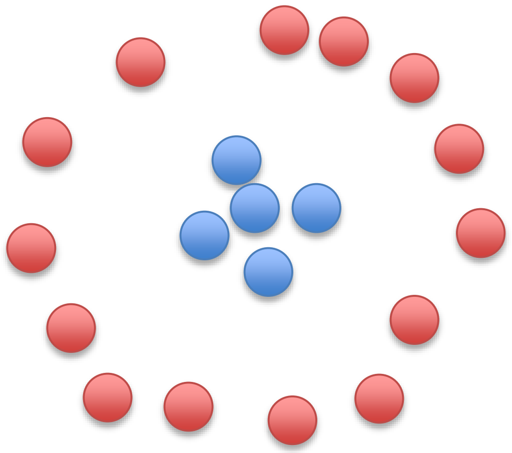
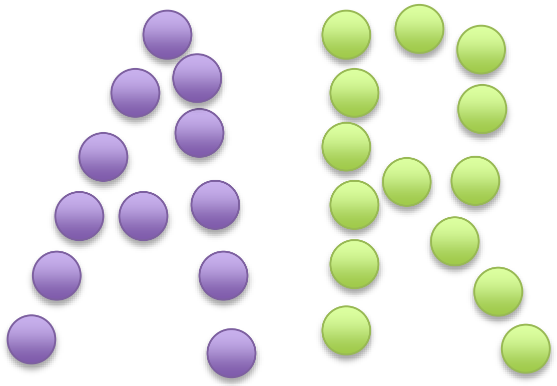
Partitional Clustering

- How do we partition a space to make the best clusters?
- Proximity to a cluster centroid.



Difficult Clusterings

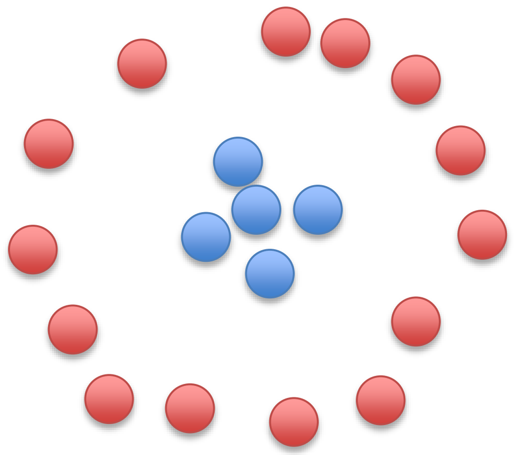
- But some clusterings don’t lend themselves to a “centroid” based definition of a cluster.

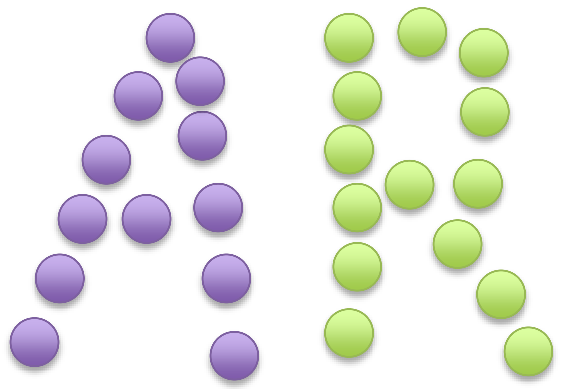



- Spectral clustering allows us to address these sorts of clusters.

Difficult Clusterings

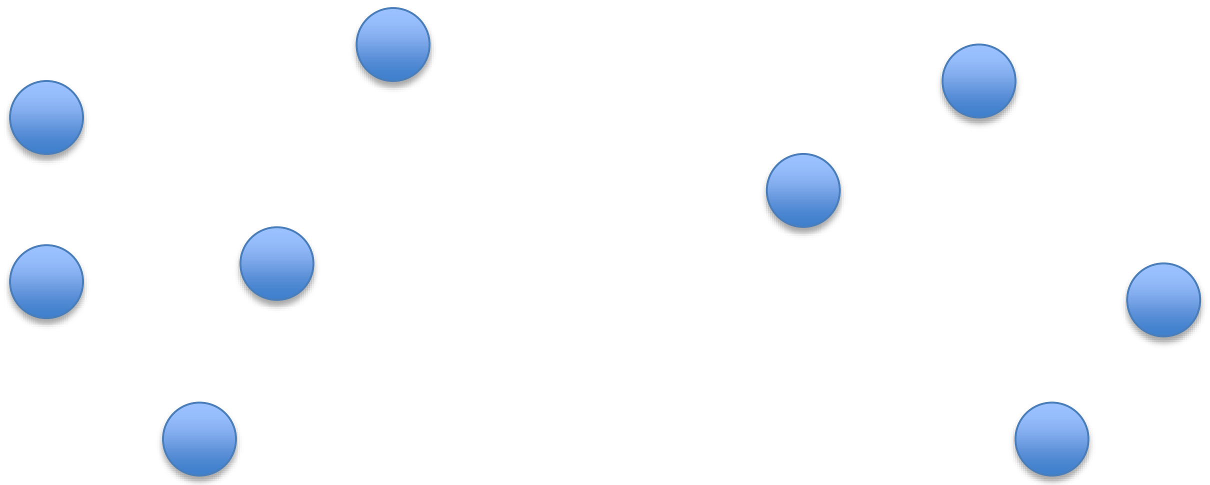
- These kinds of clusters are defined by points that are close **any** member in the cluster, rather than the **average** member of the cluster.





Graph Representation

- We can represent the relationships between data points in a graph.

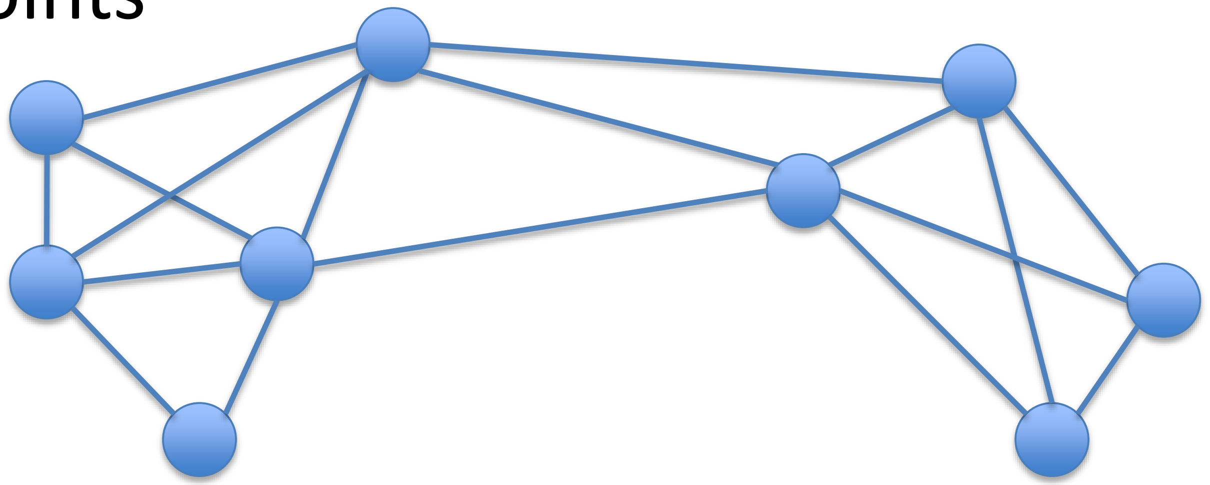


Graph definitions

- ε-neighborhood graph
  - Identify a threshold value, ε, and include edges if the affinity between two points is greater than ε.
- k-nearest neighbors
  - Insert edges between a node and its k-nearest neighbors.
  - Each node will be connected to (at least) k nodes.
- Fully connected
  - Insert an edge between every pair of nodes.

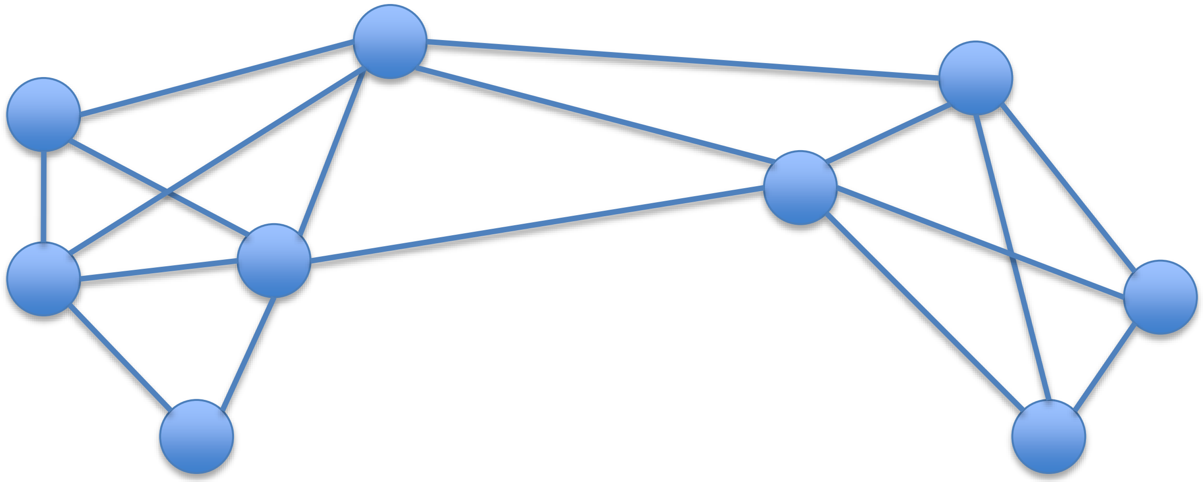
Graph Representation

- We can represent the relationships between data points in a graph.
- Weight the edges by the similarity between points



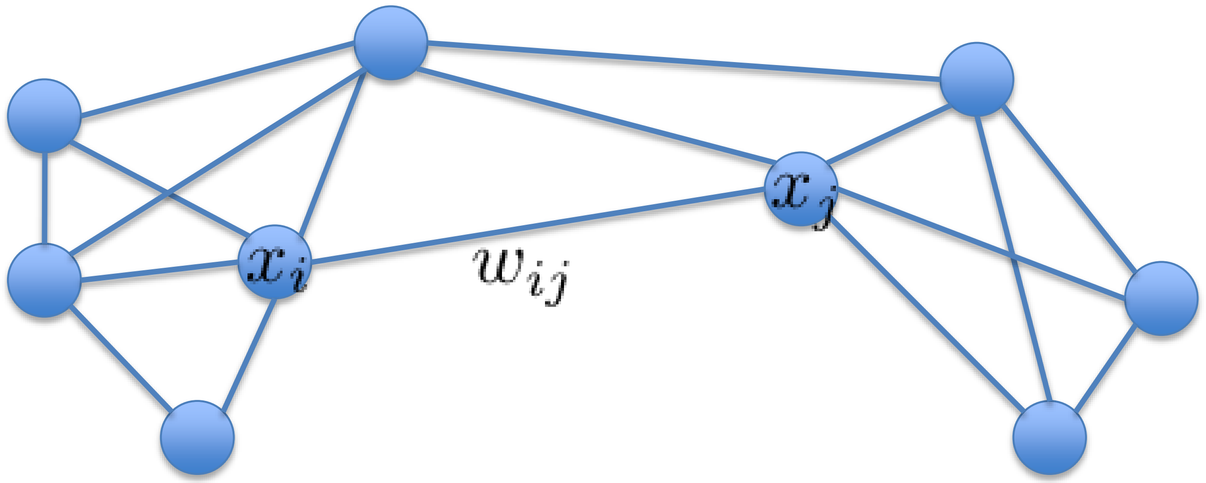
Representing data in a graph

- What is the best way to calculate similarity between two data points?
- Distance based: $d(x_i, x_j) = \exp\left\{\frac{\|x_i - x_j\|}{\sigma^2}\right\}$



Graphs

- Nodes and Edges
- Edges can be directed or undirected
- Edges can have weights associated with them



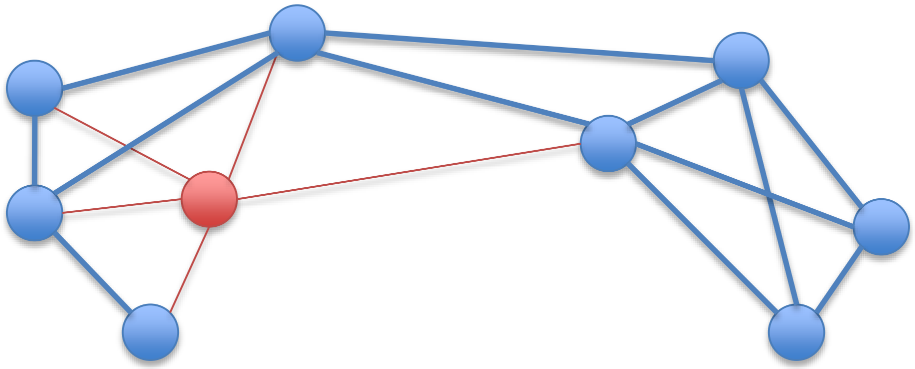
- Here the weights correspond to **pairwise affinity**

