University of Colorado **Boulder**

Department of Computer Science

CSCI 5622: Machine Learning

Chris Ketelsen

Lecture 23:
Spectral Clustering

Unsupervised Learning

Find **hidden structure** in data

Structure that can't be formally fully observed

$$\underset{\textit{Data}}{\mathbf{X}} \rightarrow \underset{\textit{Hidden Structure}}{\mathbf{Z}} \rightarrow \underset{\textit{Answer}}{\mathbf{Y}}$$

Data is simply $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{m}$

Don't know $\mathbf{Y}$. Try to get at $\mathbf{Z}$

Clustering

One important unsupervised method is **clustering**

Goal: Organize data into classes such that

- data have high in-class similarity
- data have low out-of-class similarity

Clustering - Similarity

similarity

[sim-uh-lar-i-tee]

noun

the state of being similar; likeness; resemblance; an aspect, trait, or feature like or resembling another

Clustering - Similarity

Usually we just know it when we see it

Clustering - Similarity

We'll call $d(\mathbf{x}, \mathbf{y})$ the similarity measure of $\mathbf{x}$ and $\mathbf{y}$

Examples:

Euclidean Distance: $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$

Edit Distance: $d(\mathbf{x}, \mathbf{y})$ = # replace, insert, deletes to turn $\mathbf{x}$ into $\mathbf{y}$

$$d(\texttt{kitten}, \texttt{sitting}) = 3$$

$$\texttt{kitten} \rightarrow \texttt{sitten} \rightarrow \texttt{sittin} \rightarrow \texttt{sitting}$$

What properties make a good similarity measure?

Clustering - Similarity

We'll call $d(\mathbf{x}, \mathbf{y})$ the similarity measure of $\mathbf{x}$ and $\mathbf{y}$

Properties:

$$\begin{aligned}
\text{Symmetry} \quad & d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x}) \\
\text{Self-Consistency} \quad & d(\mathbf{x}, \mathbf{x}) = 0 \\
\text{Positivity} \quad & d(\mathbf{x}, \mathbf{y}) = 0 \text{ iff } \mathbf{x} = \mathbf{y} \\
\text{Triangle Inequality} \quad & d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})
\end{aligned}$$

OK, so say we have a good similarity measure.

How do we find clusters in the data?

K-Means

General Idea:

pick K initial cluster means

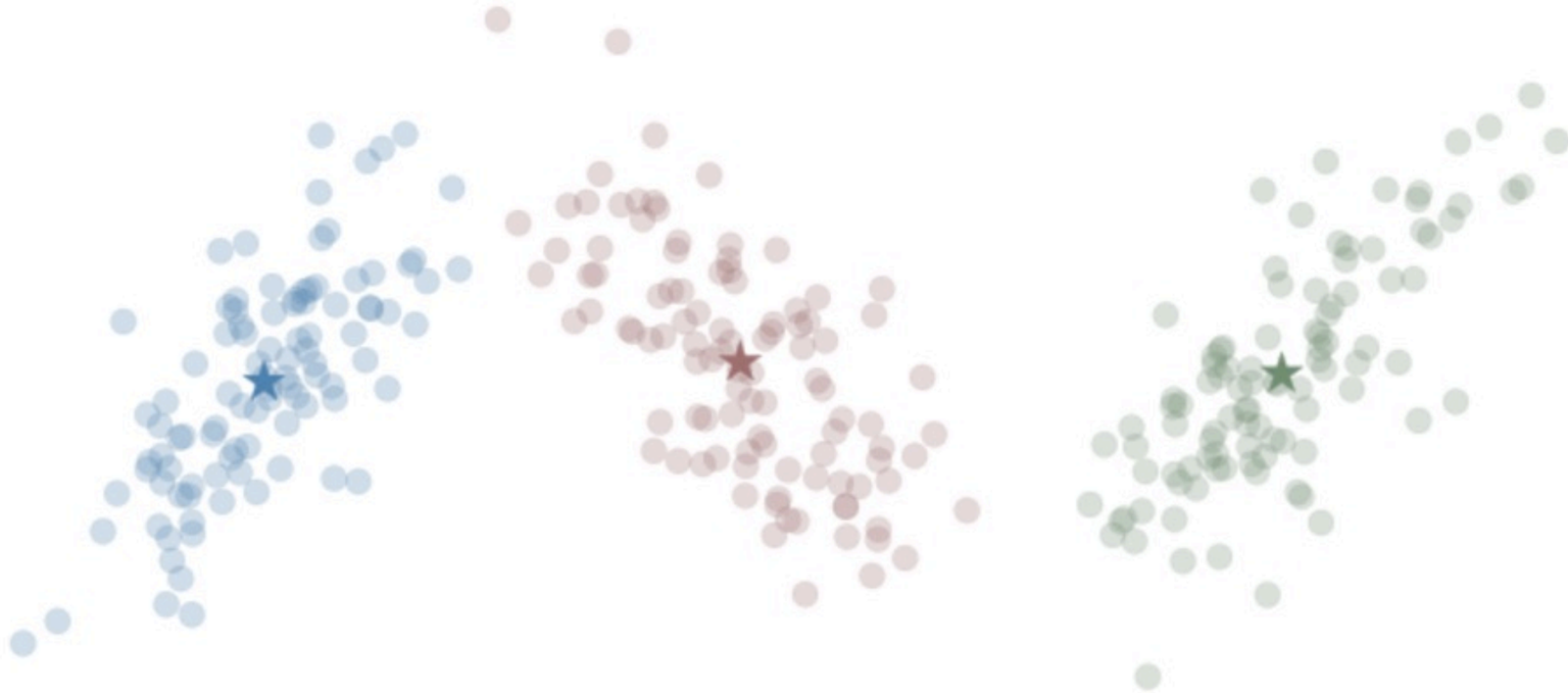
do until convergence ...

- associate examples closest to mean k with cluster k
- update cluster means with current examples in cluster k

Stop when:

- cluster assignments don't change
- cluster means don't change (too much)

K-Means Example



K-Means - Algorithm

Strengths:

- Simple to understand
- Efficient - time complexity $\mathcal{O}(nKT)$ for $\mathbf{x} \in \mathbb{R}^n$
- Simple to implement

K-Means

Weaknesses:

- Doesn't really work with categorical data
- Usually only converges to local minimum
- Have to determine number of clusters
- Can be sensitive to outliers
- Only generates convex clusters

Gaussian Mixture Models

Guassian Mixture Models (or GMMs) are a probabilistic generalization of K-Means

In K-Means we made *hard* cluster assignments.

That is, we said $\mathbf{x}_i$ definitely belongs to cluster k

GMM is utilizes *soft* cluster assignments

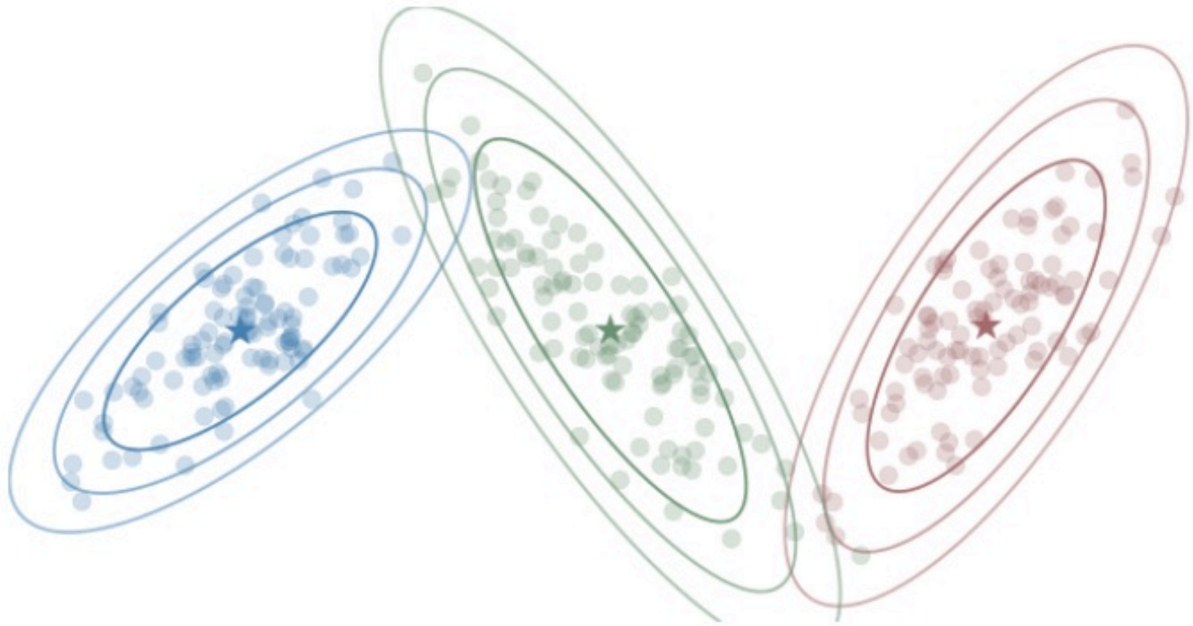
That is, we'll say $\mathbf{x}_i$ belongs to cluster $k = \{1, \ldots, K\}$ with some probability

