



Clustering

Lecture 11

Data Science Decal

Hosted by Machine Learning at Berkeley

Agenda

K Nearest Neighbors

Supervised vs. Unsupervised Learning

K-means Clustering

Gaussian Mixture Models

Expectation-Maximization Algorithm

Distance Functions

Hierarchical Clustering

Spectral Clustering

Questions

K Nearest Neighbors

KNN Problem Setup



You have a friend who loves Netflix.

KNN Problem Setup



You have a friend who loves Netflix.

You say: Hey, I just watched the new season of Stranger Things.
It's amazing.

KNN Problem Setup



You have a friend who loves Netflix.

You say: Hey, I just watched the new season of Stranger Things.
It's amazing.

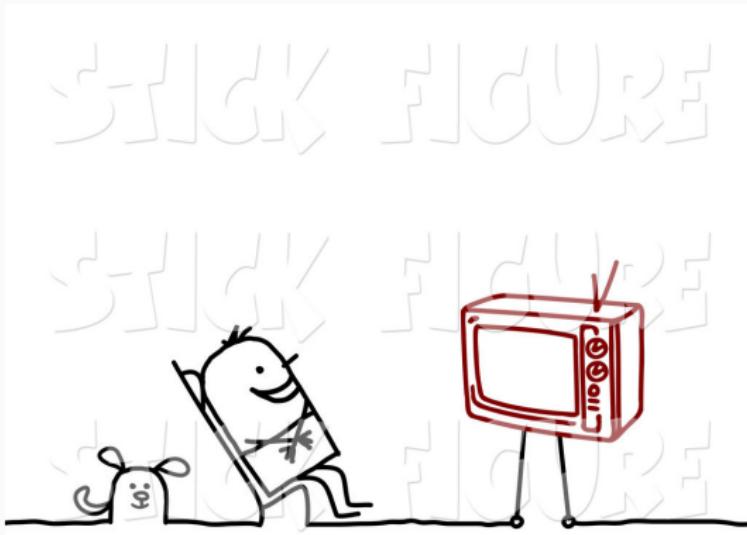
Friend goes home and watches both seasons.

KNN Problem Setup

You have a friend who loves Netflix.

You say: Hey, I just watched the new season of Stranger Things.
It's amazing.

Friend goes home and watches both seasons.



KNN Problem Setup



Friend the next day: OMG that was great!! I want to watch more shows like it, can you suggest?

Friend the next day: OMG that was great!! I want to watch more shows like it, can you suggest?

You: uhhhhh. (You're a hardworking Cal student, so ain't nobody got time for more than 1 TV show).

Friend the next day: OMG that was great!! I want to watch more shows like it, can you suggest?

You: uhhhhh. (You're a hardworking Cal student, so ain't nobody got time for more than 1 TV show).

Darn. What can you do?

KNN Problem Setup



Imagine you have this black box: it can featurize any TV show you give it.

Imagine you have this black box: it can featurize any TV show you give it.

That is, if given an input of a TV show, it spits back a vector a numbers, where each element corresponds to the weight of a particular feature.

Imagine you have this black box: it can featurize any TV show you give it.

That is, if given an input of a TV show, it spits back a vector a numbers, where each element corresponds to the weight of a particular feature.

So you obtain a feature vector for every single TV show on Netflix.

Imagine you have this black box: it can featurize any TV show you give it.

That is, if given an input of a TV show, it spits back a vector a numbers, where each element corresponds to the weight of a particular feature.

So you obtain a feature vector for every single TV show on Netflix. You find the point closest to Stranger Things and tell him to watch that show.

Training time: do nothing. Store the labeled data that you have.

Training time: do nothing. Store the labeled data that you have.

Test time: you are given an unlabeled point. To predict its label, look at the labels of the **k nearest** training points and make an estimate.

Training time: do nothing. Store the labeled data that you have.

Test time: you are given an unlabeled point. To predict its label, look at the labels of the **k nearest** training points and make an estimate.

- Categorical data: majority vote on the class.

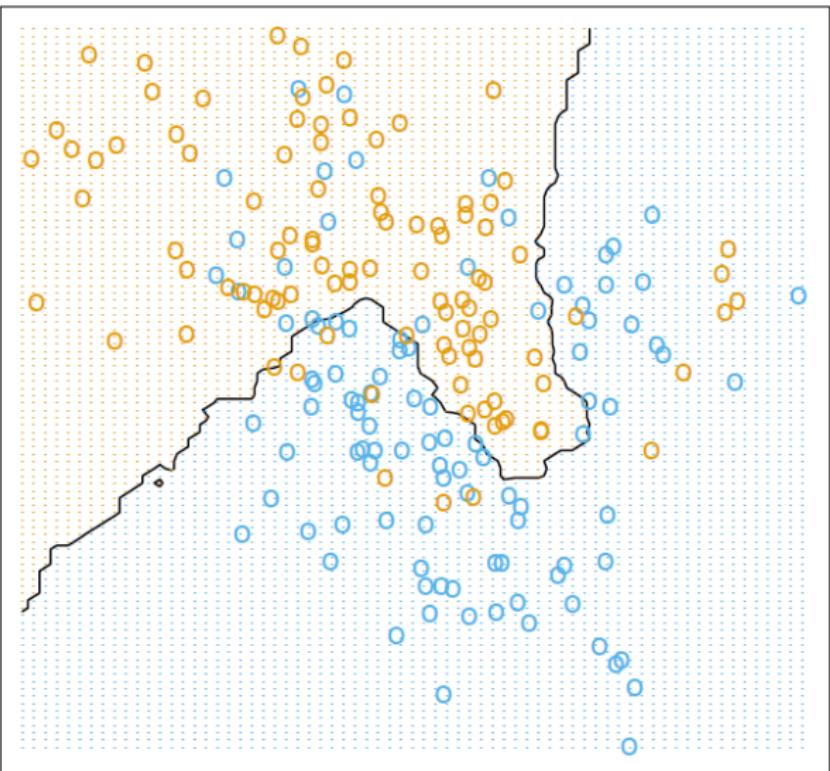
Training time: do nothing. Store the labeled data that you have.

Test time: you are given an unlabeled point. To predict its label, look at the labels of the **k nearest** training points and make an estimate.

- Categorical data: majority vote on the class.
- Numerical data: take an average. Perhaps weighted by distance.

KNN Visualization

15-Nearest Neighbor Classifier



What is K?



K is a **hyperparameter**. You choose it.

What is K?



K is a **hyperparameter**. You choose it.

Here's a nice visualization of the effect of K on classification.

What is K?

K is a **hyperparameter**. You choose it.

Here's a nice visualization of the effect of K on classification.

- If $K = 1$, classify point based on only 1 nearest. Moving a little can cause you to flip classes (high variance). But each training point classified correctly (low bias).

What is K?

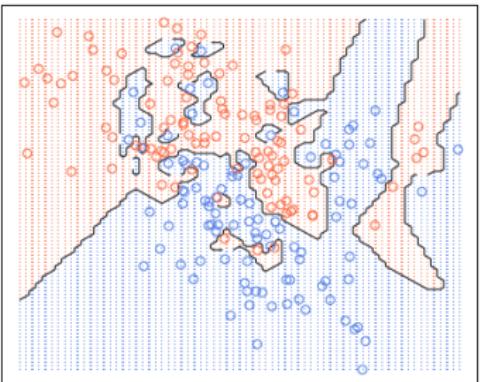
K is a **hyperparameter**. You choose it.