$$w_{ij} = d(x_i, x_j) = \exp\left\{\frac{\|x_i - x_j\|}{\sigma^2}\right\}$$

Graphs

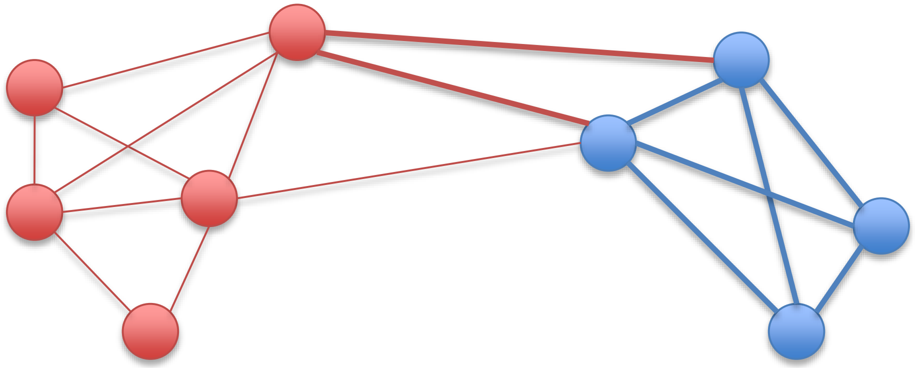
- Degree

$$D(x_i) = \sum_{j \in V} w_{ij}$$



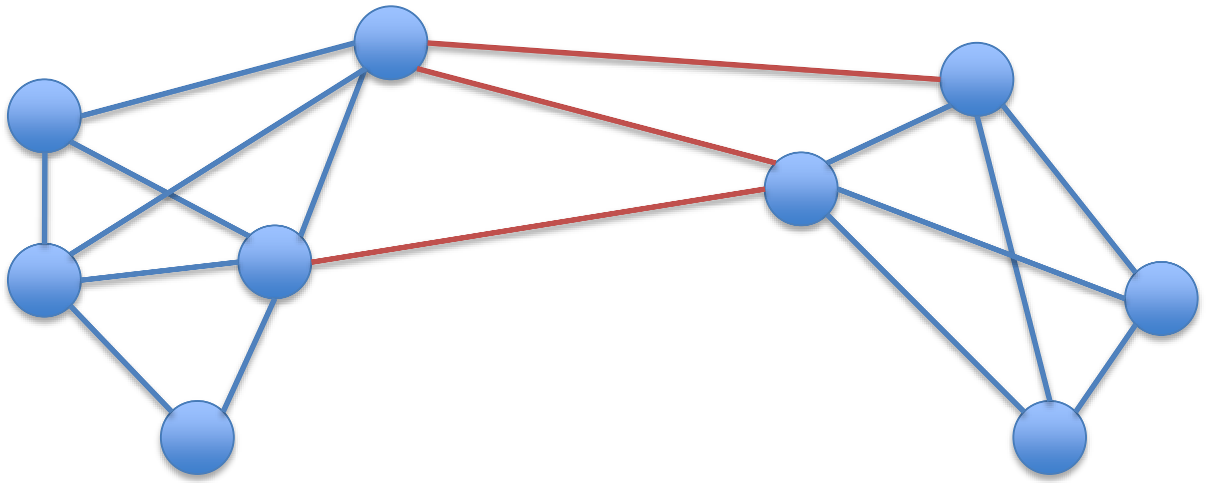
- Volume of a set

$$Vol(C) = \sum_{i \in C} D(x_i)$$



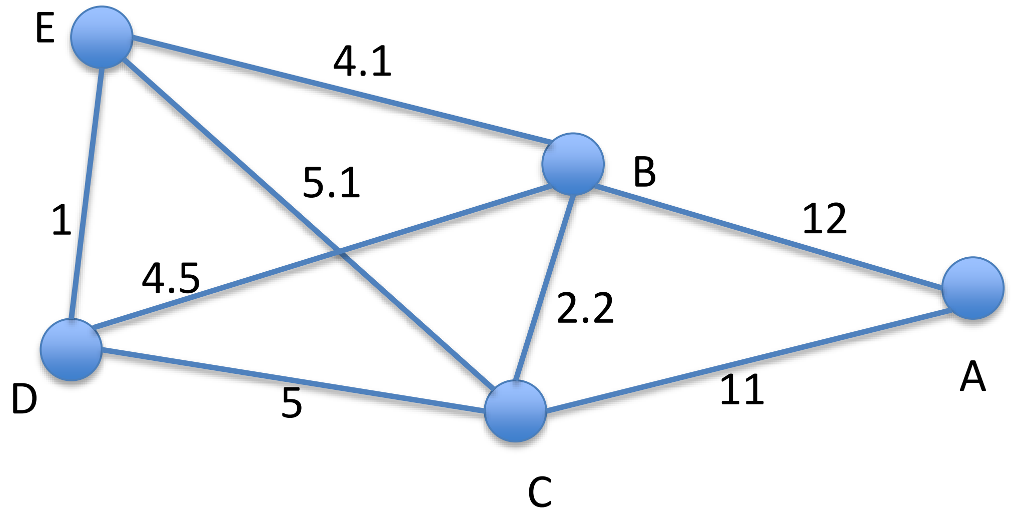
Graph Cuts

- The **cut** between two subgraphs is calculated as follows



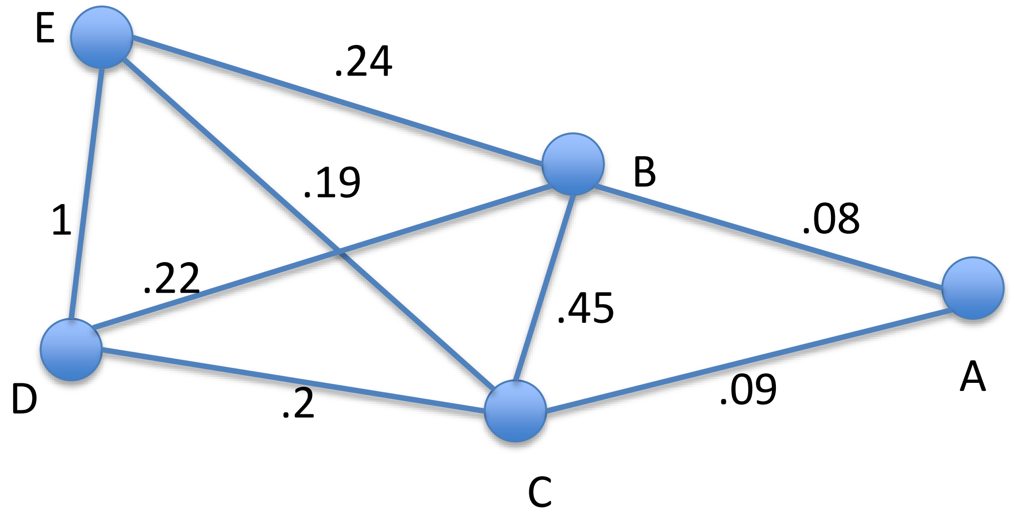
$$Cut(C_1, C_2) = \sum_{i \in C_1} \sum_{j \in C_2} w_{ij}$$

Graph Examples - Distance



Height	Weight
20	5
8	6
9	4
4	4
4	5

Graph Examples - Similarity



Height	Weight
20	5
8	6
9	4
4	4
4	5

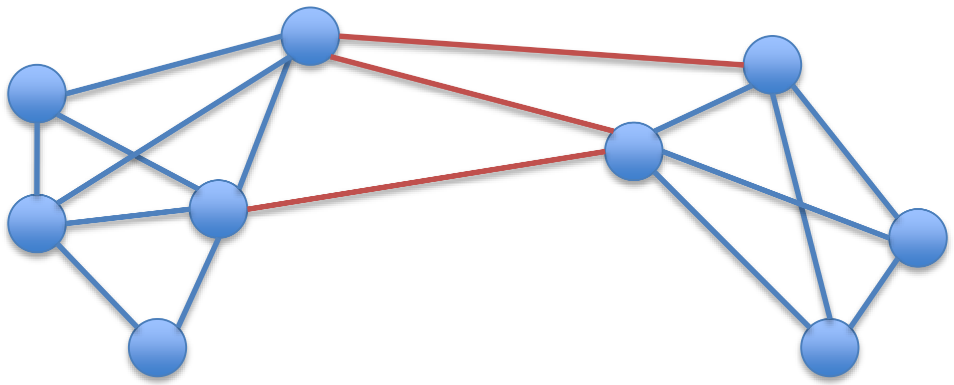
Intuition

- The minimum cut of a graph identifies an optimal partitioning of the data.
- Spectral Clustering
  - Recursively partition the data set
    - Identify the minimum cut
    - Remove edges
    - Repeat until k clusters are identified

Graph Cuts

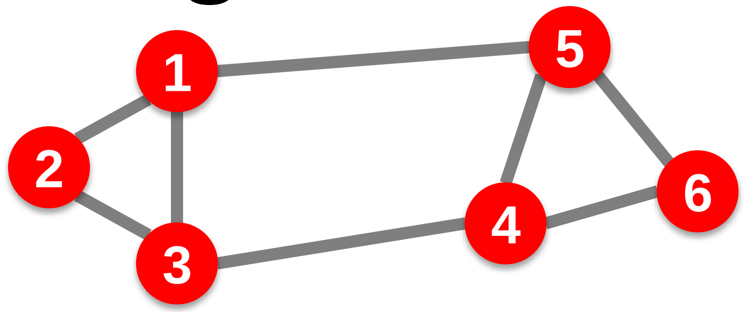
- Minimum (bipartitional) cut

$$\min Cut(C_1, C_2) = \sum_{i \in C_1} \sum_{j \in C_2} w_{ij}$$

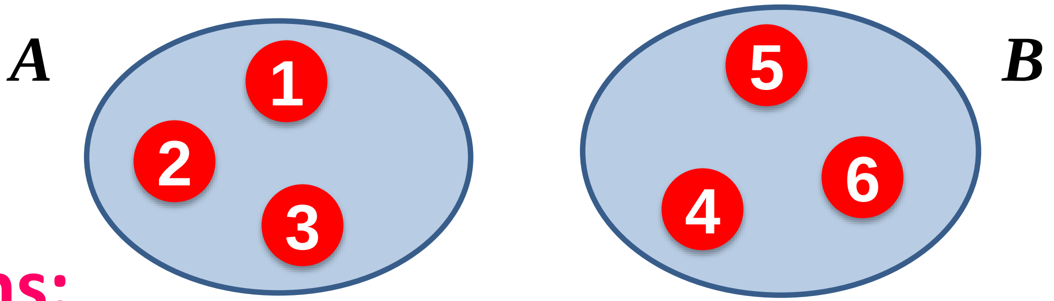


Graph Partitioning

- **Undirected graph**



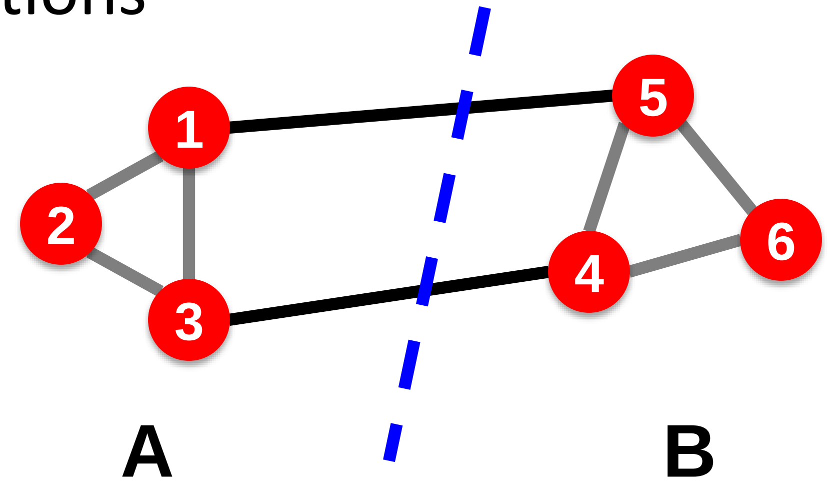
- **Bi-partitioning task:**
  - Divide vertices into two disjoint groups



- **Questions:**
  - How can we define a "good" partition of ?
  - How can we efficiently identify such a partition?

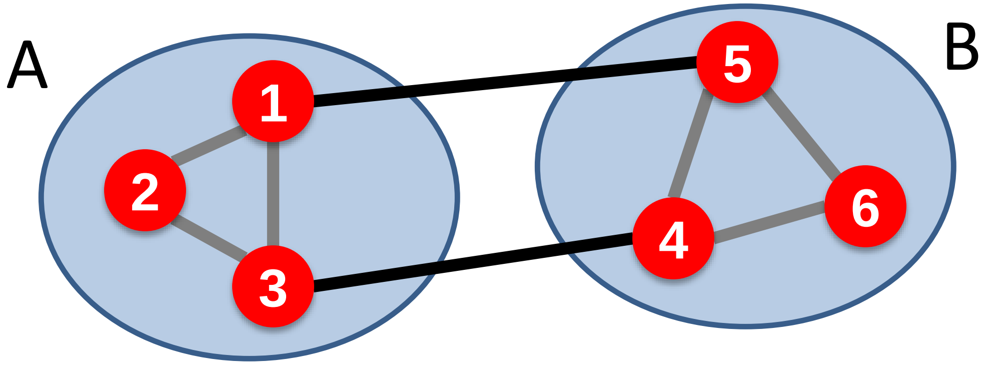
Graph Partitioning

- **What makes a good partition?**
  - Maximize the number of within-group connections
  - Minimize the number of between-group connections



Graph Cuts

- **Express partitioning objectives as a function of the "edge cut" of the partition**
- **Cut:** Set of edges with only one vertex in a group: $cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$



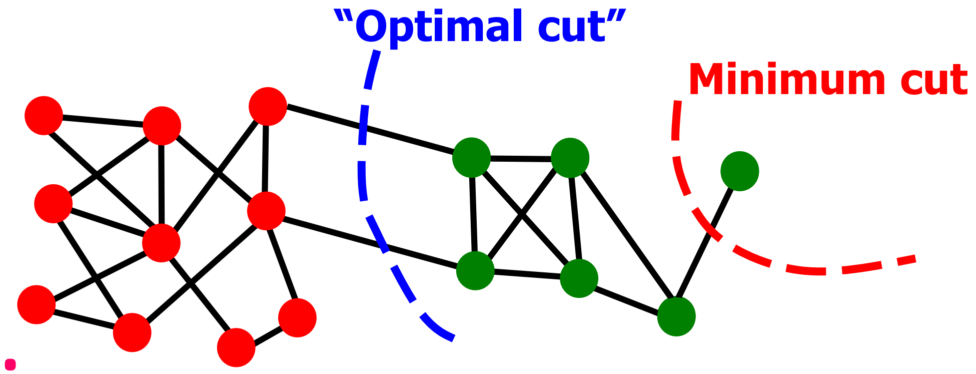
$cut(A,B) = 2$

Graph Cut Criterion

- **Criterion: Minimum-cut**
  - Minimize weight of connections between groups

$$\arg\min_{A,B} cut(A,B)$$

- **Degenerate case:**

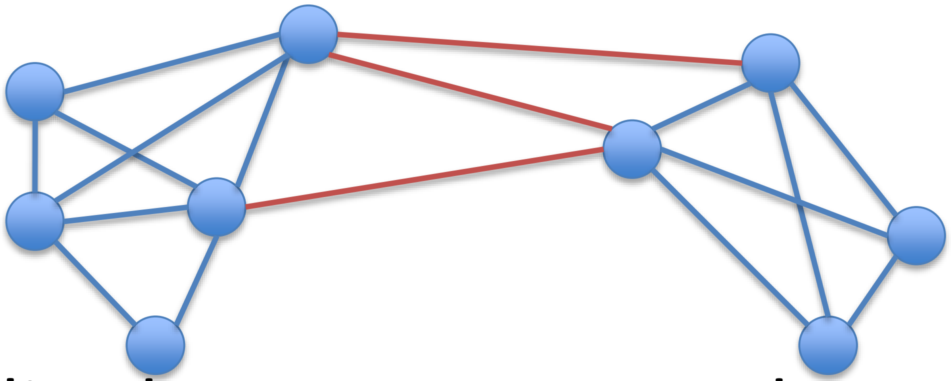


- **Problem:**
  - Only considers external cluster connections
  - Does not consider internal cluster connectivity

Graph Cuts

- Minimal (bipartitional) normalized cut.

$$\min \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)} = \min \left(\frac{1}{Vol(C_1)} + \frac{1}{Vol(C_2)} \right) Cut(C_1, C_2)$$



- Unnormalized cuts are attracted to outliers.

