COMS 4771

Clustering

Nakul Verma

Supervised Learning

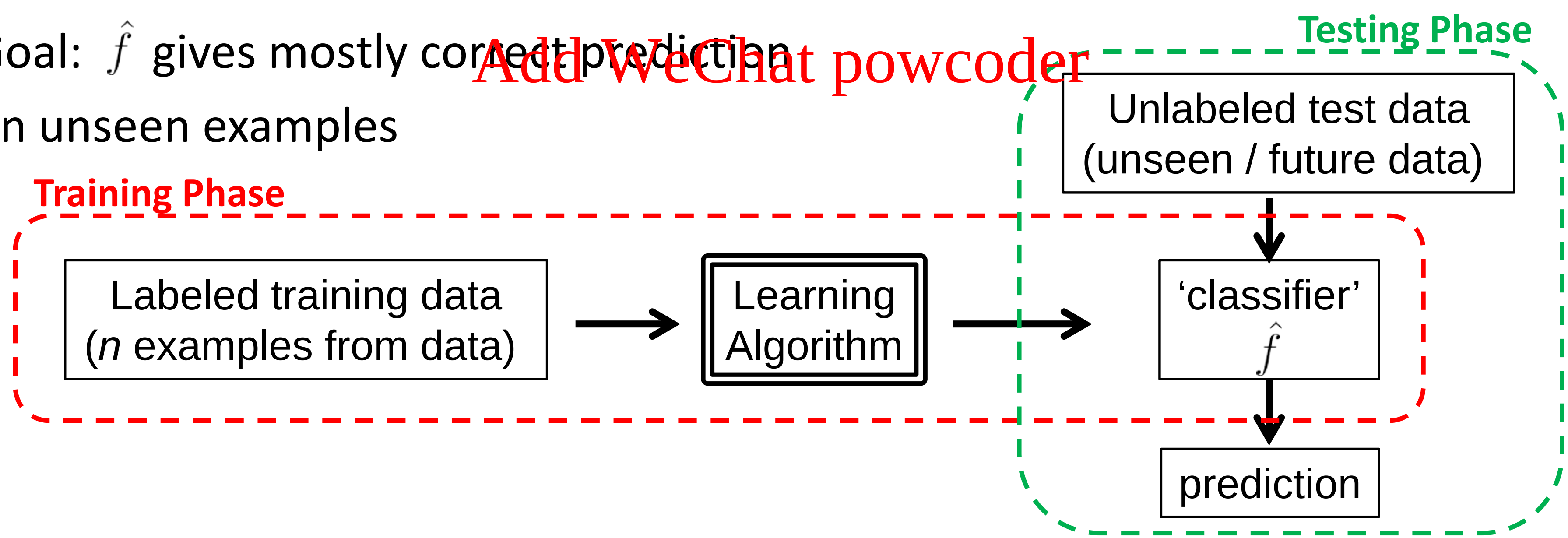
Supervised learning

Data: $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots \in \mathcal{X} \times \mathcal{Y}$

Assumption: there is a (relatively simple) function $f^* : \mathcal{X} \to \mathcal{Y}$
such that $f^*(\vec{x}_i) = y_i$ for most *i*

Learning task: given *n* examples from the data, find an approximation $\hat{f} \approx f^*$

Goal: $\hat{f}$ gives mostly correct prediction on unseen examples



Unsupervised Learning

Unsupervised learning

Data: $\vec{x}_1, \vec{x}_2, \ldots \in \mathcal{X}$

Assumption: there is an underlying structure in $\mathcal{X}$

Learning task: discover the structure given *n* examples from the data

Goal: come up with the summary of the data using the discovered structure

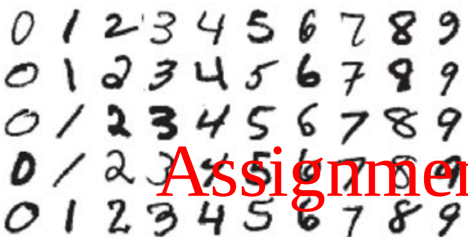
Partition the data into meaningful structures — **clustering**

Find a low-dimensional representation that retains important information, and suppresses irrelevant/noise information — **Dimensionality reduction**