Here's a nice visualization of the effect of K on classification.

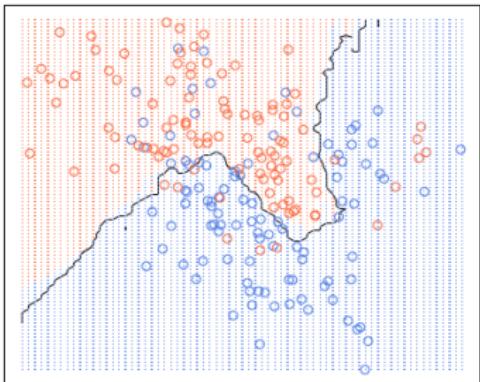
- If $K = 1$, classify point based on only 1 nearest. Moving a little can cause you to flip classes (high variance). But each training point classified correctly (low bias).
- If $K = N$ (# training points), every point gets the same classification as it looks at entire training set. Low variance. But you will surely get many things wrong (high bias).

Visualizing KNN

nearest neighbour ($k = 1$)

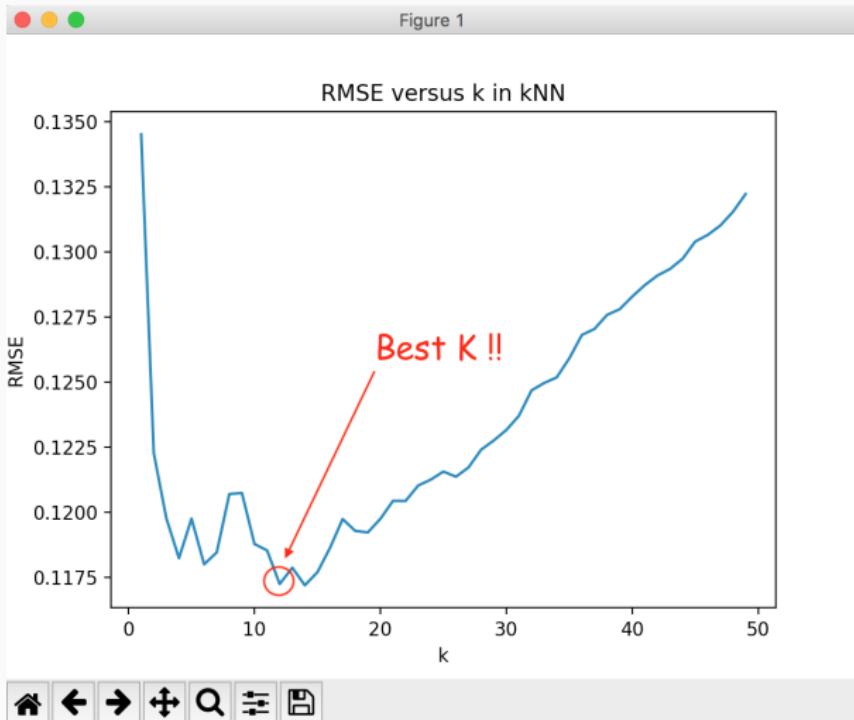


20-nearest neighbour



How to determine the correct K?

Cross validation!



Supervised vs. Unsupervised Learning

Supervised vs. Unsupervised Learning



- So far, you've only touched on supervised learning

Supervised vs. Unsupervised Learning



- So far, you've only touched on supervised learning
- Supervised learning: you are given data that is **labeled**. For every input, you are given the target output

Supervised vs. Unsupervised Learning



- So far, you've only touched on supervised learning
- Supervised learning: you are given data that is **labeled**. For every input, you are given the target output
- Unsupervised learning: you are given **unlabeled** data. This means you just have a bunch of inputs

Supervised vs. Unsupervised Learning



- So far, you've only touched on supervised learning
- Supervised learning: you are given data that is **labeled**. For every input, you are given the target output
- Unsupervised learning: you are given **unlabeled** data. This means you just have a bunch of inputs
- So what can you do with a bunch of data and no specific target/goal?

Supervised vs. Unsupervised Learning



Q: So what can you do with a bunch of data and no specific target/goal?

Supervised vs. Unsupervised Learning



Q: So what can you do with a bunch of data and no specific target/goal?

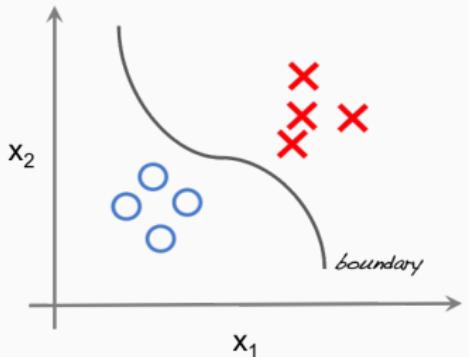
A: We make the most of what we have. We explore the data and see if we can uncover any patterns or **structure** that may be useful.

Supervised vs. Unsupervised Learning

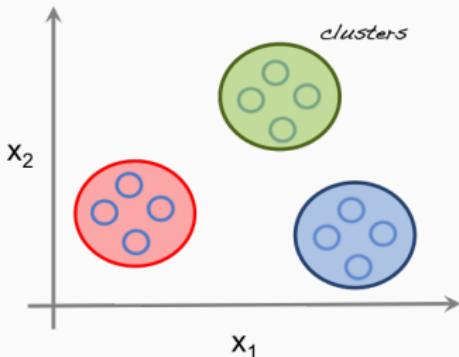
Q: So what can you do with a bunch of data and no specific target/goal?

A: We make the most of what we have. We explore the data and see if we can uncover any patterns or **structure** that may be useful.

Supervised learning



Unsupervised learning



K-means Clustering

Thinking out loud: well, similar elements should be close to one another in our dataset...

Thinking out loud: well, similar elements should be close to one another in our dataset...

- We're trying to find structure, right? Let's find clusters of similar elements!

Thinking out loud: well, similar elements should be close to one another in our dataset...

- We're trying to find structure, right? Let's find clusters of similar elements!
- K-means assumes there are **k representative clusters** in our dataset

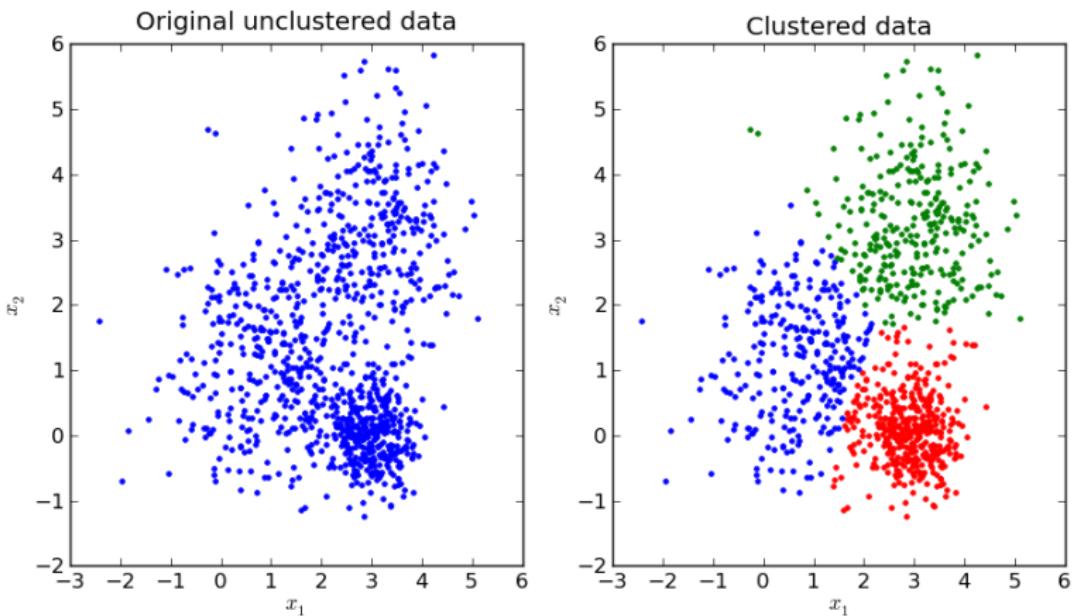
Thinking out loud: well, similar elements should be close to one another in our dataset...