Graph Cut Criteria

- **Criterion:** **Normalized-cut** [Shi-Malik, '97]
  - Connectivity between groups relative to the density of each group

$$ncut(A,B) = \frac{cut(A,B)}{vol(A)} + \frac{cut(A,B)}{vol(B)}$$

Vol(A) : total weight of the edges with at least one endpoint in :

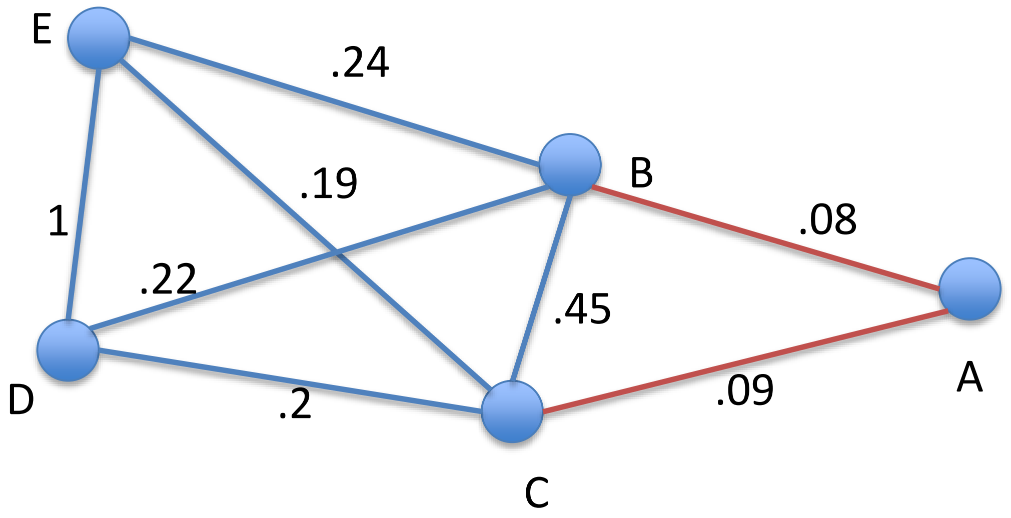
- **Why use this criterion?**
  - Produces more balanced partitions
- **How do we efficiently find a good partition?**
  - **Problem:** Computing optimal cut is NP-hard

$\boldsymbol{vol(A)}$: total weighted degree of the nodes in $\boldsymbol{A}$:

$\boldsymbol{vol(A) = \sum_{i \in A} k_i}$ (number of edge end points in $\boldsymbol{A}$)

Spectral Clustering Example

- Minimum Cut



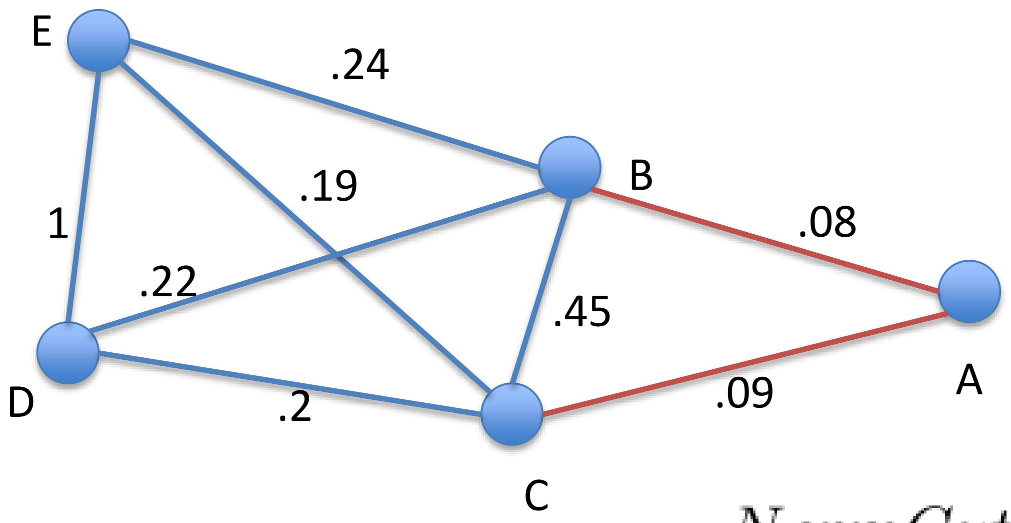
Height	Weight
20	5
8	6
9	4
4	4
4	5

$$Cut(BCDE, A) = 0.17$$

Spectral Clustering Example

- Normalized Minimum Cut

$$NormCut(C_1, C_2) = \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)}$$



Height	Weight
20	5
8	6
9	4
4	4
4	5

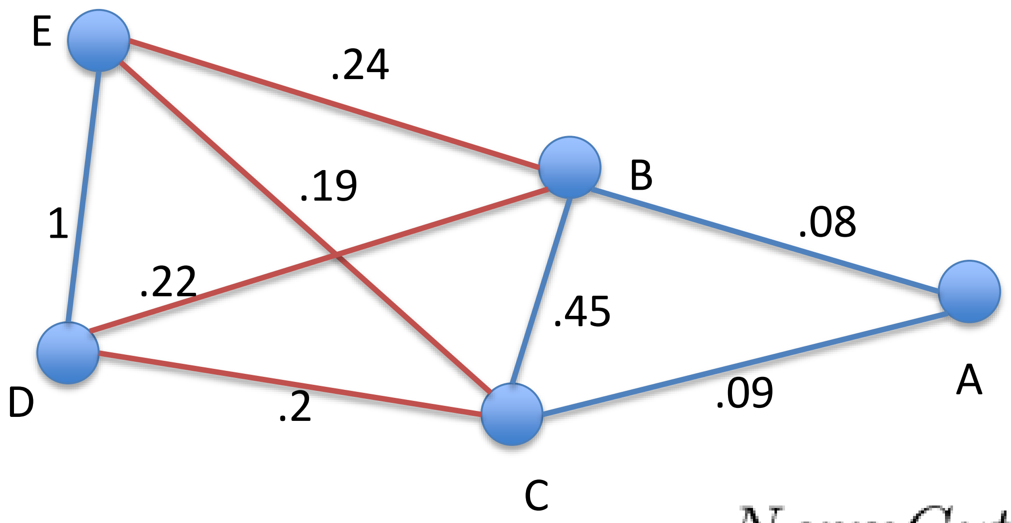
$$NormCut(BCDE, A) = 1.067$$

Spectral Clustering Example

- Normalized Minimum Cut

$$NormCut(C_1, C_2) = \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)}$$

Height	Weight
20	5
8	6
9	4
4	4
4	5



$$NormCut(BCDE, A) = 1.067$$

$$NormCut(ABC, DE) = 1.038$$

Problem

- Identifying a minimum cut is NP-hard.
- There are efficient approximations using linear algebra.
- Based on the Laplacian Matrix, or **graph Laplacian**

Spectral Graph Partitioning

- $\boldsymbol{A}$: adjacency matrix of undirected **G**
  - $\boldsymbol{A}_{ij}$ **=1** if is an edge, else **0**
- $\boldsymbol{x}$ is a vector in $\Re^n$ with components
  - Think of it as a label/value of each node of
- **What is the meaning of $A \cdot x$?**

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$y_i = \sum_{j=1}^{n} A_{ij} xj = \sum_{(i,j)\in E} x_j$$

- **Entry y_i is a sum of labels x_j of neighbors of i**

What is the meaning of *Ax*?

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- **j^{th} coordinate of $A \cdot x$:**
  - Sum of the x-values of neighbors of j
  - Make this a new value at node j

$$\boldsymbol{A \cdot x = \lambda \cdot x}$$

- **Spectral Graph Theory:**
  - Analyze the "spectrum" of matrix representing
  - **Spectrum:** Eigenvectors of a graph, ordered by the magnitude (strength) of their corresponding eigenvalues λi :

$$\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_n\}$$

$$\lambda_1 \le \lambda_2 \le \dots \le \lambda_n$$

Note : we sort Lambda's in ascending (Not Descending) Order

Example | d- Regular Graph

- Suppose all nodes in $\boldsymbol{G}$ have degree $\boldsymbol{d}$ ($\boldsymbol{G}$ is $\boldsymbol{d}$-regular) and $\boldsymbol{G}$ is connected
- **What are some eigenvalues/vectors of $\boldsymbol{G}$?**

  $\boldsymbol{A}\cdot \boldsymbol{x} = \boldsymbol{\lambda} \cdot \boldsymbol{x}$ What is $\boldsymbol{\lambda}$? What $\boldsymbol{x}$?

  - **Let's try:** $\boldsymbol{x} = (\mathbf{1}, \mathbf{1}, \dots, \mathbf{1})$
  - **Then:** $\boldsymbol{A} \cdot \boldsymbol{x} = (\boldsymbol{d}, \boldsymbol{d}, \dots, \boldsymbol{d}) = \boldsymbol{\lambda} \cdot \boldsymbol{x}$. **So:** $\boldsymbol{\lambda} = \boldsymbol{d}$
  - **We found an eigenpair of $\boldsymbol{G}$:** $\boldsymbol{x} = (\mathbf{1}, \mathbf{1}, \dots, \mathbf{1}), \boldsymbol{\lambda} = \boldsymbol{d}$
- $\boldsymbol{d}$ is the largest eigenvalue of $\boldsymbol{A}$ (see next slide)

Remember the meaning of $\boldsymbol{y} = \boldsymbol{A}\cdot \boldsymbol{x}$:

$$y_i = \sum_{j=1}^{n} A_{ij} x_j = \sum_{(i,j)\in E} x_j$$

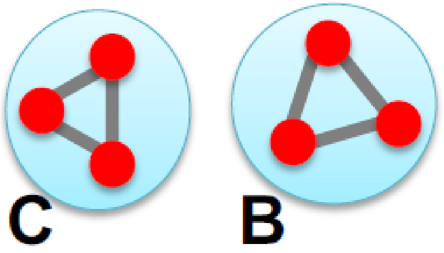
Note, this is just one eigenpair.
An *n* by *n* matrix can have up to *n* eigenpairs.

d is the largest eigen value of A

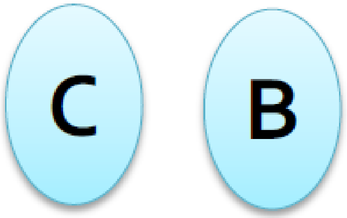
- $\boldsymbol{G}$ is $\boldsymbol{d}$-regular connected, $\boldsymbol{A}$ is its adjacency matrix
- **Claim:**
  - (1) $\boldsymbol{d}$ has multiplicity of **1** (there is only **1** eigenvector associated with eigenvalue $\boldsymbol{d}$)
  - (2) **d** is the largest eigenvalue of $\boldsymbol{A}$
- **Proof:**
  - To obtain value eigval $\boldsymbol{d}$ we needed $\boldsymbol{x_i} = \boldsymbol{x_j}$ for every i, j
  - This means $\boldsymbol{x} = c \cdot (1,1, \dots, 1)$ for some const. c
  - **Define:** Set $\boldsymbol{S}$ = nodes $\boldsymbol{i}$ with maximum value of $\boldsymbol{x_i}$
  - Then consider some vector $\boldsymbol{y}$ which is not a multiple of vector $(\mathbf{1}, \dots, \mathbf{1})$. So not all nodes $\boldsymbol{i}$ (with labels $\boldsymbol{y_i}$) are in $\boldsymbol{S}$
  - Consider some node $\boldsymbol{j} \in \boldsymbol{S}$ and a neighbor $\boldsymbol{i} \notin \boldsymbol{S}$ then node $\boldsymbol{j}$ gets a value strictly less than $\boldsymbol{d}$
  - So y is not eigenvector! And so $\boldsymbol{d}$ is the largest eigenvalue!

Example: Graph on 2 Components

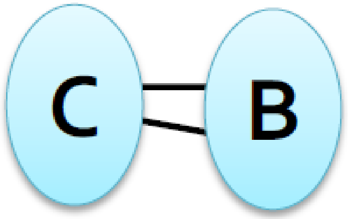
- **What if G is not connected?**
  - G has **2** components, each d-regular



- **What are some eigenvectors?**
  - $x =$ Put all $\mathbf{1}$s on C and $\mathbf{0}$s on B or vice versa
    - $x' = (\underbrace{1, \dots, 1}_{|C|}, \underbrace{0, \dots, 0}_{|B|})^T$ **then** $A \cdot x' = (d, \dots, d, 0, \dots, 0)^T$
    - $x'' = (0, \dots, 0, 1, \dots, 1)^T$ **then** $A \cdot x'' = (0, \dots, 0, d, \dots, d)^T$
    - And so in both cases the corresponding $\lambda = d$
- **A bit of intuition:**



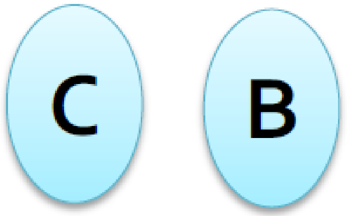
$\lambda_n = \lambda_{n-1}$



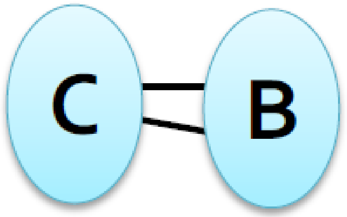
$\lambda_n - \lambda_{n-1} \approx 0$

2nd largest eigval. λ_{n-1} now has value very close to λ_n

More intuition



$\lambda_n = \lambda_{n-1}$



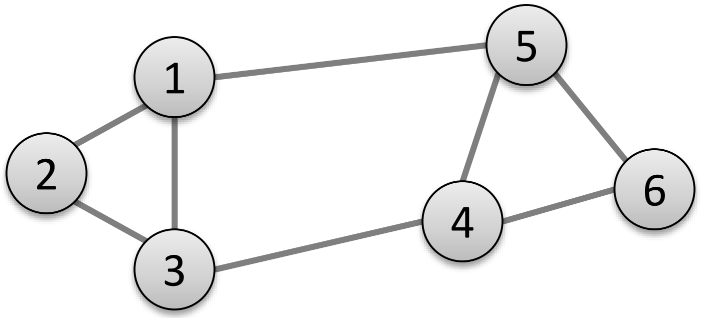
$\lambda_n - \lambda_{n-1} \approx 0$

2nd largest eigval. λ_{n-1} now has value very close to λ_n

- If the d-regular graph is connected (right example) then we already know that $x_n = (1, \dots 1)$ is an eigenvector
- Eigenvectors are orthogonal so then the components of x_{n-1} must sum to $\mathbf{0}$
  - Why? $x_n \cdot x_{n-1} = 0$ then $\sum_i x_n[i] \cdot x_{n-1}[i]$ = $\sum_i x_{n-1}[i]$ =0
  - x_{n-1} "splits" the nodes into two groups
    - $x_{n-1}[i] > 0$ vs. $x_{n-1}[i] < 0$
  - So we in principle could look at the eigenvector of the 2nd largest eigenvalue and declare nodes with positive label in **C** and negative label in **B.** (but there are still many details for us to figure out here)

Matrix Representations

- **Adjacency matrix (A):**
  - $n \times n$ matrix
  - $A=[a_{ij}]$, $a_{ij}=1$ if edge between node i and j

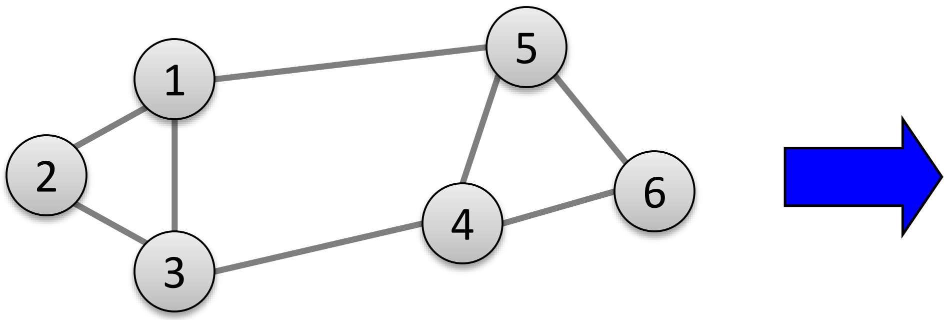


	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

- **Important properties:**
  - Symmetric matrix
  - Eigenvectors are real and orthogonal

Matrix Representations

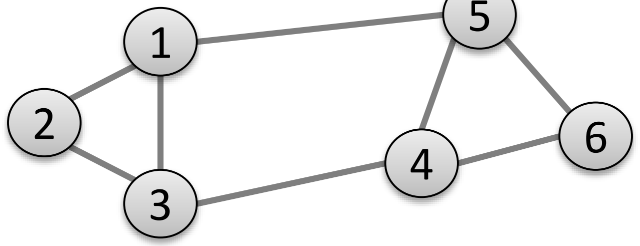
- **Degree matrix (D):**
  - $n \times n$ diagonal matrix
  - $D=[d_{ii}]$, d_{ii} = degree of node i



	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

Matrix Representations

- **Laplacian matrix (L):**
  - $n \times n$ symmetric matrix



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

$$\boldsymbol{L = D - A}$$

- **What is trivial eigenpair?**
- **Important properties:**
  - **Eigenvalues** are non-negative real numbers
  - **Eigenvectors** are real and orthogonal

3 Facts about The Laplacian L

(a) All eigenvalues are ≥ 0

(b) $x^T L x = \sum_{ij} L_{ij} x_i x_j \geq 0$ for every x

(c) L can be written as $L = N^T \cdot N$

- That is, L is positive semi-definite

- **Proof: (the 3 facts are saying the same thing)**
  - **(c)⇒(b):** $x^T L x = x^T N^T N x = (Nx)^T (Nx) \geq 0$
    - As it is just the square of length of Nx
  - **(b)⇒(a):** Let $\boldsymbol{\lambda}$ be an eigenvalue of $\boldsymbol{L}$. Then by **(b)** $x^T L x \geq 0$ so $x^T L x = x^T \lambda x = \lambda x^T x \Rightarrow \boldsymbol{\lambda \geq 0}$
  - **(a)⇒(c):** is also easy! Do it yourself.

