

CSCI 4360/6360 Data Science II

Spectral Clustering

High Dimensional Data

- Given a cloud of data points we want to understand its structure

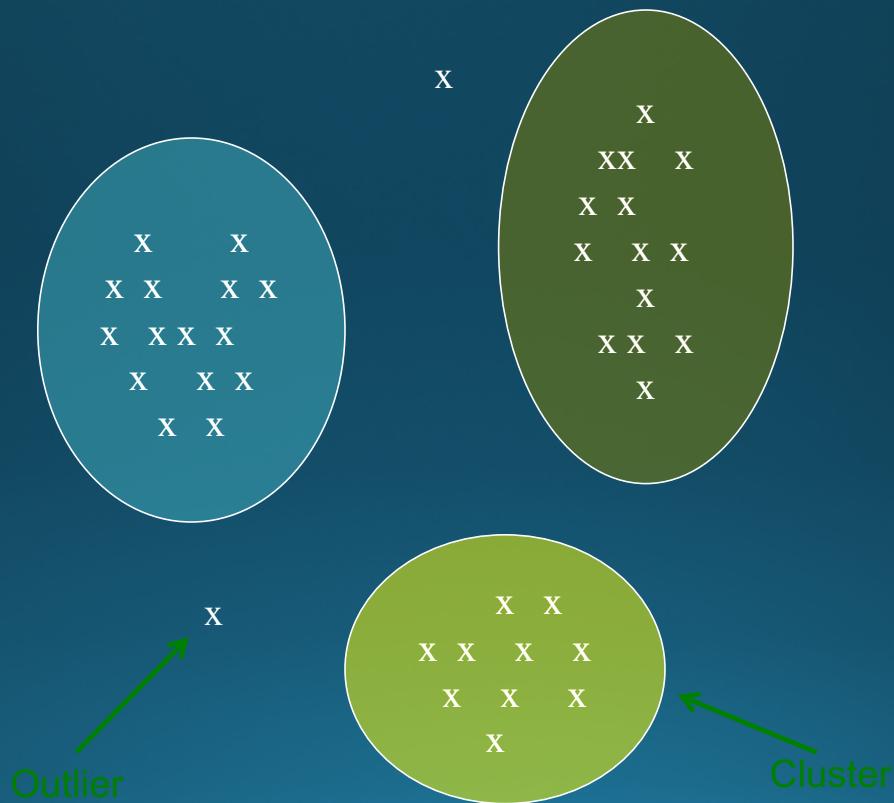


J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

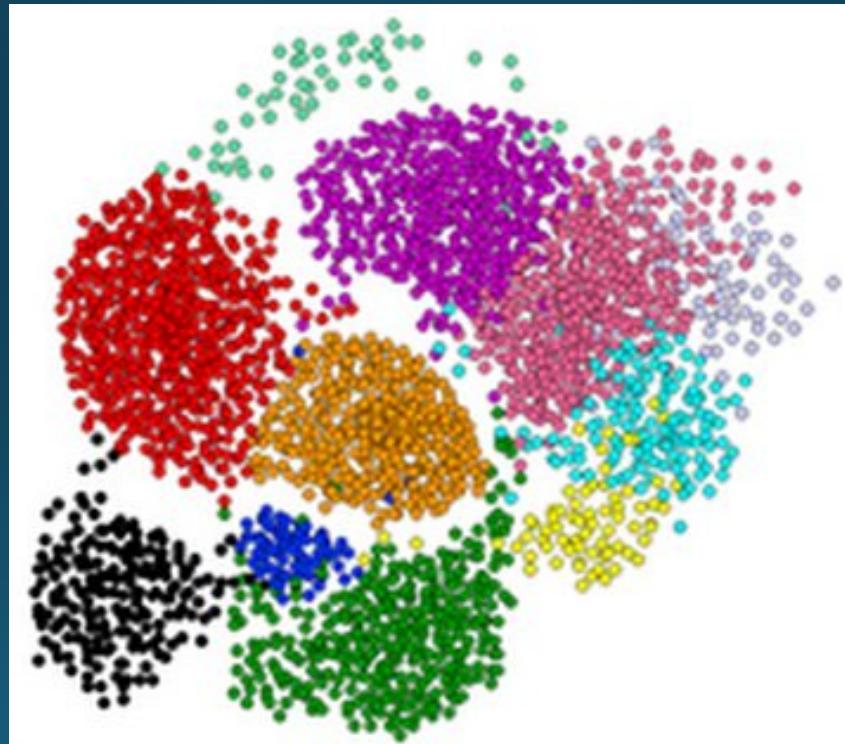
The Problem of Clustering

- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of **clusters**, so that
 - Members of a cluster are close/similar to each other
 - Members of different clusters are dissimilar
- **Usually:**
 - Points are in a high-dimensional space
 - Similarity is defined using a distance measure
 - Euclidean, Cosine, Jaccard, edit distance, ...

Example: Clusters & Outliers



Clustering is a hard problem!



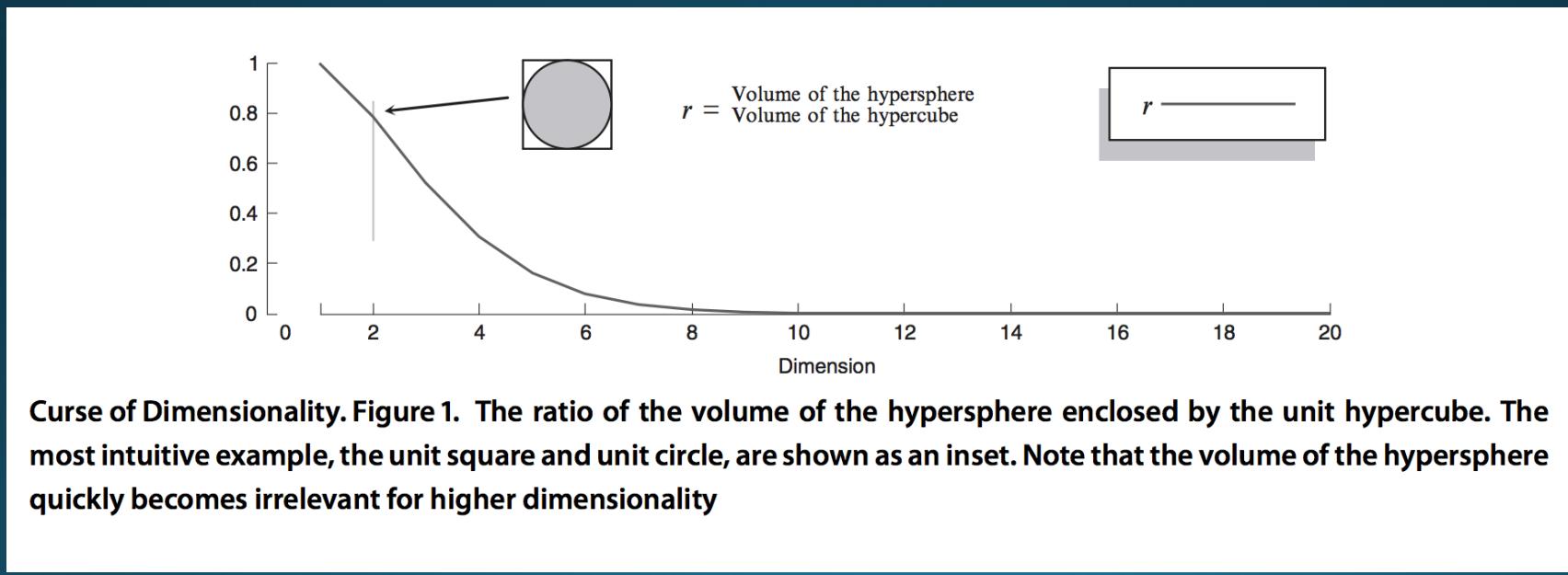
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

Why is it hard?

- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy
- And in most cases, looks are not deceiving
- Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:** Almost all pairs of points are at about the same distance

Curse of dimensionality

- “Vastness” of Euclidean space



Clustering Problem: Galaxies

- A catalog of 2 billion “sky objects” represents objects by their radiation in 7 dimensions (frequency bands)
- **Problem:** Cluster into similar objects, e.g., galaxies, nearby stars, quasars, etc.
- Sloan Digital Sky Survey



Clustering Problem: Music CDs

- **Intuitively:** Music divides into categories, and customers prefer a few categories
 - But what are categories really?
- Represent a CD by a set of customers who bought it:
- Similar CDs have similar sets of customers, and vice-versa

Clustering Problem: Music CDs

Space of all CDs:

- Think of a space with one dimension for each customer
 - Values in a dimension may be 0 or 1 only
 - A CD is a point in this space (x_1, x_2, \dots, x_k) , where $x_i = 1$ iff the i^{th} customer bought the CD
- For Amazon, the dimension is tens of millions
- **Task:** Find clusters of similar CDs

Clustering Problem: Documents

Finding topics:

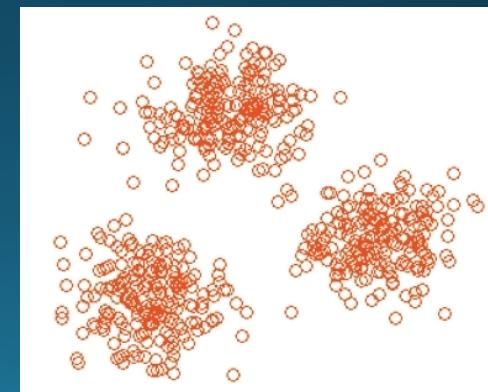
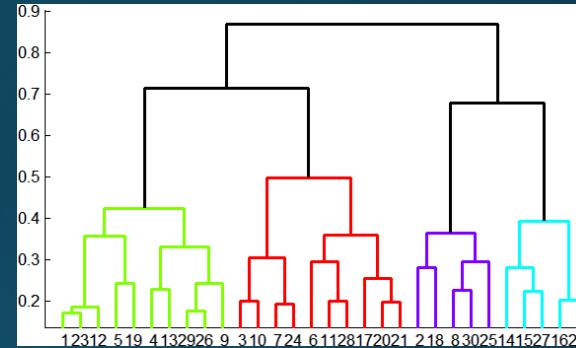
- Represent a document by a vector (x_1, x_2, \dots, x_k) , where $x_i = 1$ iff the i^{th} word (in some order) appears in the document
 - It actually doesn't matter if k is infinite; i.e., we don't limit the set of words
- **Documents with similar sets of words may be about the same topic**

Cosine, Jaccard, and Euclidean

- As with CDs we have a choice when we think of documents as sets of words or shingles:
 - **Sets as vectors:** Measure similarity by the **cosine distance**
 - **Sets as sets:** Measure similarity by the **Jaccard distance**
 - **Sets as points:** Measure similarity by **Euclidean distance**

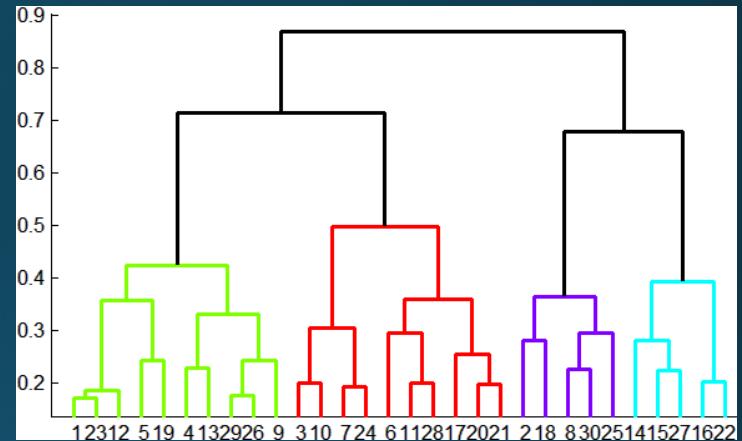
Overview: Methods of Clustering

- **Hierarchical:**
 - **Agglomerative** (bottom up):
 - Initially, each point is a cluster
 - Repeatedly combine the two “nearest” clusters into one
 - **Divisive** (top down):
 - Start with one cluster and recursively split it
- **Point assignment:**
 - Maintain a set of clusters
 - Points belong to “nearest” cluster



Hierarchical Clustering

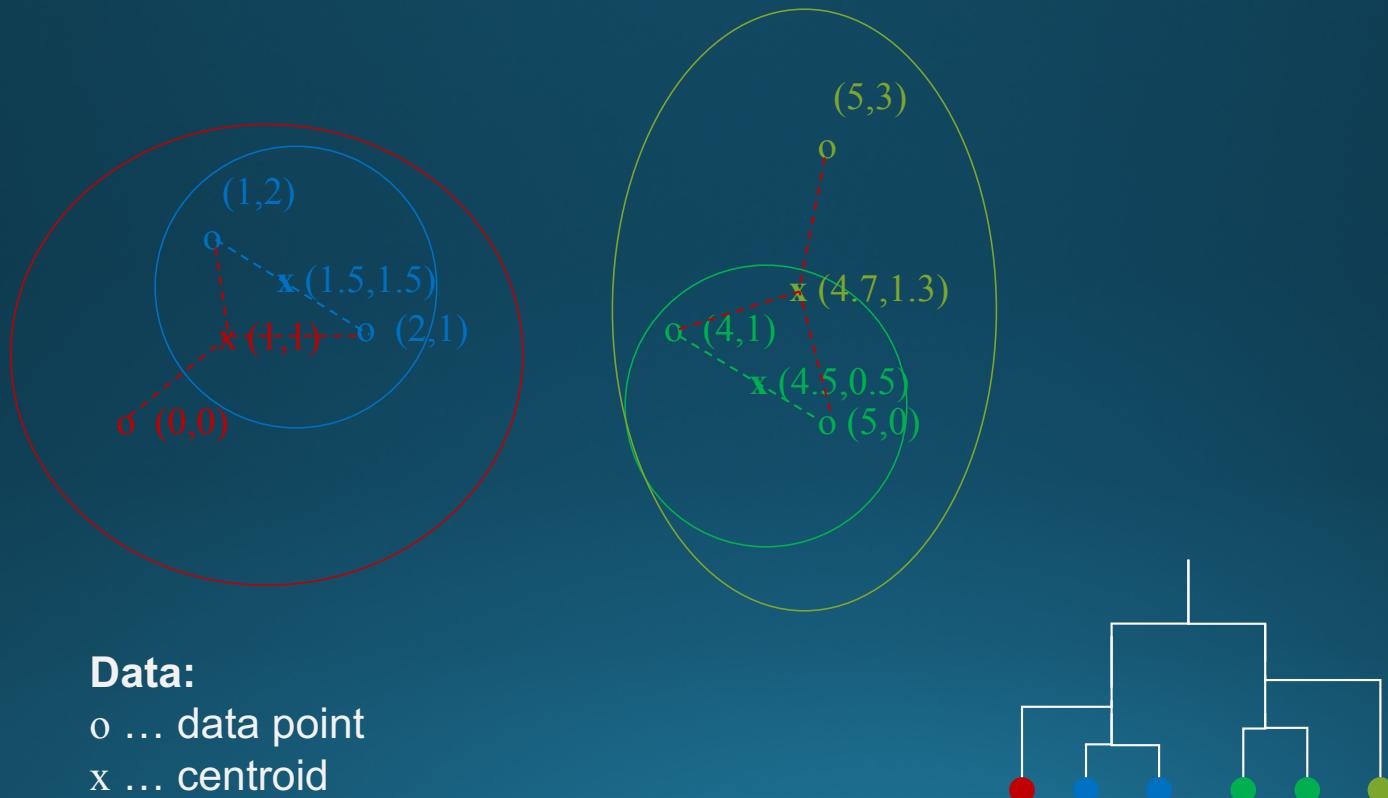
- **Key operation:**
Repeatedly combine two nearest clusters
- **Three important questions:**
 - 1) How do you represent a cluster of more than one point?
 - 2) How do you determine the “nearness” of clusters?
 - 3) When to stop combining clusters?