We can then estimate that probability for all k and, if need be, assign $\mathbf{x}_i$ to the cluster with the highest probability

Gaussian Mixture Models

Example: Consider our toy data set again

After 9 EM iterations



GMMs and the EM Algorithm

GMMs with the EM Algorithm suffer from some of the same problems as K-Means

- Doesn't really work with categorical data
- Usually only converges to a local minimum
- Have to determine the number of clusters
- Only generates convex clusters

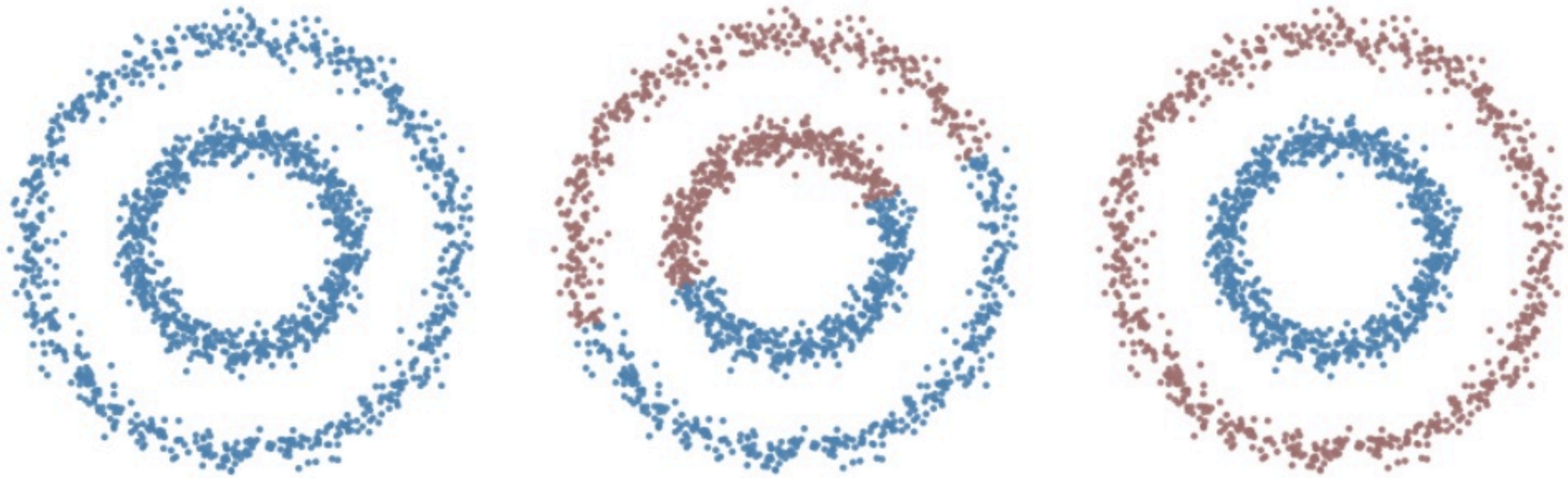
But, it also has certain advantages

- The clusters are allowed different shapes
- We get a soft partitioning of the data

Spectral Clustering - Motivation

Why Spectral Clustering?

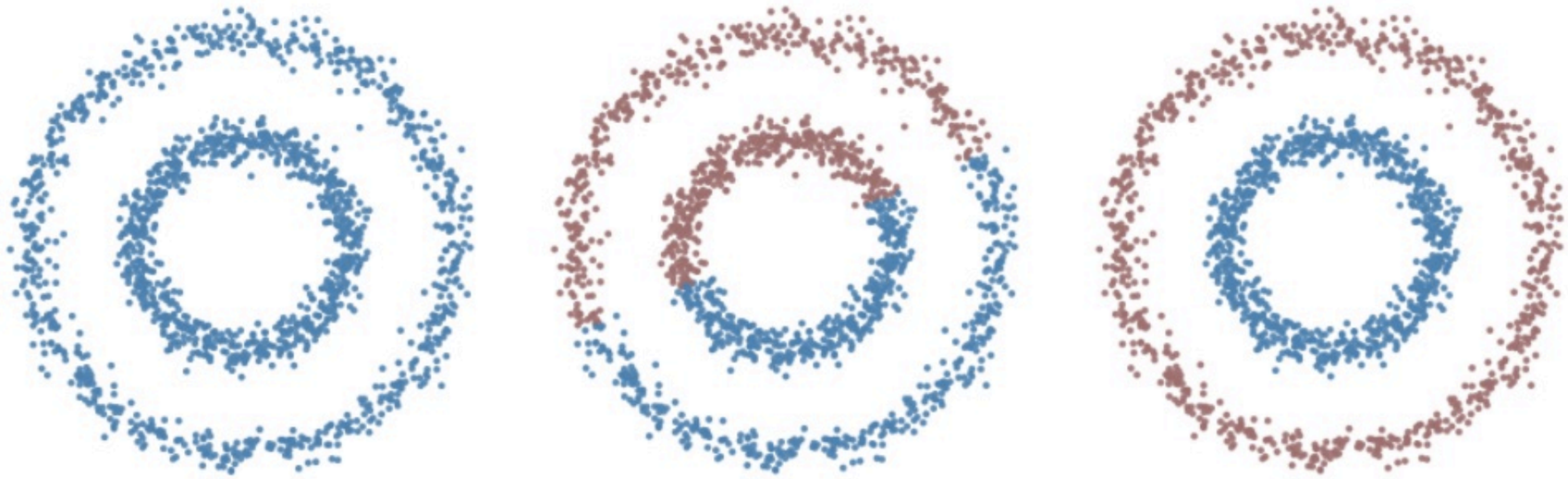
- Unlike GMM/K-Means, Spectral Clustering makes no assumptions about shapes of clusters



Spectral Clustering - Motivation

Why Spectral Clustering?

- Unlike GMM/K-Means, Spectral Clustering makes no assumptions about shapes of clusters



- GMM/K-Means were iterative, sensitive to starting points

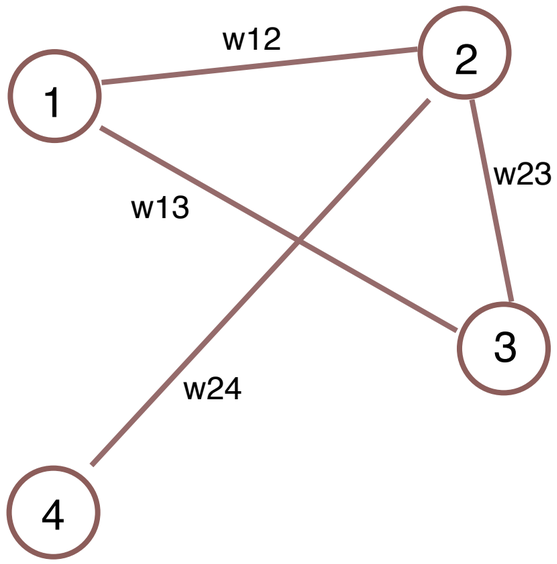
Spectral Clustering - RoadMap

1. Embed data in a similarity graph
2. Compute eigenvectors of matrix associated with graph
3. Embed data in low-dimensional space
4. Apply classic clustering methods to low-dimensional data