L2 as Optimization Problem

- **Fact: For symmetric matrix M:**

$$\lambda_2 = \min_{x\,:\,x^T w_1 = 0} \frac{x^T M x}{x^T x}$$

(**w₁** is eigenvector corresponding to λ_1)

See next slid
for the proof.
Deriving this
a HW proble

- **What is the meaning of min $x^T L x$ on G?**
  - $x^T L\,x = \sum_{i,j=1}^{n} L_{ij}\, x_i x_j = \sum_{i,j=1}^{n} (D_{ij} - A_{ij})\, x_i x_j$
  - $= \sum_i D_{ii} x_i^2 - \sum_{(i,j)\in E} 2 x_i x_j$
  - $= \sum_{(i,j)\in E} (\underbrace{x_i^2 + x_j^2} - 2 x_i x_j) = \sum_{(i,j)\in E} (x_i - x_j)^2$

Node i has degree d_i. So, value x_i^2 needs to be summed up d_i times.
But each edge (i,j) has two endpoints so we need $x_i^2 + x_j^2$

Proof: $\lambda_2 = \min\limits_{x\,:\,x^T w_1 = 0} \frac{x^T M x}{x^T x}$

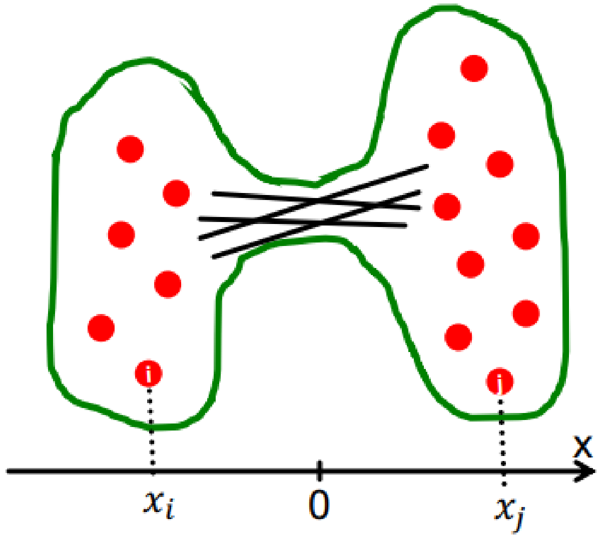
- Write x in basis of eigenvecs $w_1, w_2, \dots, w_n$ of $\boldsymbol{M}$ and λ_i are corresponding eigenvalues. So, $x = \sum_i^n \alpha_i w_i$
- Then we get: $Mx = \sum_i \alpha_i \underbrace{M w_i}_{\lambda_i w_i} = \sum_i \alpha_i \lambda_i w_i$
- So, what is $\boldsymbol{x^T M x}$?
  - $x^T M x = \overbrace{\left(\sum_i \alpha_i w_i\right)^T}^{x^T} \overbrace{\left(\sum_i \alpha_i \lambda_i w_i\right)}^{Mx} = \sum_{ij} \alpha_i \lambda_j \alpha_j$
    $= \sum_i \alpha_i^2 \lambda_i \underbrace{w_i^T w_i}_{= 0 \text{ if } i \neq j,\ 1 \text{ otherwise}} = \boldsymbol{\sum_i \lambda_i \alpha_i^2}$
  - Want minimize this over all unit vectors $\boldsymbol{w}$:
    $\boldsymbol{w}$ = min over choices of $(\alpha_1, \dots \alpha_n)$ so that:
    - $x^T w_1 = 0$, rewrite it as $\left(\sum_i \alpha_i w_i\right) \cdot w_1 = 0$ and remember that $w_i^T w_j = 0$ (because w are eigenvectors). Then $\alpha_1 = 0$
    - $\sum \alpha_i^2 = 1$ (unit length)
  - So, to minimize this, set $\boldsymbol{\alpha_2 = 1}$ and the rest to 0 $\boldsymbol{\sum_i \lambda_i \alpha_i^2 = \lambda_2}$

Finding X that Solves $\lambda_2 = \min_{x\,:\,x^T w_1 = 0} \frac{x^T M x}{x^T x}$

- **What else do we know about x?**
  - $\boldsymbol{x}$ is unit vector: $\sum_i x_i^2 = 1$
  - $\boldsymbol{x}$ is orthogonal to 1st eigenvector $(\mathbf{1}, \dots, \mathbf{1})$ thus: $\sum_i x_i \cdot 1 = \sum_i x_i = 0$
- **Remember:**

$$\lambda_2 = \min_{\substack{\text{All labelings} \\ \text{of nodes } i \text{ so} \\ \text{that } \sum x_i = 0}} \frac{\sum_{(i,j)\in E} (x_i - x_j)^2}{\sum_i x_i^2}$$

We want to assign values x_i to nodes i such that few edges cross 0.
(we want x_i and x_j to subtract each other)



Balance to minimize

Finding Optimal Cut (Fidler 73)

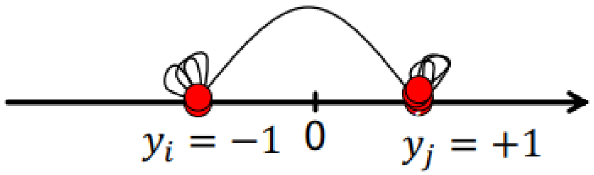
- **Back to finding the optimal cut**
- **Express partition (A,B) as a vector**

$$\boldsymbol{y_i = \begin{cases} +1 & if\ i \in A \\ -1 & if\ i \in B \end{cases}}$$

- Enforce that **|A| = |B|** → $\boldsymbol{\Sigma_i y_i = 0}$
  - Equivalent to being orthogonal to the trivial eigenvector $(\mathbf{1}, \dots, \mathbf{1})$
- We can minimize the cut of the partition by finding a vector y that **minimizes**:

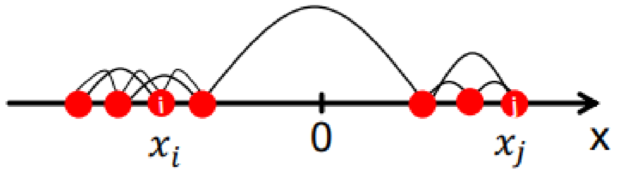
$$\arg \min_{y \in \{-1,+1\}^n} f(y) = \sum_{(i,j) \in E} (y_i - y_j)^2$$

Can't solve exactly. Let's relax y and allow it to take any real value.



Rayleigh Theorem $\lambda_2 = \min_{x\,:\,x^T w_1 = 0} \frac{x^T M x}{x^T x}$

$$\min_{\substack{\in \mathbb{R}^n \,:\, \sum_i y_i = 0 \\ \sum_i y_i^2 = 1}} f(y) = \sum_{(i,j)\in E} (y_i - y_j)^2 = y^T L y$$



- $\boldsymbol{\lambda_2 = \min_y f(y)}$: The minimum value of $f(y)$ is given by the 2nd smallest eigenvalue λ_2 of the Laplacian matrix L
- $\mathbf{x} = \arg\min_y f(y)$: The optimal solution for y is given by the eigenvector x corresponding to λ_2, referred to as the **Fiedler vector**
- Can use sign of x_i to determine cluster assignment of node i

Approx. Guarantee of Spectral

- Suppose there is a partition of **G** into **A** and **B** where $|A| \leq |B|$, s.t. "conductance" of the cut (A,B) is $\boldsymbol{\beta} = \frac{(\#\ edges\ from\ A\ to\ B)}{|A|}$ then $\boldsymbol{\lambda_2} \leq 2\beta$

  Note: |A|<|B|

  - This is the approximation guarantee of the spectral clustering: Spectral finds a cut that has at most **twice the conductance** as the optimal one of conductance $\boldsymbol{\beta}$.
- **Proof:**
  - Let: $\boldsymbol{a} = |\boldsymbol{A}|, \boldsymbol{b} = |\boldsymbol{B}|$ and $\boldsymbol{e} = \#$ edges from $\boldsymbol{A}$ to $\boldsymbol{B}$
  - Enough to choose some $\boldsymbol{x_i}$ based on $\boldsymbol{A}$ and $\boldsymbol{B}$ such that:

$$\lambda_2 \leq \underbrace{\frac{\sum (x_i - x_j)^2}{\sum_i x_i^2}} \leq 2\beta \text{ (while also } \textstyle\sum_i x_i = 0)$$

Act

Approx. Guarantee of Spectral

- **Proof (continued):**
  - **1) Let's set:** $x_i = \begin{cases} -\frac{1}{a} & \text{if } i \in A \\ +\frac{1}{b} & \text{if } i \in B \end{cases}$

    Note: |A|<|B|

    - Let's quickly verify that $\sum_i x_i = 0$: $a\left(-\frac{1}{a}\right) + b\left(\frac{1}{b}\right) = \mathbf{0}$
  - **2) Then:** $\frac{\sum (x_i - x_j)^2}{\sum_i x_i^2} = \frac{\sum_{i \in A, j \in B}\left(\frac{1}{b}+\frac{1}{a}\right)^2}{a\left(-\frac{1}{a}\right)^2 + b\left(\frac{1}{b}\right)^2} = \frac{e \cdot \left(\frac{1}{a}+\frac{1}{b}\right)^2}{\frac{1}{a}+\frac{1}{b}} =$

$$e\left(\frac{1}{a}+\frac{1}{b}\right) \leq e\left(\frac{1}{a}+\frac{1}{a}\right) = e\frac{2}{a} \leq 2\beta$$

e … number of edges between A and B

Which proves that the cost achieved by spectral is better than twice the OPT cost

Approx. Guarantee of Spectral

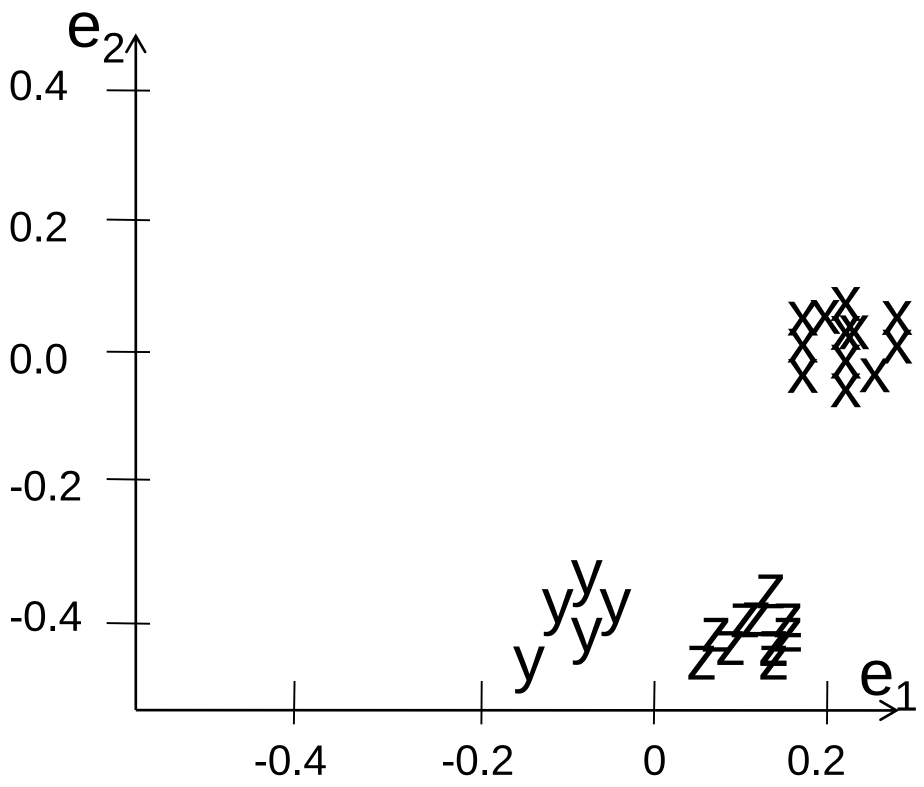
- **Putting it it all together: The Cheeger inequality**

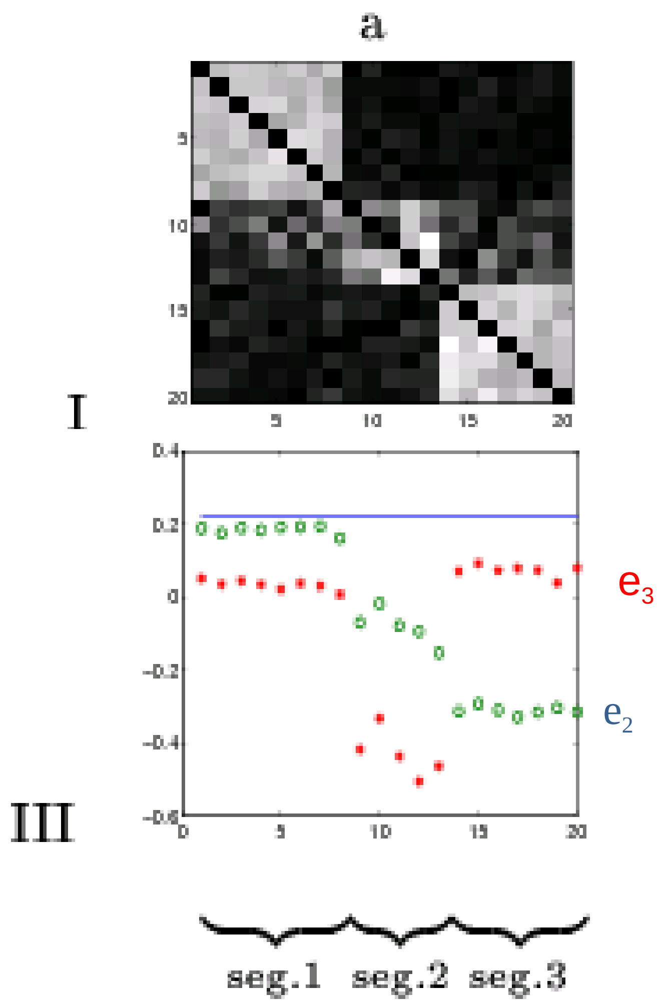
$$\frac{\beta^2}{2k_{max}} \leq \lambda_2 \leq 2\beta$$