Hierarchical Clustering

- **Key operation:** Repeatedly combine two nearest clusters
- **(1) How to represent a cluster of many points?**
 - **Key problem:** As you merge clusters, how do you represent the “location” of each cluster, to tell which pair of clusters is closest?
- **Euclidean case:** each cluster has a *centroid* = average of its (data)points
- **(2) How to determine “nearness” of clusters?**
 - Measure cluster distances by distances of centroids

Example: Hierarchical clustering



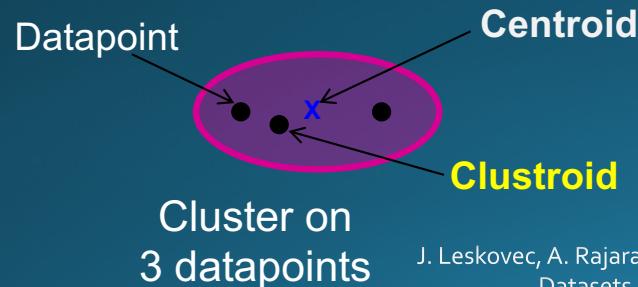
And in the Non-Euclidean Case?

What about the Non-Euclidean case?

- The only “locations” we can talk about are the points themselves
 - i.e., there is no “average” of two points
- Approach 1:
 - (1) How to represent a cluster of many points?
clustroid = (data)point “closest” to other points
 - (2) How do you determine the “nearness” of clusters? Treat clustroid as if it were centroid, when computing inter-cluster distances

“Closest” Point?

- (1) How to represent a cluster of many points?
clustroid = point “closest” to other points
- Possible meanings of “closest”:
 - Smallest maximum distance to other points
 - Smallest average distance to other points
 - Smallest sum of squares of distances to other points
 - For distance metric d clustroid c of cluster C is:



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

$$\min_c \sum_{x \in C} d(x, c)^2$$

Centroid is the avg. of all (data)points in the cluster. This means centroid is an “artificial” point.

Clustroid is an **existing** (data)point that is “closest” to all other points in the cluster.

Defining “Nearness” of Clusters

- (2) How do you determine the “nearness” of clusters?
 - Approach 2:
Intercluster distance = minimum of the distances between any two points, one from each cluster
 - Approach 3:
Pick a notion of “**cohesion**” of clusters, e.g., maximum distance from the clustroid
 - Merge clusters whose **union** is most cohesive

Cohesion

- **Approach 3.1:** Use the **diameter** of the merged cluster = maximum distance between points in the cluster
- **Approach 3.2:** Use the **average distance** between points in the cluster
- **Approach 3.3:** Use a **density-based approach**
 - Take the diameter or avg. distance, e.g., and divide by the number of points in the cluster

Implementation

- **Naïve implementation of hierarchical clustering:**
 - At each step, compute pairwise distances between all pairs of clusters, then merge
 - $O(N^3)$
- Careful implementation using priority queue can reduce time to $O(N^2 \log N)$
 - **Still too expensive for really big datasets that do not fit in memory**

k -means Algorithm(s)

- Assumes Euclidean space/distance
- Start by picking k , the number of clusters
- Initialize clusters by picking one point per cluster
 - **Example:** Pick one point at random, then $k-1$ other points, each as far away as possible from the previous points

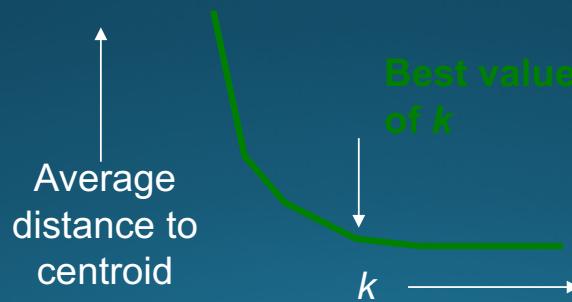
Populating Clusters

- 1) For each point, place it in the cluster whose current centroid it is nearest
- 2) After all points are assigned, update the locations of centroids of the k clusters
- 3) Reassign all points to their closest centroid
 - Sometimes moves points between clusters
- **Repeat 2 and 3 until convergence**
 - **Convergence:** Points don't move between clusters and centroids stabilize

Getting the k right

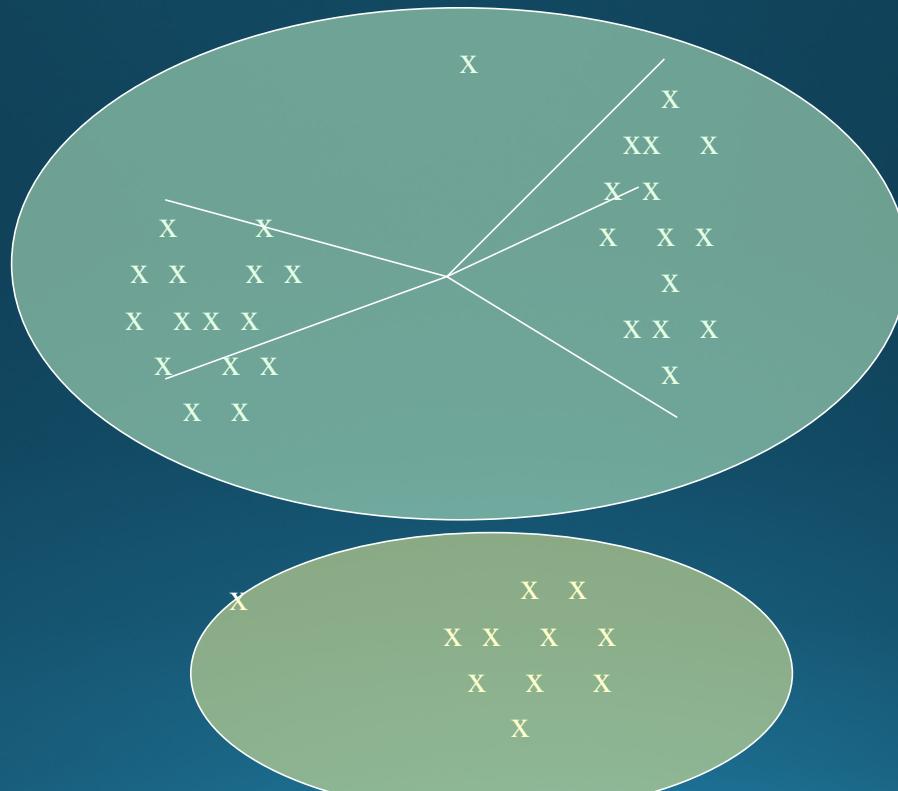
How to select k ?

