Clustering

Lecture 8

Machine Learning Decal

Hosted by Machine Learning at Berkeley

ML@B

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

ML@B

You have a friend who loves Netflix.

KNN Problem Setup

ML@B

You have a friend who loves Netflix.

You say: Hey, I just watched the new season of Black Mirror. It's amazing!

KNN Problem Setup

ML@B

You have a friend who loves Netflix.

You say: Hey, I just watched the new season of Black Mirror. It's amazing!

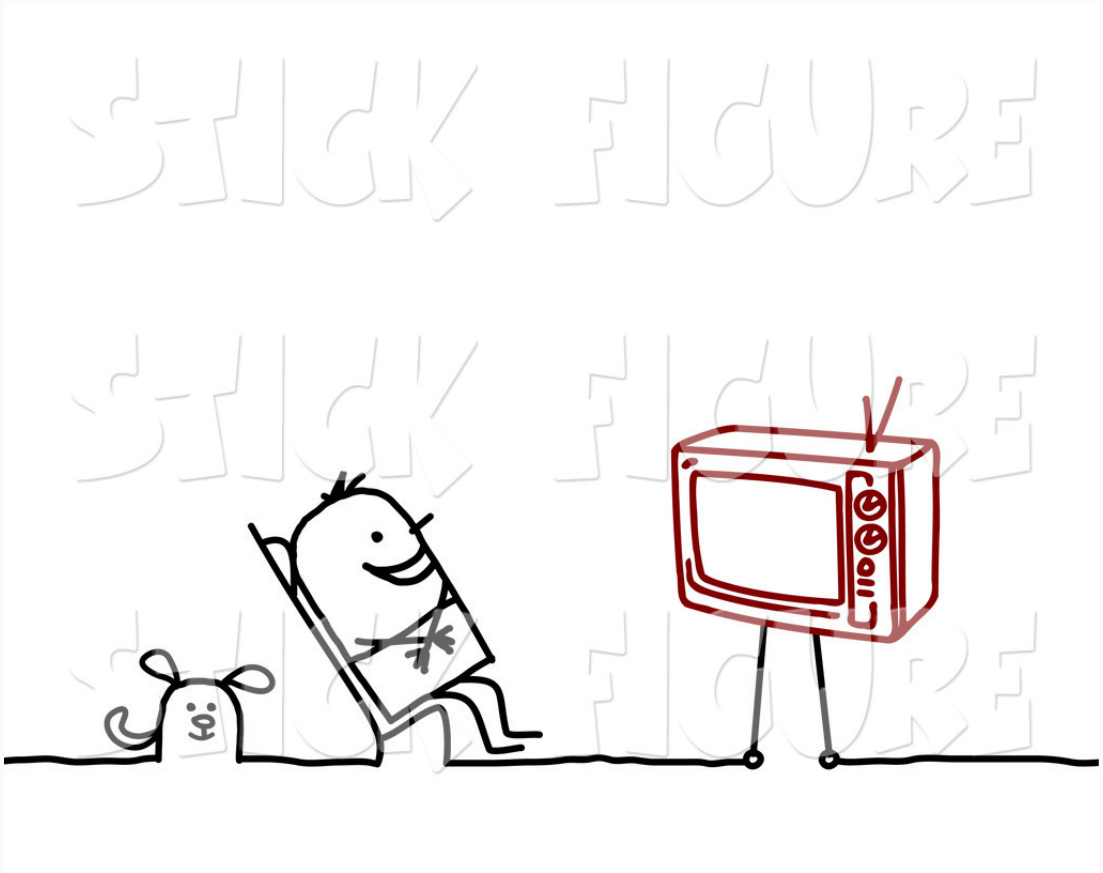
Friend goes home and binge watches all 4 seasons.

KNN Problem Setup

You have a friend who loves Netflix.

You say: Hey, I just watched the new season of Black Mirror. It's amazing!

Friend goes home and binge watches all 4 seasons.



KNN Problem Setup

ML@B

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

KNN Problem Setup

ML@B

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).

KNN Problem Setup

ML@B

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

ML@B

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

KNN Problem Setup

ML@B

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.

KNN Problem Setup

ML@B

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.

KNN Problem Setup

ML@B

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 Black Mirror and tell him to watch that show.

KNN

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

KNN

ML@B

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.

KNN

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.

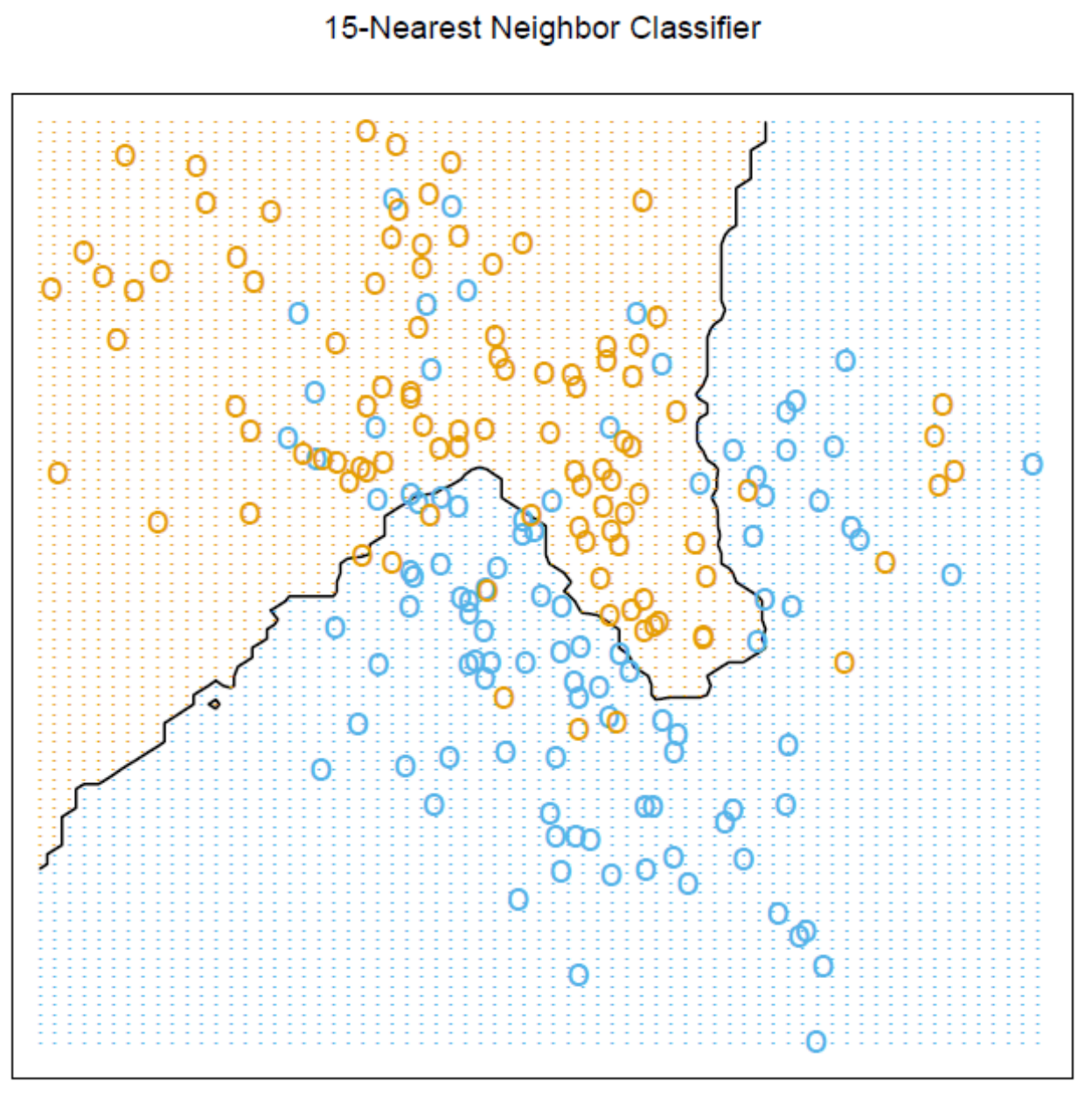
- Categorical data: majority vote on the class.

KNN

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



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.

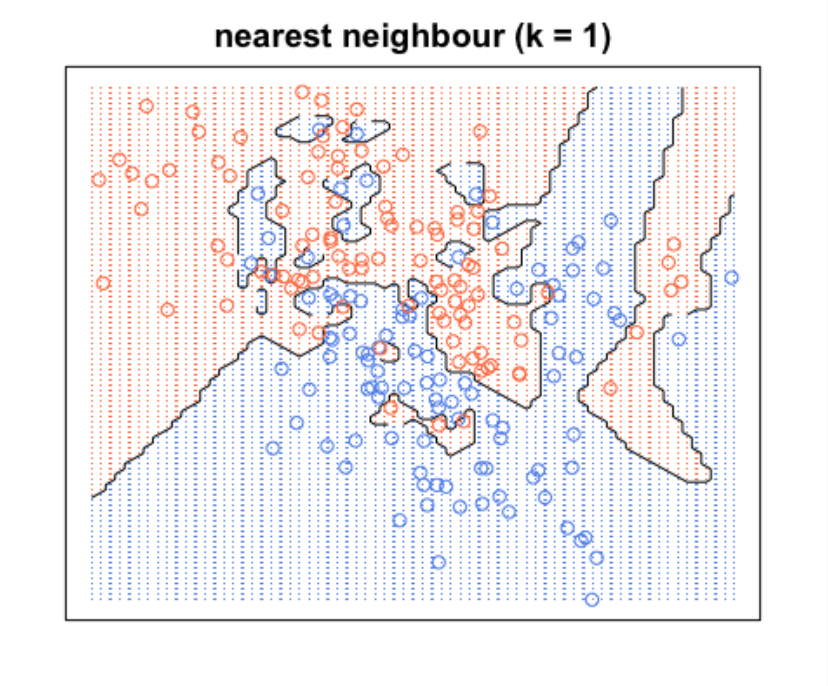
- If $K = 1$, classify point based on only 1 nearest. Moving a little can cause you to flip classes (high variance). But each training point classified correctly (low bias).