Let’s take a closer look using an example…

Example: Handwritten digits revisited

Handwritten digit data, but with no labels

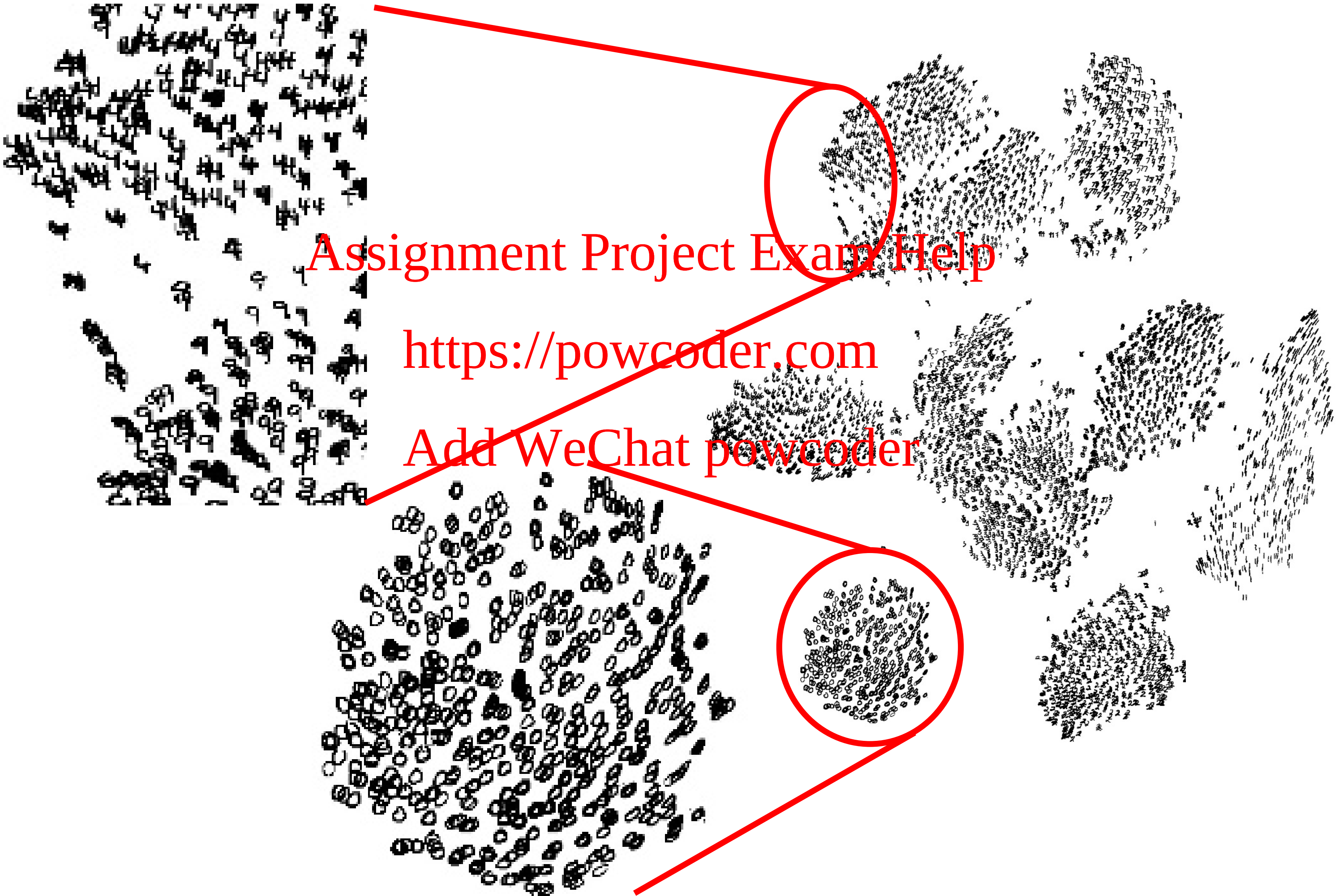


What can we do?

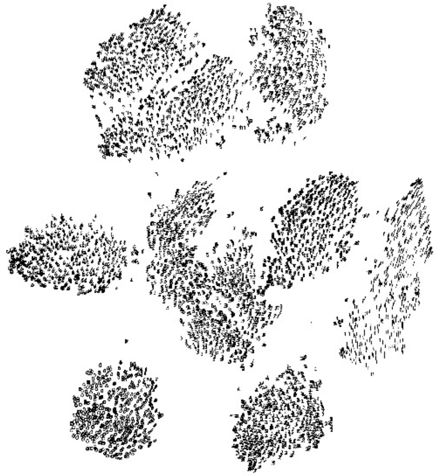
- Suppose know that there are 10 groupings, can we ***find the groups***?
- What if we don't know there are 10 groups?
- How can we *discover/explore* other structure in such data?

A 2D visualization of digits dataset

Handwritten digits visualization



Grouping The Data, aka Clustering



Data: $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathcal{X}$

Given: known target number of groups *k*

Output: Partition $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n$ into *k* groups.

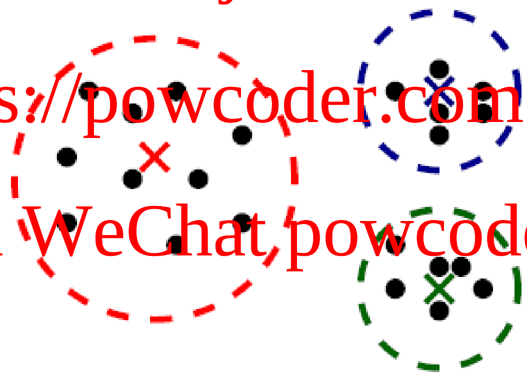
This is called the clustering problem,
also known as unsupervised classification, or quantization

k-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Idea:

find a set of representatives $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ such that data is **close to** some representative



Optimization:

$$\text{minimize}_{c_1,\ldots,c_k} \quad \left[\sum_{i=1}^{n} \min_{j=1,\ldots,k} \|\vec{x}_i - \vec{c}_j\|^2 \right]$$

Unfortunately this is NP-hard
Even for d=2 and k=2

How do we optimize this?

How do we solve for
d=1 or k=1 case?