- Try different k , looking at the change in the average distance to centroid as k increases
- Average falls rapidly until right k , then changes little



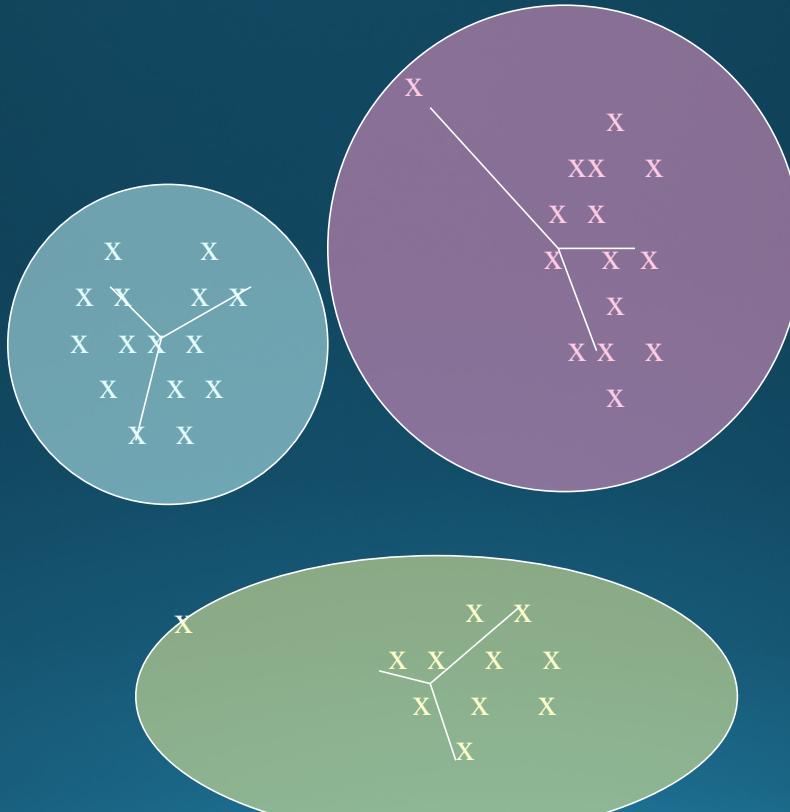
Example: Picking k

Too few;
many long
distances
to centroid.



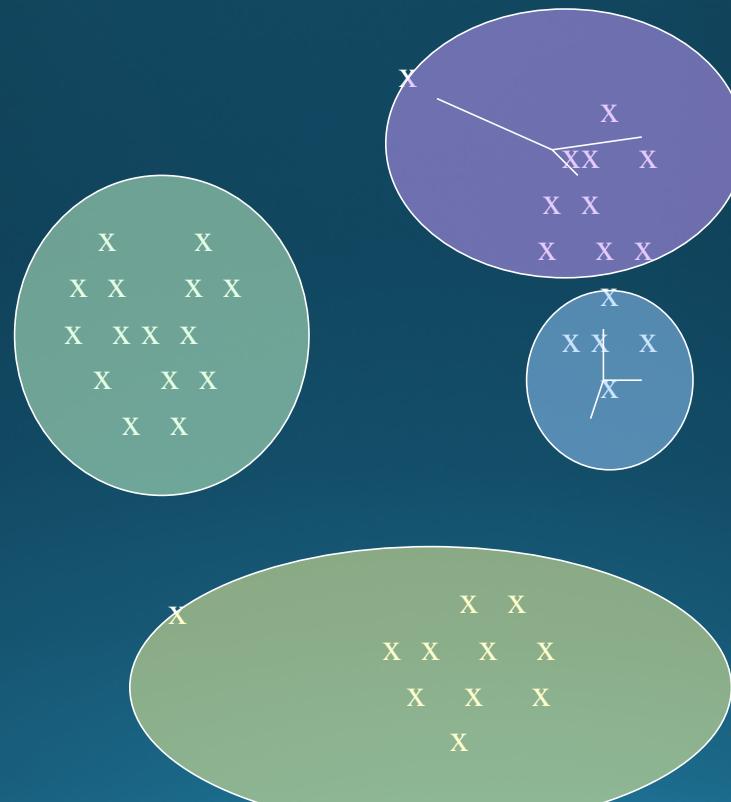
Example: Picking k

Just right;
distances
rather short.



Example: Picking k

Too many;
little improvement
in average
distance.



More K-means examples

- <http://www.naftaliharris.com/blog/visualizing-k-means-clustering/>

Graph Partitioning

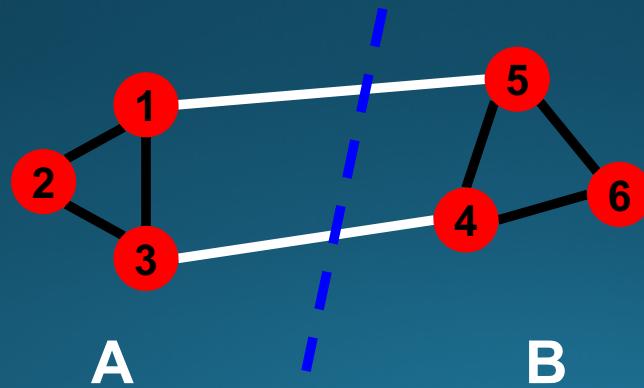
- Undirected graph
- Bi-partitioning task:
 - Divide vertices into two disjoint groups



- Questions:
 - How can we define a “good” partition of?
 - How can we efficiently identify such a partition?

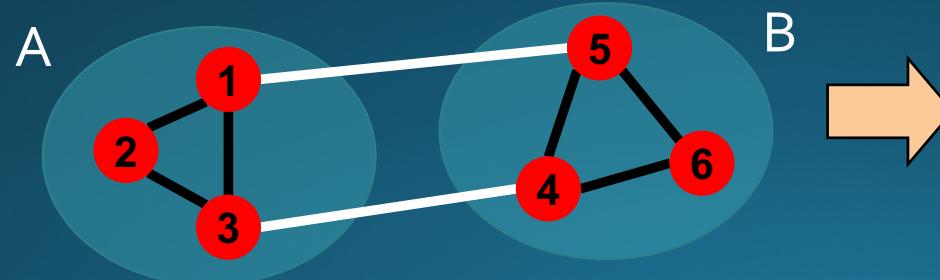
Graph Partitioning

- **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:



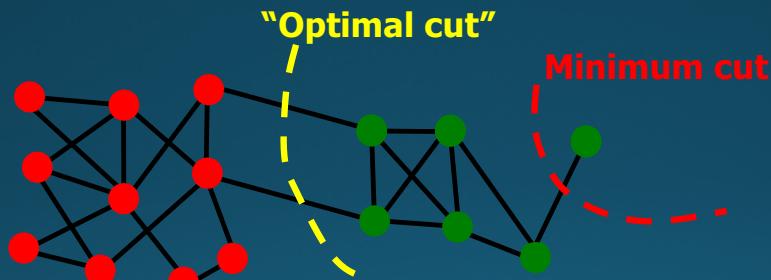
$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}$$

$$cut(A, B) = 2$$

Graph Cut Criterion

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

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



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

Graph Cut Criteria

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

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

: 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

Spectral Graph Partitioning

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

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad y_i = \sum_{j=1}^n A_{ij} x_j = \sum_{(i,j) \in E} x_j$$

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

What is the meaning of Ax ?