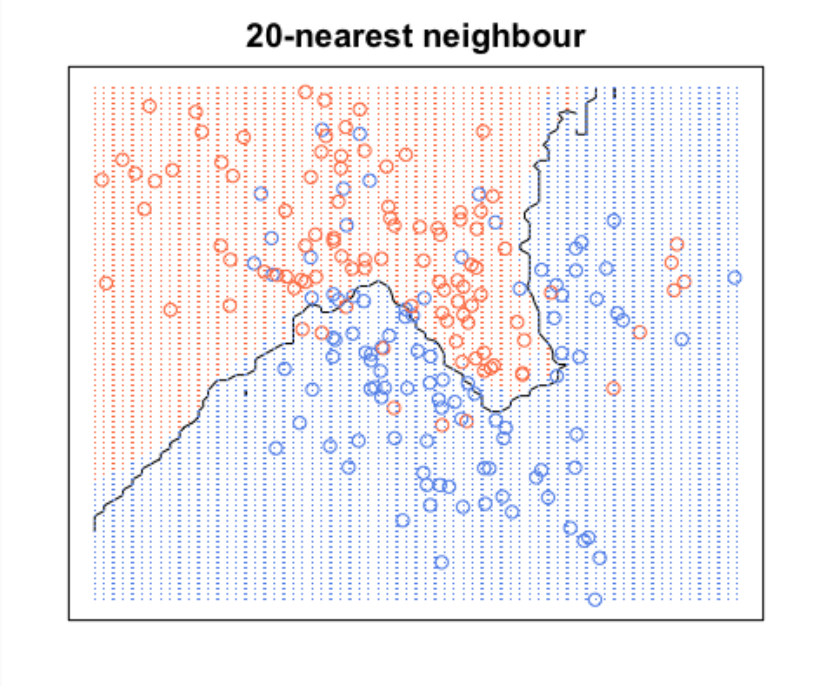
What is K?

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

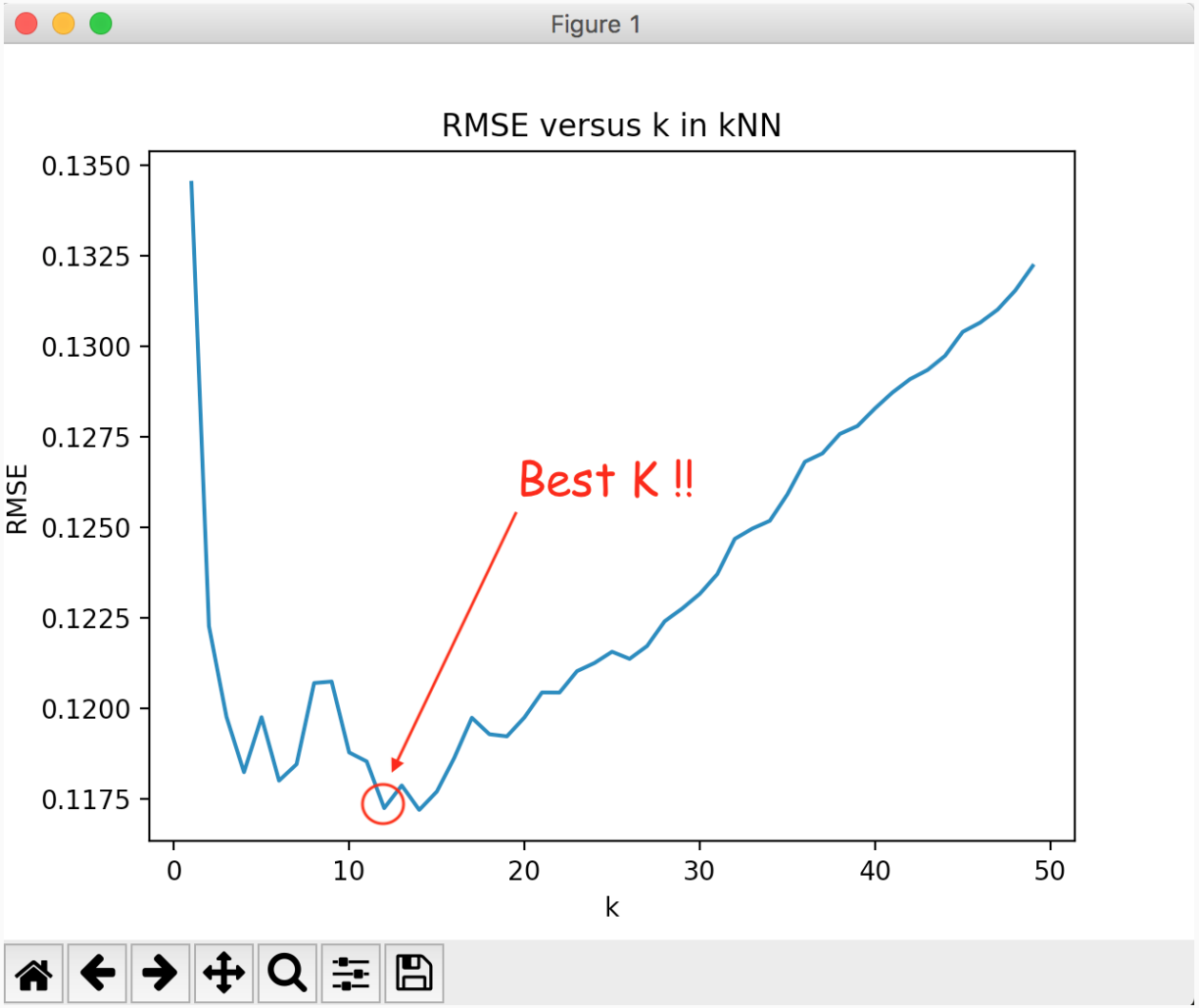




How to determine the correct K?

Finding K

Cross validation!



Supervised vs. Unsupervised Learning

Supervised vs. Unsupervised Learning

ML@B

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

Supervised vs. Unsupervised Learning

ML@B

- 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

ML@B

- 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

ML@B

- 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

ML@B

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

Supervised vs. Unsupervised Learning

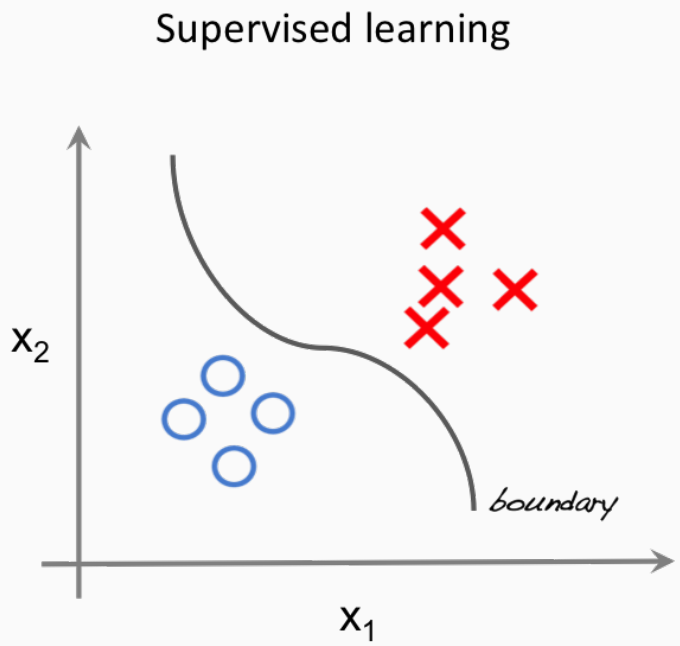
Q: So what can you do with a bunch of data and no specific target/goal?

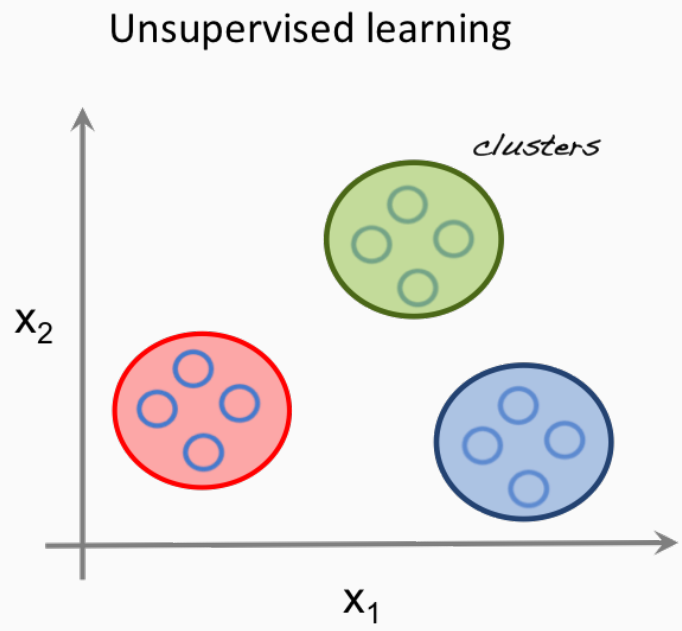
A: We make the most of what we have. We explore the data and see if we can uncover any patterns or **structure** that may be useful.

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.