- We're trying to find structure, right? Let's find clusters of similar elements!
- K-means assumes there are **k representative clusters** in our dataset
- Each cluster can be thought of as a bunch of similar points centered around a **mean**

Thinking out loud: well, similar elements should be close to one another in our dataset...

- We're trying to find structure, right? Let's find clusters of similar elements!
- K-means assumes there are **k representative clusters** in our dataset
- Each cluster can be thought of as a bunch of similar points centered around a **mean**
- We need to find this mean

K-means



How do we find these clusters?

Algorithm:

Algorithm:

1. Randomly choose k points initially

Algorithm:

1. Randomly choose k points initially
2. For every point in training set, find which of the k points it is closest to and label it

Algorithm:

1. Randomly choose k points initially
2. For every point in training set, find which of the k points it is closest to and label it
3. We have k different groups of points, "k clusters."

Algorithm:

1. Randomly choose k points initially
2. For every point in training set, find which of the k points it is closest to and label it
3. We have k different groups of points, "k clusters."
4. Find the **mean** of each cluster (average the data points in each cluster).

Algorithm:

1. Randomly choose k points initially
2. For every point in training set, find which of the k points it is closest to and label it
3. We have k different groups of points, "k clusters."
4. Find the **mean** of each cluster (average the data points in each cluster).
5. These means are the new k points. Repeat from step 2 until algorithm converges.

Algorithm:

1. Randomly choose k points initially
2. For every point in training set, find which of the k points it is closest to and label it
3. We have k different groups of points, "k clusters."
4. Find the **mean** of each cluster (average the data points in each cluster).
5. These means are the new k points. Repeat from step 2 until algorithm converges.

Algorithm:

1. Randomly choose k points initially
2. For every point in training set, find which of the k points it is closest to and label it
3. We have k different groups of points, "k clusters."
4. Find the **mean** of each cluster (average the data points in each cluster).
5. These means are the new k points. Repeat from step 2 until algorithm converges.

Closest is a metric we pick (L2 norm for instance).

Demo. Better Demo.

Jupyter notebook!

Applications:

Applications:

- Image quantization (what we saw in the notebook)

Applications:

- Image quantization (what we saw in the notebook)
- Finding similar documents/images/audio/etc.

Applications:

- Image quantization (what we saw in the notebook)
- Finding similar documents/images/audio/etc.
- Many other things

Formalizing K-means.

We want to find k subsets of our data points, $S = \{S_1, S_2, \dots, S_k\}$

Formalizing K-means.

We want to find k subsets of our data points, $S = \{S_1, S_2, \dots, S_k\}$

Minimize the distance from each point to its cluster center.

Formalizing K-means.

We want to find k subsets of our data points, $S = \{S_1, S_2, \dots, S_k\}$

Minimize the distance from each point to its cluster center.

$$\operatorname{argmin}_S \sum_{i=1}^k \sum_{x \in S_i} \| (x - \mu_i) \|^2$$

Formalizing K-means.

We want to find k subsets of our data points, $S = \{S_1, S_2, \dots, S_k\}$

Minimize the distance from each point to its cluster center.

$$\operatorname{argmin}_S \sum_{i=1}^k \sum_{x \in S_i} \| (x - \mu_i) \|^2$$

No closed-form solution to this, so use iterative algorithm!

Guaranteed to converge (although could converge to local minima).

Formalizing K-means.

We want to find k subsets of our data points, $S = \{S_1, S_2, \dots, S_k\}$

Minimize the distance from each point to its cluster center.

$$\operatorname{argmin}_S \sum_{i=1}^k \sum_{x \in S_i} \| (x - \mu_i) \|^2$$

No closed-form solution to this, so use iterative algorithm!

Guaranteed to converge (although could converge to local minima).

Why does this even work?

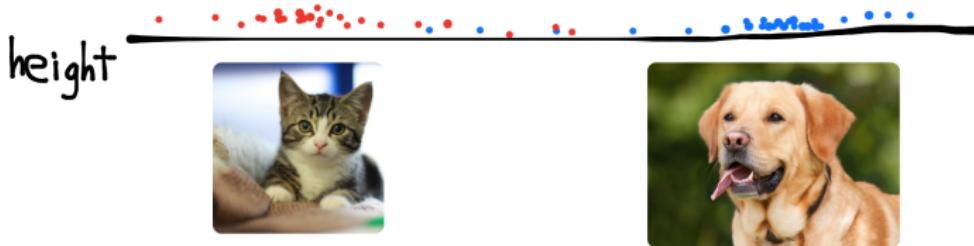
Gaussian Mixture Models

Background

Here's the setup: I go to a park. I find a bunch of dogs, and record their heights. I also find a bunch of cats, and record their heights.

Background

Here's the setup: I go to a park. I find a bunch of dogs, and record their heights. I also find a bunch of cats, and record their heights.



Background

We can estimate a probability distribution for each.

$$\mathcal{N}(\mu, \sigma^2)$$

Background

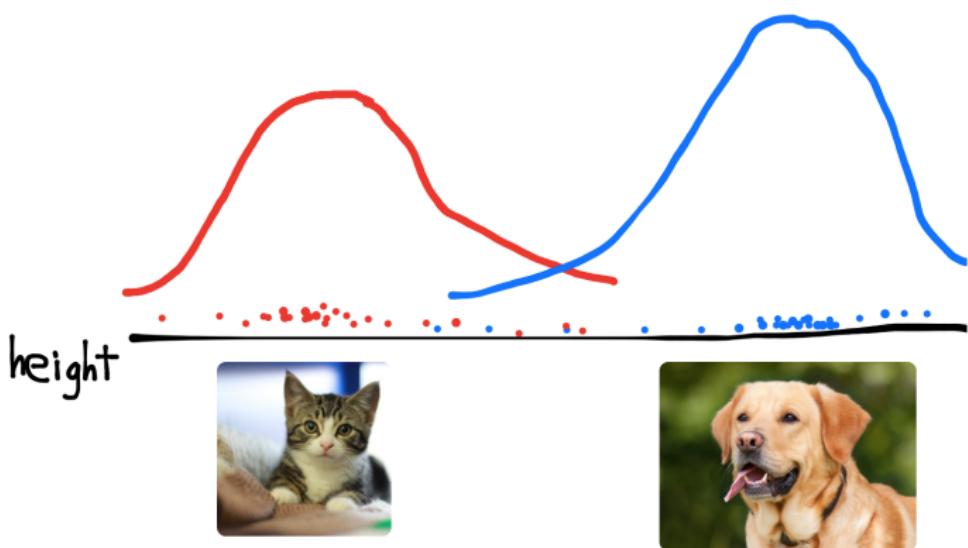
We can estimate a probability distribution for each.