- j^{th} coordinate of $A \cdot x$:

- Sum of the x -values of neighbors of j
- Make this a new value at node j

- **Spectral Graph Theory:**

- Analyze the “spectrum” of matrix representing
- **Spectrum:** Eigenvectors of a graph, ordered by the magnitude (strength) of their corresponding eigenvalues :

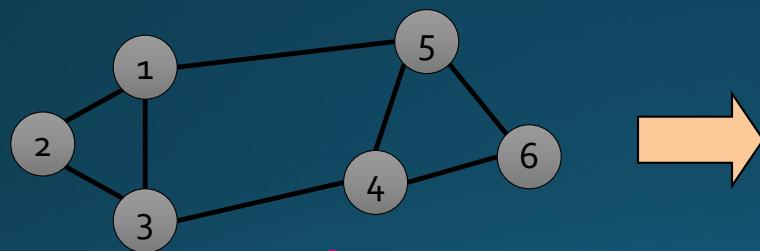
$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$$A \cdot x = \lambda \cdot x$$

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

Matrix Representations

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

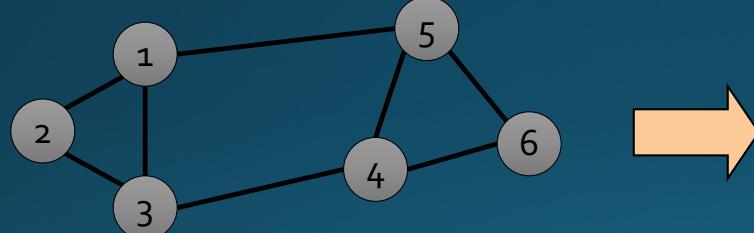


	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

- **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

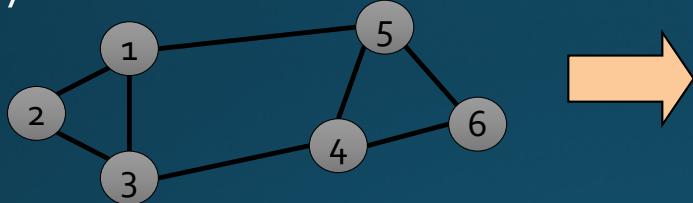


	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

Matrix Representations

- **Laplacian matrix (L):**

- $n \times n$ symmetric matrix



- **What is trivial eigenpair?**

- **Important properties:**

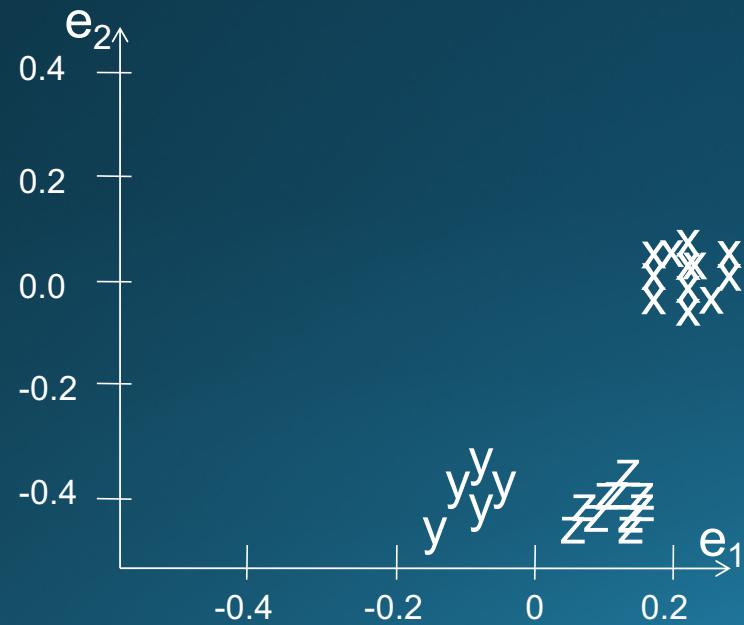
- **Eigenvalues** are non-negative real numbers
- **Eigenvectors** are real and orthogonal

	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

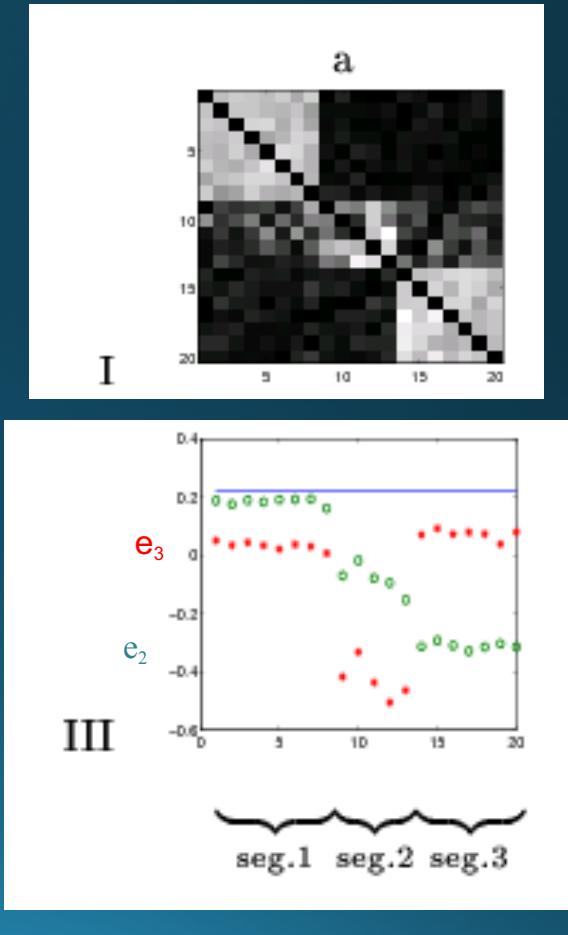
$$L = D - A$$

Spectral Clustering

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

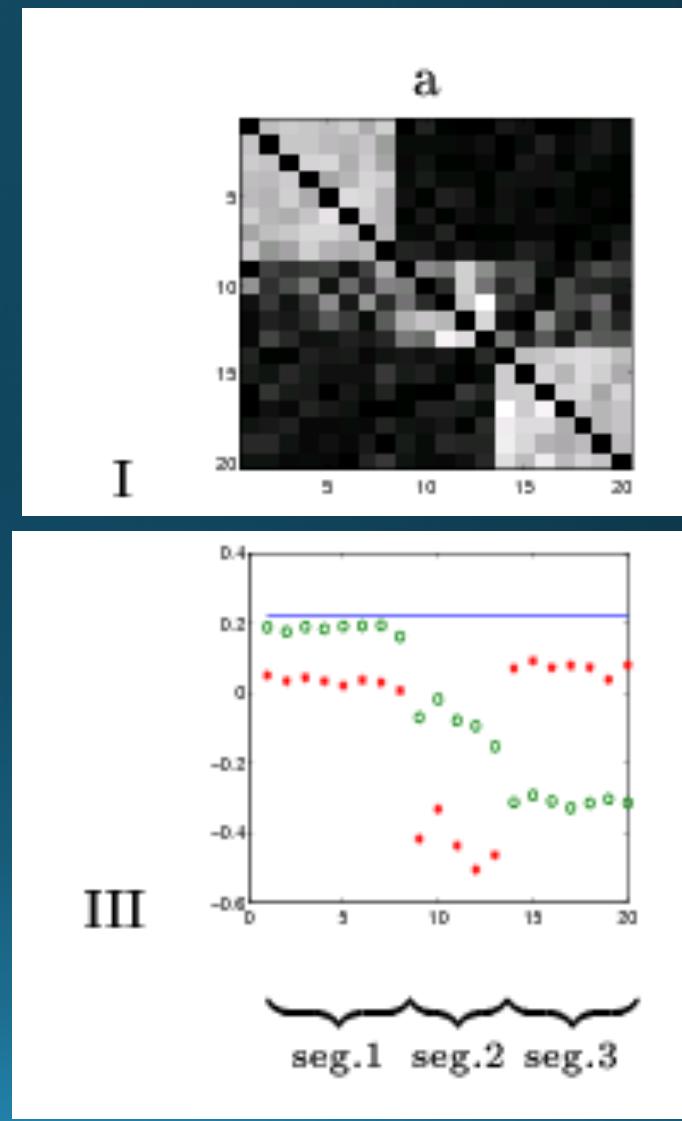


[Shi & Meila, 2002]