  - where k_{max} is the maximum node degree in the graph
    - Note we only provide the 1st part: $\lambda_2 \leq 2\beta$
    - We did not prove $\frac{\beta^2}{2k_{max}} \leq \lambda_2$
  - Overall this always certifies that λ_2 always gives a useful bound

Spectral Clustering: Graph = Matrix

$W{*}v_1 = v_2$ “propogates weights from neighbors”





[Shi & Meila, 2002]

Spectral Clustering: Graph = Matrix

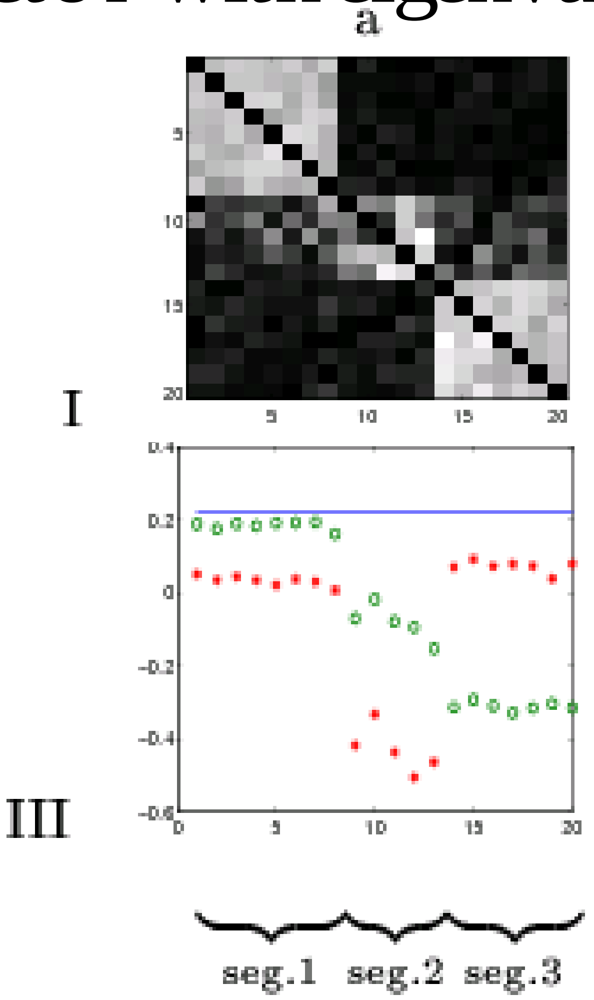
$W^*v_1 = v_2$ "propagates weights from neighbors"

$$\mathbf{W} \cdot \mathbf{v} = \lambda \mathbf{v} : \mathbf{v} \text{ is an eigenvector with eigenvalue } \lambda$$

If Wis connected but roughly block diagonal with *k* blocks then

• the top eigenvector is a constant vector

• the next *k* eigenvectors are roughly piecewise constant with "pieces" corresponding to blocks



M

Spectral Clustering: Graph = Matrix

$W*v_1 = v_2$ "propagates weights from neighbors"

$$\mathbf{W} \cdot \mathbf{v} = \lambda \mathbf{v} : \mathbf{v} \text{ is an eigenvector with eigenvalue } \lambda$$

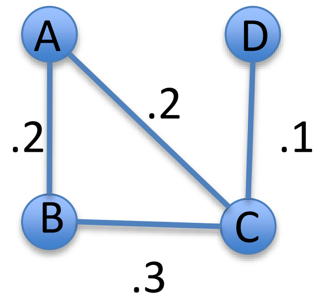
If **W** is connected but roughly block diagonal with *k* blocks then
• the "top" eigenvector is a constant vector
• the next *k* eigenvectors are roughly piecewise constant with "pieces" corresponding to blocks

Spectral clustering:
• Find the top *k+1* eigenvectors $\mathbf{v}_1,\ldots,\mathbf{v}_{k+1}$
• Discard the "top" one
• Replace every node *a* with *k*-dimensional vector $x_a = <\mathbf{v}_2(a),\ldots,\mathbf{v}_{k+1}(a)>$
• Cluster with *k*-means

M

Spectral Clustering

- Construct an **affinity** matrix



$W =$

	A	**B**	**C**	**D**
A	.4	.2	.2	0
B	.2	.5	.3	0
C	.2	.3	.6	.1
D	0	0	.1	.1

Spectral Clustering

- Construct the graph Laplacian

$$D = diag(D_0, D_1, \ldots, D_n)$$

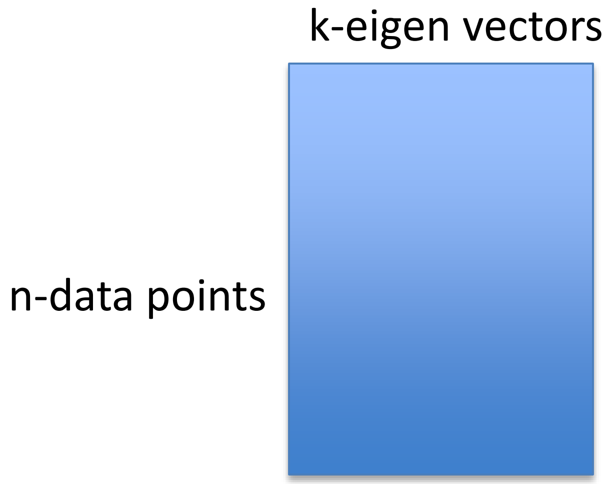
$$L = D - W$$

- Identify eigenvectors of the affinity matrix

$$Lv = \lambda v$$

Spectral Clustering

- K-Means on eigenvector transformation of the data.



Each Row represents a data point in the eigenvector space

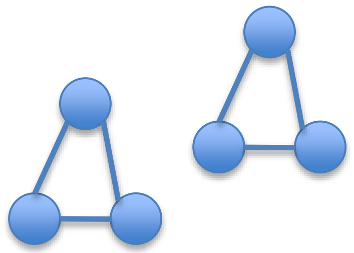
- Project back to the initial data representation.

Overview: what are we doing?

- Define the affinity matrix
- Identify eigenvalues and eigenvectors.
- K-means of transformed data
- Project back to original space

Why does this work?

- Ideal Case



1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

$$Lv = \lambda v$$

1	0
1	0
1	0
0	1
0	1
0	1

- What are we optimizing? Why do the eigenvectors of the laplacian include cluster identification information

Normalized Graph Cuts view

- Minimal (bipartitional) normalized cut.

$$\min \frac{Cut(C_1, C_2)}{Vol(C_1)} + \frac{Cut(C_1, C_2)}{Vol(C_2)} = \min \left(\frac{1}{Vol(C_1)} + \frac{1}{Vol(C_2)} \right) Cut(C_1, C_2)$$

$$NCut(A, B) = \frac{y^T(D - W)y}{y^T Dy}$$

$$\min_y y^T(D - W)y \text{ subject to } y^T Dy = 1$$

$$(D - W)y = \lambda Dy$$

- Eigenvalues of the laplacian are approximate solutions to mincut problem.

The Laplacian Matrix

- L = D-W
- Positive semi-definite $x^T L x \geq 0$
- The lowest eigenvalue is 0, eigenvector is $\vec{1}$
- The second lowest contains the solution
  - The corresponding eigenvector contains the cluster indicator for each data point

$$\lambda_2 = \frac{Cut(A,B)}{|A|} + \frac{Cut(A,B)}{|B|}$$

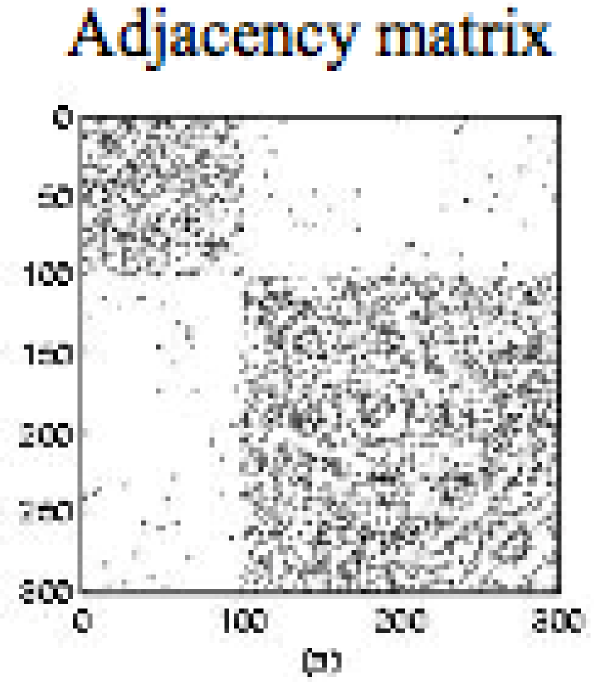
Using eigenvectors to partition

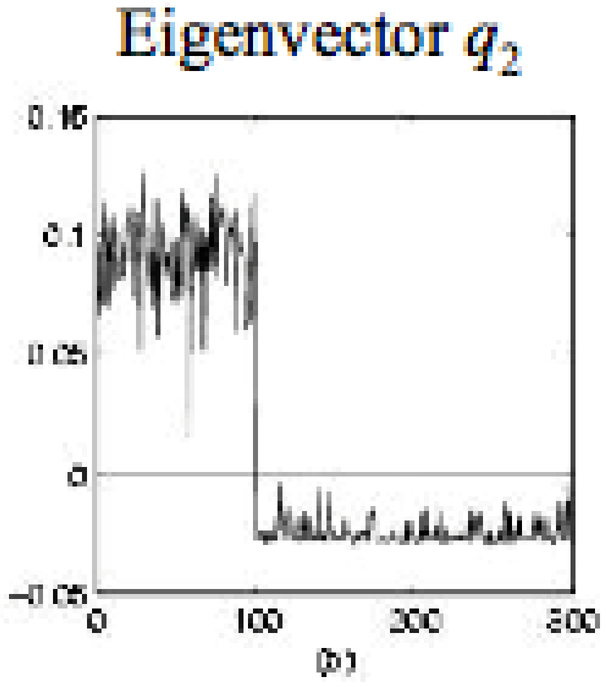
- Each eigenvector partitions the data set into two clusters.
- The entry in the second eigenvector determines the first cut.

- Subsequent eigenvectors can be used to further partition into more sets.