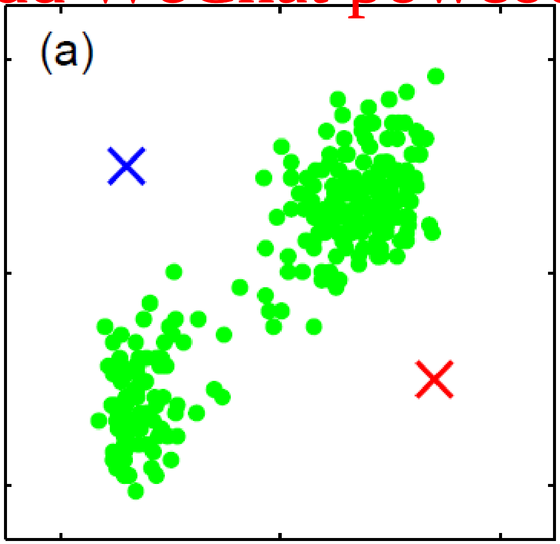
Algorithm to approximate *k*-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Alternating optimization algorithm:

- Initialize cluster centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (say randomly)
- Repeat till no more changes occur
  - Assign data to its closest center (this creates a partition) (assume centers are fixed)
  - Find the optimal centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (assuming the data partition is fixed)

Demo:



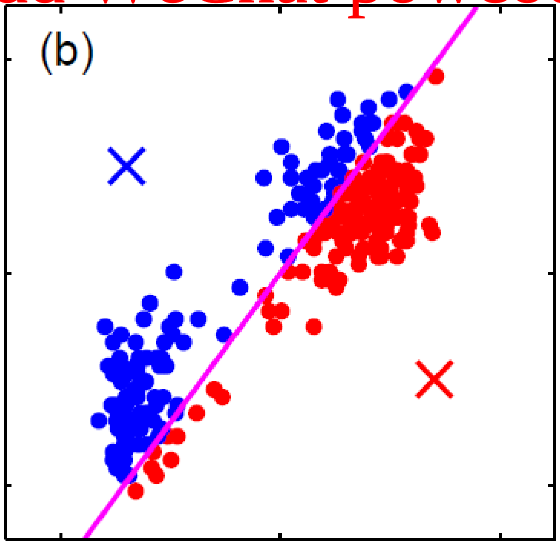
Algorithm to approximate *k*-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Alternating optimization algorithm:

- Initialize cluster centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (say randomly)
- Repeat till no more changes occur
  - Assign data to its closest center (this creates a partition) (assume centers are fixed)
  - Find the optimal centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (assuming the data partition is fixed)

Demo:



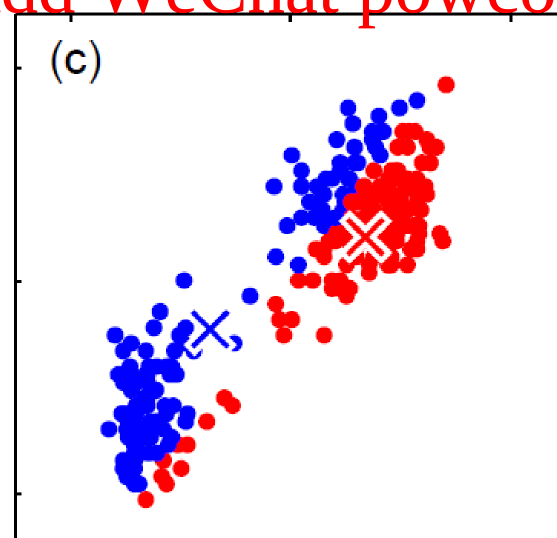
Algorithm to approximate *k*-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Alternating optimization algorithm:

- Initialize cluster centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (say randomly)
- Repeat till no more changes occur
  - Assign data to its closest center (this creates a partition) (assume centers are fixed)
  - Find the optimal centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (assuming the data partition is fixed)

Demo:



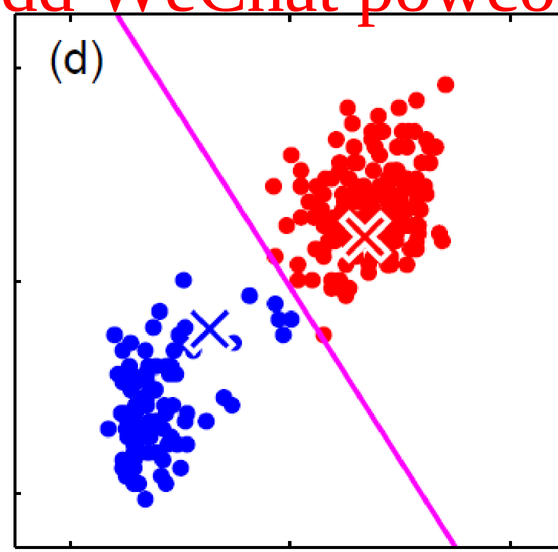
Algorithm to approximate *k*-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Alternating optimization algorithm:

- Initialize cluster centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (say randomly)
- Repeat till no more changes occur
  - Assign data to its closest center (this creates a partition) (assume centers are fixed)
  - Find the optimal centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (assuming the data partition is fixed)

Demo:



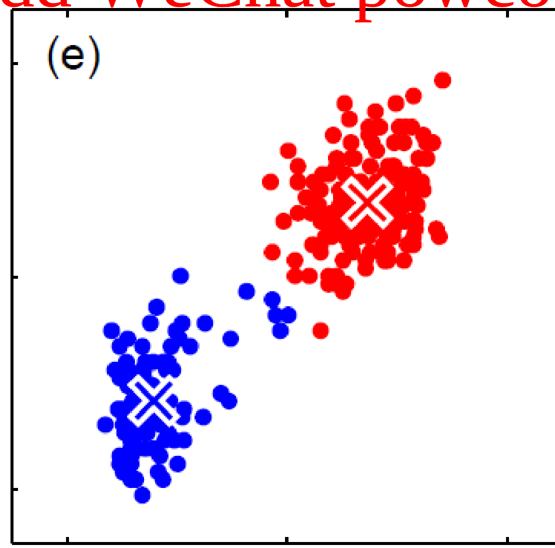
Algorithm to approximate *k*-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Alternating optimization algorithm:

- Initialize cluster centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (say randomly)
- Repeat till no more changes occur
  - Assign data to its closest center (this creates a partition) (assume centers are fixed)
  - Find the optimal centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (assuming the data partition is fixed)

Demo:



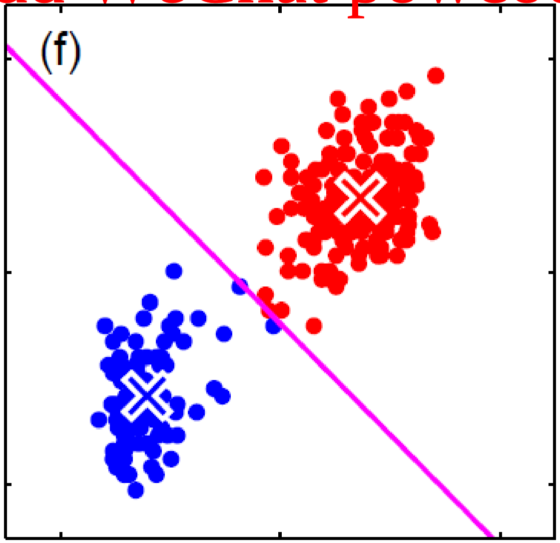
Algorithm to approximate *k*-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Alternating optimization algorithm:

- Initialize cluster centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (say randomly)
- Repeat till no more changes occur
  - Assign data to its closest center (this creates a partition) (assume centers are fixed)
  - Find the optimal centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (assuming the data partition is fixed)

Demo:



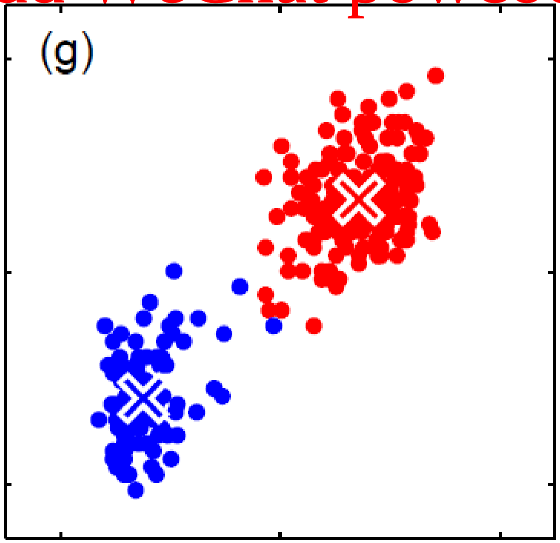
Algorithm to approximate *k*-means

Given: data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$, and intended number of groupings *k*

Alternating optimization algorithm:

- Initialize cluster centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (say randomly)
- Repeat till no more changes occur
  - Assign data to its closest center (this creates a partition) (assume centers are fixed)
  - Find the optimal centers $\vec{c}_1, \vec{c}_2, \ldots \vec{c}_k$ (assuming the data partition is fixed)

Demo:



k-means

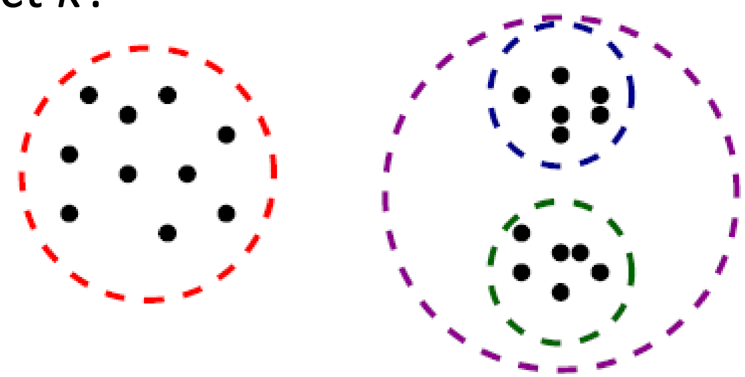
Some properties of this alternating updates algorithm:

- The approximation can be arbitrarily bad, compared to the best cluster assignment!

- Performance quality heavily dependent on the initialization!

k-means:

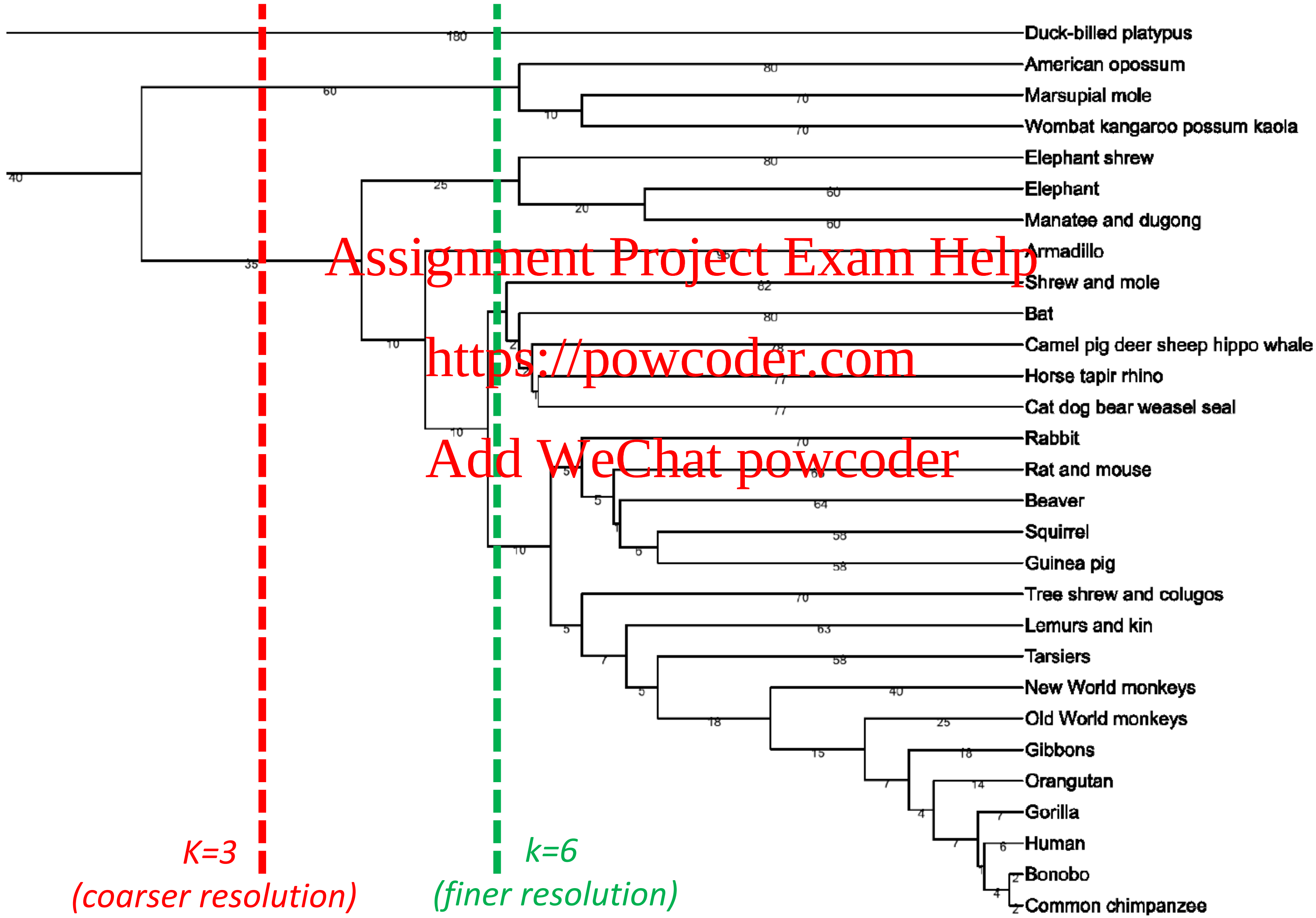
- How to select *k*?



is the right k=2 or k=3?

***Solution**: encode clustering for all values of k! (hierarchical clustering)*

Example: Clustering Without Committing to *k*



Hierarchical Clustering

Two approaches:

Top Down (divisive):

- Partition data into two groups (say, by k-means, with k=2)
- Recurse on each part
- Stop when cannot partition data anymore (ie single points left)

Bottom Up (agglomerative):

- Start by each data sample as its own cluster (so initial number of clusters is *n*)
- Repeatedly merge "closest" pair of clusters
- Stop when only one cluster is left

Clustering via Probabilistic Mixture Modeling

Alternative way to cluster data:

Given: $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$ and number of intended number of clusters *k*.

Assume a joint probability distribution (X, C) over the joint space $\mathbf{R}^d \times [k]$

$$C \sim \begin{pmatrix} \pi_1 \\ \vdots \\ \pi_k \end{pmatrix}$$ *Discrete distribution over the clusters P[C=i] =* π_i

$X|C = i \sim$ *Some multivariate distribution, e.g.* $N(\vec{\mu}_i, \Sigma_i)$

Parameters: $\theta = (\pi_1, \vec{\mu}_1, \Sigma_1, \ldots, \pi_k, \vec{\mu}_k, \Sigma_k)$ ***looks familiar?***

Modeling assumption data (x_1,c_1),…, (x_n,c_n) i.i.d. from $\mathbf{R}^d \times [k]$

BUT only get to see partial information: $x_1, x_2, \ldots, x_n$ ($c_1, \ldots, c_n$ hidden!)

Gaussian Mixture Modeling (GMM)

Given: $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^d$ and *k*.

Assume a joint probability distribution (X, C) over the joint space $\mathbf{R}^d \times [k]$

$$C \sim \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_k \end{bmatrix} \qquad X | C = i \sim N(\vec{\mu}_i, \Sigma_i)$$

Gaussian Mixture Model

$$\theta = (\pi_1, \vec{\mu}_1, \Sigma_1, \ldots, \pi_k, \vec{\mu}_k, \Sigma_k)$$

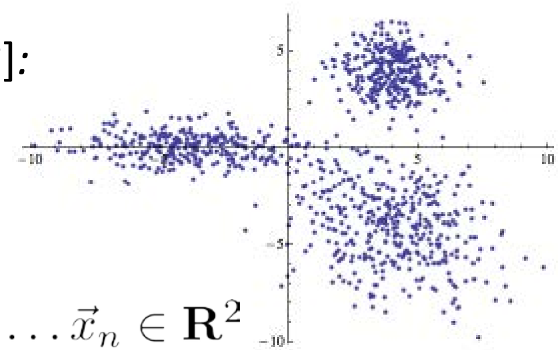
$$P[\vec{x} \mid \theta] = \sum_{i=1}^{k} \pi_i \frac{1}{\sqrt{(2\pi)^d \det(\Sigma_i)}} \exp\left\{ -\frac{1}{2} (\vec{x} - \vec{\mu}_i)^{\mathsf{T}} \Sigma_i^{-1} (\vec{x} - \vec{\mu}_i) \right\}$$