K-means Clustering

K-means

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

K-means

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

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

K-means

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**

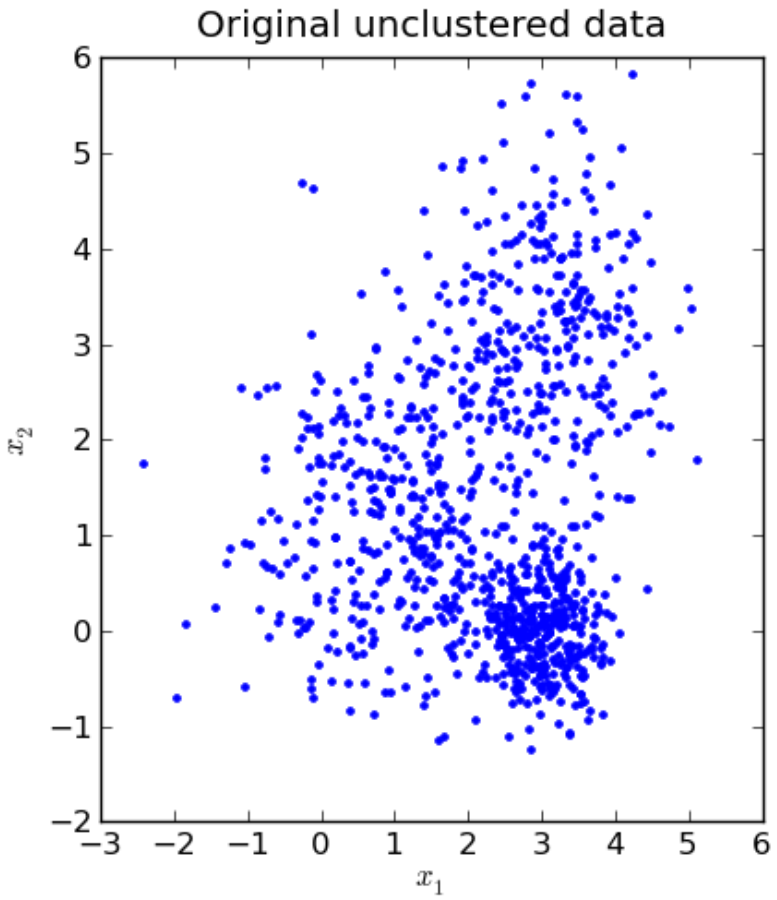
K-means

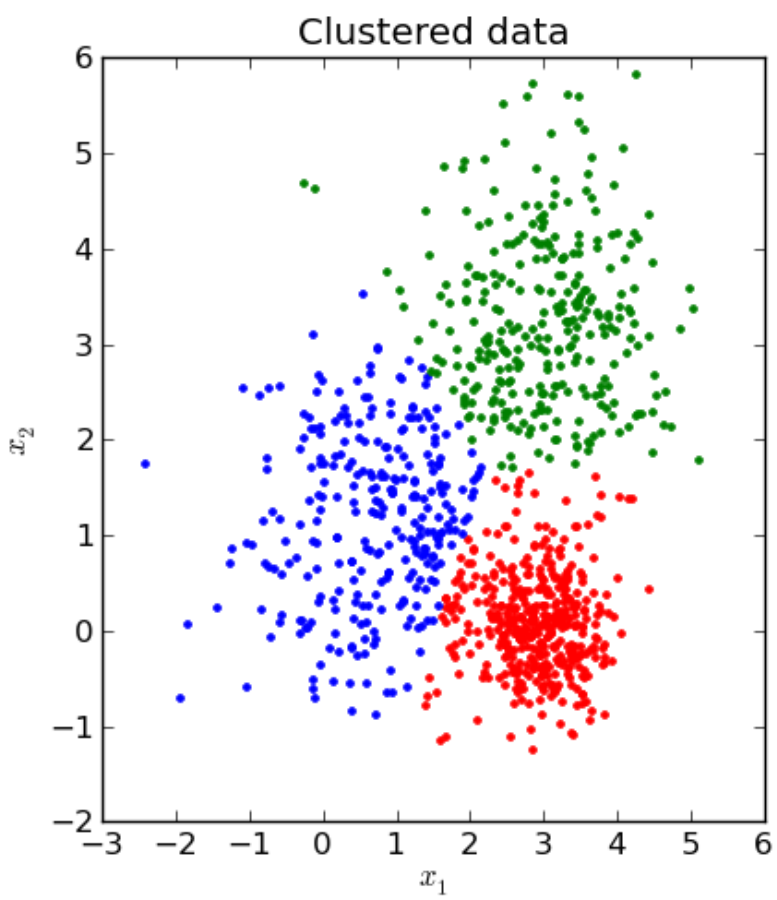
ML@B

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?

K-means

Algorithm:

K-means

Algorithm:

1. Randomly choose **k** points initially

K-means

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

K-means

ML@B

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."

K-means

ML@B

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).

K-means

ML@B

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.

K-means

ML@B

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.

K-means

ML@B

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).

ML@B

Demo. Better Demo.

Jupyter notebook!

K-means

Applications:

K-means

Applications:

- Image quantization (what we saw in the notebook)

K-means

ML@B

Applications:

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

K-means

ML@B

Applications:

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

K-means

Formalizing K-means.

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

K-means

Formalizing K-means.

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

Minimize the distance from each point to its cluster center.

K-means

Formalizing K-means.

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

Minimize the distance from each point to its cluster center.

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

K-means

Formalizing K-means.

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

Minimize the distance from each point to its cluster center.

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

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

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

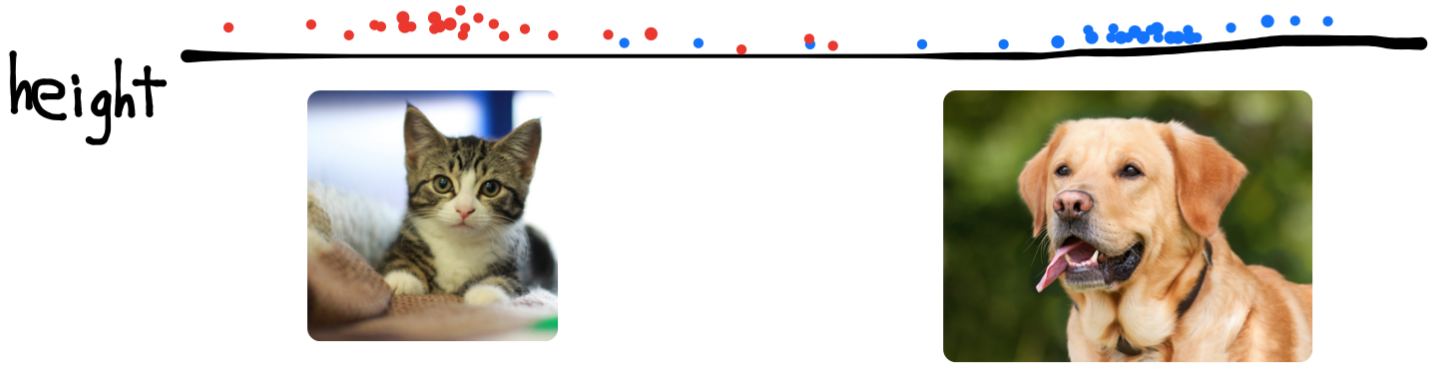
Gaussian Mixture Models

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.

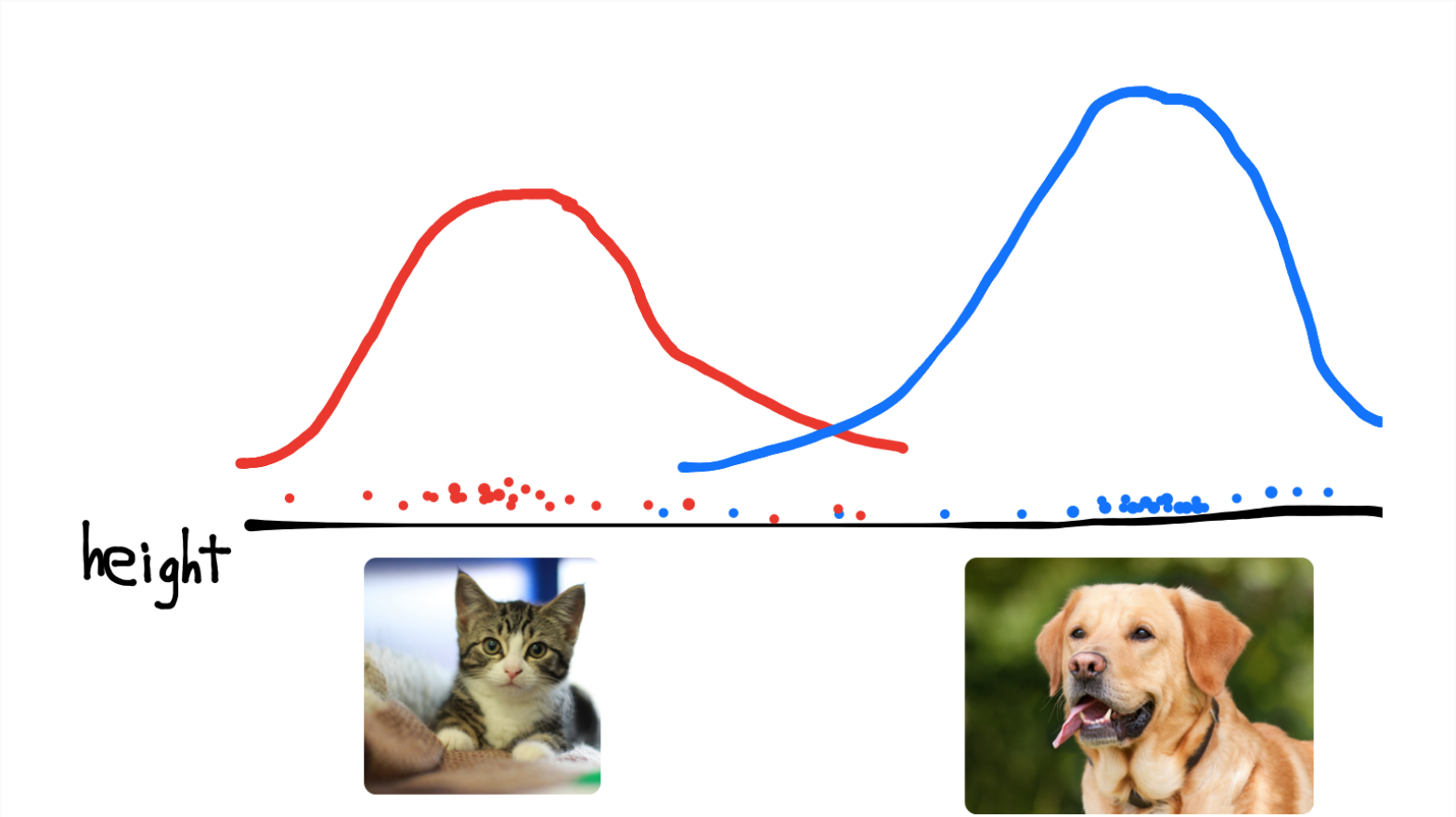
$$\mathcal{N}(\mu, \sigma^2)$$

We can estimate a probability distribution for each.