Representing a Graph as a Matrix

- Adjacency Matrix: $n \times n$ and symmetric

$$W_{ij} = \begin{cases} w_{ij} & \text{weight of edge connecting nodes } i \text{ and } j \\ 0 & \text{if no edge between nodes } i \text{ and } j \end{cases}$$



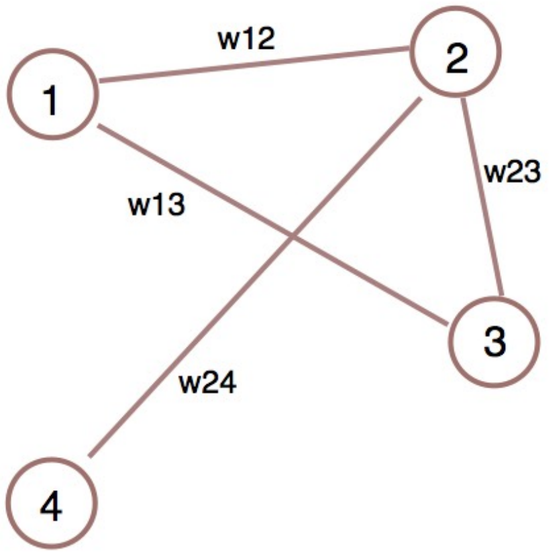
$$W =$$

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
$\mathbf{x}_1$	0	w_{12}	w_{13}	0
$\mathbf{x}_2$	w_{12}	0	w_{23}	w_{24}
$\mathbf{x}_3$	w_{13}	w_{23}	0	0
$\mathbf{x}_4$	0	w_{24}	0	0

Representing a Graph as a Matrix

- Graph Laplacian Matrix: $n \times n$ and symmetric

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -w_{ij} & \text{if } i \neq j \end{cases} \quad \text{where} \quad d_i = \sum_{j \mid i \leftrightarrow j} w_{ij}$$



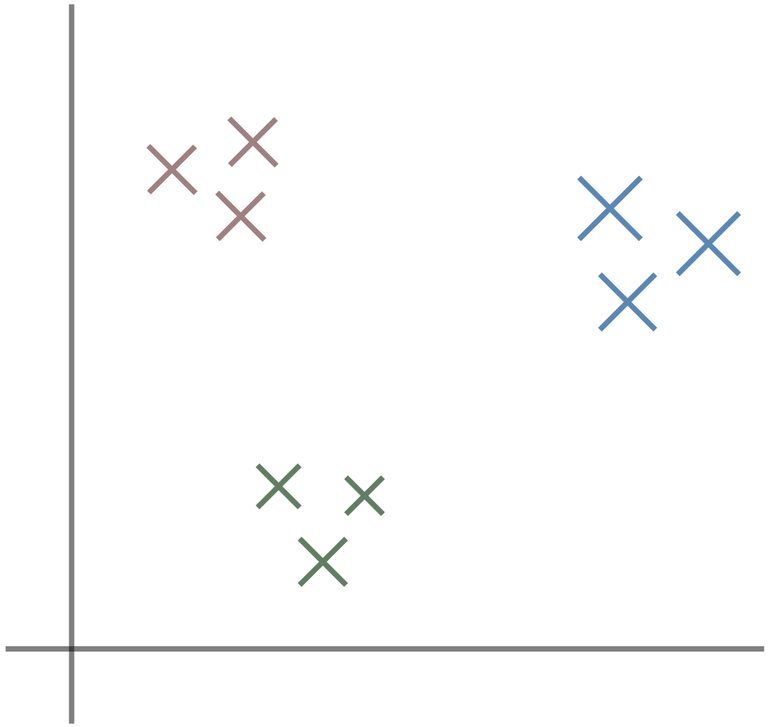
$$L =$$

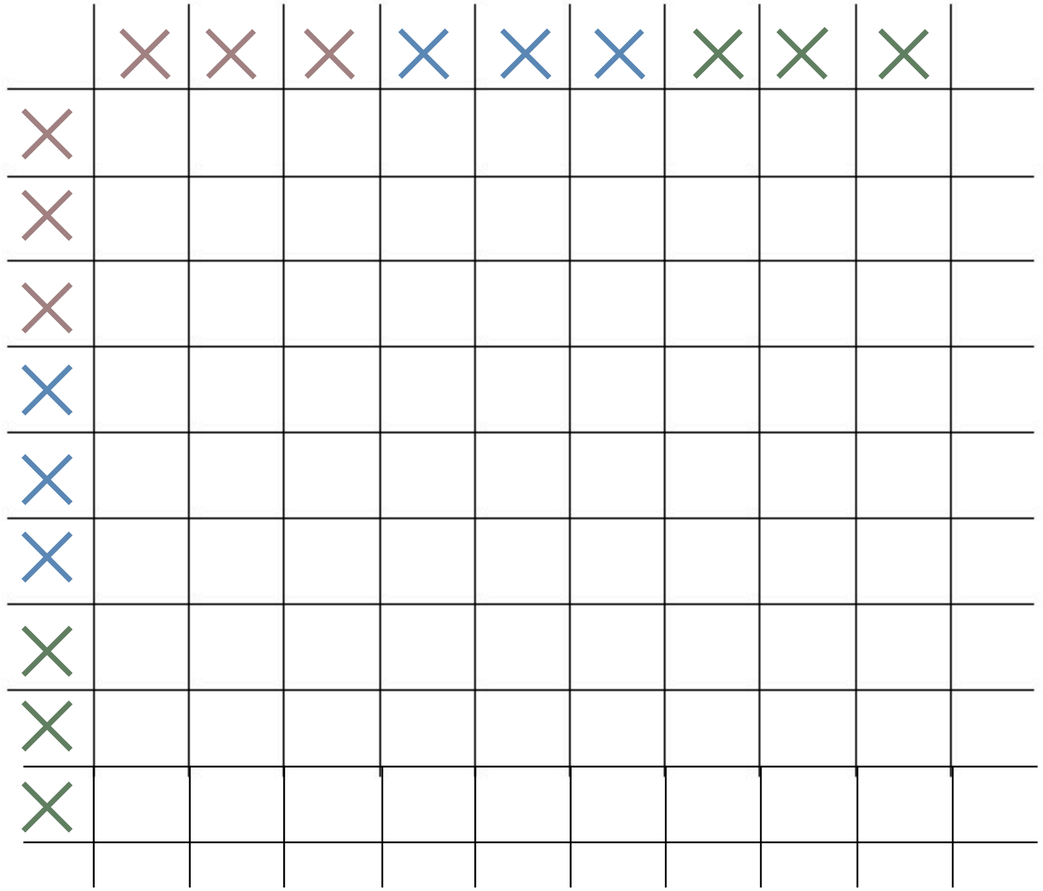
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
$\mathbf{x}_1$	d_1	$-w_{12}$	$-w_{13}$	0
$\mathbf{x}_2$	$-w_{12}$	d_2	$-w_{23}$	$-w_{24}$
$\mathbf{x}_3$	$-w_{13}$	$-w_{23}$	d_3	0
$\mathbf{x}_4$	0	$-w_{24}$	0	d_4

Representing a Graph as a Matrix

Representing Data as a Graph (and then a Matrix)

- Each example becomes a node
- Weights/edges between nodes come from similarity





Representing Data as a Graph (and then a Matrix)

But how do we compute the edge weights, w_{ij}?

