University of Colorado **Boulder**

Department of Computer Science

CSCI 5622: Machine Learning

Chris Ketelsen

Lecture 22: K-Means Clustering and Gaussian Mixture Models

Learning Objectives

- Learn about general Clustering
- Learn about the K-Means algorithm
- Learn about Gaussian Mixture Models
- Learn about the EM algorithm

Unsupervised Learning

Find **hidden structure** in data

Structure that can't be formally fully observed

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

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

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

Clustering

One important unsupervised method is **clustering**

Goal: Organize data into classes such that

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

Clustering - Documents into Topics

Clustering - Image Segmentation

Clustering - Market Segmentation

Clustering

Goal: Organize data into classes such that

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

Need to define what we mean by **similarity**...

Clustering - Similarity

similarity

[sim-uh-lar-i-tee]

noun

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

Clustering - Similarity

Usually we just know it when we see it

Clustering - Similarity

Usually we just know it when we see it

Clustering - Similarity

Usually we just know it when we see it

But we'll need something more formal if we're going to do math

Clustering - Similarity

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

Examples:

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

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

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

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

What properties make a good similarity measure?

Clustering - Similarity

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

Properties:

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

OK, so say we have a good similarity measure.

How do we find clusters in the data?

K-Means

Simplest clustering method

Iterative in nature

Reasonably fast

Very popular in practice (though with more bells and whistles)

Requires real-valued data

K-Means

General Idea:

pick K initial cluster means

do until convergence ...

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

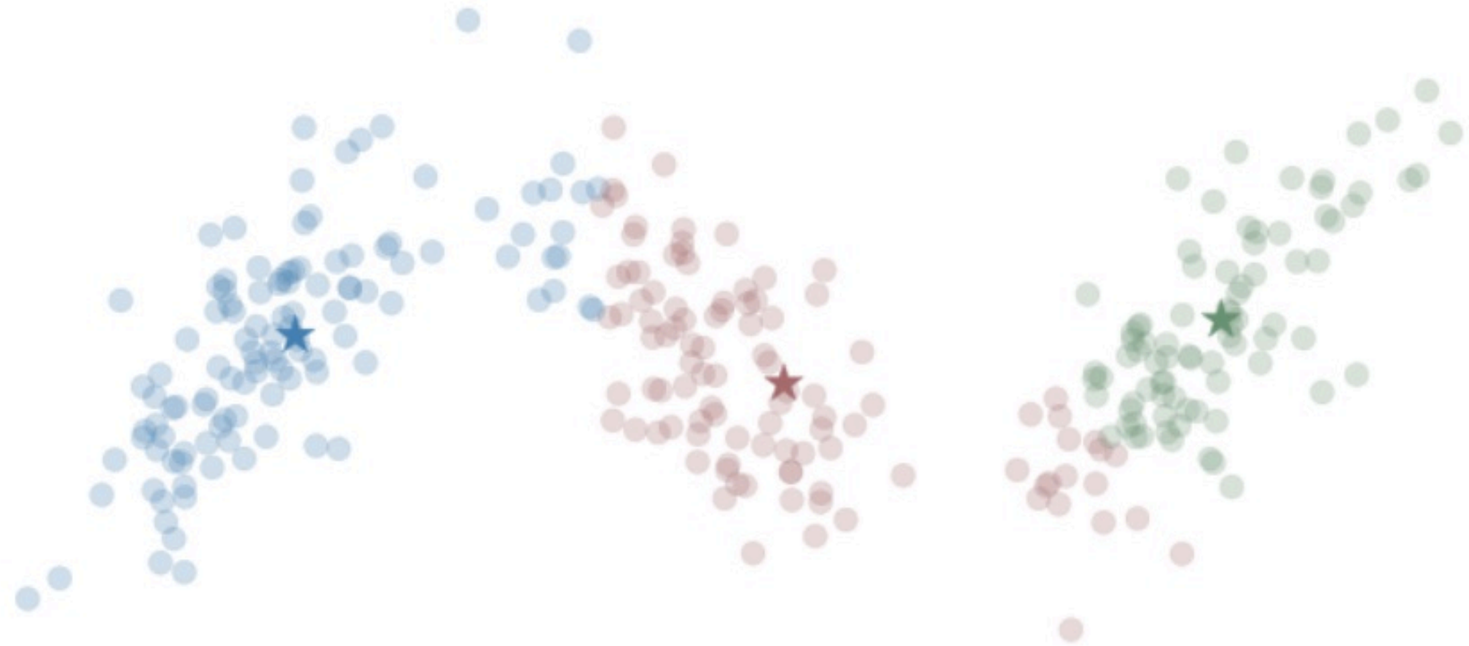
Stop when:

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

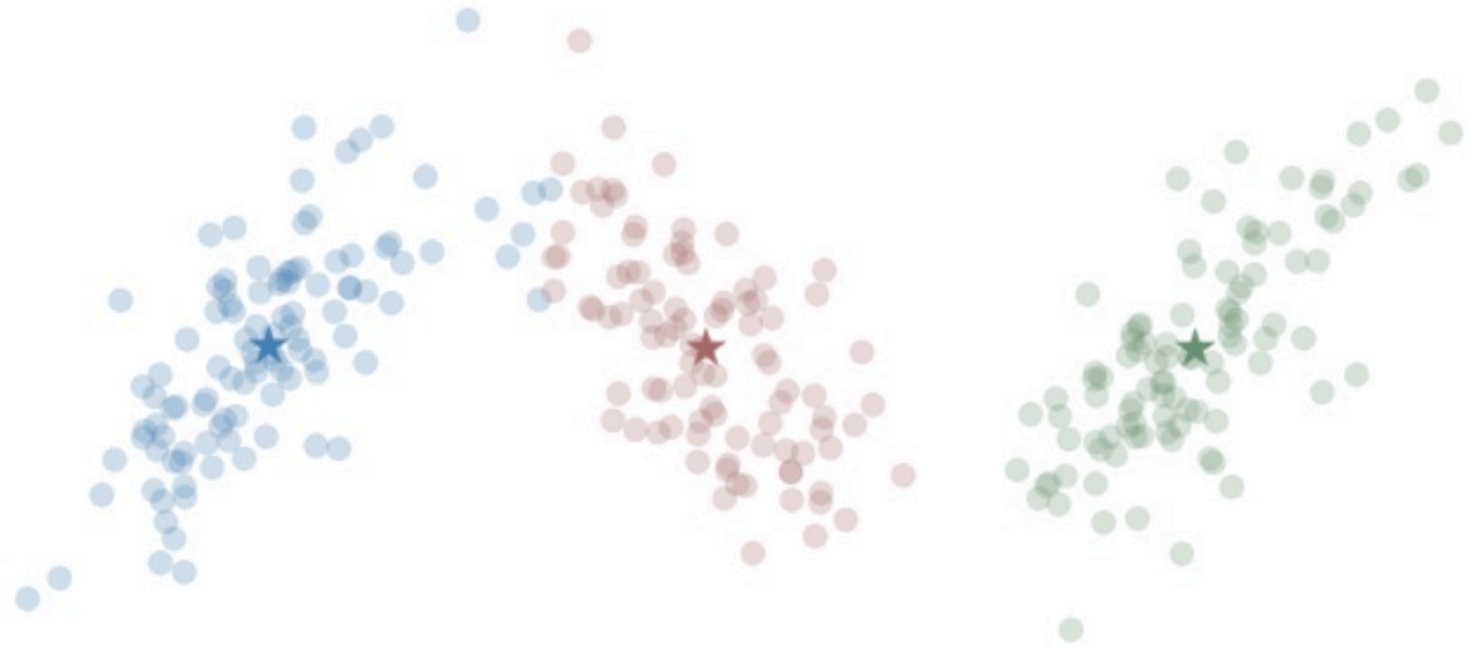
K-Means Example

K-Means Example

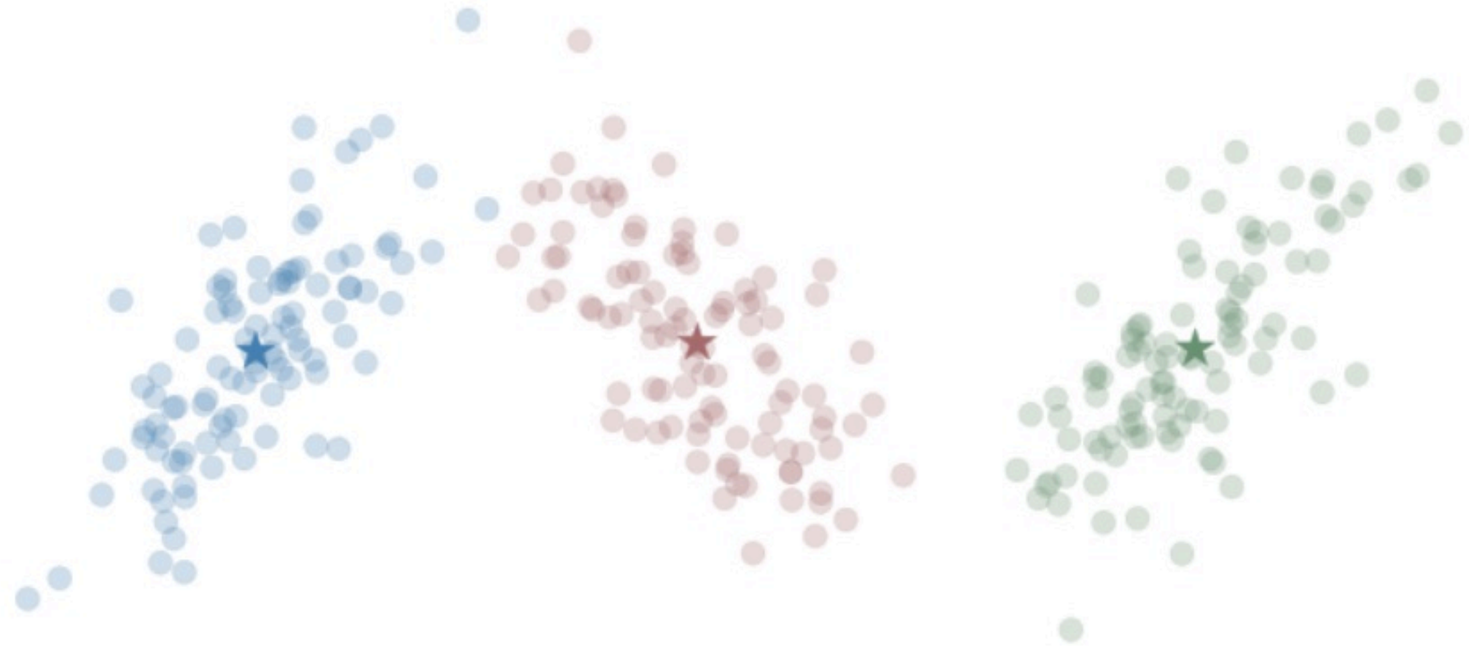
K-Means Example



K-Means Example



K-Means Example



K-Means Example

K-Means Example

K-Means - Algorithm

Strengths:

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

K-Means - Algorithm

In [1]:
```
def KMeans(X, K, max_it=500):

    # Initialize cluster means to K samples from data
    rstart = choice(range(X.shape[0]), size=(K), replace=False)
    mu, muold = X[rstart,:], 1000*np.ones(X[rstart,:].shape)

    its = 0
    while its < max_it and np.linalg.norm(mu-muold) > 1e-4 :

        # compute distance b/w each point and centroid
        dist = np.array([[np.linalg.norm(x-m) for m in mu] for x in X]

        # compute new cluster assignments
        z = np.array([np.argmin(d) for d in dist])

        # move centroids
        muold = mu
        mu = np.array([np.mean(X[z==k, :], axis=0) for k in range(K)])
        its += 1

    return mu, z
```