Example

- Dense clusters with some sparse connections

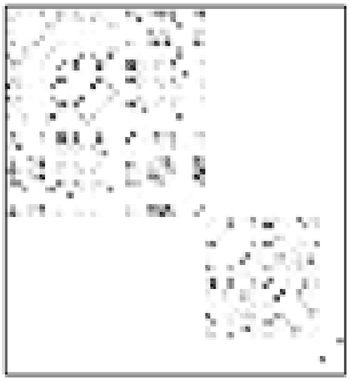




3 class Example

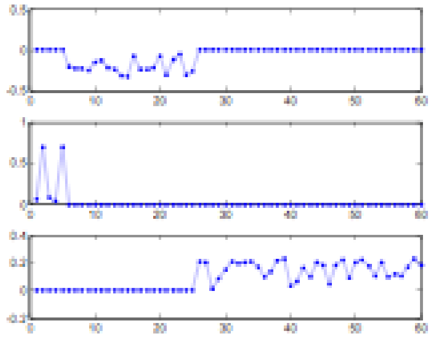
Affinity matrix

$W; A$



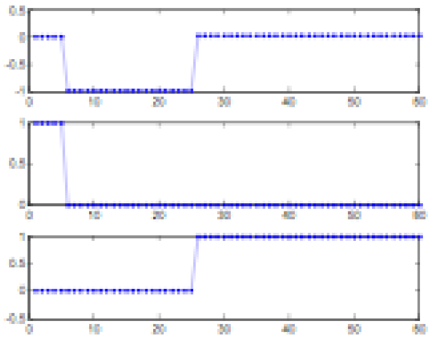
eigenvectors

$V = [v_1, v_2, v_3]$

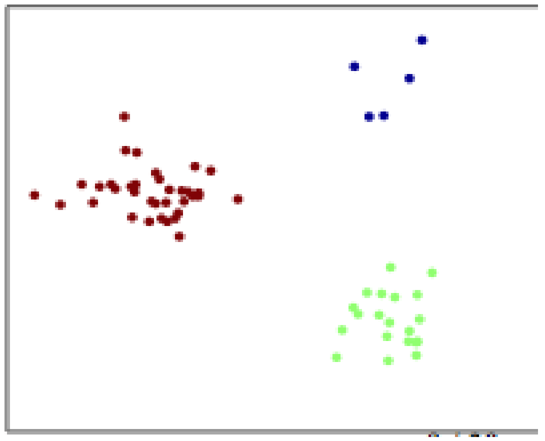


row normalization

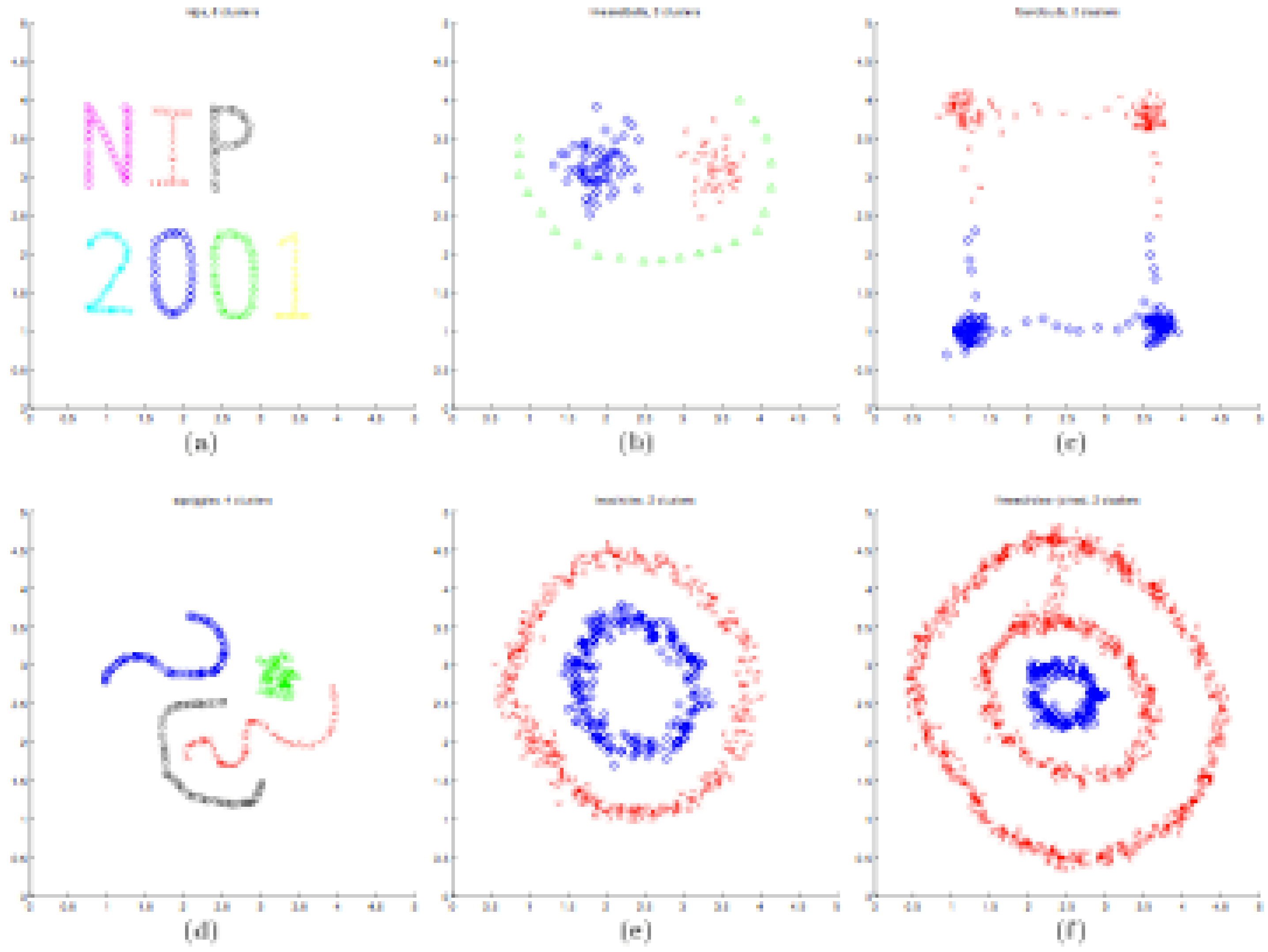
$U = [u_1, u_2, u_3]$



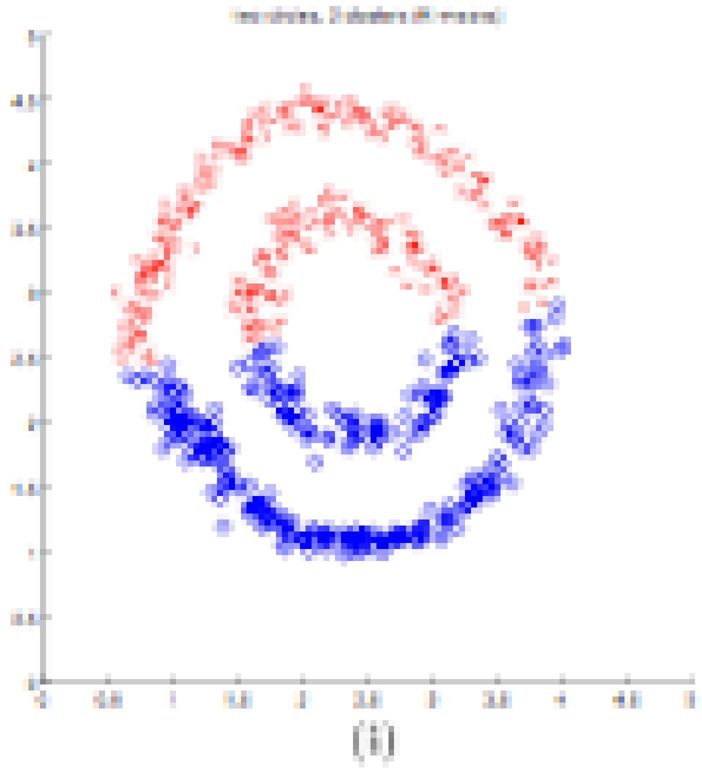
output



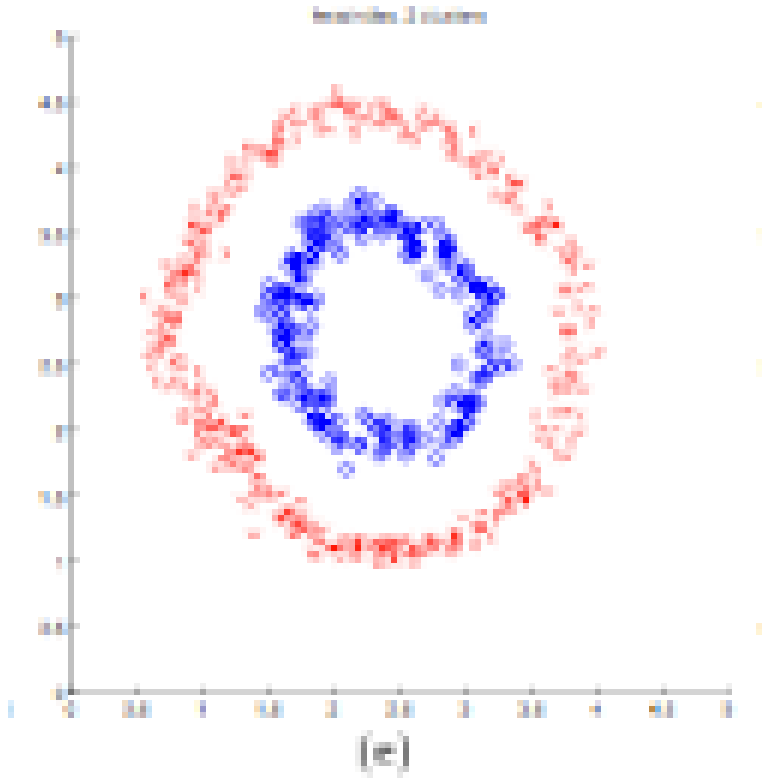
Example [Ng et al. 2001]



k-means vs. Spectral Clustering



K-means



Spectral Clustering

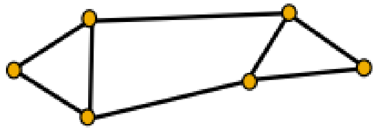
So far…

- **How to define a "good" partition of a graph?**
  - Minimize a given graph cut criterion
- **How to efficiently identify such a partition?**
  - Approximate using information provided by the eigenvalues and eigenvectors of a graph
- **Spectral Clustering**

Spectral Clustering Algorithms

- **Three basic stages:**
  - 1) **Pre-processing**
    - Construct a matrix representation of the graph
  - 2) **Decomposition**
    - Compute eigenvalues and eigenvectors of the matrix
    - Map each point to a lower-dimensional representation based on one or more eigenvectors
  - 3) **Grouping**
    - Assign points to one (or more) clusters, based on the new representation

- **1) Pre-processing:**
  - Build Laplacian matrix L of the graph



	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

- **2) Decomposition:**
  - Find eigenvalues λ and eigenvectors x of the matrix L
  - Map vertices to corresponding components of x_2

λ=

0.0
1.0
3.0
3.0
4.0
5.0

X =

0.4	0.3	-0.5	-0.2	-0.4	-0.5
0.4	0.6	0.4	-0.4	0.4	0.0
0.4	0.3	0.1	0.6	-0.4	0.5
0.4	-0.3	0.1	0.6	0.4	-0.5
0.4	-0.3	-0.5	-0.2	0.4	0.5
0.4	-0.6	0.4	-0.4	-0.4	0.0

1	0.3
2	0.6
3	0.3
4	-0.3
5	-0.3
6	-0.6

How do we now find the clusters?

- **3) Grouping:**
  - Sort components of reduced 1-dimensional vector
  - Identify clusters by splitting the sorted vector in two
- **How to choose a splitting point?**
  - Naïve approaches:
    - Split at **0** or median value
  - More expensive approaches:
    - Attempt to minimize normalized cut in 1-dimension (sweep over ordering of nodes induced by the eigenvector)

1	0.3
2	0.6
3	0.3
4	-0.3
5	-0.3
6	-0.6

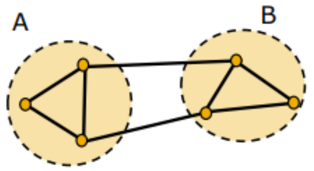
Split at 0:

Cluster A: Positive points

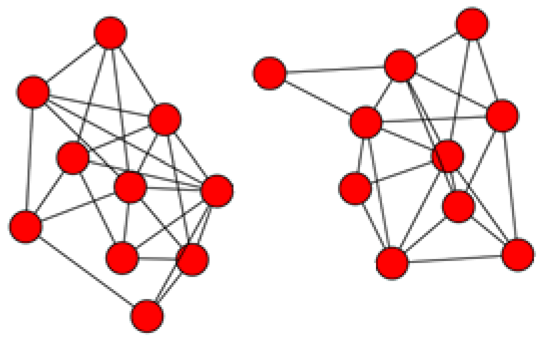
Cluster B: Negative points

1	0.3
2	0.6
3	0.3

4	-0.3
5	-0.3
6	-0.6



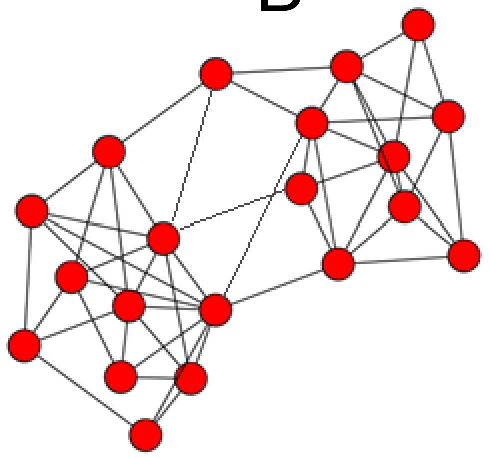
Graphs and Eigenvalues



A

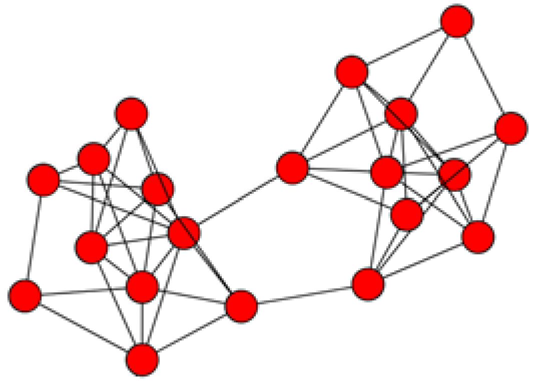
$0 = \lambda_1 = \lambda_2$

B



$\lambda_2(C) < \lambda_2(B)$

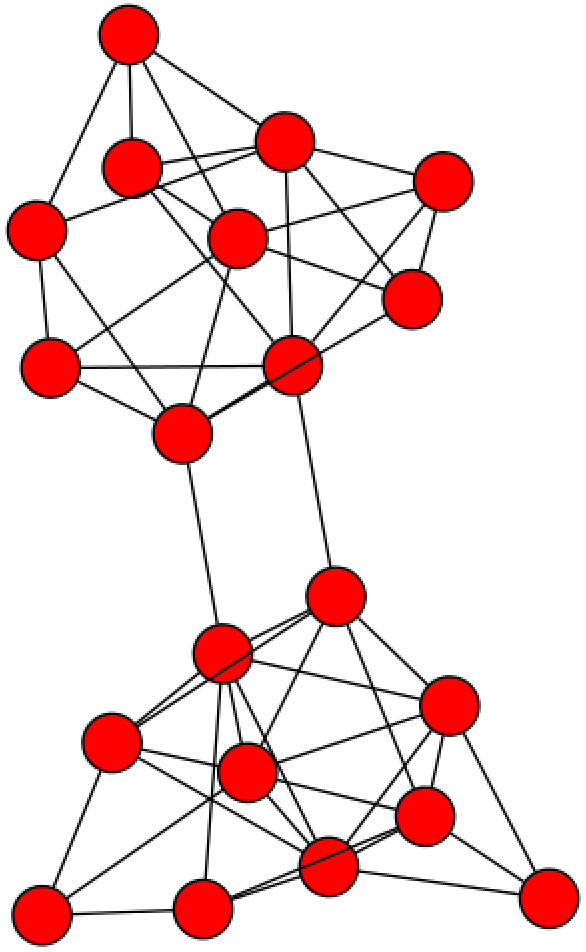
C

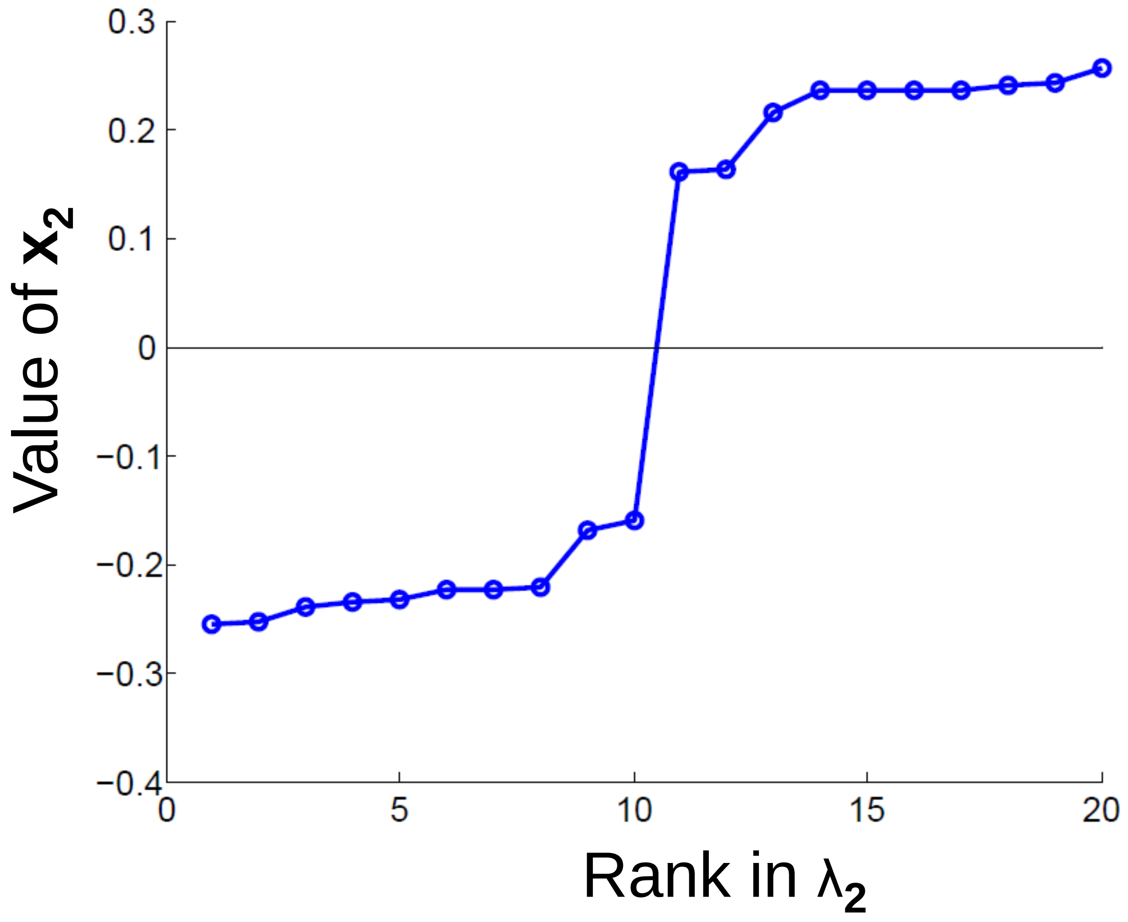


$0 = \lambda_1, \lambda_2 > 0$

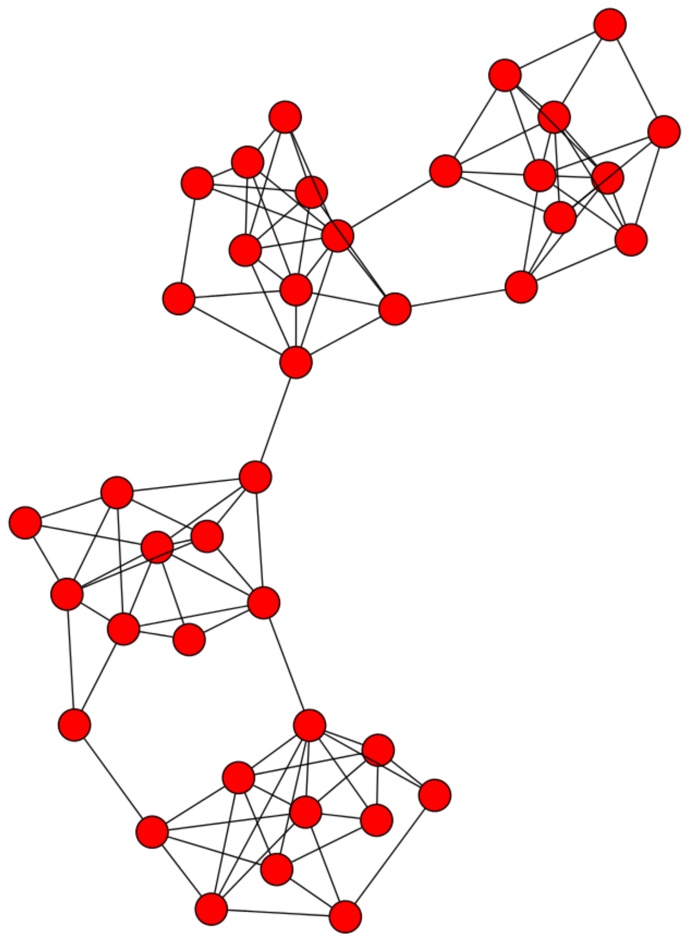
$0 = \lambda_1, \lambda_2 > 0$

Example: Spectral Partitioning

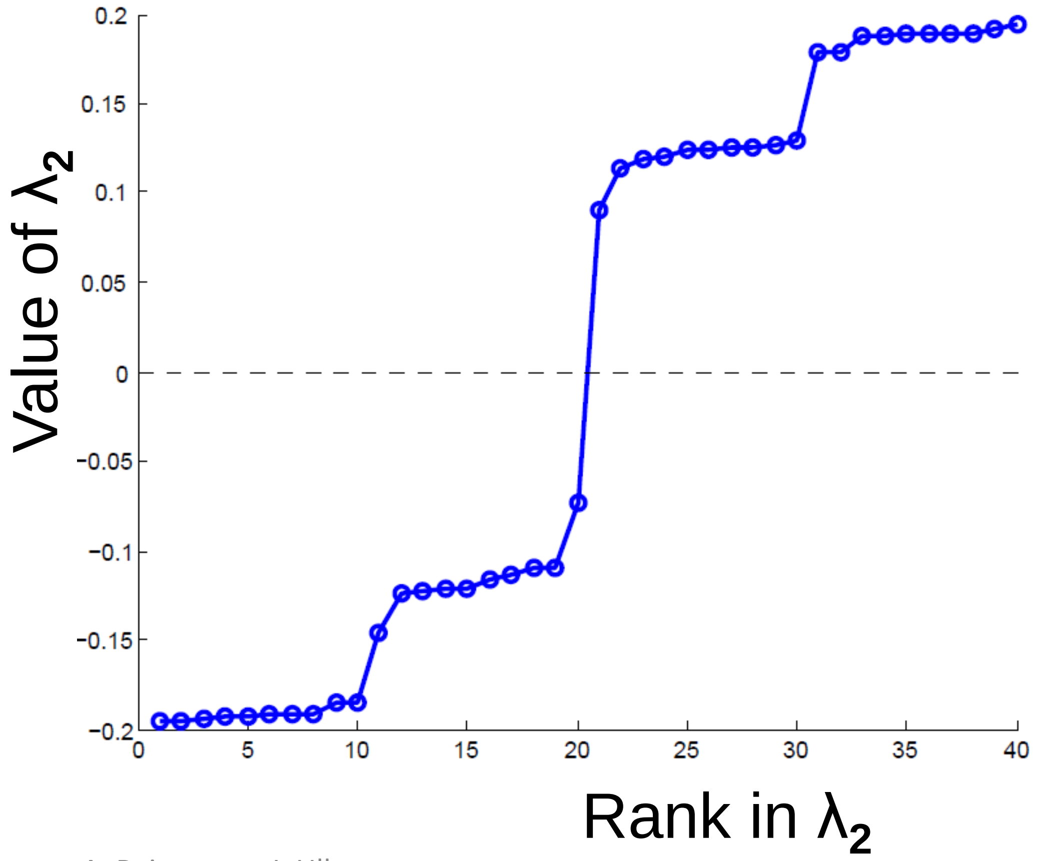




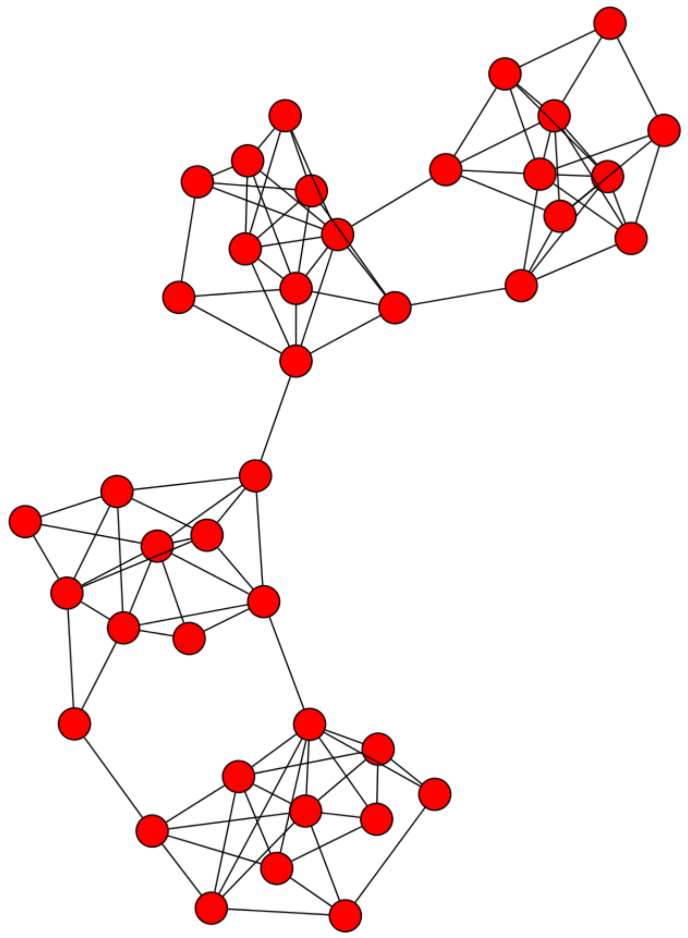
Example: Spectral Partitioning

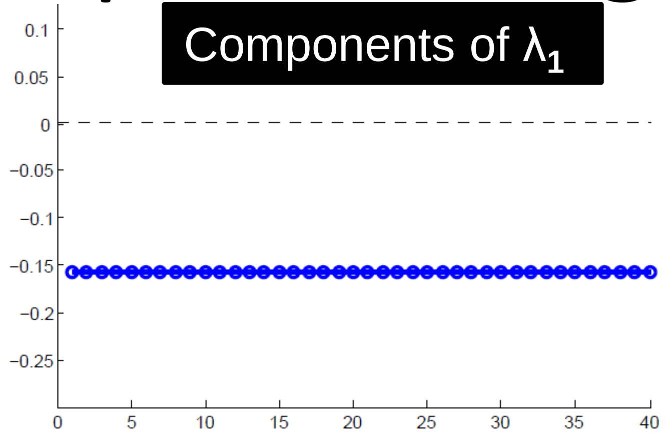


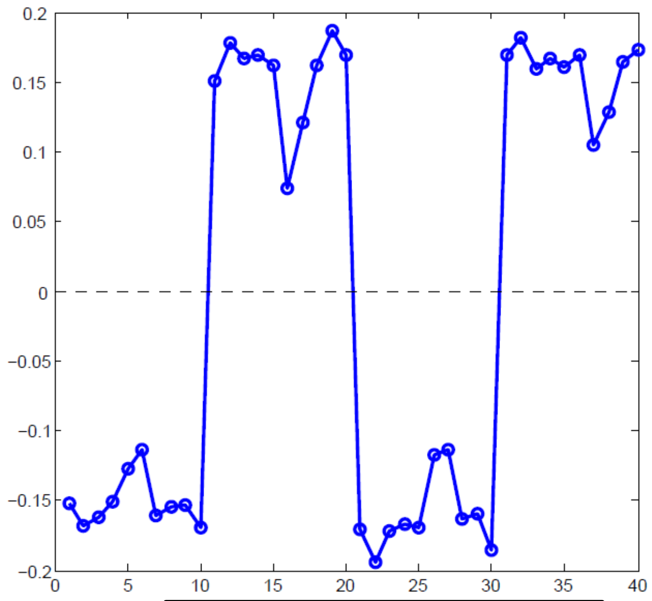
Components of $\mathbf{x_2}$



Example: Spectral partitioning







Components of λ_3

- How do we partition a graph into k clusters?

- Two basic approaches:
  - **Recursive bi-partitioning** [Hagen et al., '92]
    - Recursively apply bi-partitioning algorithm in a hierarchical divisive manner
    - Disadvantages: Inefficient, unstable
  - **Cluster multiple eigenvectors** [Shi-Malik, '00]
    - Build a reduced space from multiple eigenvectors
      - Each node is now represented by $\boldsymbol{k}$ numbers
      - We then cluster (apply k-means) the nodes based on their $\boldsymbol{k}$-dim representation
    - Commonly used in recent papers
    - A preferable approach…

k-Way Spectral Clustering

- **How do we partition a graph into *k* clusters?**

- **Two basic approaches:**
  - **Recursive bi-partitioning** [Hagen et al., ’92]
    - Recursively apply bi-partitioning algorithm in a hierarchical divisive manner
    - Disadvantages: Inefficient, unstable
  - **Cluster multiple eigenvectors** [Shi-Malik, ’00]
    - Build a reduced space from multiple eigenvectors
    - Commonly used in recent papers
    - A preferable approach…

Why use multiple eigenvectors?

- **Approximates the optimal cut** [Shi-Malik, '00]
  - Can be used to approximate optimal k-way normalized cut
- **Emphasizes cohesive clusters**
  - Increases the unevenness in the distribution of the data
  - Associations between similar points are amplified, associations between dissimilar points are attenuated
  - The data begins to "approximate a clustering"
- **Well-separated space**
  - Transforms data to a new "embedded space", consisting of $\boldsymbol{k}$ orthogonal basis vectors
- Multiple eigenvectors prevent instability due to information loss

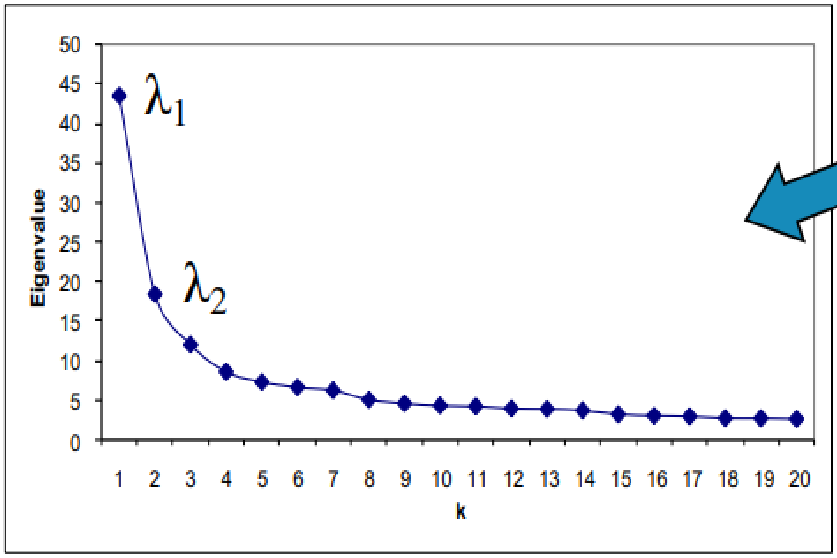
How to select K?

- **Eigengap:**
  - The difference between two consecutive eigenvalues
- **Most stable clustering is generally given by the value k that maximizes eigengap Δ_k:**

$$\Delta_k = |\lambda_k - \lambda_{k-1}|$$

Note eigenvalues are sorted in descending order

- **Example:**



$$\max \Delta_k = |\lambda_2 - \lambda_1|$$

$$\Rightarrow \textbf{Choose } k = 2$$

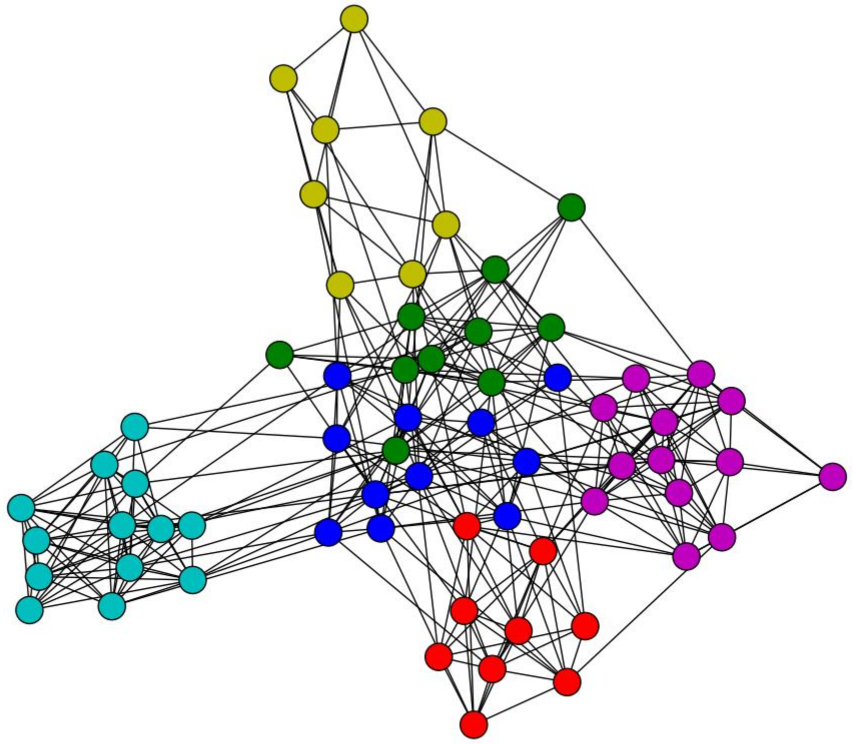
Spectral Clustering: Graph = Matrix

$W*v_1 = v_2$ “propogates weights from neighbors”

$$\mathbf{W}\cdot\mathbf{v} = \lambda\mathbf{v} : \mathbf{v} \text{ is an eigenvector with eigenvalue } \lambda$$

- smallest eigenvecs of D-A are largest eigenvecs of A
- smallest eigenvecs of I-W are largest eigenvecs of W

Q: How do I pick **v** to be an eigenvector for a block-stochastic matrix?