Several popular choices:

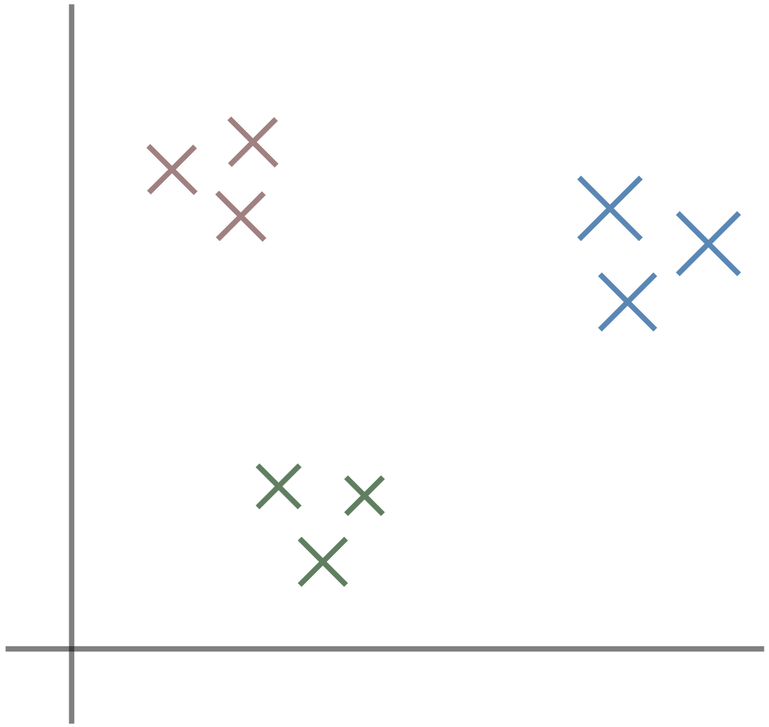
- Gaussian Kernel:

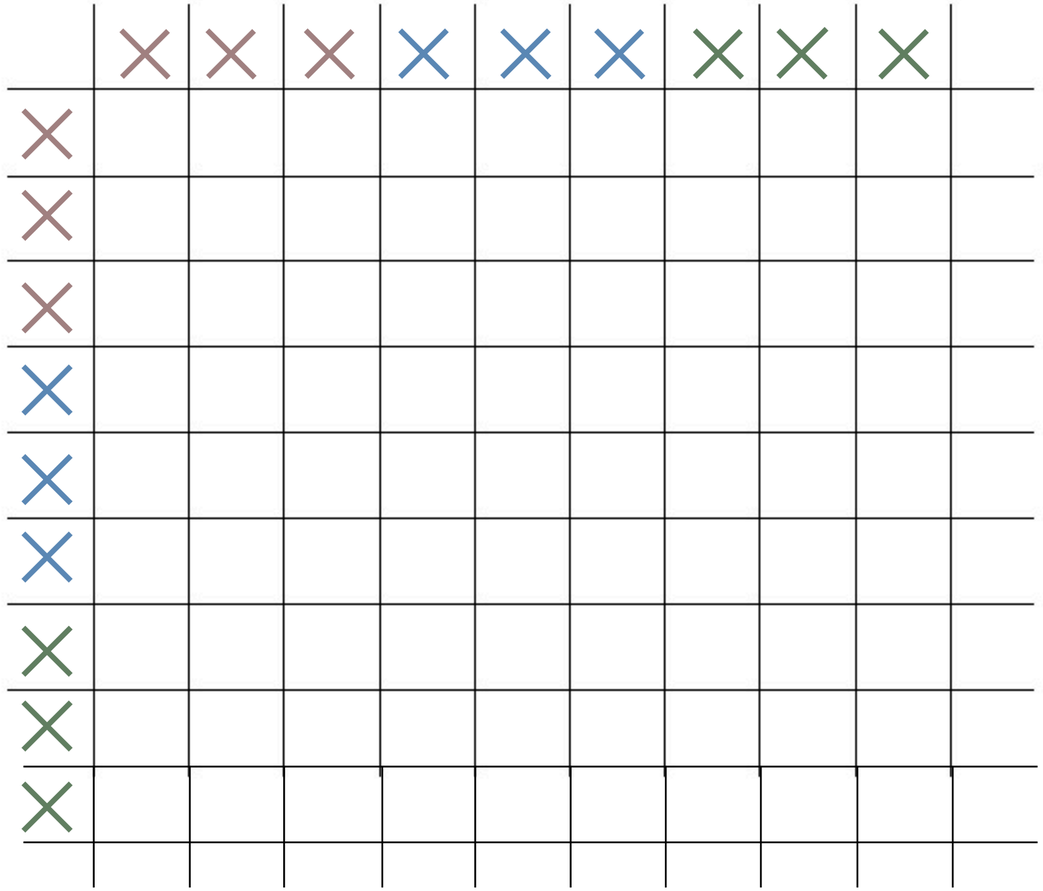
$$w_{ij} = \exp\left\{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right\}$$

- γ is a tuning parameter
- Results in a fully connected graph (dense L)
- Can threshold to make it sparser

Representing Data as a Graph (and then a Matrix)

- Each example becomes a node
- Weights/edges between nodes come from similarity





Representing Data as a Graph (and then a Matrix)

But how do we compute the edge weights, w_{ij}?

Several popular choices:

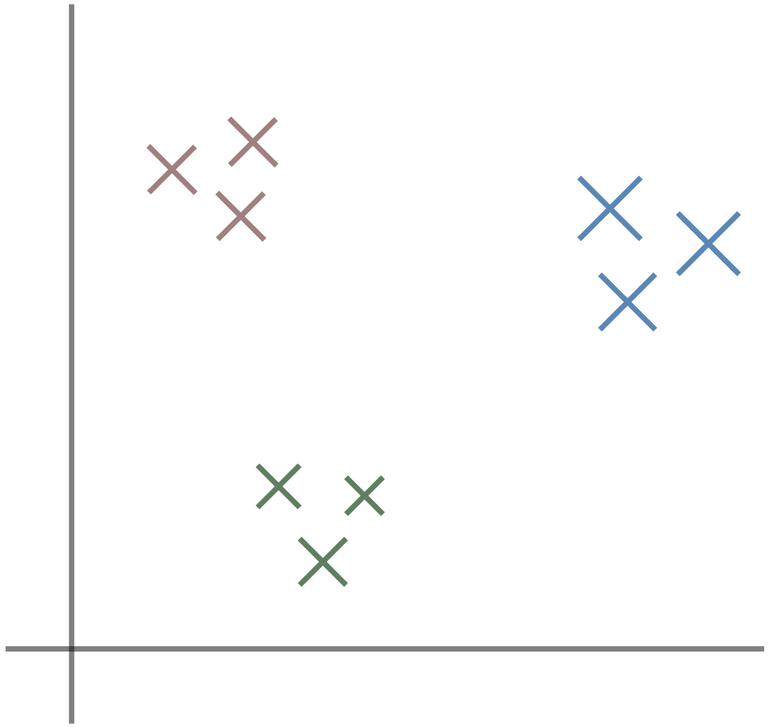
- Symmetric KNN:

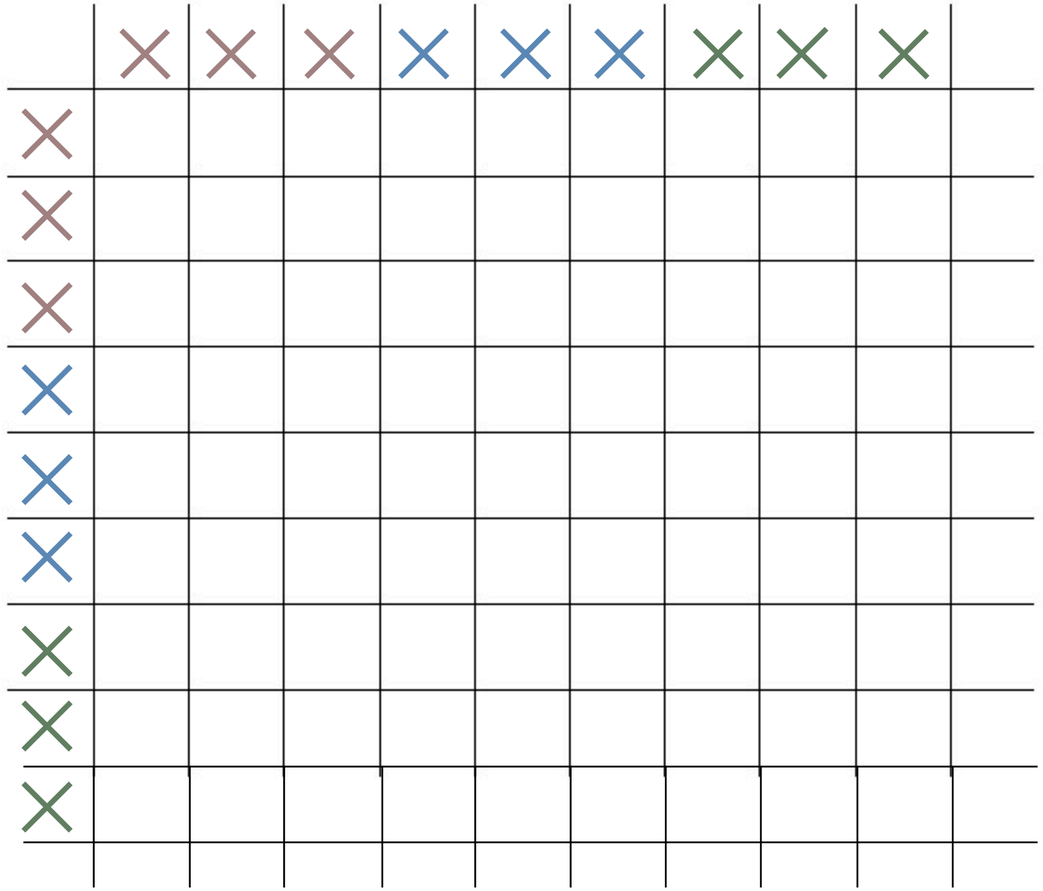
$$w_{ij} = \begin{cases} 1 & \text{if } i \text{ is SNN of } j \\ 0 & \text{otherwise} \end{cases}$$