Mixing weight

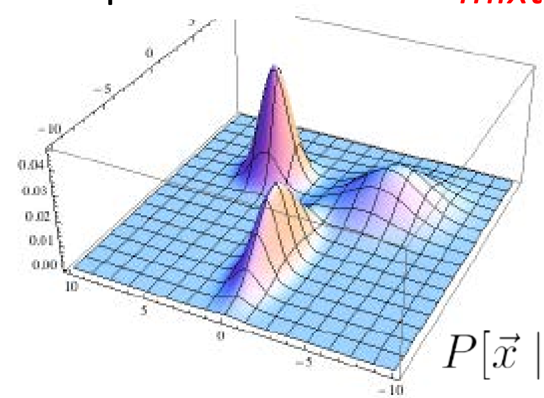
Mixture component

(this is called a mixture model)

Example in **R**^{2} *x [3]:*



$\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbf{R}^2$



$P[\vec{x} \mid \theta]$

GMM: Parameter Learning

$$P[\vec{x} \mid \theta] = \sum_{i=1}^{k} \pi_i \frac{1}{\sqrt{(2\pi)^d \det(\Sigma_i)}} \exp\left\{ -\frac{1}{2}(\vec{x}-\vec{\mu}_i)^{\mathsf{T}} \Sigma_i^{-1} (\vec{x}-\vec{\mu}_i) \right\}$$

$$\theta = (\pi_1, \vec{\mu}_1, \Sigma_1, \ldots, \pi_k, \vec{\mu}_k, \Sigma_k)$$

So... how to learn the parameters θ?

MLE approach:

Given data $\vec{x}_1, \vec{x}_2, \ldots \vec{x}_n \in \mathbb{R}^d$ i.i.d.

$$\theta_{\text{MLE}} := \arg\max_{\theta} \sum_{i=1}^{n} \ln P[\vec{x}_i \mid \theta]$$

$$= \arg\max_{\theta} \sum_{i=1}^{n} \ln \left[\sum_{j=1}^{n} \pi_j \frac{1}{\sqrt{(2\pi)^d \det(\Sigma_j)}} \exp\left\{ -\frac{1}{2}(\vec{x}-\vec{\mu}_j)^{\mathsf{T}} \Sigma_j^{-1} (\vec{x}-\vec{\mu}_j) \right\} \right]$$

Cannot really simplify further!

ummm.... now what?

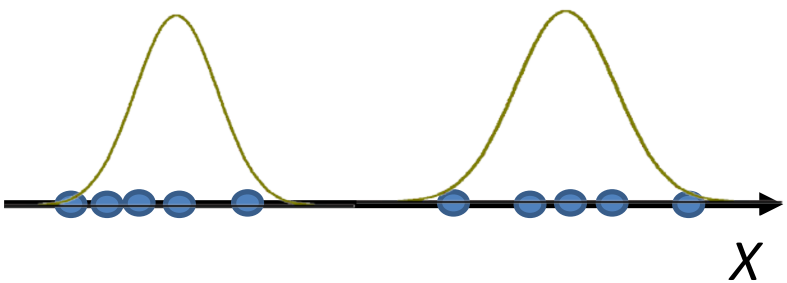
GMM: Maximum Likelihood

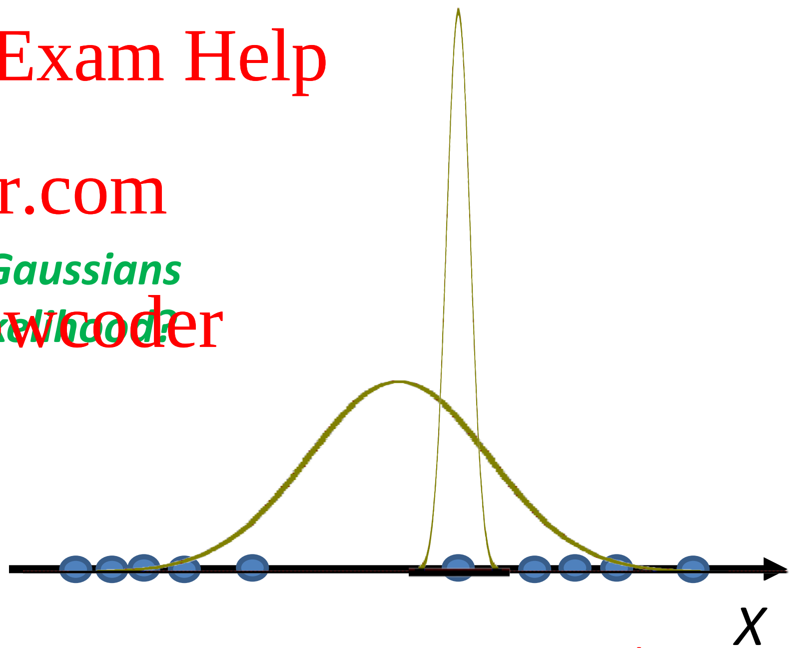
MLE for Mixture modeling (like GMMs) is NOT a convex optimization problem

In fact **Maximum** Likelihood Estimate for GMMs is degenerate!