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Background

We can estimate a probability distribution for each.

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = p(x|\mu, \sigma^2)$$



Background

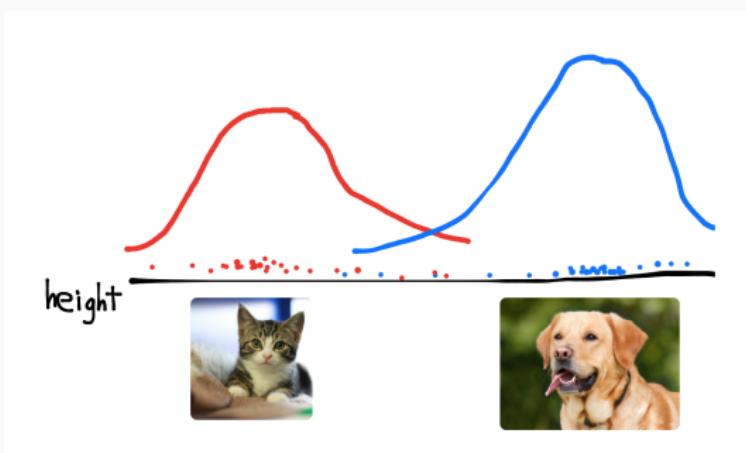
Someone gives us a new datapoint. Asks if it is dog or cat. How do we respond?

Background

Someone gives us a new datapoint. Asks if it is dog or cat. How do we respond?

Check the probability of each point under each distribution!

dog if $p(x|\mu_{dog}, \sigma^2_{dog}) > p(x|\mu_{cat}, \sigma^2_{cat})$ else cat



Background

Life is great when we know these distributions! But what if we didn't know which points were cats and which were dogs?

Background

Life is great when we know these distributions! But what if we didn't know which points were cats and which were dogs?
Are we stuck?

Life is great when we know these distributions! But what if we didn't know which points were cats and which were dogs?
Are we stuck?

1. If we knew the distributions, we can classify each point based on which distribution it's most likely under

Life is great when we know these distributions! But what if we didn't know which points were cats and which were dogs?
Are we stuck?

1. If we knew the distributions, we can classify each point based on which distribution it's most likely under
2. If we knew the labeling of the datapoints, we could estimate the distribution

Life is great when we know these distributions! But what if we didn't know which points were cats and which were dogs?
Are we stuck?

1. If we knew the distributions, we can classify each point based on which distribution it's most likely under
2. If we knew the labeling of the datapoints, we could estimate the distribution

Life is great when we know these distributions! But what if we didn't know which points were cats and which were dogs?
Are we stuck?

1. If we knew the distributions, we can classify each point based on which distribution it's most likely under
2. If we knew the labeling of the datapoints, we could estimate the distribution

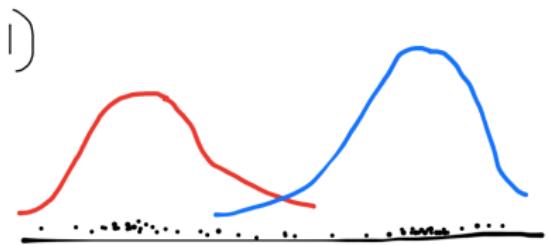
We have **neither**. The labels (dog/cat in this case) are called **latent** or **hidden** variables.

Background - The Dilemma

We observe:

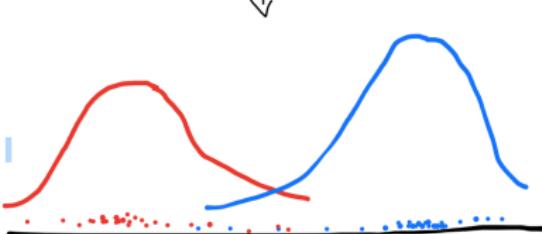


If only we could get one of these:

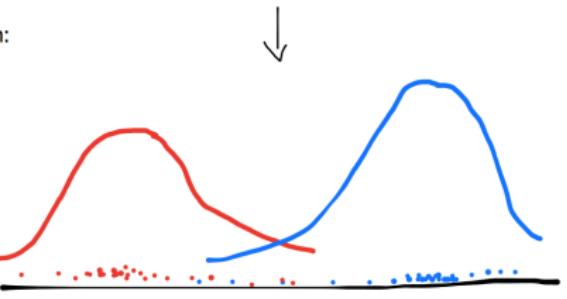


II)

Then:



II)



Gaussian Mixture Models



Why don't we just **assume** we have one of them?

Why don't we just **assume** we have one of them?

Before we move on, let's review what we know:

Why don't we just **assume** we have one of them?

Before we move on, let's review what we know:

- We have a normal distribution (gaussian) in 1d. Now think of this distribution in multiple dimensions, and we have a multivariate gaussian $\mathcal{N}(\vec{\mu}, \Sigma)$

Why don't we just **assume** we have one of them?

Before we move on, let's review what we know:

- We have a normal distribution (gaussian) in 1d. Now think of this distribution in multiple dimensions, and we have a multivariate gaussian $\mathcal{N}(\vec{\mu}, \Sigma)$
- The gaussians explain latent variables. In our example, one gaussian for cats, one for dogs.

Why don't we just **assume** we have one of them?

Before we move on, let's review what we know:

- We have a normal distribution (gaussian) in 1d. Now think of this distribution in multiple dimensions, and we have a multivariate gaussian $\mathcal{N}(\vec{\mu}, \Sigma)$
- The gaussians explain latent variables. In our example, one gaussian for cats, one for dogs.
- Goal: we want to estimate the distributions. Find $(\vec{\mu}_i, \Sigma_i) \forall i$

Expectation-Maximization Algorithm

Input: A bunch of unlabeled data. We suspect clusters in the data.
Explainable by latent variables.

Input: A bunch of unlabeled data. We suspect clusters in the data.
Explainable by latent variables.

Goal: Estimate latent factors by approximating the gaussian distributions. k gaussian distributions.

Input: A bunch of unlabeled data. We suspect clusters in the data.
Explainable by latent variables.

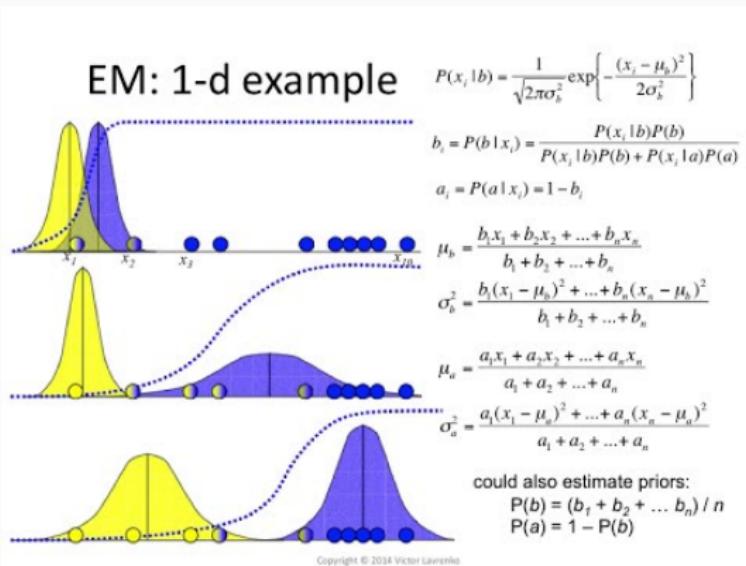
Goal: Estimate latent factors by approximating the gaussian distributions. **k** gaussian distributions.