- Guarantees sparse graph
- Have to be careful with symmetry

Representing Data as a Graph (and then a Matrix)

- Each example becomes a node
- Weights/edges between nodes come from similarity





Eigenvalues and Eigenvectors

OK, so we've constructed our Graph Laplacian based on our data

Now what?!

Def: Let A be a square $n \times n$ matrix. A nonzero n-length vector $\mathbf{v}$ is an eigenvector of A if there exists some scalar λ s.t. $A\mathbf{v} = \lambda\mathbf{v}$.

Fact 1: Except in pathological cases, A has n eigenvalue-eigenvector pairs.

Fact 2: If A is symmetric, its eigenvalues are real

Fact 3: If A is symmetric, its eigenvectors are orthogonal

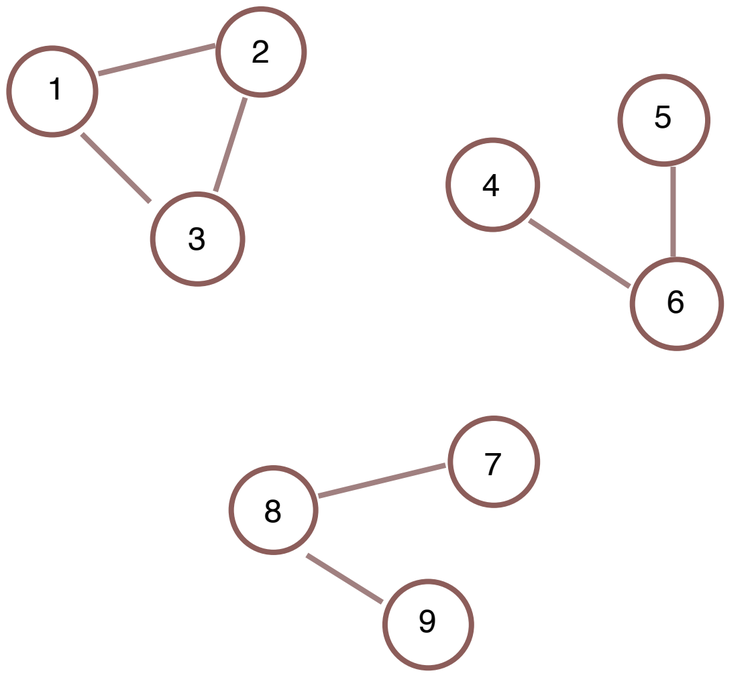
Eigenvalues and Eigenvectors

Proof of Fact 3:

Eigenvalues and Eigenvectors

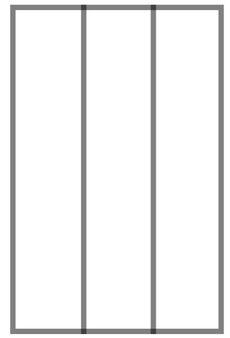
Fact 4: $\mathbf{v}_1 = \mathbb{1}$ is an evec of L with $\lambda_1 = 0$

Fact 5: If a graph G has K connected components, then the associated L has $\lambda_1 = \cdots = \lambda_k = 0$ and $\lambda_{k+1} > 0$



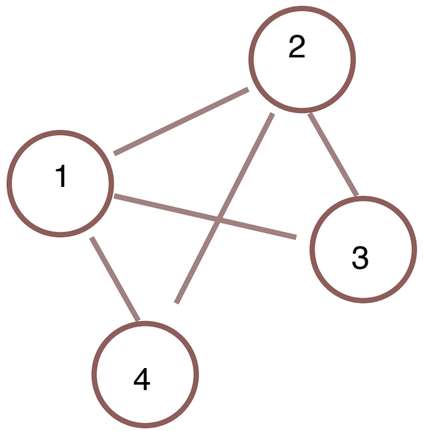
$$[\mathbf{v}_1 \quad \mathbf{v}_2 \quad \mathbf{v}_3]$$

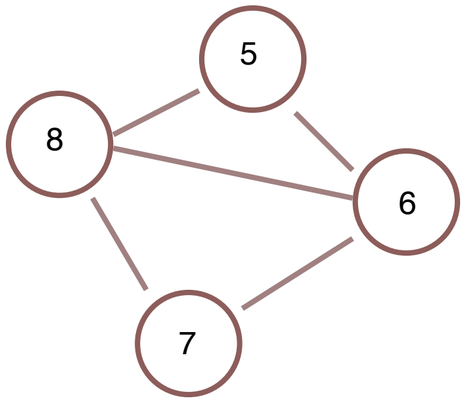
$$L = \begin{bmatrix} L_1 & 0 & 0 \\ 0 & L_2 & 0 \\ 0 & 0 & L_3 \end{bmatrix}$$



Towards the Fully Connected Case

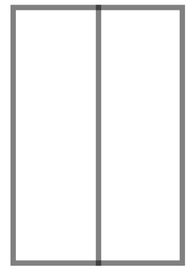
Suppose you have a graph with two connected components:





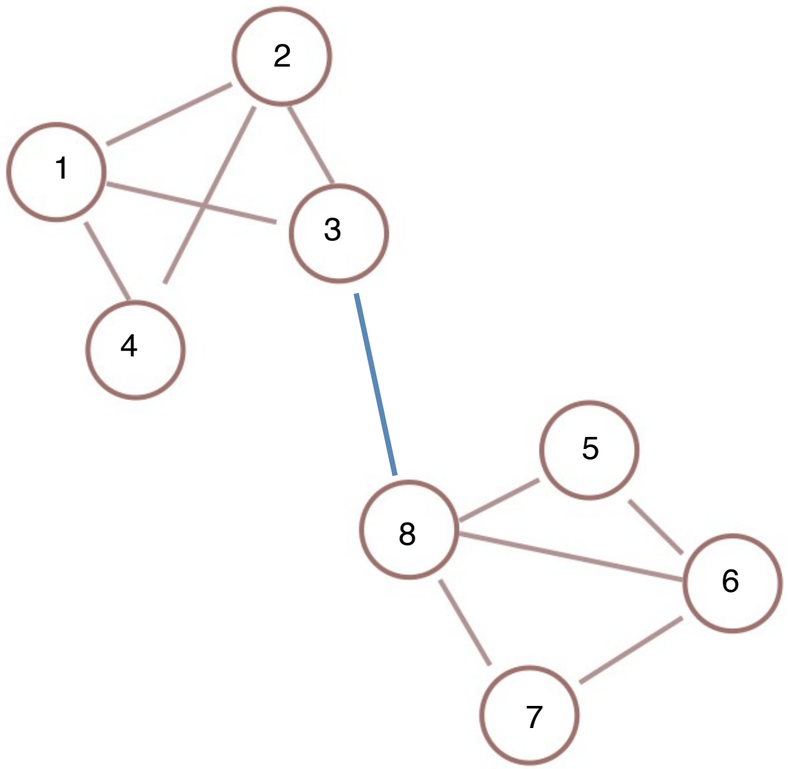
$$L = \begin{bmatrix} L_1 & 0 \\ 0 & L_2 \end{bmatrix}$$

$$[\mathbf{v}_1 \quad \mathbf{v}_2]$$



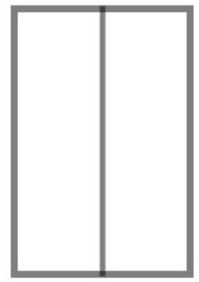
Towards the Fully Connected Case

Suppose suppose we connect them just a wee bit:



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

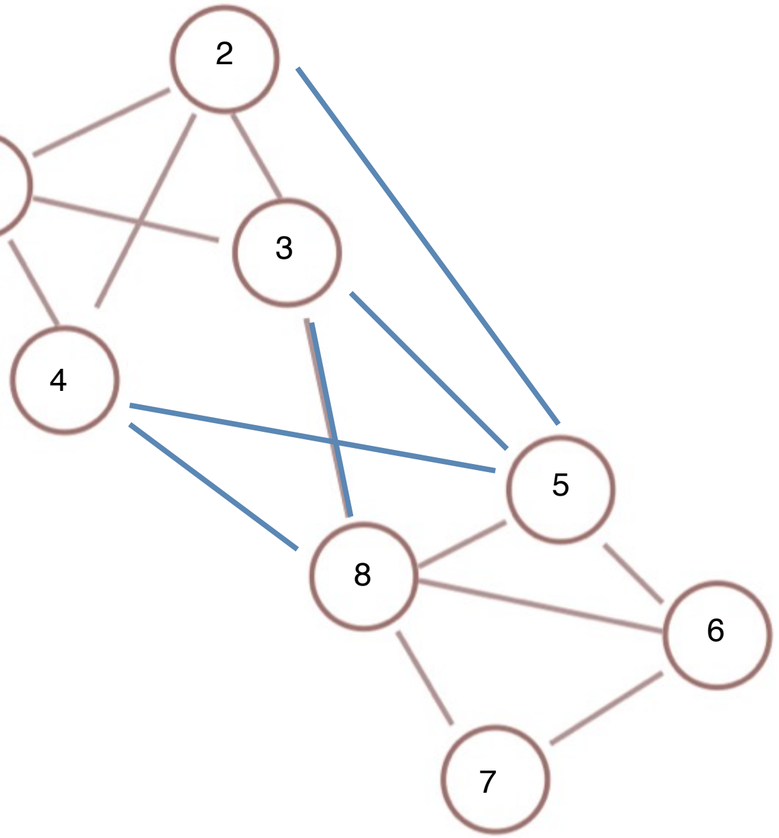
$[\mathbf{v}_1 \quad \mathbf{v}_2]$



We have $\lambda_1 = 0, \lambda_2 > 0$ and size of λ_2 tells you stength of conn.

Towards the Fully Connected Case

Suppose suppose we connect them just a wee bit:



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

$$[\mathbf{v}_1 \quad \mathbf{v}_2]$$

We have $\lambda_1 = 0, \lambda_2 > 0$ and size of λ_2 tells you stength of conn.

The Fiedler Vector and Bipartitioning

In the connected case, the first eigenvector is $\mathbf{v}_1 = \mathbb{1}$

The second eigenvector allows us to biparition the nodes

The eigenvector $\mathbf{v}_2$ is called the Fiedler Vector

K-Cluster Partitioning

If we want to find K clusters we get K eigenvectors

K-Cluster Partitioning

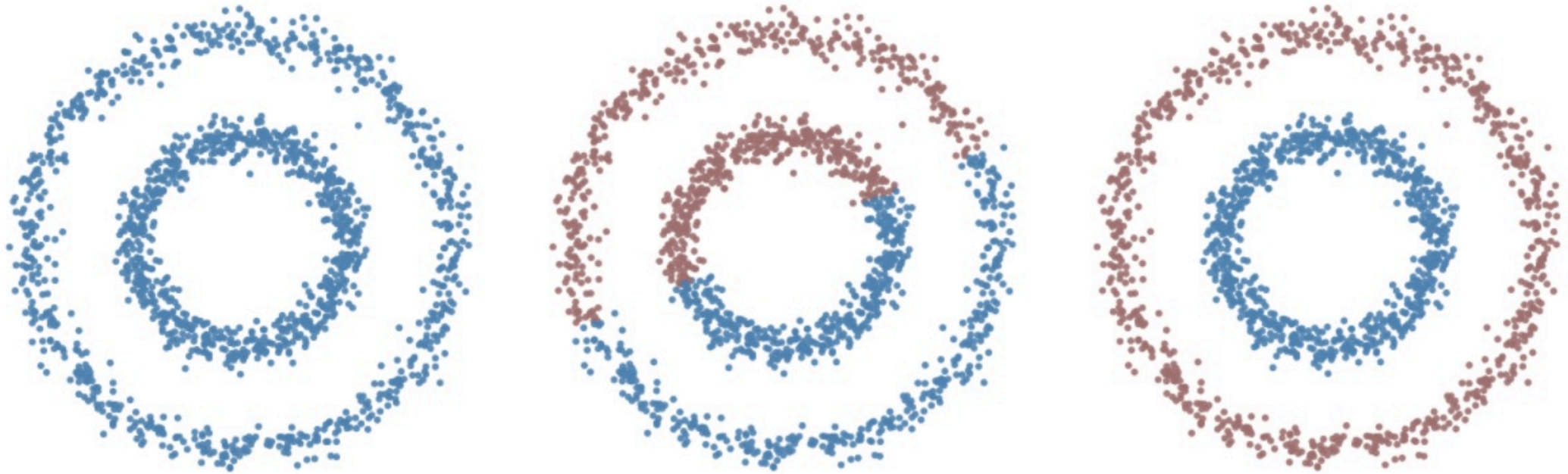
If we want to find K clusters we get K eigenvectors