X = **R**, k = 2 (fit two Gaussian in 1d):

Which pair of Gaussians gives higher likelihood?





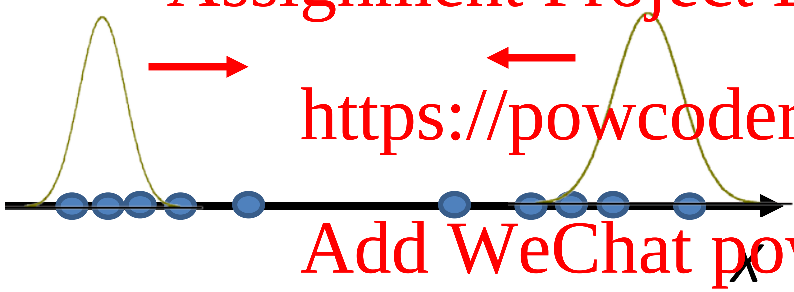
as $\sigma \to 0$, MLE $\to \infty$!

Aside: why doesn't this occur when fitting one Gaussian?

GMM: (local) Maximum Likelihood

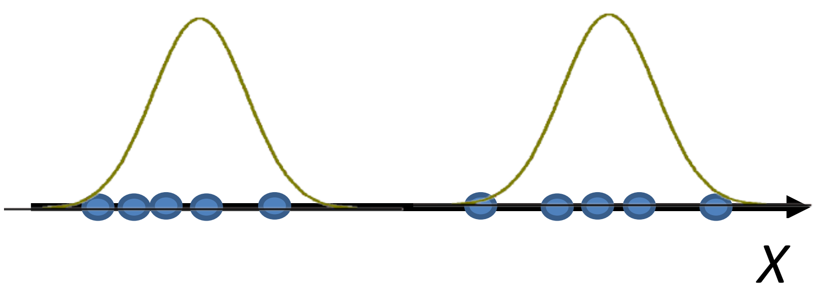
So, can we make any progress?

Observation: even though a global MLE maximizer is not appropriate, several local maximizers are desirable!



An example non-maximized likelihood

(do a few steps of gradient ascent)



Reaches a desirable local maximum!

A better algorithm for finding good parameters: Expectation Maximization (EM)

Expectation Maximization (EM) Algorithm

Similar in spirit to the alternating update for *k*-means algorithm

Idea:

- Initialize the parameters arbitrarily
- Given the current setting of parameters find the best (soft) assignment of data samples to the clusters (**Expectation-step**)
- Update all the parameters with respect to the current (soft) assignment that maximizes the likelihood (**Maximization-step**)
- Repeat until no more progress is made.

EM for GMM

Initialize $\theta = (\pi_1, \vec{\mu}_1, \Sigma_1, \ldots, \pi_k, \vec{\mu}_k, \Sigma_k)$ arbitrarily

Expectation-step: For each $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k\}$ compute the assignment $w_j^{(i)}$ of data x_i to cluster j

$$w_j^{(i)} := \frac{\pi_j \sqrt{\det(\Sigma_j^{-1})} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_j)^{\mathsf{T}} \Sigma_j^{-1} (\vec{x} - \vec{\mu}_j)\right)}{\sum_{j'=1}^{k} \pi_{j'} \sqrt{\det(\Sigma_{j'}^{-1})} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_{j'})^{\mathsf{T}} \Sigma_{j'}^{-1} (\vec{x} - \vec{\mu}_{j'})\right)}$$

Maximization-step: Maximize the log-likelihood of the parameters

$$n_j := \sum_{i=1}^{n} w_j^{(i)}$$

Effective number of points assigned to cluster j

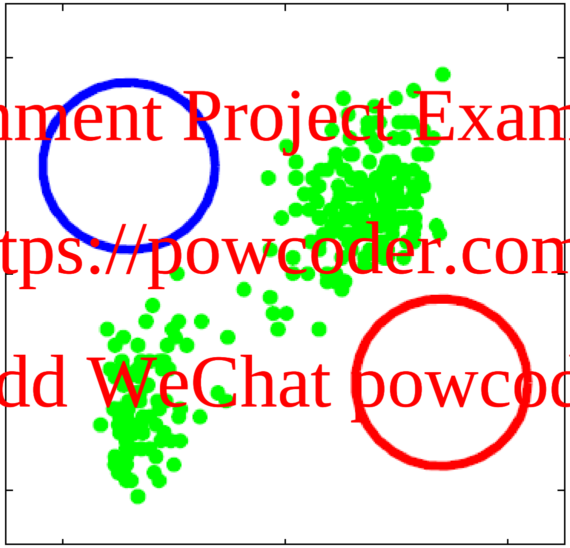
$$\pi_j := \frac{n_j}{n}$$

$$\vec{\mu}_j := \frac{1}{n_j} \sum_{i=1}^{n} w_j^{(i)} \vec{x}_i$$

$$\Sigma_j := \frac{1}{n_j} \sum_{i=1}^{n} w_j^{(i)} (\vec{x}_i - \vec{\mu}_j)(\vec{x}_i - \vec{\mu}_j)^{\mathsf{T}}$$

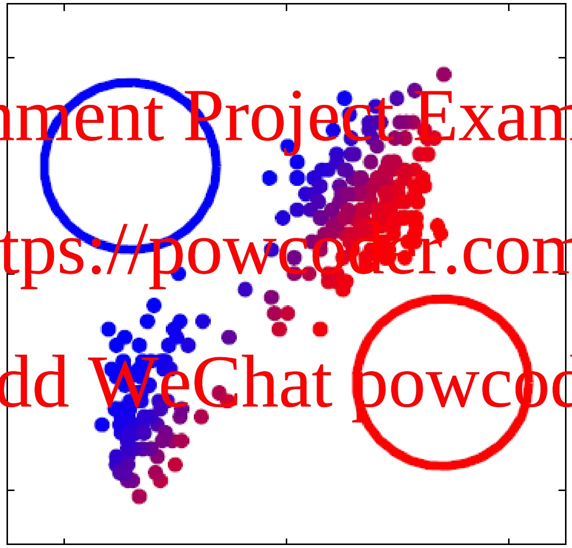
Why?