Think of this as k-means, but instead of finding just the means, you are finding the means and covariances.

Demo.

Does this kind of make sense now? The important thing to remember here is that each point is not hard-assigned to a cluster; instead, each point has a **probability** of being in each cluster. These are "soft" assignments to clusters, and this weighting is taken into account when recomputing the means and covariances.

EM Algorithm



Problem Setup:

$$Given : X = \{x_1, x_2, \dots, x_n\}$$

Problem Setup:

Given : $X = \{x_1, x_2, \dots, x_n\}$

Model : $(x, z) \sim p_\theta$ for unknown θ, z

Problem Setup:

$$Given : X = \{x_1, x_2, \dots, x_n\}$$

$$Model : (x, z) \sim p_\theta \text{ for unknown } \theta, z$$

$$Goal : \operatorname{argmax}_\theta P(\theta | X)$$

Problem Setup:

$$Given : X = \{x_1, x_2, \dots, x_n\}$$

$$Model : (x, z) \sim p_\theta \text{ for unknown } \theta, z$$

$$Goal : \operatorname{argmax}_\theta P(\theta|X)$$

$$Problem : P(\theta|X) = \sum_z P(\theta|X, z) \text{ hard to maximize}$$

Problem Setup:

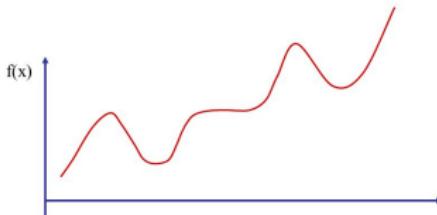
Given : $X = \{x_1, x_2, \dots, x_n\}$

Model : $(x, z) \sim p_\theta$ for unknown θ, z

Goal : $\underset{\theta}{\operatorname{argmax}} P(\theta|X)$

Problem : $P(\theta|X) = \sum_z P(\theta|X, z)$ hard to maximize

Example of Non-Convex Function



Algorithm:

1. Initialize θ_0

Algorithm:

1. Initialize θ_0
2. Repeat until convergence:

Algorithm:

1. Initialize θ_0
2. Repeat until convergence:
 - 2.1 E-step:

$$Q(\theta_{t+1}, \theta_t) = E[P(X, Z|\theta_t)|X, \theta_t]$$

Algorithm:

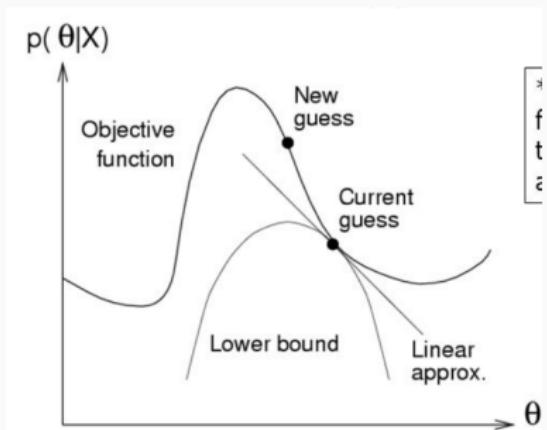
1. Initialize θ_0
2. Repeat until convergence:
 - 2.1 E-step:

$$Q(\theta_{t+1}, \theta_t) = E[P(X, Z | \theta_t) | X, \theta_t]$$

2.2 M-step:

$$\theta_{t+1} = \operatorname{argmax}_{\theta_{t+1}} Q(\theta_{t+1}, \theta_t)$$

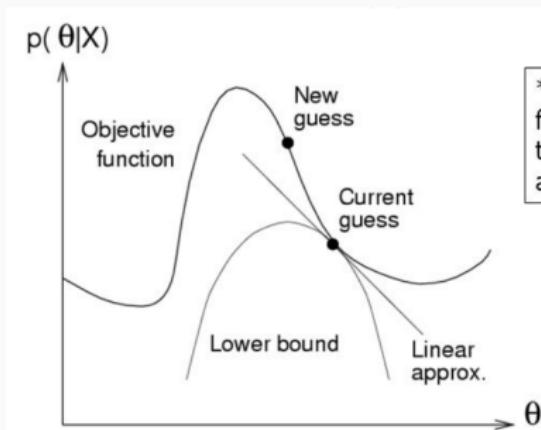
Alternate form of gradient descent.



6/28/2015

Fatih Gelgi, ASU'05

Alternate form of gradient descent.



6/28/2015

Fatih Gelgi, ASU'05

E-step calculates the lower bounding function. M-step maximizes.

Pros:

- $P_{\theta_{t+1}}(X) > P_{\theta_t}(X)$, guaranteed to converge

Cons:

Pros:

- $P_{\theta_{t+1}}(X) > P_{\theta_t}(X)$, guaranteed to converge
- Works well in practice

Cons:

Pros:

- $P_{\theta_{t+1}}(X) > P_{\theta_t}(X)$, guaranteed to converge
- Works well in practice

Cons:

- Could converge to local maximum.

Pros:

- $P_{\theta_{t+1}}(X) > P_{\theta_t}(X)$, guaranteed to converge
- Works well in practice

Cons:

- Could converge to local maximum.

Pros:

- $P_{\theta_{t+1}}(X) > P_{\theta_t}(X)$, guaranteed to converge
- Works well in practice

Cons:

- Could converge to local maximum.

Observation: we used Gaussian Mixture Models to illustrate, but EM algorithm can be implemented many different ways. You will notice that **k-means is just a special case** of the EM algorithm!

K-means vs GMM:

- GMM accounts for **uncertainty** in clustering assignments.
Provides probabilities for each class

K-means vs GMM:

- GMM accounts for **uncertainty** in clustering assignments.
Provides probabilities for each class
- K-means more sensitive to outliers

K-means vs GMM:

- GMM accounts for **uncertainty** in clustering assignments.
Provides probabilities for each class
- K-means more sensitive to outliers
- K-means assumes all points are clustered in identical spheres
(not ideal!), but GMM adapts unique distributions for each class

Distance Functions

Can you think of many different distance functions for K-means or KNN.
Common choice is L2 norm (euclidean distance).

Can you many different distance functions for K-means or KNN.

Common choice is L2 norm (euclidean distance).