Spectral Clustering

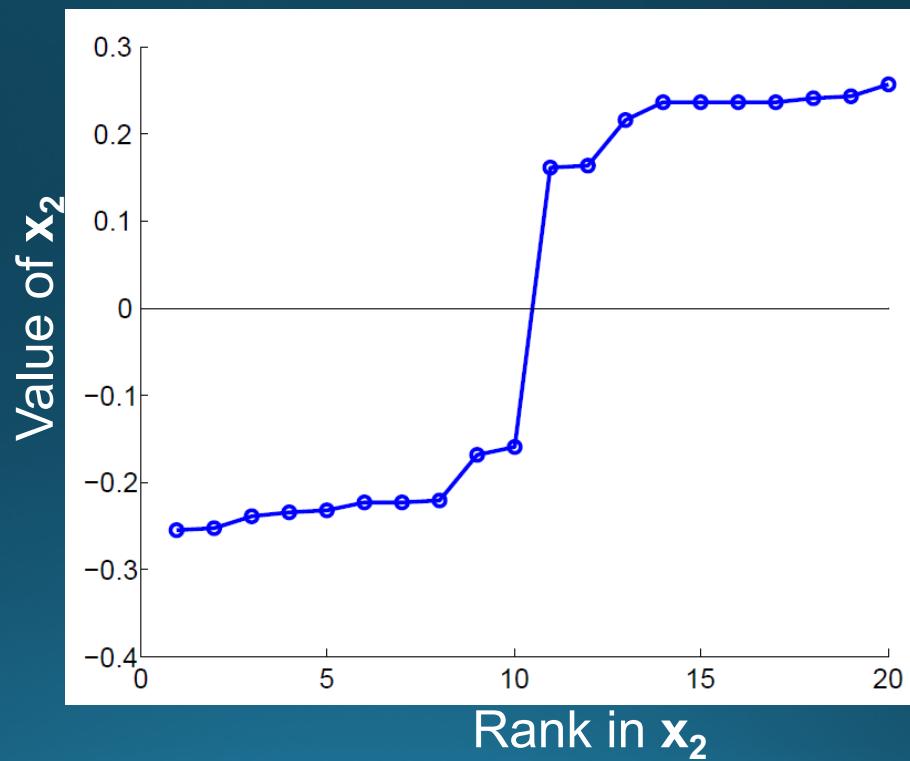
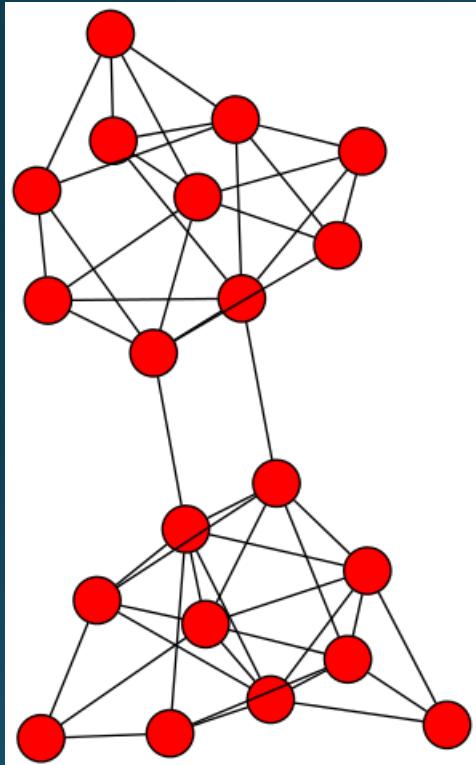
- If W is connected but roughly block diagonal with k blocks, then
- the top eigenvector is a constant vector
- the next k eigenvectors are roughly piecewise constant with “pieces” corresponding to blocks



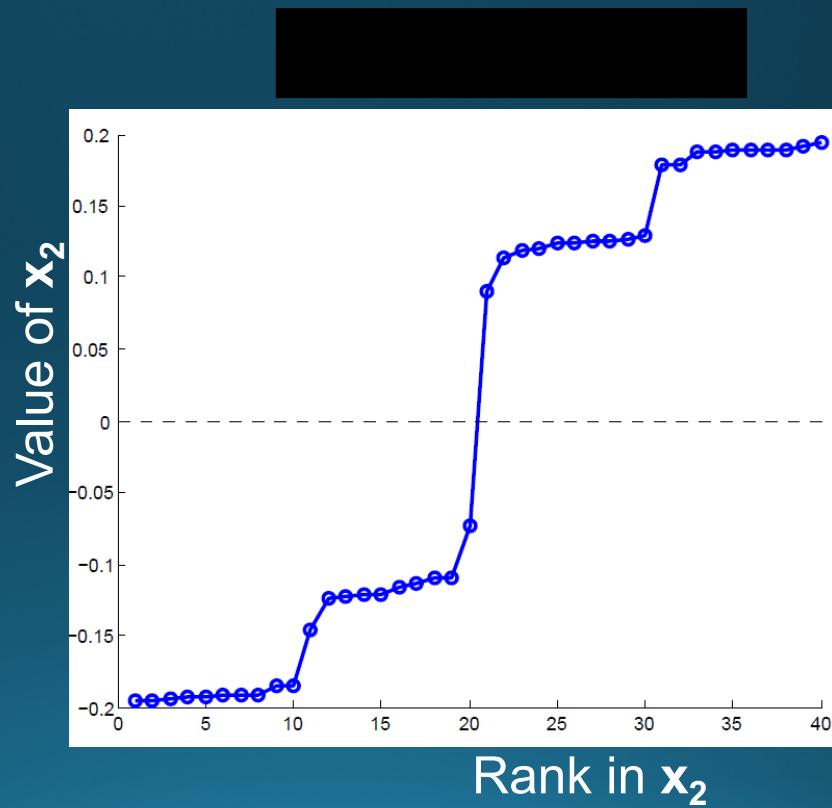
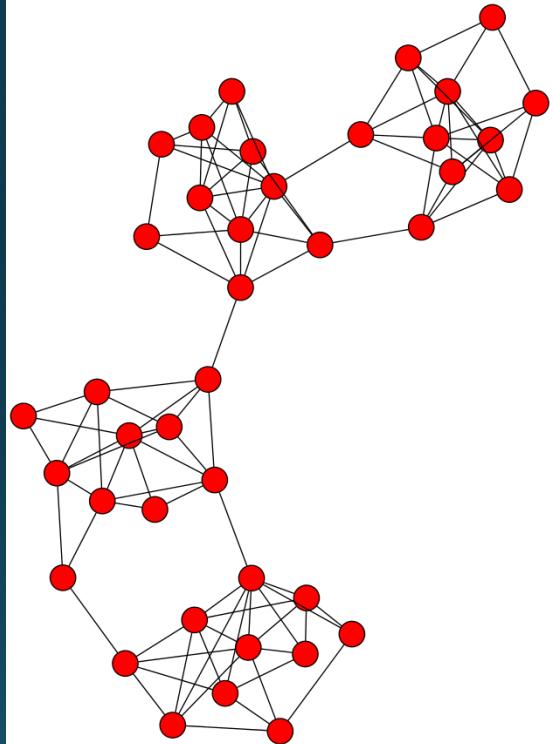
Spectral Clustering

- Outline of the algorithm:
 1. Find the top $k+1$ eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_{k+1}$
 2. Discard the “top” one (the “trivial pair”)
 3. Replace every node a with k -dimensional vector $x_a = \langle \mathbf{v}_2(a), \dots, \mathbf{v}_{k+1}(a) \rangle$
 4. Cluster with k -means