EM for GMM in Action



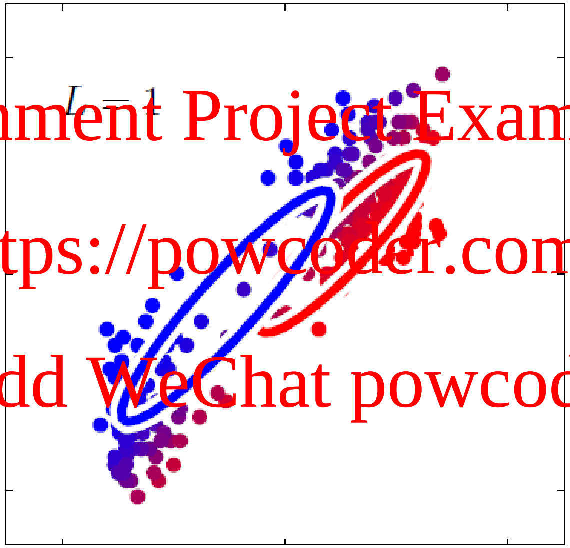
Arbitrary θ assignment

EM for GMM in Action



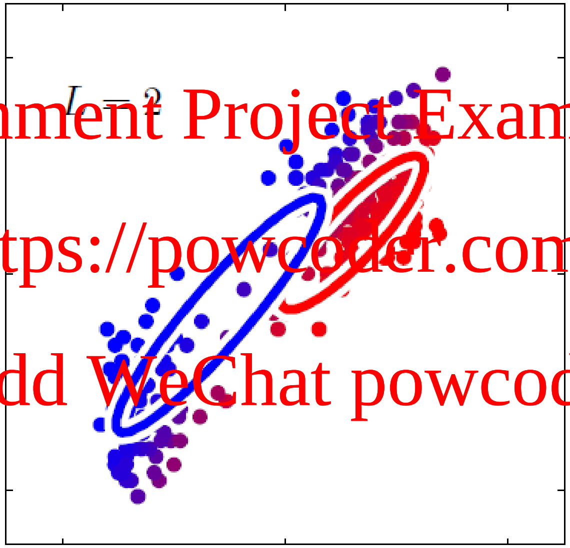
E step: soft assignment of data

EM for GMM in Action



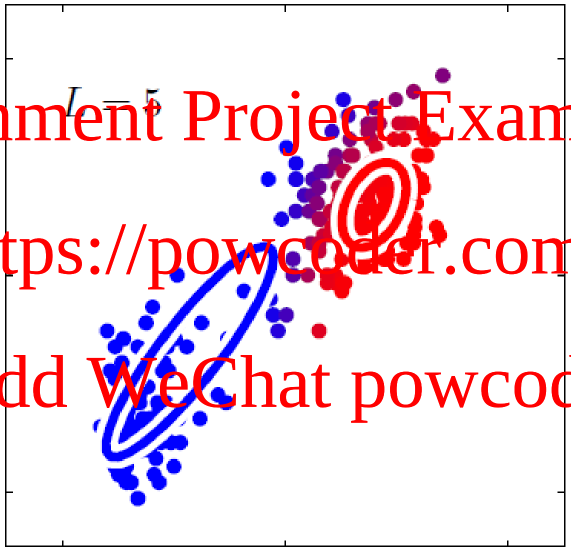
M step: Maximize parameter estimate

EM for GMM in Action



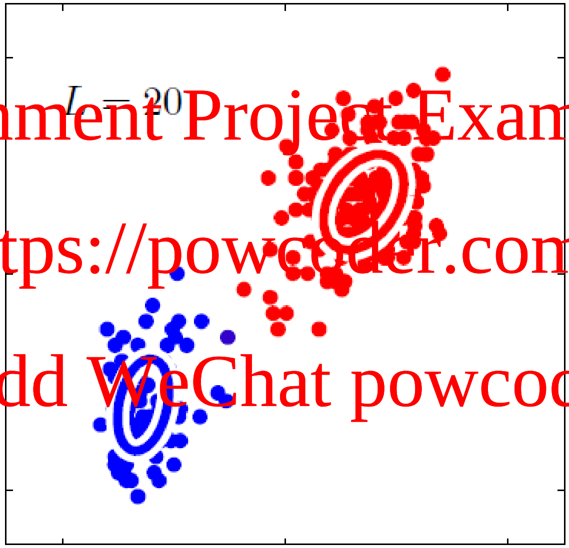
After two rounds

EM for GMM in Action



After five rounds

EM for GMM in Action



After twenty rounds

What We Learned…

- Unsupervised Learning problems:

    Clustering and Dimensionality Reduction

- K-means

- Hierarchical Clustering

- Gaussian Mixture Models
- EM algorithm

Questions?

Next time…

Dimension reduction!