Must satisfy:

$$d(x, y) \geq 0 \text{ with equality iff } x = y$$

$$d(x, y) = d(y, x)$$

$$d(x, y) \leq d(x, z) + d(z, y)$$

Hierarchical Clustering

Agglomerative Clustering



Main Idea: merge the most similar instances, and repeat

Agglomerative Clustering



Main Idea: merge the most similar instances, and repeat
Algorithm:

1. Initialize each datapoint to its own cluster

Main Idea: merge the most similar instances, and repeat
Algorithm:

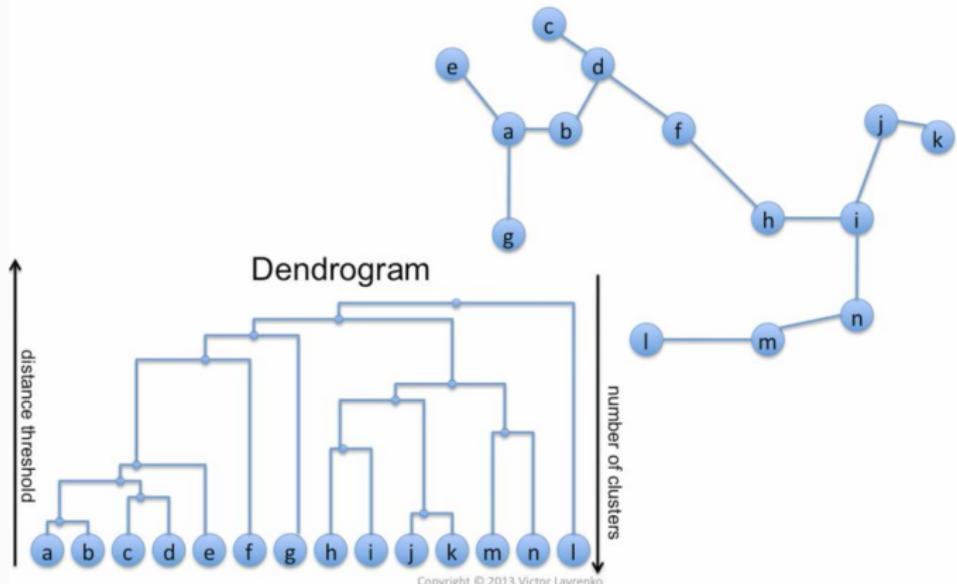
1. Initialize each datapoint to its own cluster
2. Greedily pick the 2 smallest and merge

Main Idea: merge the most similar instances, and repeat
Algorithm:

1. Initialize each datapoint to its own cluster
2. Greedily pick the 2 smallest and merge
3. Repeat above until only 1 cluster

"Bottom up" approach, builds Dendrogram.

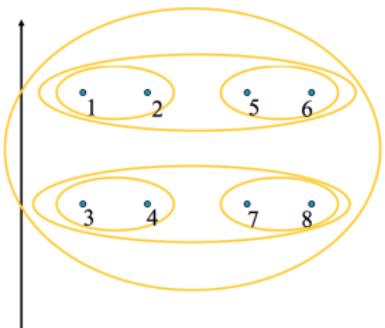
Agglomerative clustering: example



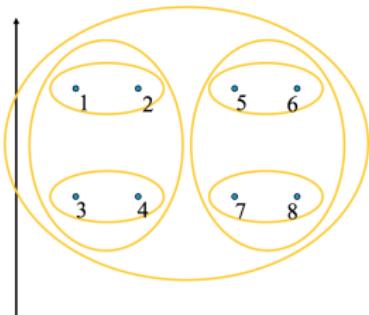
Agglomerative Clustering

Many ways to define distance between two instances.

Closest pair
(single-link clustering)



Farthest pair
(complete-link clustering)



[Pictures from Thorsten Joachims]

Top down approach, still builds dendrogram.

1. Start with everything in one cluster

Top down approach, still builds dendrogram.

1. Start with everything in one cluster
2. Split into two clusters, maximizing distance

Top down approach, still builds dendrogram.

1. Start with everything in one cluster
2. Split into two clusters, maximizing distance
3. Recurse on each subhalf

Top down approach, still builds dendrogram.

1. Start with everything in one cluster
2. Split into two clusters, maximizing distance
3. Recurse on each subhalf

Top down approach, still builds dendrogram.

1. Start with everything in one cluster
2. Split into two clusters, maximizing distance
3. Recurse on each subhalf

Can be more efficient, as can stop algorithm short (no need to cluster down to single leaf).

Spectral Clustering

Spectral Clustering



My favorite! :)

Very new way of thinking about clustering

Super efficient

Produces amazing results

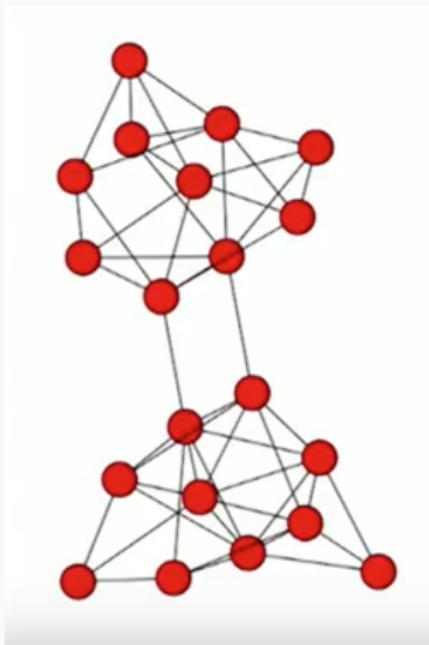
Spectral Clustering

Key point: think of each data sample as a vertex in a graph

Spectral Clustering

Key point: think of each data sample as a vertex in a graph

Connect similar points with an edge, larger weights mean more similar.



Spectral Clustering

How to weight the edges?

- Any weighting which gives a higher score for similar vertices

How to weight the edges?

- Any weighting which gives a higher score for similar vertices
- Common: use gaussian distance

$$e^{\frac{||x_i - x_j||^2}{\sigma^2}}$$

How to weight the edges?

- Any weighting which gives a higher score for similar vertices
- Common: use gaussian distance

$$e^{\frac{||x_i - x_j||^2}{\sigma^2}}$$

How to weight the edges?

- Any weighting which gives a higher score for similar vertices
- Common: use gaussian distance

$$e^{\frac{||x_i - x_j||^2}{\sigma^2}}$$

Which edges to connect?

- Can connect every edge

How to weight the edges?

- Any weighting which gives a higher score for similar vertices
- Common: use gaussian distance

$$e^{\frac{||x_i - x_j||^2}{\sigma^2}}$$

Which edges to connect?

- Can connect every edge
- Connect to k nearest neighbors

How to weight the edges?

- Any weighting which gives a higher score for similar vertices
- Common: use gaussian distance

$$e^{\frac{||x_i - x_j||^2}{\sigma^2}}$$

Which edges to connect?

- Can connect every edge
- Connect to k nearest neighbors
- Connect to points within ϵ (some threshold)

How to weight the edges?

- Any weighting which gives a higher score for similar vertices
- Common: use gaussian distance

$$e^{\frac{||x_i - x_j||^2}{\sigma^2}}$$

Which edges to connect?

- Can connect every edge
- Connect to k nearest neighbors
- Connect to points within ϵ (some threshold)

How to weight the edges?

- Any weighting which gives a higher score for similar vertices
- Common: use gaussian distance

$$e^{\frac{||x_i - x_j||^2}{\sigma^2}}$$

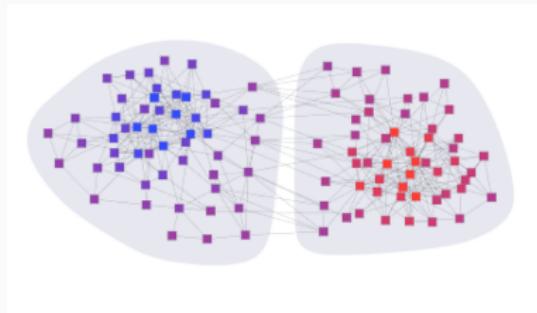
Which edges to connect?

- Can connect every edge
- Connect to k nearest neighbors
- Connect to points within ϵ (some threshold)

Different graphs have different properties, give different results.