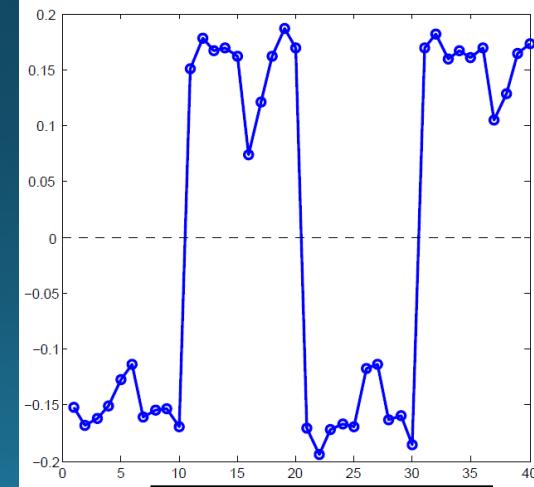
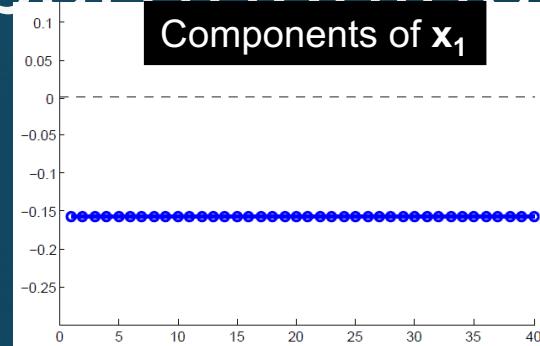
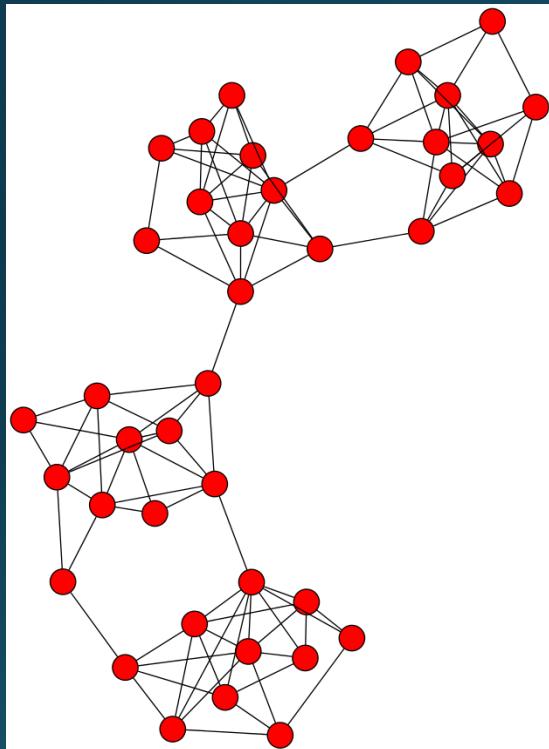
Example: Spectral Partitioning



Example: Spectral Partitioning



Example: Spectral partitioning



J. Leskovec, A. Rajaraman, J. Ullman: Mining of
Datasets, <http://www.mmds.org>

Components of x_3

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 k orthogonal basis vectors
- Multiple eigenvectors prevent instability due to information loss

More terms

- If A is an adjacency matrix (maybe weighted) and D is a (diagonal) matrix giving the degree of each node
- Then $L_u = D - A$ is the (*unnormalized*) *Laplacian*
 - $W = AD^{-1}$ is a *probabilistic adjacency matrix*
- $L_n = I - D^{-1/2}AD^{-1/2}$ is the (*normalized or random-walk*) *Laplacian*
- The largest eigenvectors of W correspond to the smallest eigenvectors of L_n
 - So sometimes people talk about “*bottom eigenvectors of the Laplacian*”

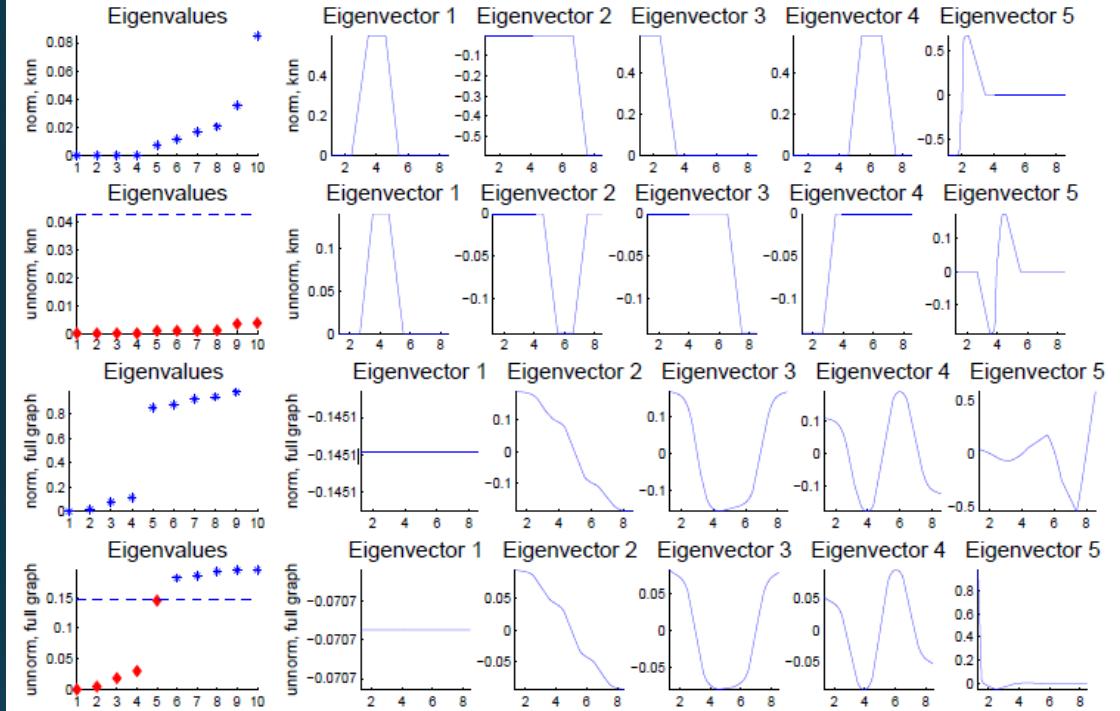
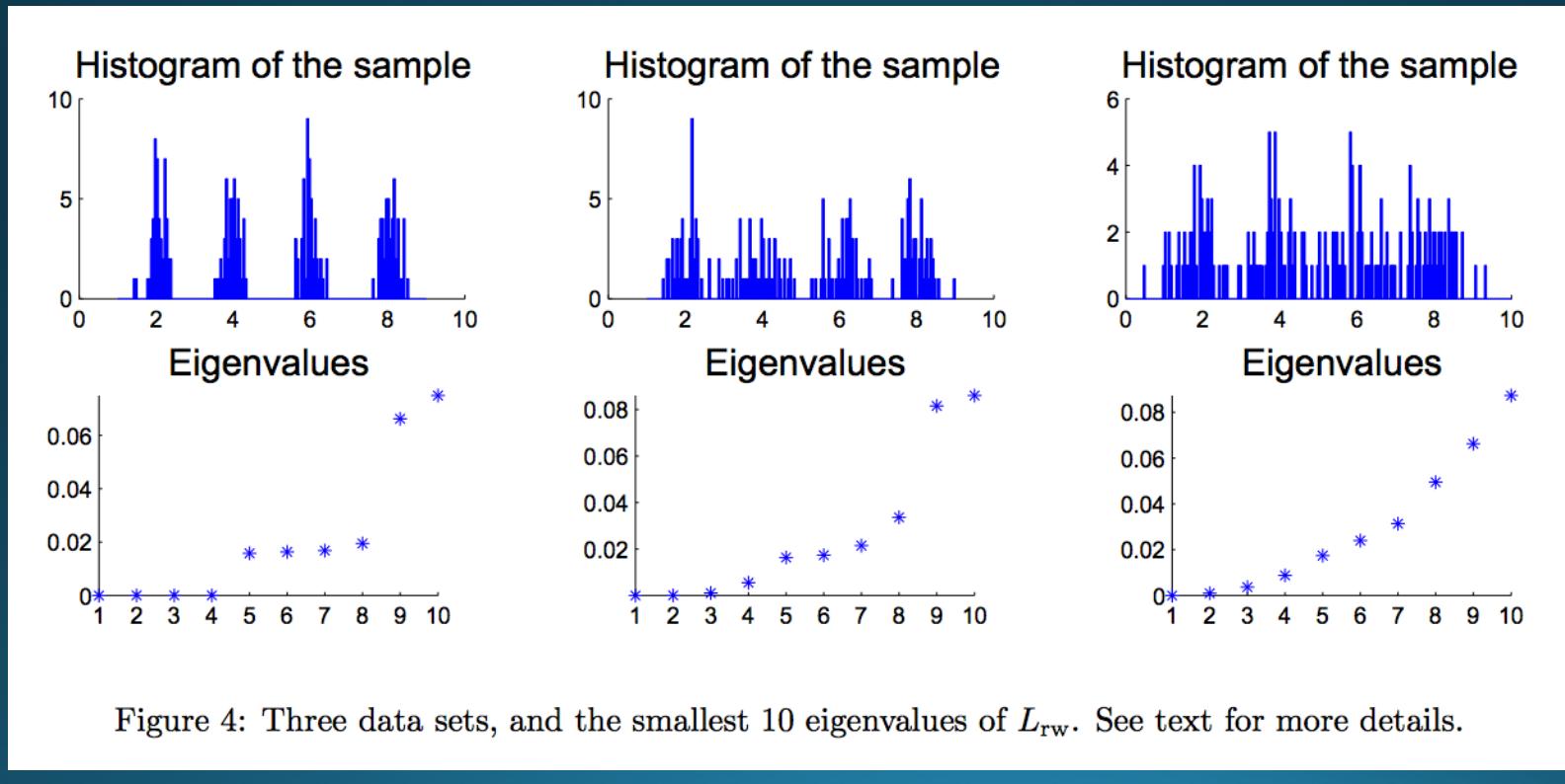