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

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

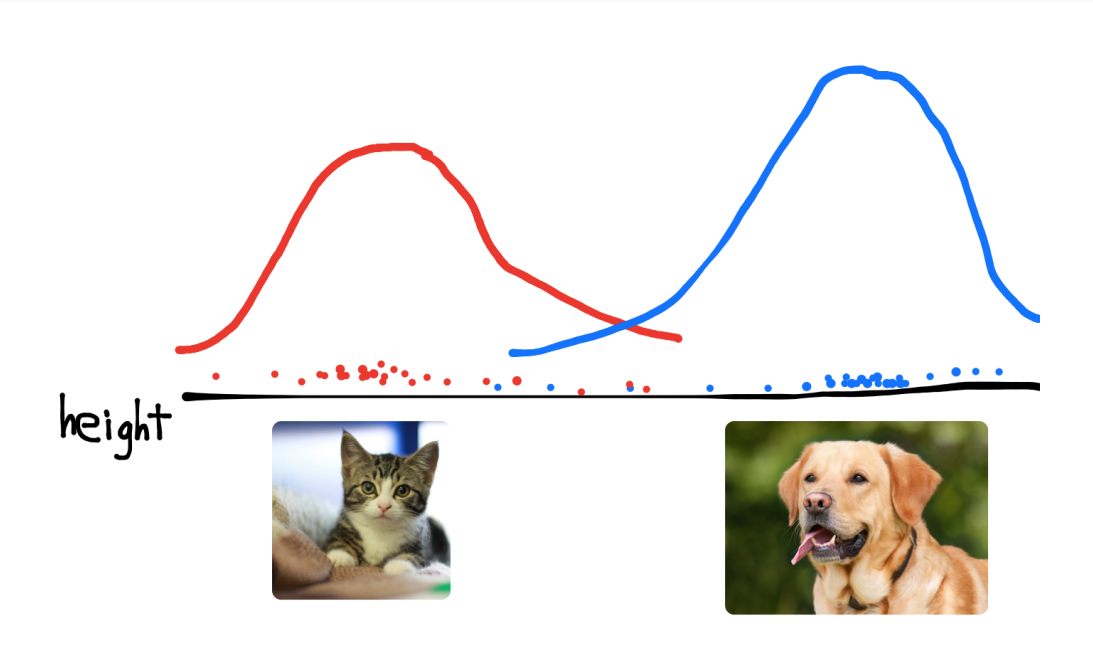
Someone gives us a new datapoint. Asks if it is dog or cat. How do we respond?

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!

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



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?

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?

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

Background

ML@B

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

Background

ML@B

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

Background

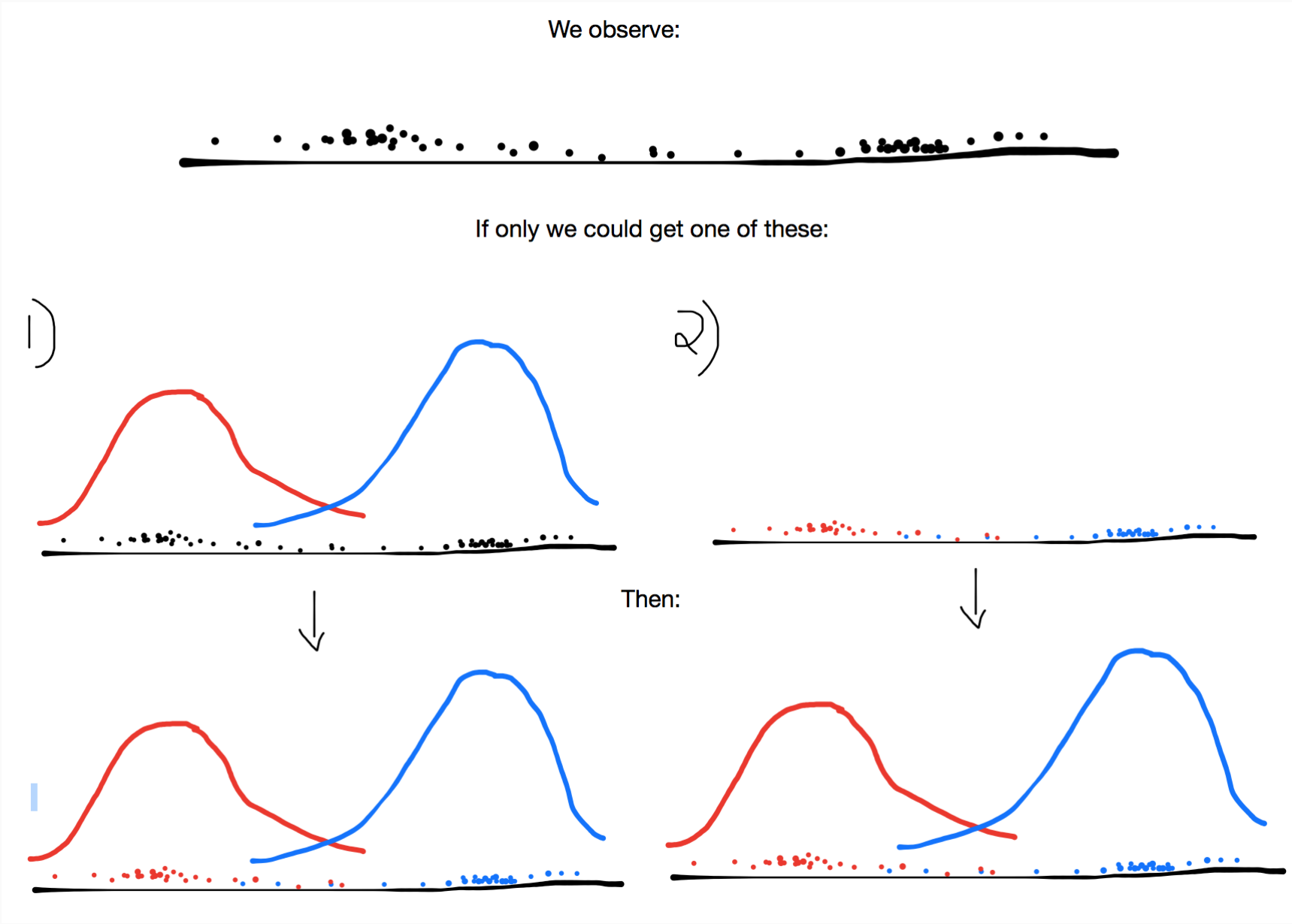
ML@B

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



Gaussian Mixture Models

ML@B

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 $\mathcal{N}(\vec{\mu}, \Sigma)$

Gaussian Mixture Models

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.

Gaussian Mixture Models

ML@B

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

EM Algorithm

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

EM Algorithm

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.

EM Algorithm

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.

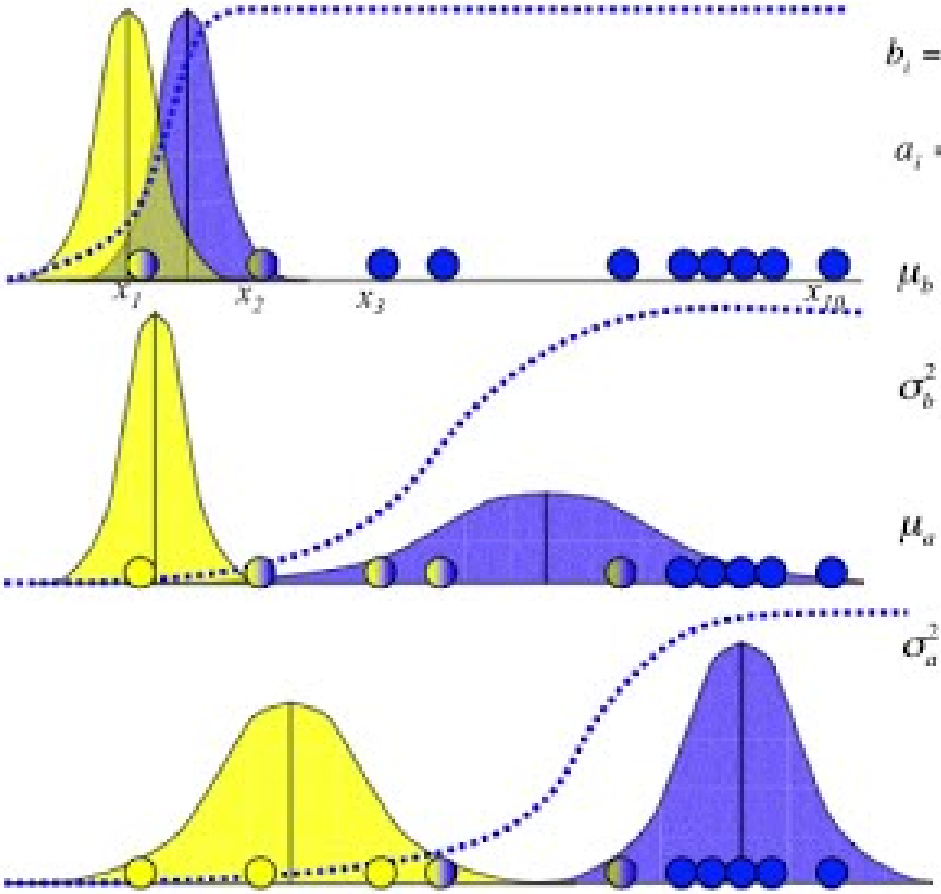
EM Algorithm

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

EM: 1-d example



$$P(x_i \mid b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left\{-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right\}$$

$$b_i = P(b \mid x_i) = \frac{P(x_i \mid b)P(b)}{P(x_i \mid b)P(b) + P(x_i \mid a)P(a)}$$

$$a_i = P(a \mid x_i) = 1 - b_i$$

$$\mu_b = \frac{b_1 x_1 + b_2 x_2 + \ldots + b_n x_n}{b_1 + b_2 + \ldots + b_n}$$

$$\sigma_b^2 = \frac{b_1 (x_1 - \mu_b)^2 + \ldots + b_n (x_n - \mu_b)^2}{b_1 + b_2 + \ldots + b_n}$$

$$\mu_a = \frac{a_1 x_1 + a_2 x_2 + \ldots + a_n x_n}{a_1 + a_2 + \ldots + a_n}$$

$$\sigma_a^2 = \frac{a_1 (x_1 - \mu_a)^2 + \ldots + a_n (x_n - \mu_a)^2}{a_1 + a_2 + \ldots + a_n}$$

could also estimate priors:

$P(b) = (b_1 + b_2 + \ldots b_n) / n$

$P(a) = 1 - P(b)$

EM Algorithm

Problem Setup:

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

EM Algorithm

Problem Setup:

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

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

EM Algorithm

Problem Setup:

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

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

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

EM Algorithm

Problem Setup:

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

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

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

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

EM Algorithm

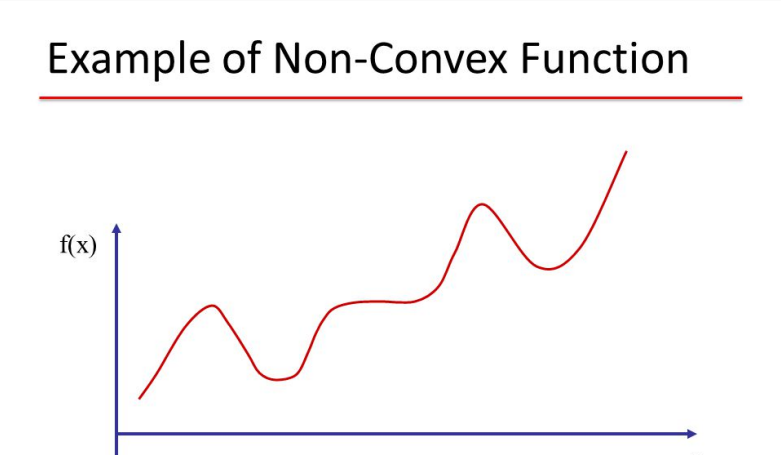
Problem Setup:

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

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

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

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



EM Algorithm

Algorithm:

1. Initialize θ_0

EM Algorithm

Algorithm:

1. Initialize θ_0
2. Repeat until convergence:

EM Algorithm

Algorithm:

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

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

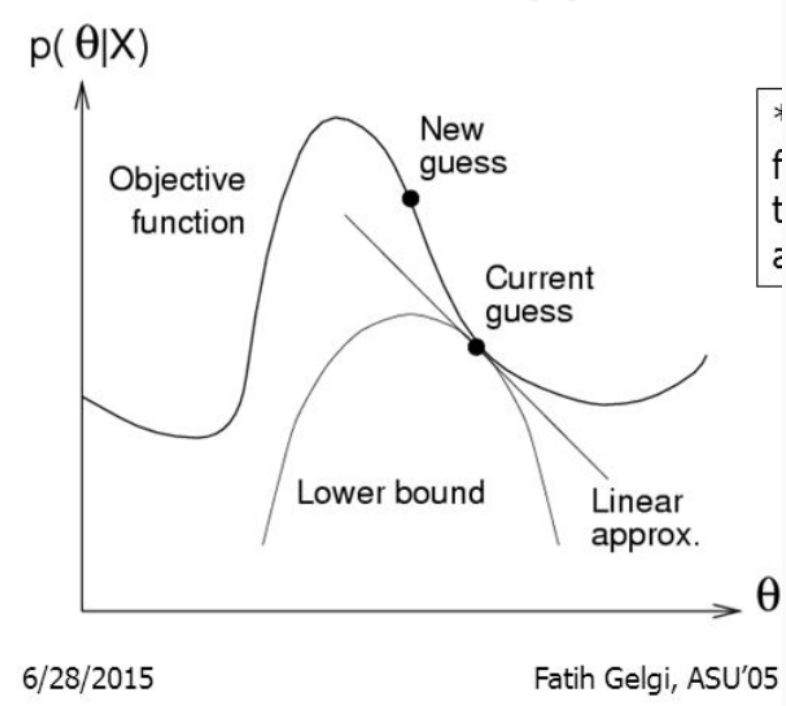
EM Algorithm

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

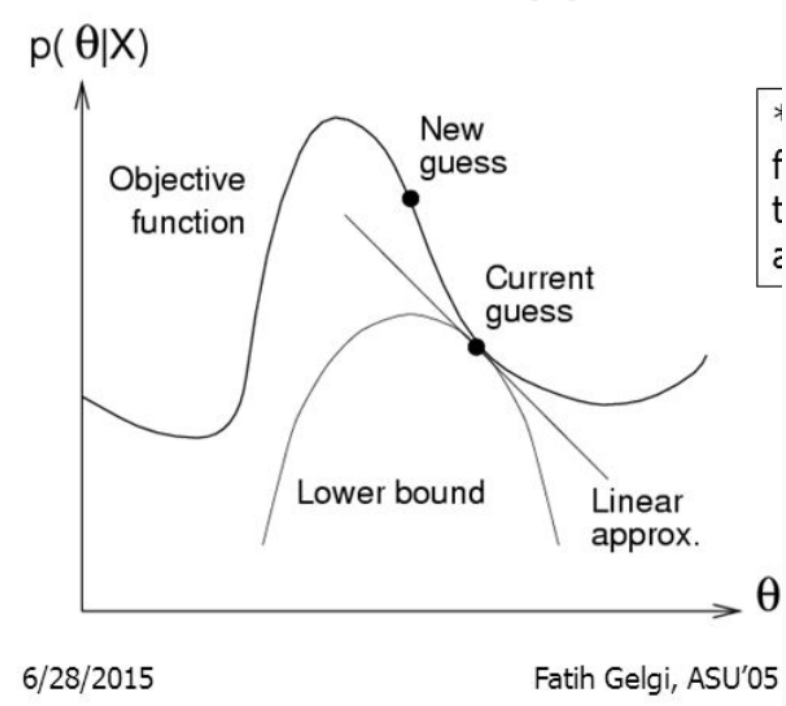
EM Algorithm

Alternate form of gradient descent.



EM Algorithm

Alternate form of gradient descent.



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

EM Algorithm

Pros:

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

Cons:

EM Algorithm

Pros:

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

Cons:

EM Algorithm

Pros:

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

Cons:

- Could converge to local maximum.

EM Algorithm

Pros:

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

Cons:

- Could converge to local maximum.

EM Algorithm

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!

EM Algorithm

K-means vs GMM:

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

EM Algorithm

K-means vs GMM:

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

EM Algorithm

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

Distance Functions

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

Common choice is L2 norm (euclidean distance).

Distance Functions

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

Common choice is L2 norm (euclidean distance).

Must satisfy:

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

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

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

Hierarchical Clustering

Agglomerative Clustering

Main Idea: merge the most similar instances, and repeat

Agglomerative Clustering

ML@B

Main Idea: merge the most similar instances, and repeat

Algorithm:

1. Initialize each datapoint to its own cluster

Agglomerative Clustering

ML@B

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

Agglomerative Clustering

ML@B

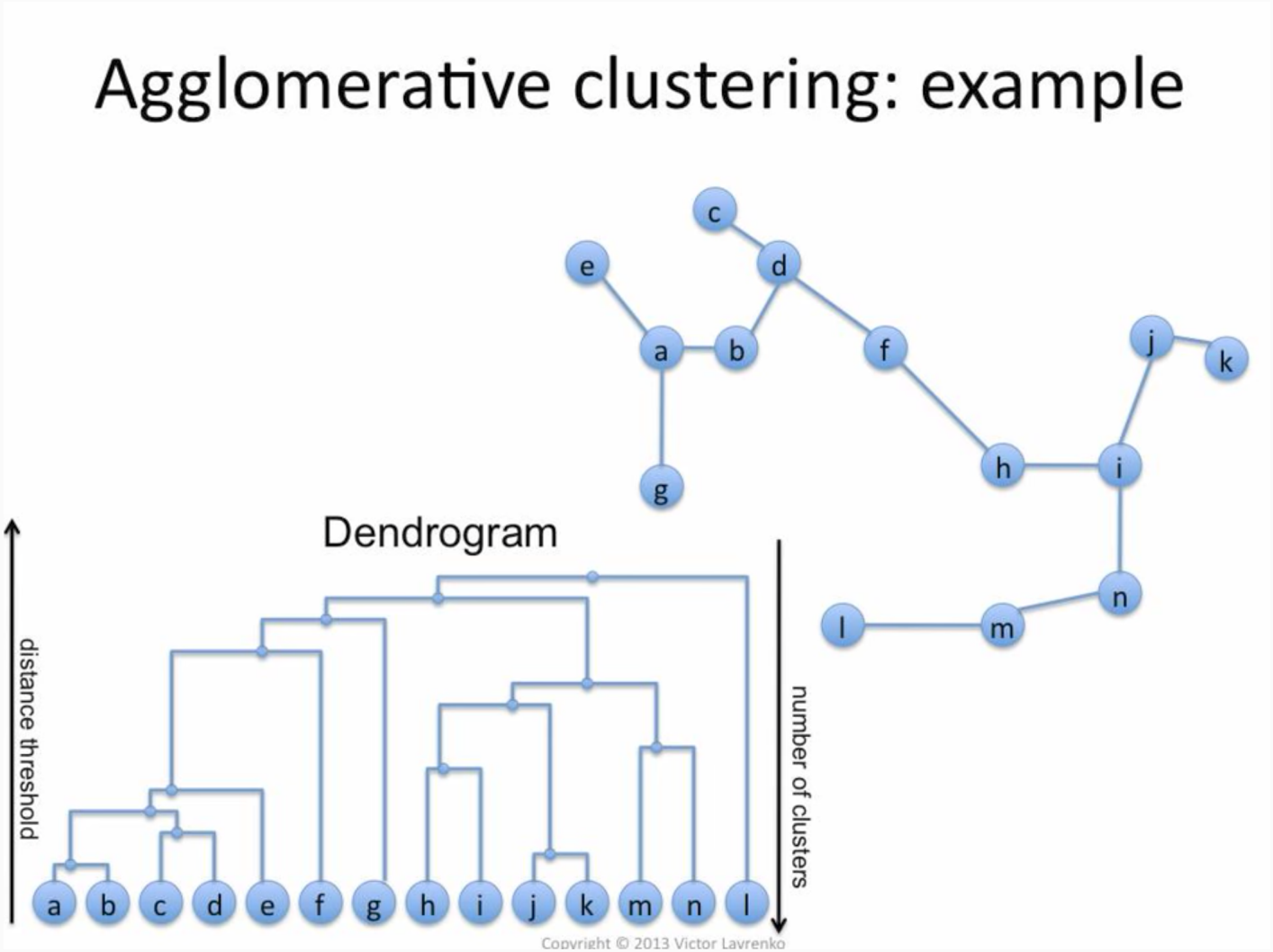
Main Idea: merge the most similar instances, and repeat

Algorithm:

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

Agglomerative Clustering

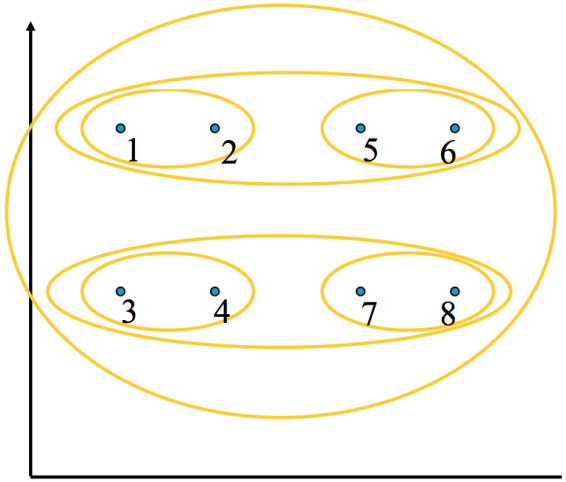
"Bottom up" approach, builds Dendogram.