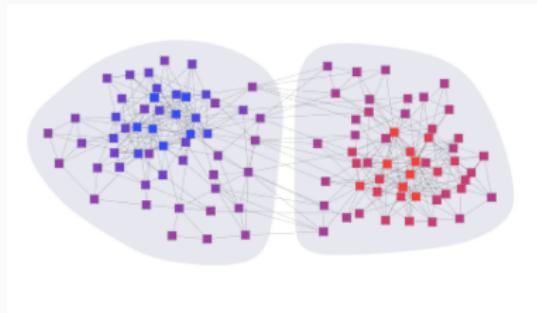
Spectral Graphs

So now we have this graph.



Spectral Graphs

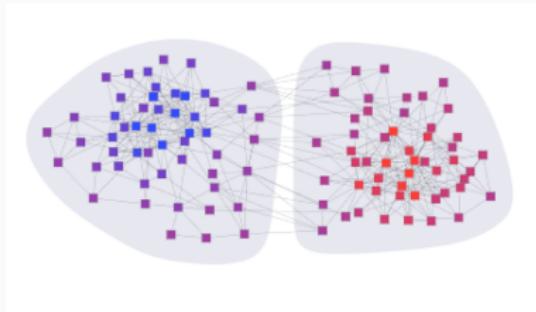
So now we have this graph.



What do we want to find? Min cut!

Spectral Graphs

So now we have this graph.

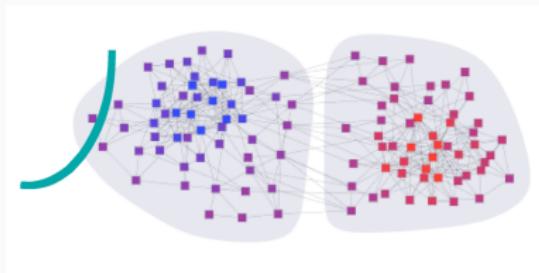


What do we want to find? Min cut!

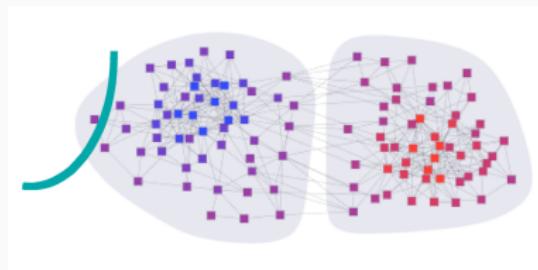
$$\operatorname{argmin}_{S_1, S_2 : \{S_1, S_2\} = V} \sum_{i \in S_1, j \in S_2} d(i, j)$$

Spectral Graphs

Problem:

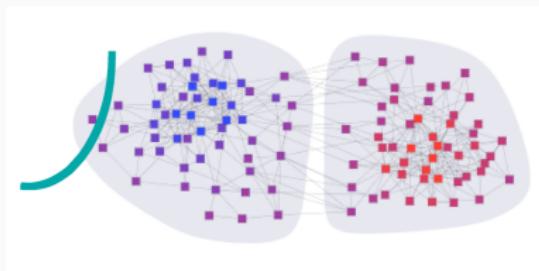


Problem:



We need **balanced** clusters. So normalize the cuts!

Problem:



We need **balanced** clusters. So normalize the cuts!

$$\operatorname{argmin}_{S_1, S_2 : \{S_1, S_2\} = V} \frac{\sum_{i \in S_1, j \in S_2} d(i, j)}{\sum_{i \in S_1} d(i, j)} + \frac{\sum_{i \in S_1, j \in S_2} d(i, j)}{\sum_{i, j \in S_2} d(i, j)}$$

Spectral Graphs



Normalized mincut in NP-Hard.

Normalized mincut in NP-Hard.

We can approximate it with a matrix based on the graph. Called Laplacian.

Normalized mincut in NP-Hard.

We can approximate it with a matrix based on the graph. Called Laplacian.

Laplacian is also a low-rank approximation to the gaussian kernel.

Spectral Clustering



First step: construct the Laplacian.

$$L = D - W$$

Spectral Clustering



First step: construct the Laplacian.

$$L = D - W$$

W = adjacency matrix

$$W_{ij} = d(i,j) \text{ if } \text{connected}(i,j) \text{ else } 0$$

Spectral Clustering

First step: construct the Laplacian.

$$L = D - W$$

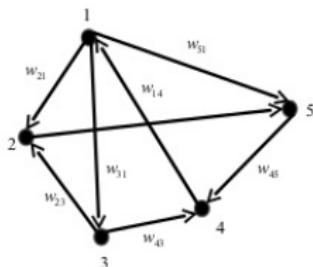
W = adjacency matrix

$$W_{ij} = d(i,j) \text{ if } \text{connected}(i,j) \text{ else } 0$$

D = diagonal matrix

$$D_{ii} = \sum_{j=\text{neighbors}(i)} d(i,j)$$

Graph Dynamics (Diagraph)



Adjacency Matrix

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 \\ 5 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\text{or } A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & w_{14} & 0 \\ w_{21} & 0 & w_{23} & 0 & 0 \\ w_{31} & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{43} & 0 & w_{45} \\ w_{51} & w_{52} & 0 & 0 & 0 \end{bmatrix}$$

Diagonal Matrix

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

Laplacian matrix

$$L = D - A$$

$$L = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ -1 & -1 & 0 & 0 & 2 \end{bmatrix}$$

Note that $(I + L)$ is row stochastic

Spectral Clustering



Step two: eigen decomposition

$$L = U \Sigma V^T$$

Spectral Clustering



Step two: eigen decomposition

$$L = U \Sigma V^T$$

Take a look at eigenvalues! Measure of "connectivity" of graph.
For k connected components:

$$\lambda_{n-k}, \dots, \lambda_n = 0$$

Spectral Clustering



Step two: eigen decomposition

$$L = U \Sigma V^T$$

Take a look at eigenvalues! Measure of "connectivity" of graph.
For k connected components:

$$\lambda_{n-k}, \dots, \lambda_n = 0$$

We want to focus on smallest nonzero eigenvalue - will tell us how to separate the graph best.

- 2)
Decomposition:
 - Find eigenvalues λ and eigenvectors V of the matrix L
 - Map vertices to corresponding components of λ_2



0.0
1.0
1.0
1.0
1.0
0.0

0.4	0.7	0.4	0.3	0.4	0.3
0.4	0.7	0.4	0.3	0.4	0.3
0.4	0.7	0.4	0.3	0.4	0.3
0.4	0.7	0.4	0.3	0.4	0.3
0.4	0.7	0.4	0.3	0.4	0.3
0.4	0.7	0.4	0.3	0.4	0.3