Stack the $\mathbf{v}$'s in a matrix

Rows of matrix are coordinates in eigenvector space

Perform K-Means on this lower-dimensional data

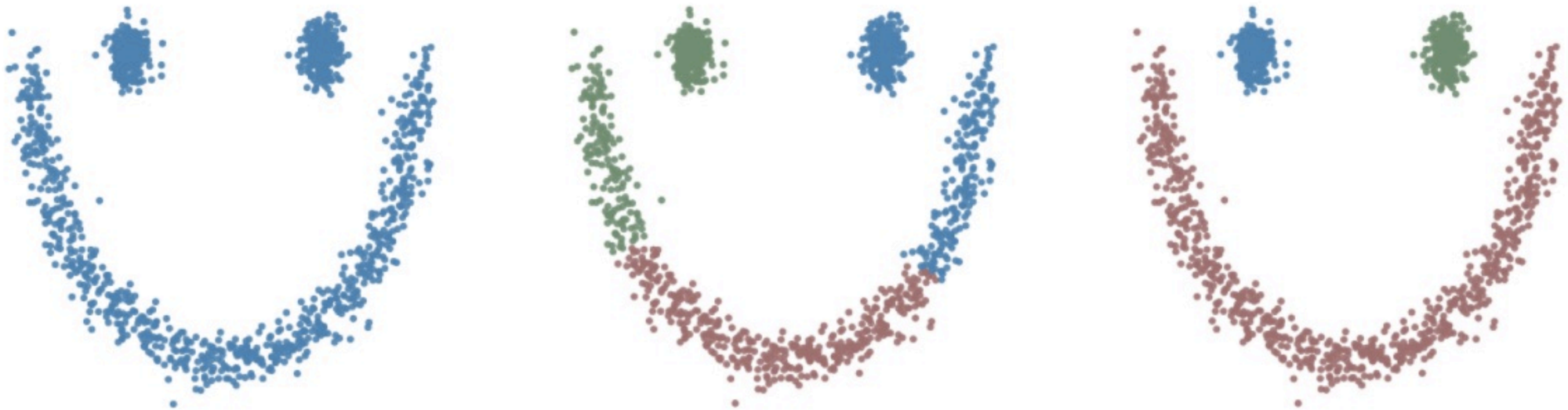
Example - Concentric Rings



Example - Moons

Example - Clustering Happiness

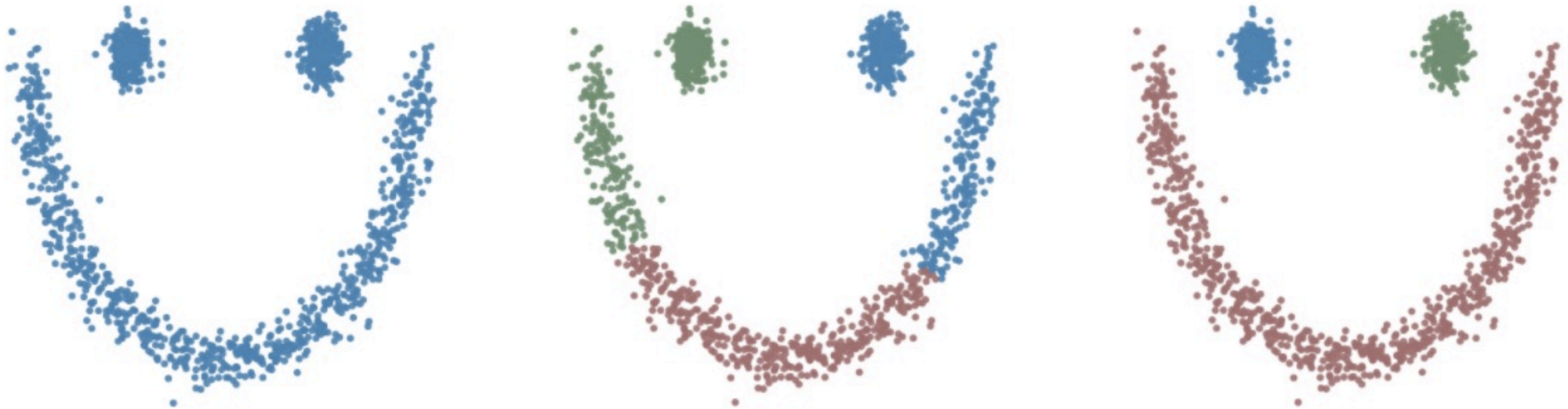
This time we had to go look for three clusters



Still that darned problem of figuring out what K is ...

Example - Clustering Happiness

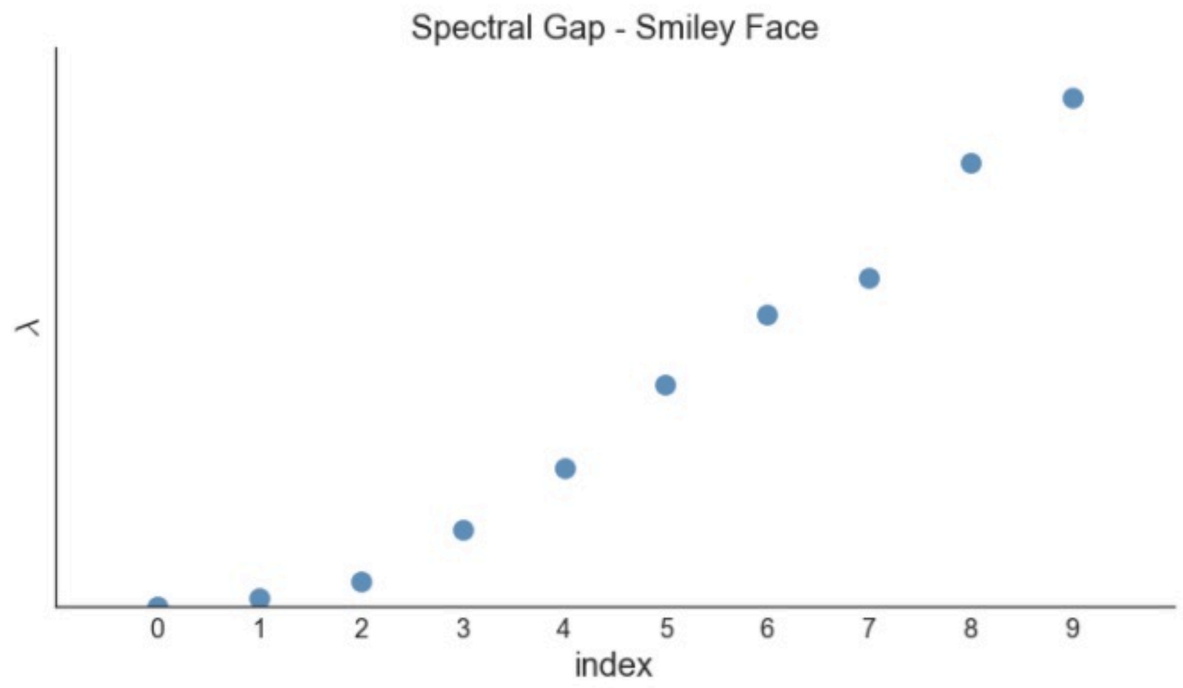
This time we had to go look for three clusters



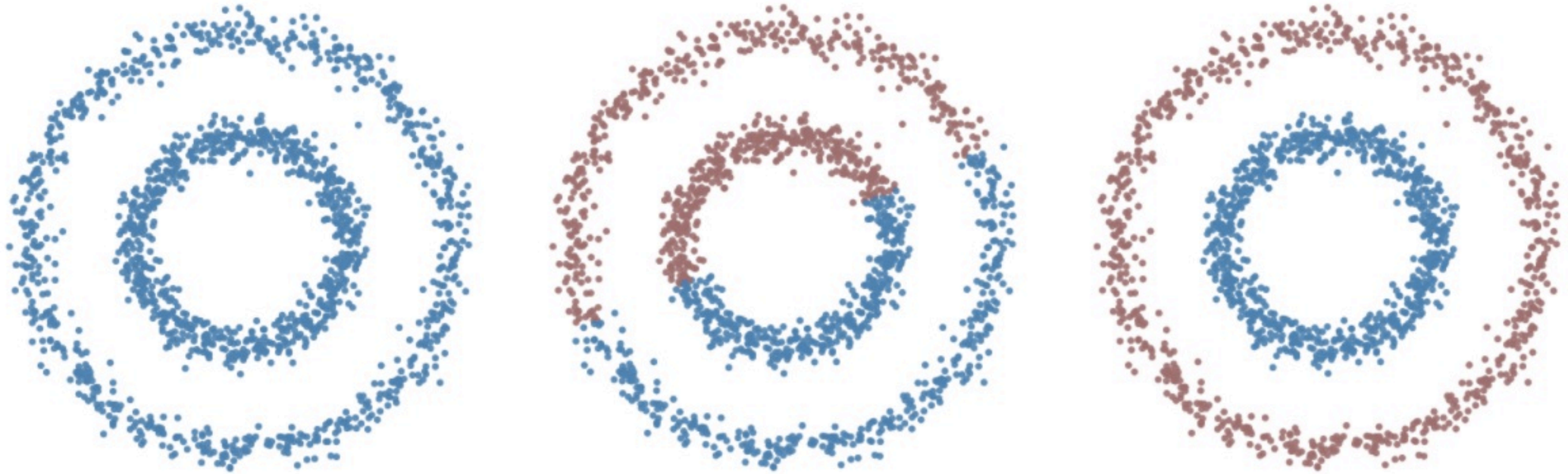
Maybe the eigenvalues can shed some light?