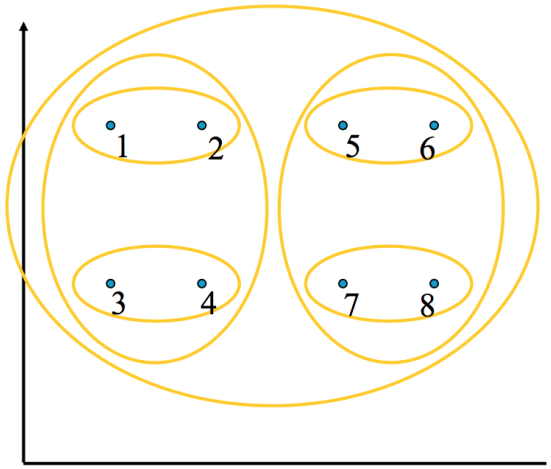
Agglomerative Clustering

Many ways to define distance between two instances.

Closest pair
(single-link clustering)



Farthest pair
(complete-link clustering)



[Pictures from Thorsten Joachims]

Divisive Clustering

Top down approach, still builds dendogram.

1. Start with everything in one cluster

Divisive Clustering

Top down approach, still builds dendogram.

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

Divisive Clustering

Top down approach, still builds dendogram.

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

Divisive Clustering

Top down approach, still builds dendogram.

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

Divisive Clustering

Top down approach, still builds dendogram.

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

ML@B

My favorite! :)

Very new way of thinking about clustering

Super efficient

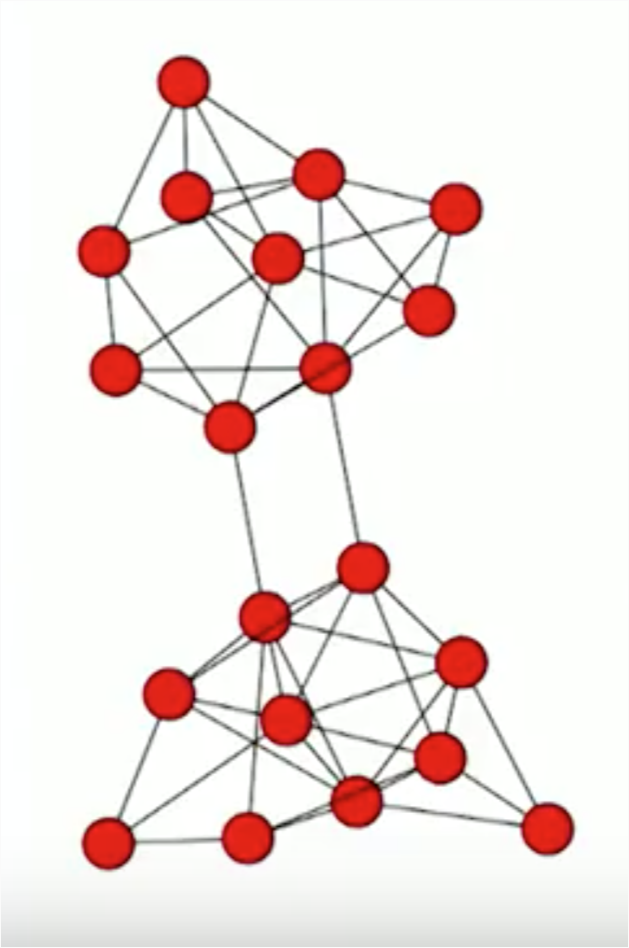
Produces amazing results

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

Spectral Clustering

How to weight the edges?

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

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

Spectral Clustering

How to weight the edges?

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

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

Spectral Clustering

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

Spectral Clustering

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

Spectral Clustering

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)

Spectral Clustering

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)

Spectral Clustering

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?

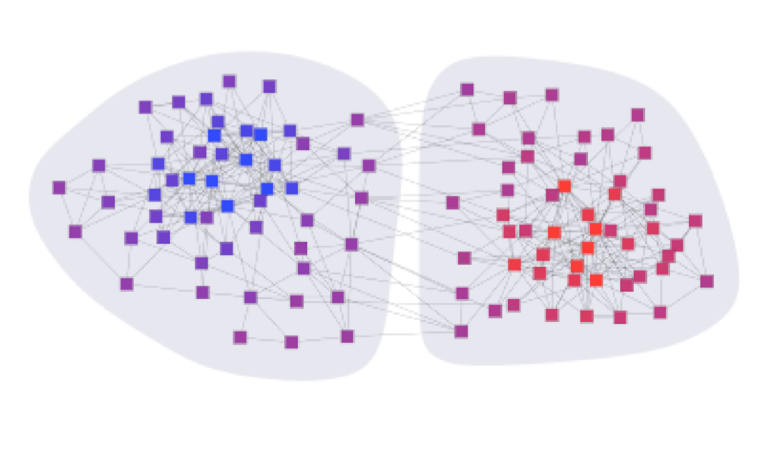
- Can connect every edge
- Connect to k nearest neighbors
- Connect to points within ϵ (some threshold)

Different graphs have different properties, give different results.

Spectral Graphs

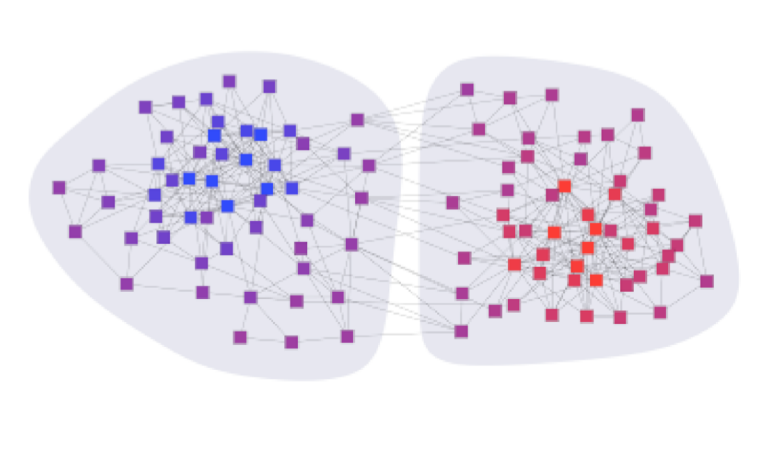
ML@B

So now we have this graph.



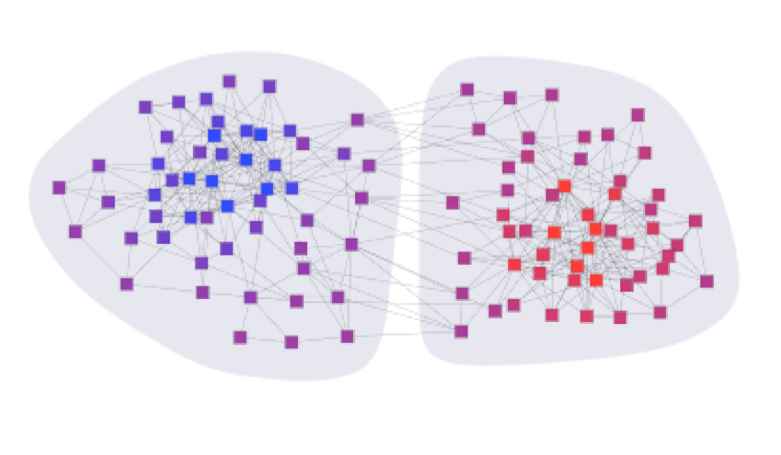
Spectral Graphs

So now we have this graph.



What do we want to find? Min cut!

So now we have this graph.

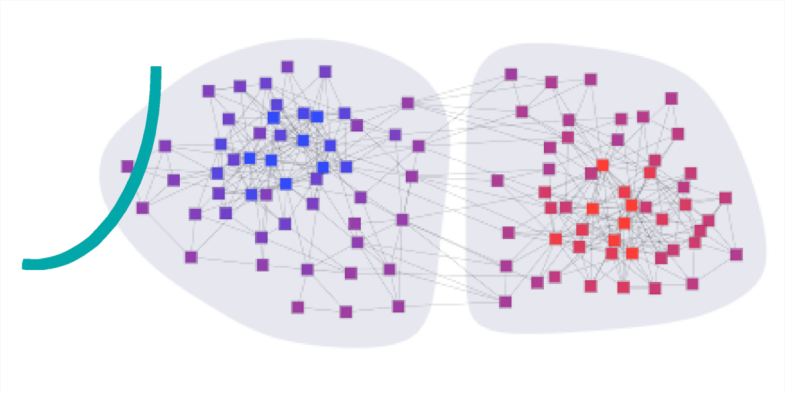


What do we want to find? Min cut!

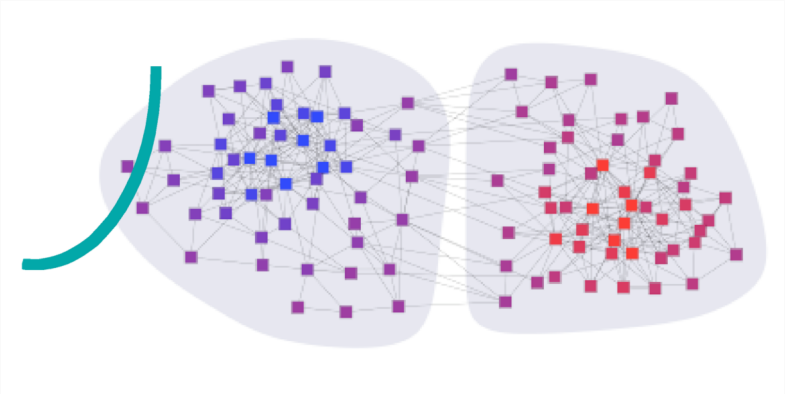
$$\underset{S_1,S_2:\{S_1,S_2\}=V}{\operatorname{argmin}} \sum_{i\in S_1, j\in S_2} d(i,j)$$

ML@B

Problem:

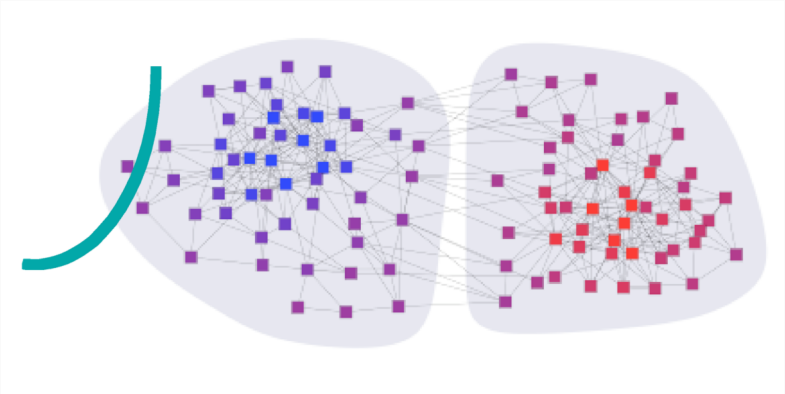


Problem:



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

Problem:



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

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

Finding normalized mincut is NP-Hard.