K-Means

Weaknesses:

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

K-Means - Weaknesses

- Doesn't really work with categorical data
- **Fix**: Do K-Modes instead

K-Means - Weaknesses

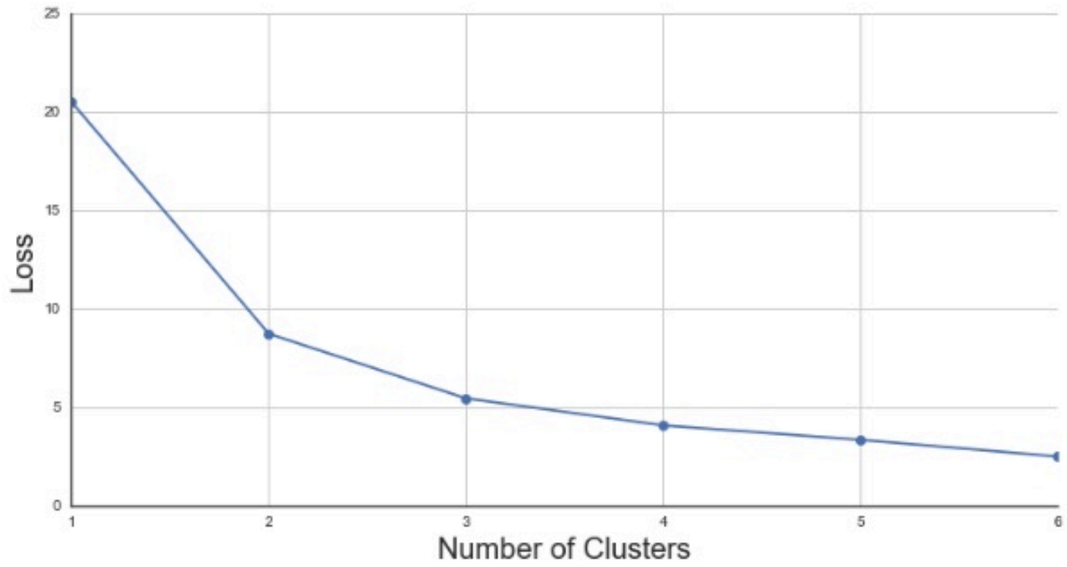
- Usually only converges to local minimum