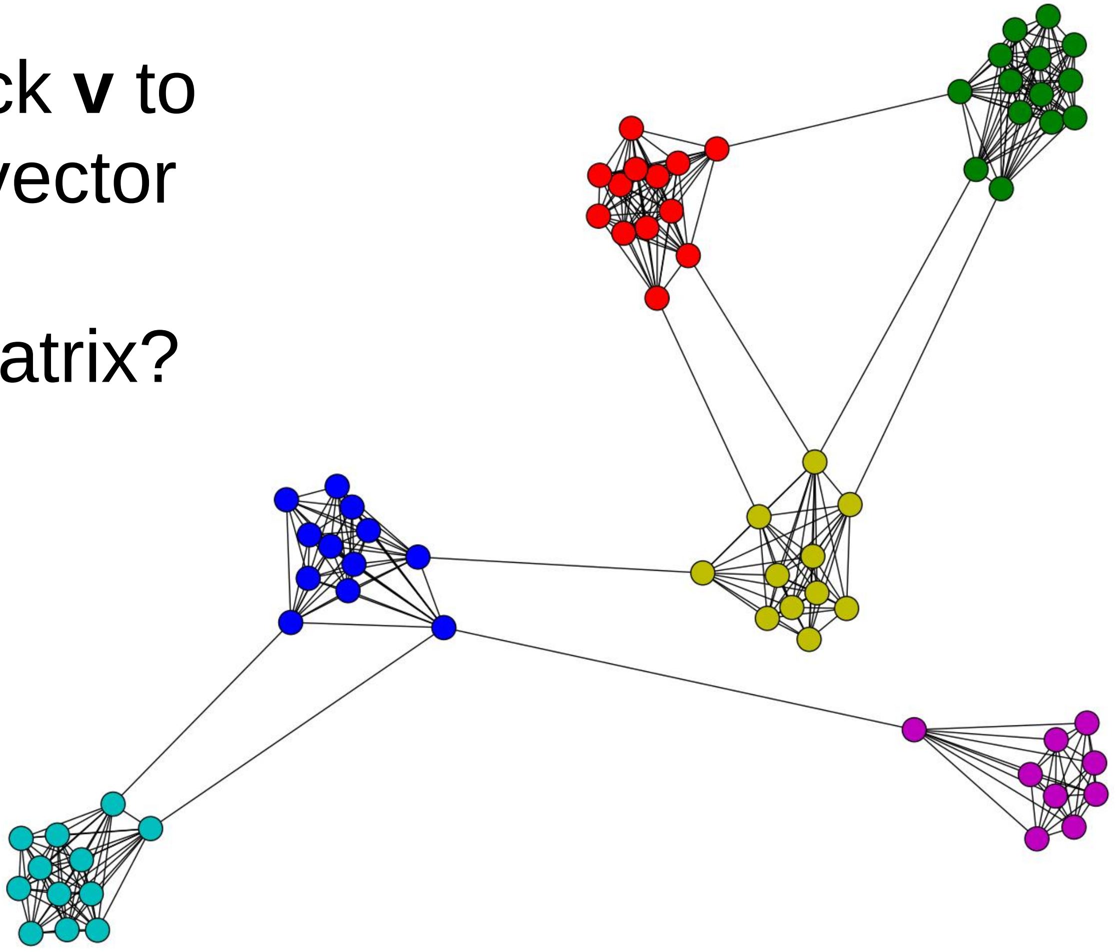


Spectral Clustering: Graph = Matrix

$W^*v_1 = v_2$ “propogates weights from neighbors”

$$\mathbf{W} \cdot \mathbf{v} = \lambda \mathbf{v} : \mathbf{v} \text{ is an eigenvector with eigenvalue } \lambda$$

How do I pick **v** to be an eigenvector for a block-stochastic matrix?



Spectral Clustering: Graph = Matrix

W*v_1 = v_2 “propogates weights from neighbors”

$$\mathbf{W} \cdot \mathbf{v} = \lambda \mathbf{v} : \mathbf{v} \text{ is an eigenvector with eigenvalue } \lambda$$

- smallest eigenvecs of D-A are largest eigenvecs of A
- smallest eigenvecs of I-W are largest eigenvecs of W

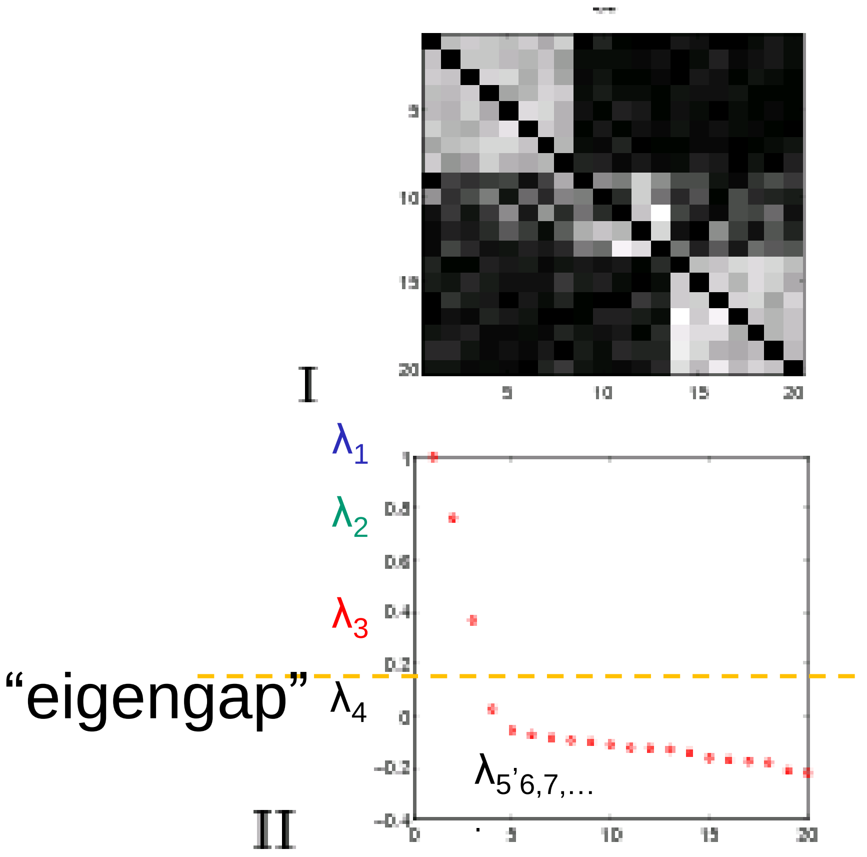
Suppose each *y(i)=+1* or *-1*:

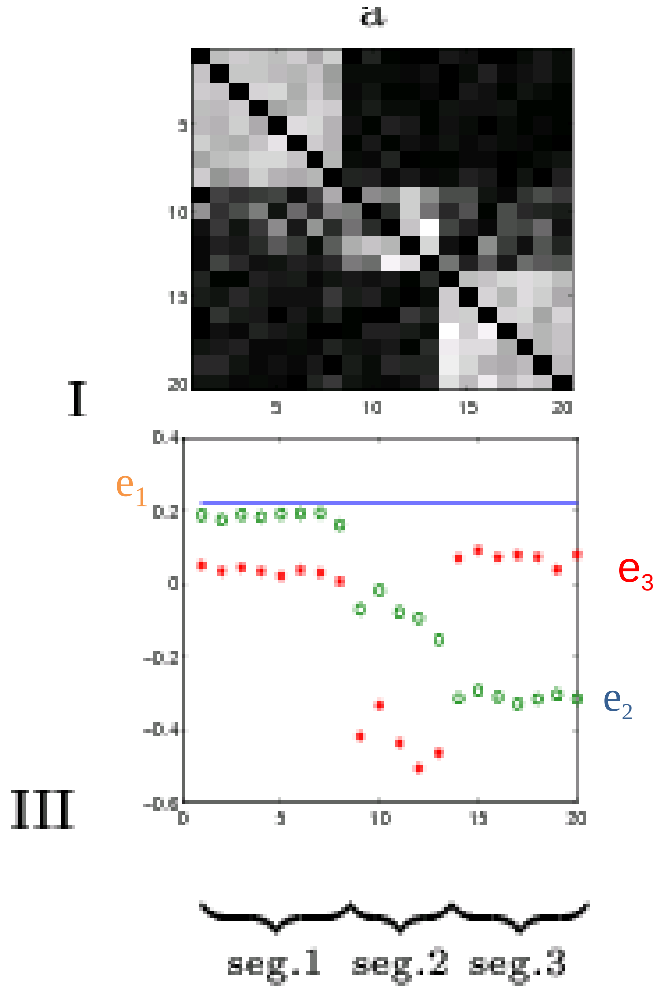
- Then *y* is a cluster indicator that cuts the nodes into two
- what is $\mathbf{y}^T$(D-A)$\mathbf{y}$? <u>The cost of the graph cut defined by **y**</u>
- what is $\mathbf{y}^T$(I-W)$\mathbf{y}$? <u>Also a cost of a graph cut defined by **y**</u>
- How to minimize it?
    - Turns out: to minimize $\mathbf{y}^T$ X $\mathbf{y}$ / ($\mathbf{y}^T\mathbf{y}$) find *smallest* eigenvector of X
    - But: this will not be +1/-1, so it’s a “relaxed” solution

Spectral Clustering: Graph = Matrix

W*v_1 = v_2 “propogates weights from neighbors”

$$\mathbf{W} \cdot \mathbf{v} = \lambda \mathbf{v} : \mathbf{v} \text{ is an eigenvector with eigenvalue } \lambda$$





[Shi & Meila, 2002]

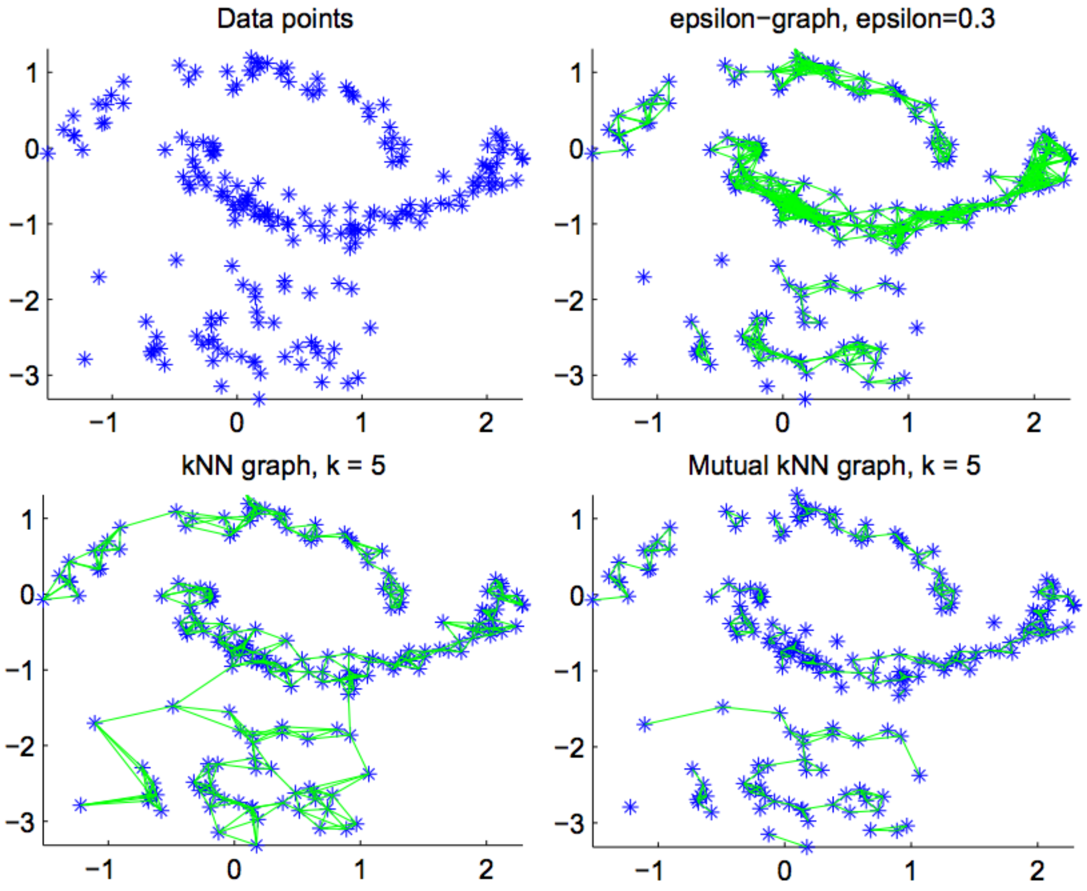


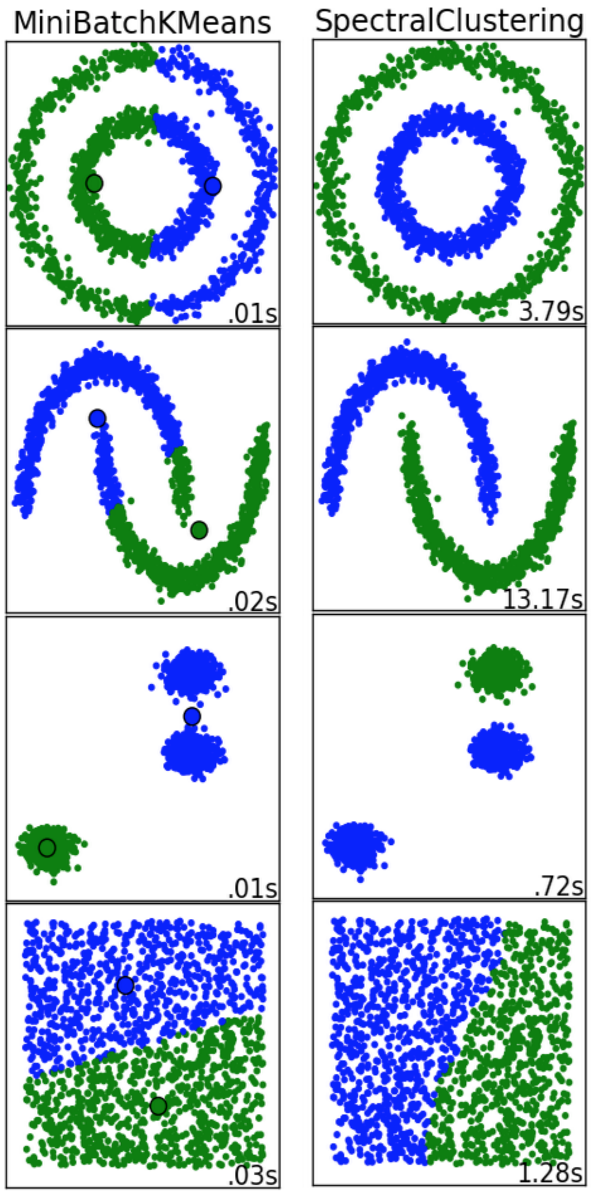
Figure 3: Different similarity graphs, see text for details.

Spectral Clustering: Pros and Cons

- Elegant, and well-founded mathematically
- Works quite well when relations are approximately transitive (like similarity)
- Very noisy datasets cause problems
  - "Informative" eigenvectors need not be in top few
  - Performance can drop suddenly from good to terrible
- Expensive for very large datasets
  - Computing eigenvectors is the bottleneck

Use cases and runtimes

- K-Means
  - FAST
  - "Embarrassingly parallel"
  - Not very useful on anisotropic data
- Spectral clustering
  - Excellent quality under many different data forms
  - Much slower than K-Means



Further Reading

- Spectral Clustering Tutorial: http://www.informatik.uni-hamburg.de/ML/contents/people/luxburg/publications/Luxburg07_tutorial.pdf