Spectral Graphs

Finding normalized mincut is NP-Hard.

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

Spectral Graphs

Finding normalized mincut is 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

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

Spectral Clustering

First step: construct the Laplacian.

$$L = D - W$$

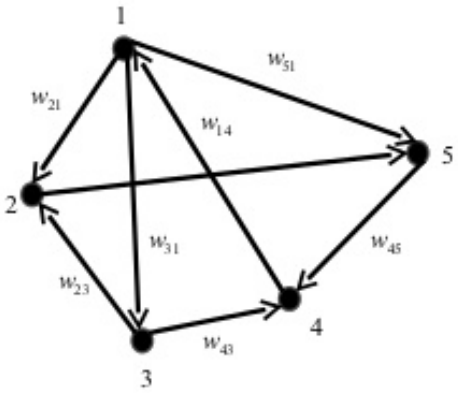
W = adjacency matrix

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

D = diagonal matrix

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

Graph Dynamics (Diagraph)



Adjacency Matrix	$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$ or $A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & w_{14} & 0 \\ w_{21} & 0 & w_{23} & 0 & 0 \\ w_{31} & 0 & 0 & 0 & 0 \\ 0 & 0 & w_{43} & 0 & w_{45} \\ w_{51} & w_{52} & 0 & 0 & 0 \end{bmatrix} \end{array}$
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:

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

Spectral Clustering

Step two: eigen decomposition

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

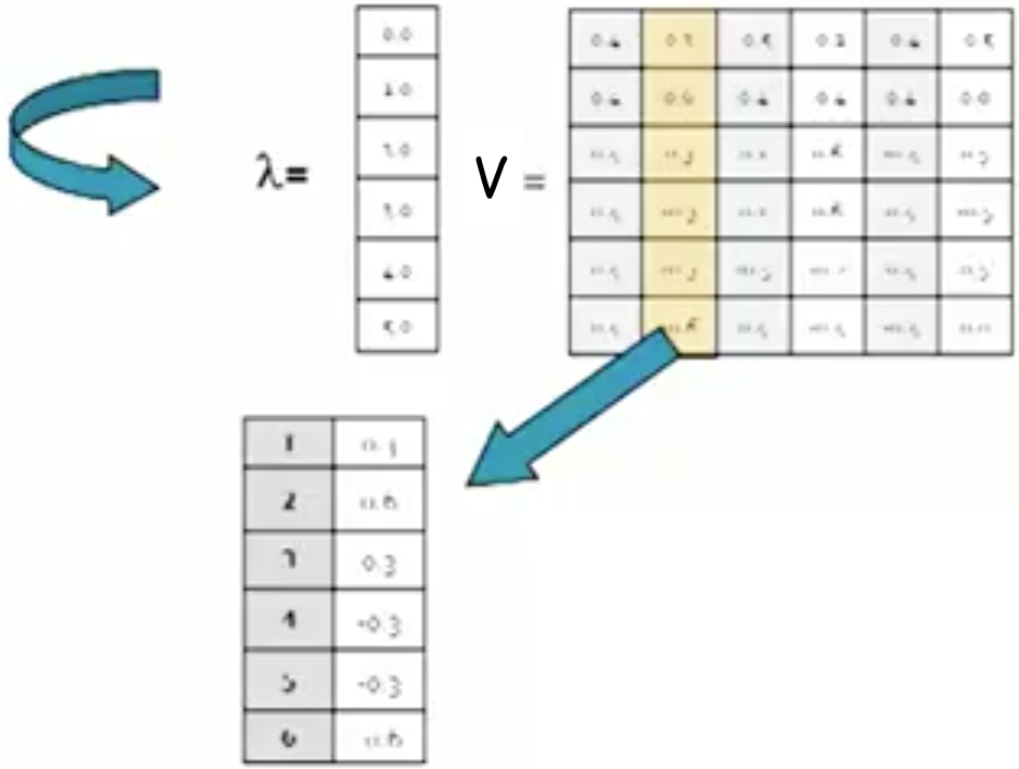
Take a look at eigenvalues! Measure of "connectivity" of graph.
For k connected components:

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

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

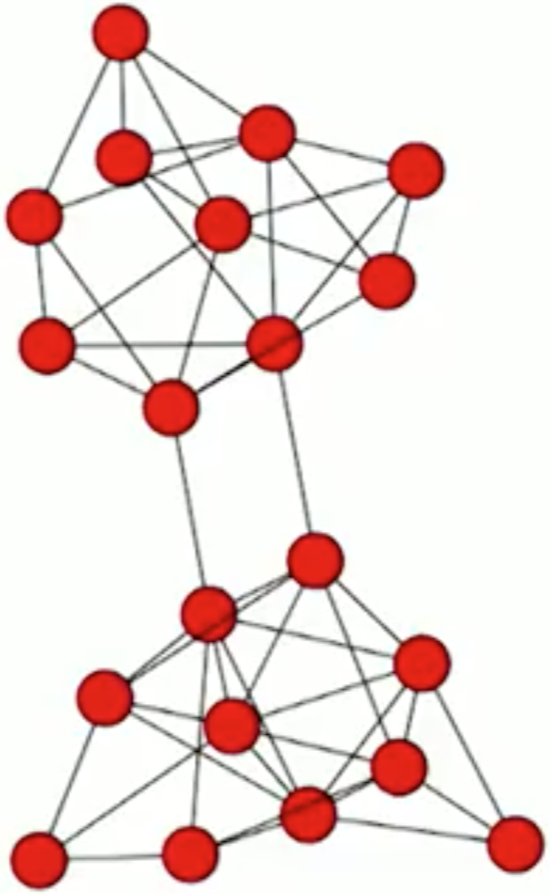
Spectral Clustering

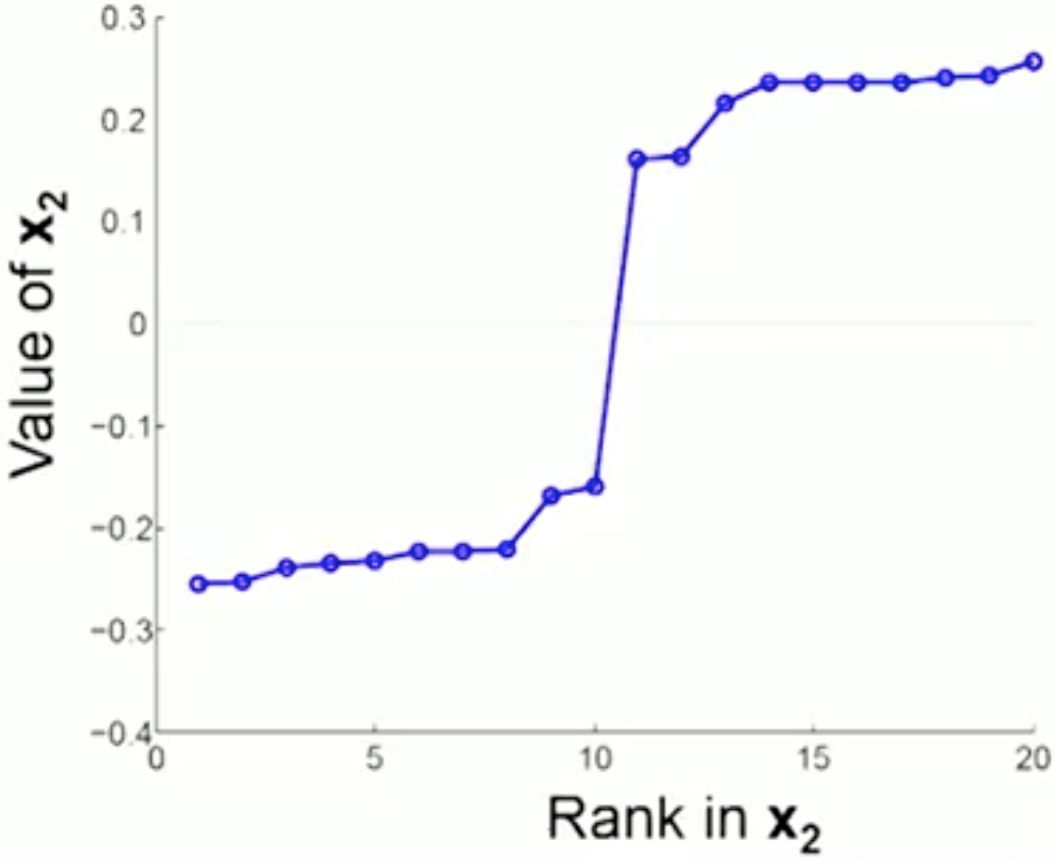
- **2) Decomposition:**
  - Find eigenvalues λ and eigenvectors V of the matrix L
  - Map vertices to corresponding components of λ_2



Material adapted from Stanford

Why is looking at v_2 reasonable?