Figure 1: Toy example for spectral clustering where the data points have been drawn from a mixture of four Gaussians on \mathbb{R} . Left upper corner: histogram of the data. First and second row: eigenvalues and eigenvectors of L_{rw} and L based on the k -nearest neighbor graph. Third and fourth row: eigenvalues and eigenvectors of L_{rw} and L based on the fully connected graph. For all plots, we used the Gaussian kernel with $\sigma = 1$ as similarity function. See text for more details.

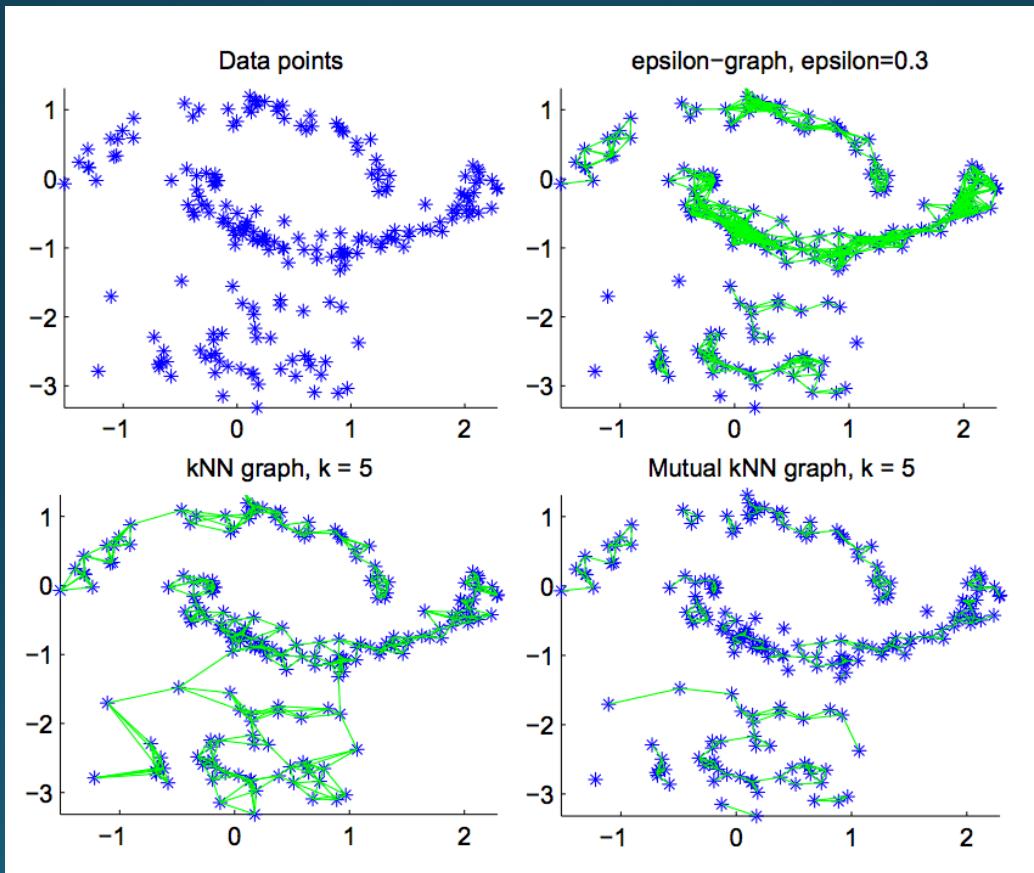
K-nn graph
(easy)

Fully
connected
graph,
weighted by
distance

Spectrum from Data



Similarity Graphs for Spectral Clustering

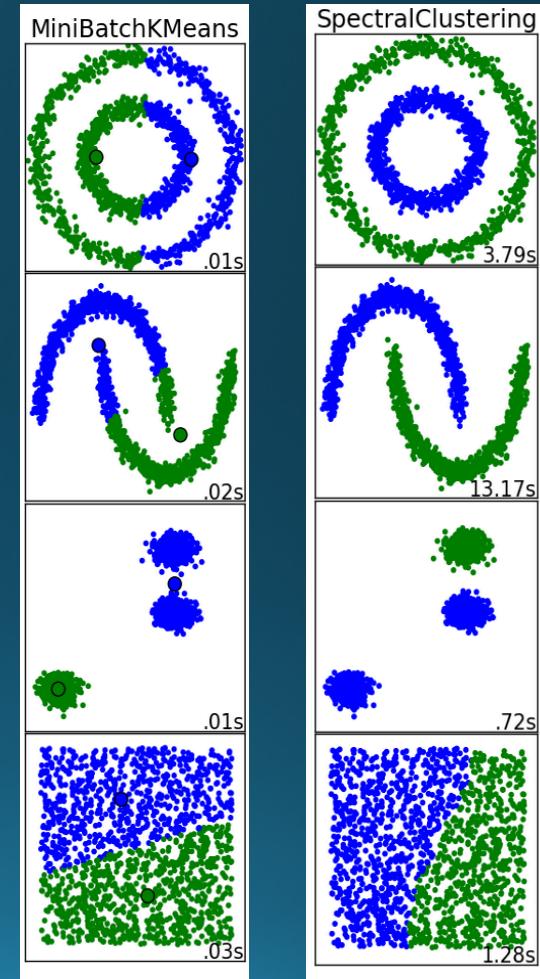


Spectral Clustering: Pros and Cons

- Elegant, and well-founded mathematically
- Works quite well when relations are approximately transitive (like similarity)
- Does not assume any form of the data (compare to K-means)
- 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



References

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