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

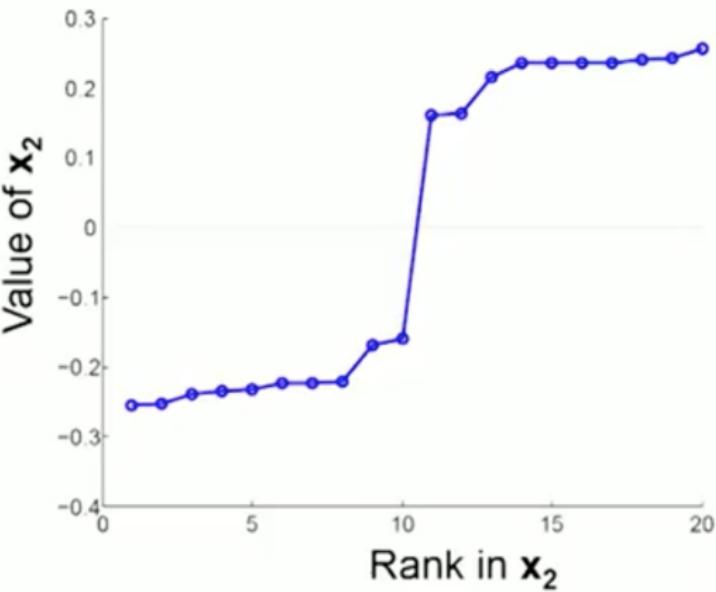
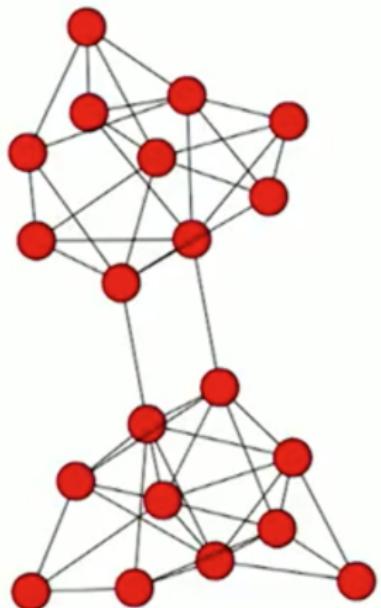
$V =$

$\lambda =$

Material adapted from Stanford

Spectral Clustering

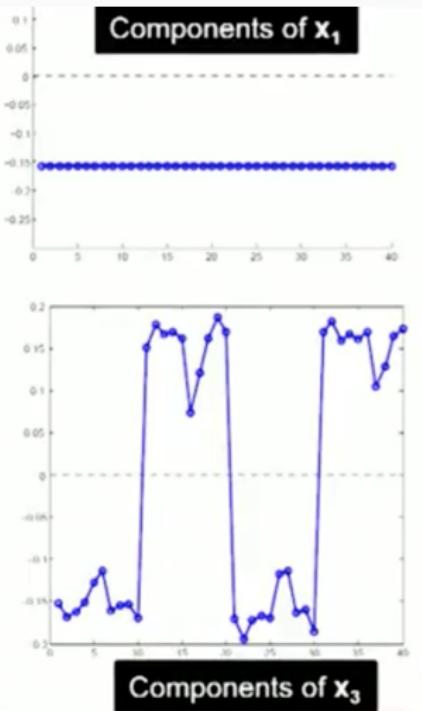
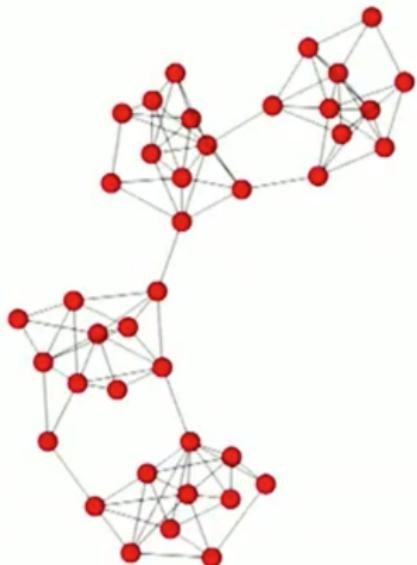
Why is looking at v_2 reasonable?



Material adapted from Stanford

Spectral Clustering

Even cooler!



Material adapted from Stanford

Spectral Clustering



Use v_2 to split vertices into 2 sets:

$$\text{Node}_i = S_1 \text{ if } v_{2i} > \epsilon \text{ else } S_2$$

Spectral Clustering



Use v_2 to split vertices into 2 sets:

$$\text{Node}_i = S_1 \text{ if } v_{2i} > \epsilon \text{ else } S_2$$

Common values for ϵ : 0, $\text{median}(v_2)$

Spectral Clustering

Use v_2 to split vertices into 2 sets:

$$\text{Node}_i = S_1 \text{ if } v_{2i} > \epsilon \text{ else } S_2$$

Common values for ϵ : 0, $\text{median}(v_2)$

Now we have 2 subsets! What if we want k clusters? Can use method **recursively**...

Alternate method:

- Instead of looking at smallest eigenvector, look at **c smallest eigenvectors**

Alternate method:

- Instead of looking at smallest eigenvector, look at **c smallest eigenvectors**
- Now we have each data point represented in **c dimensions!**
(This is essentially PCA!!)

Alternate method:

- Instead of looking at smallest eigenvector, look at **c smallest eigenvectors**
- Now we have each data point represented in **c dimensions!**
(This is essentially PCA!!)
- Do K-means in this new space

Alternate method:

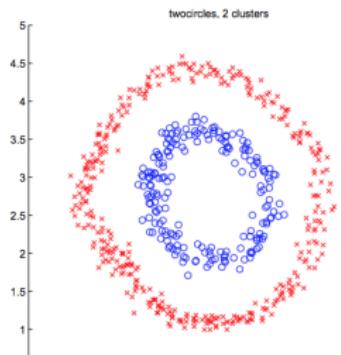
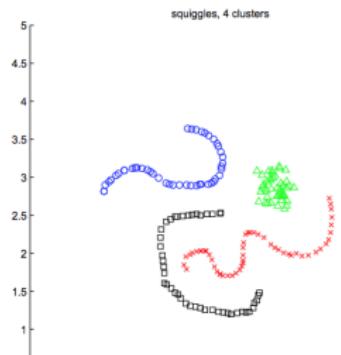
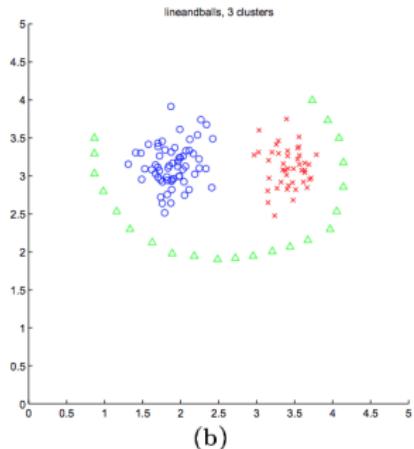
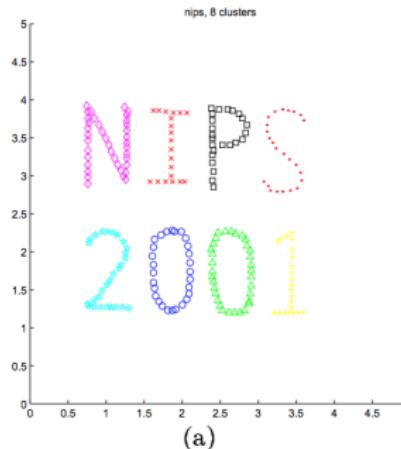
- Instead of looking at smallest eigenvector, look at **c smallest eigenvectors**
- Now we have each data point represented in **c dimensions!**
(This is essentially PCA!!)
- Do K-means in this new space

Alternate method:

- Instead of looking at smallest eigenvector, look at **c smallest eigenvectors**
- Now we have each data point represented in **c dimensions!**
(This is essentially PCA!!)
- Do K-means in this new space

Empirically, this produces best results. Amazing results, in fact.

Spectral Clustering Results



Questions

Unsupervised Learning comic

Unsupervised Learning



(c) 2012 Adam Pauls

Image (c) 2005 Ryan North www.qwantz.com

Questions?