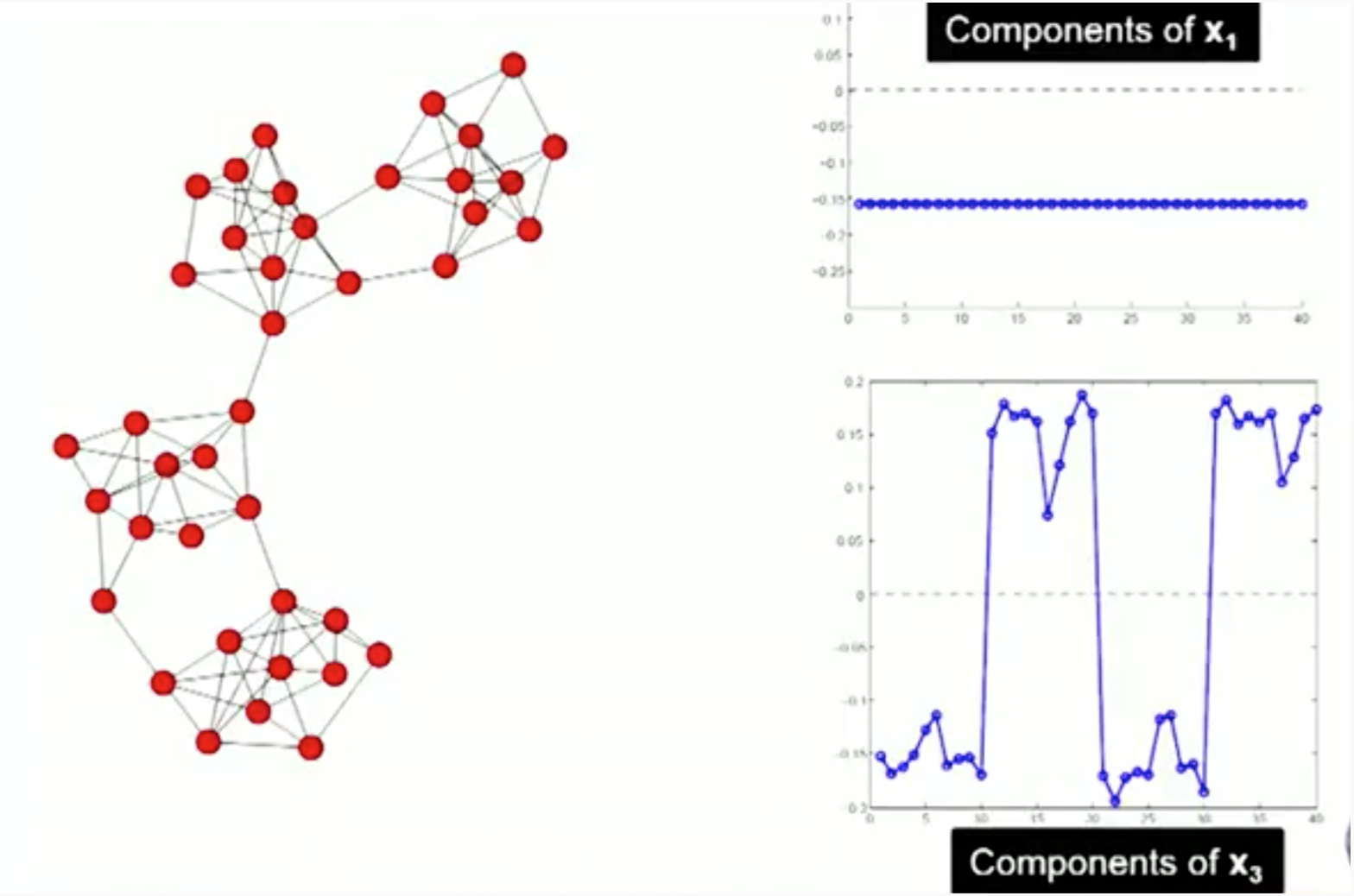




Material adapted from Stanford

Spectral Clustering

Even cooler!



Material adapted from Stanford

Spectral Clustering

Use v_2 to split vertices into 2 sets:

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

Spectral Clustering

Use v_2 to split vertices into 2 sets:

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

Common values for $\epsilon : 0, median(v_2)$

Spectral Clustering

Use v_2 to split vertices into 2 sets:

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

Common values for $\epsilon : 0, median(v_2)$

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

Spectral Clustering

Alternate method:

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

Spectral Clustering

ML@B

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!!)

Spectral Clustering

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

Spectral Clustering

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

Spectral Clustering

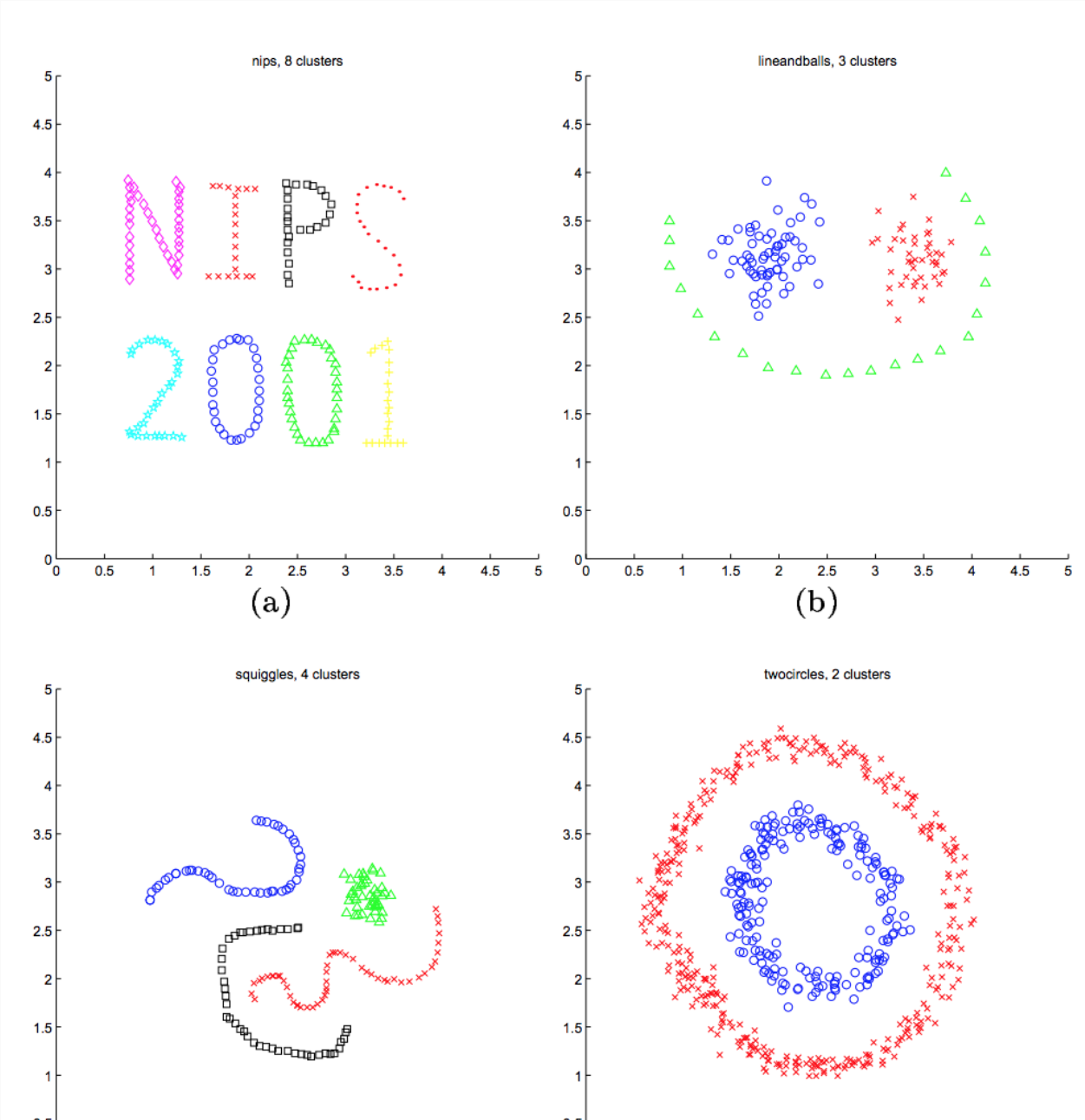
ML@B

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

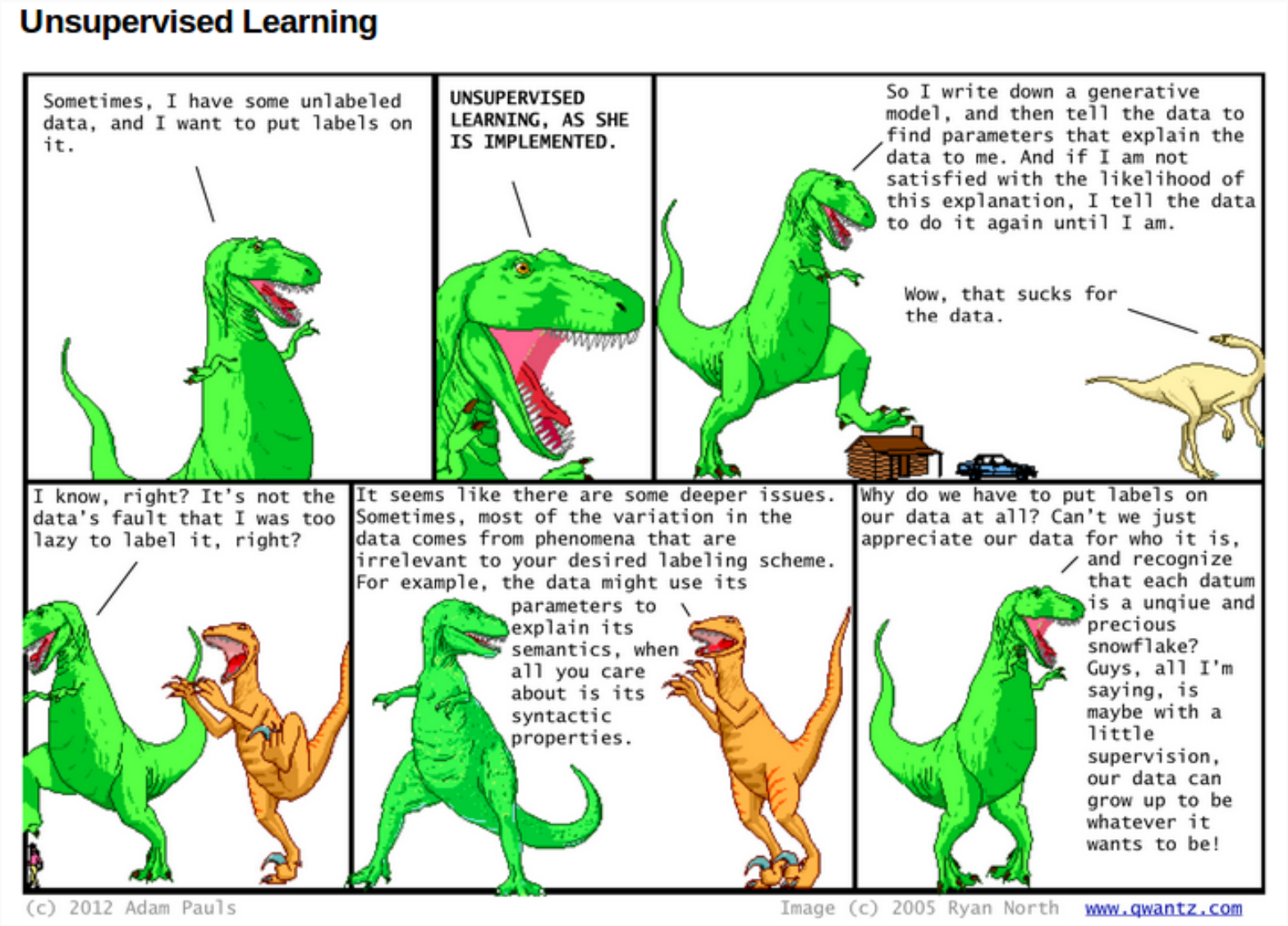
Emperically, this produces best results. Amazing results, in fact.

Spectral Clustering Results



Questions

Unsupervised Learning comic



Questions?