Example - Clustering Happiness

Here's a plot of the 9 smallest eigenvalues of L

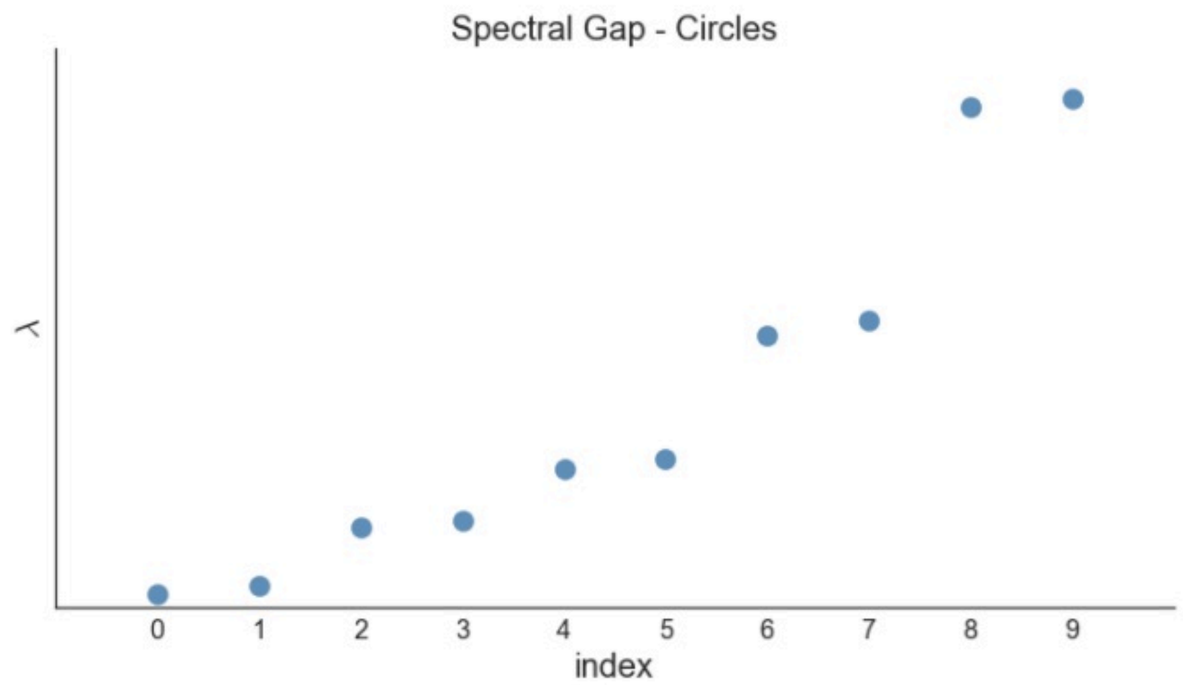


Example - Concentric Rings



Example - Concentric Rings

Here's a plot of the 9 smallest eigenvalues of L



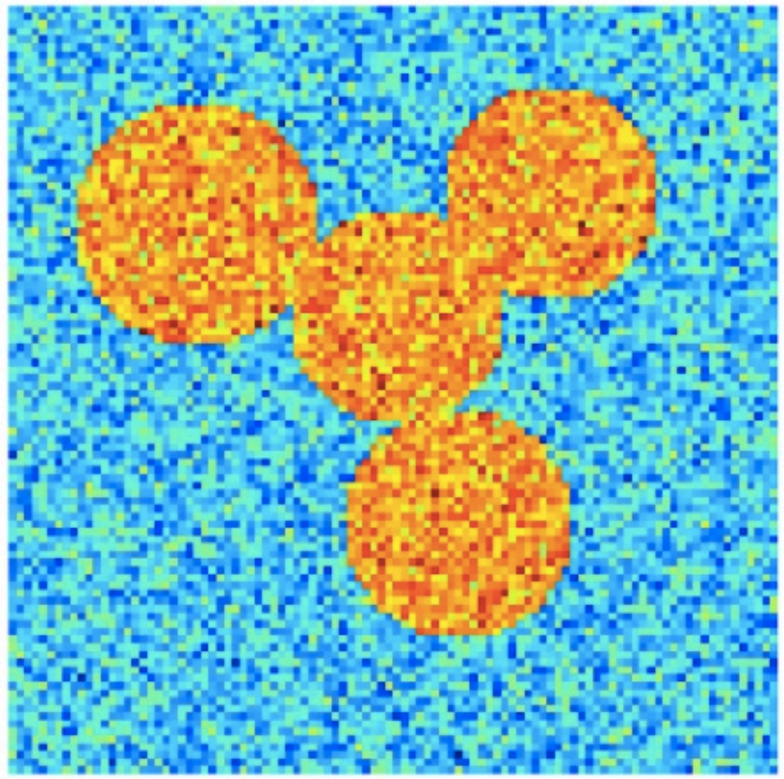
Notice anything?

Spectral Clustering in Practice (Almost)

1. Choose a Similarity Measure
2. Form Graph Laplacian
3. Find the first *several* eval-evec pairs
4. Look for the spectral gap to choose K
5. Perform K-Means on evec-coordinates
6. Profit

Example - Image Segmentation

Suppose you have an image with random additive noise



What are the features?

What is a good similarity measure?

Example - Image Segmentation

Connect neighboring pixels in the graph

Edge weights are Gaussian Kernel of intensity gradient

$$w_{i,i+1} = e^{-(I(i+1,j)-I(i,j))^2} \quad \text{for horizontal neighbors}$$

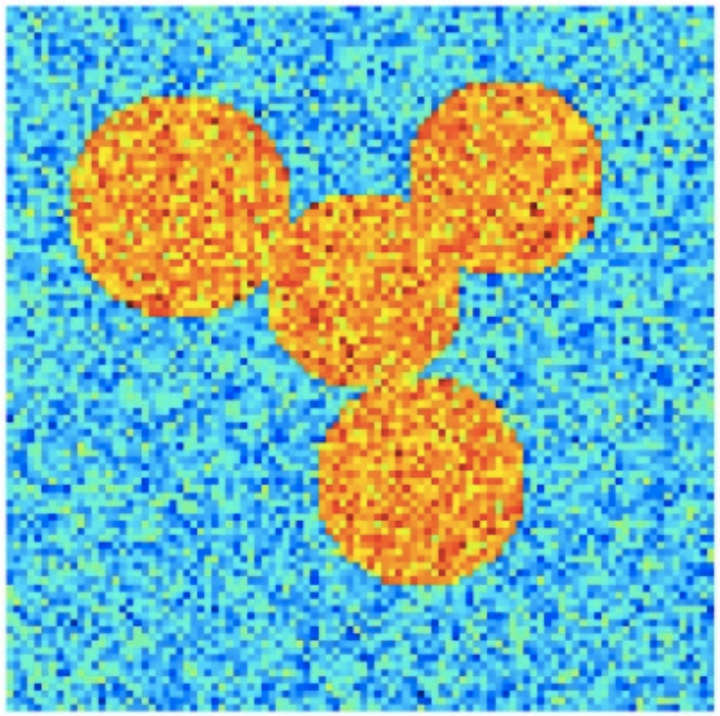
$$w_{j,j+1} = e^{-(I(i,j+1)-I(i,j))^2} \quad \text{for vertical neighbors}$$

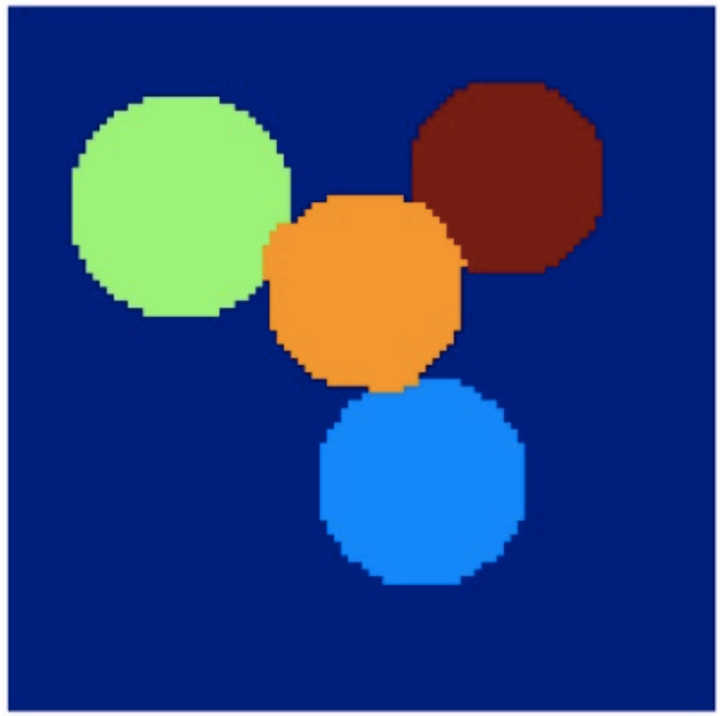
If intensity doesn't vary between neighboring pixels too much, difference is small, exponential is large.

If intensity varies a lot between neighboring pixels, difference is large, exponential is small.

Example - Image Segmentation

Not too shabby...





In Class

In Class

In Class

In Class

In Class

In Class