- **Fix**: Do several runs with random inits. and choose best

K-Means - Weaknesses

- Have to determine number of clusters
- **Fix**: Use the *elbow* method

Run K-Means for different values of K and look at loss function



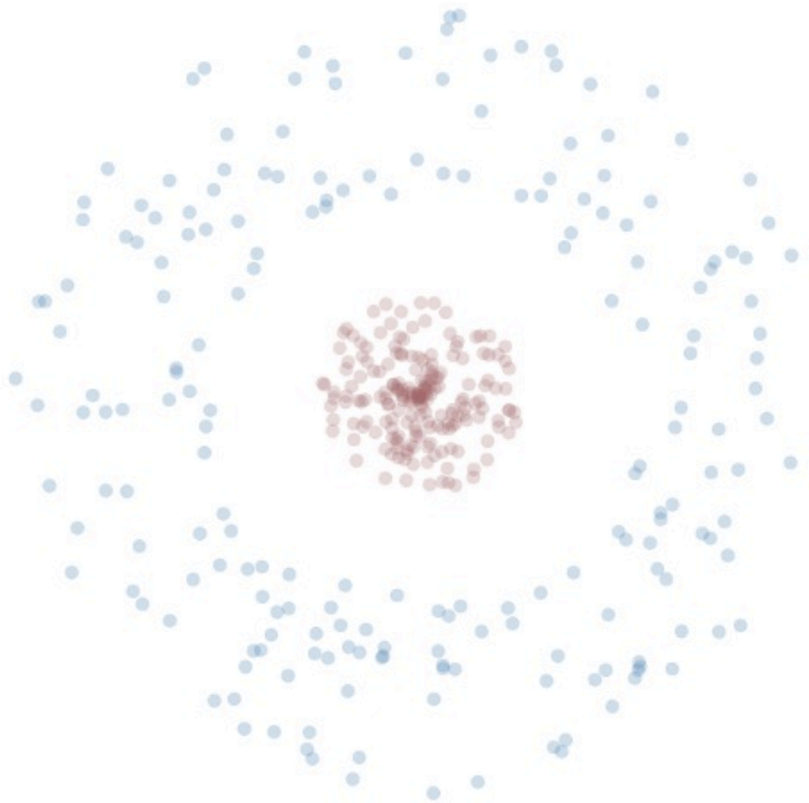
Weaknesses - Outlier Sensitivity

Weaknesses - Outlier Sensitivity

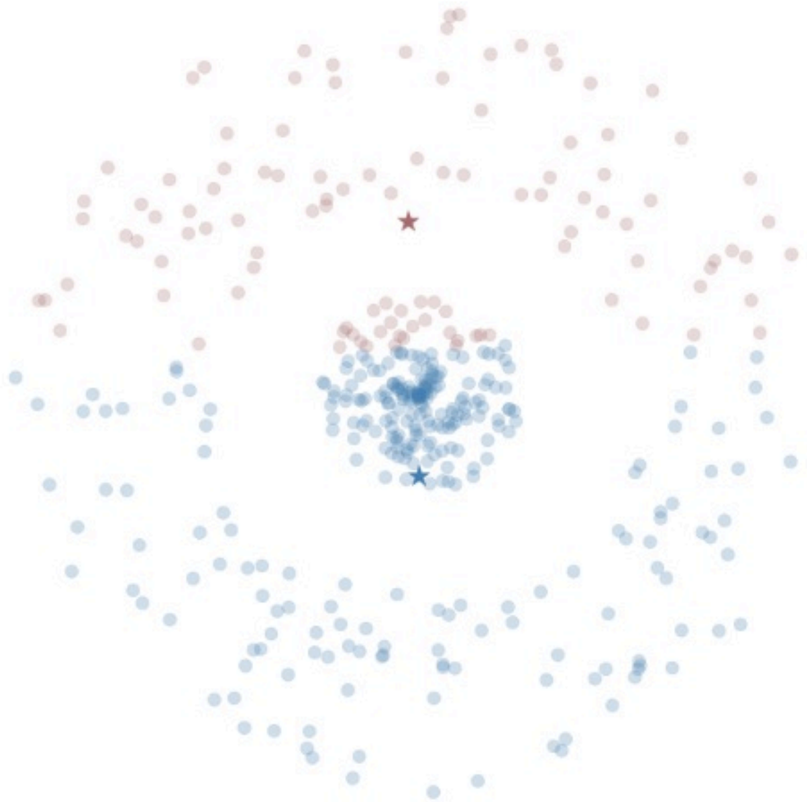
Weaknesses - Outlier Sensitivity

Weaknesses - Outlier Sensitivity

Weaknesses - Convex Clusters



Weaknesses - Convex Clusters



Gaussian Mixture Models

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

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

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

GMM is utilizes *soft* cluster assignments

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

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

Gaussian Mixture Models

The motivation behind GMMs is a generative one

We'll impose on the data a distribution of the form

$$p(\mathbf{x}_i, z_i) = p(\mathbf{x}_i \mid z_i)\, p(z_i)$$

where here z_i is the cluster that $\mathbf{x}_i$ belongs to (though, keep in mind that z_i is a random variable taking on all values in $\{1, \ldots, K\}$

We'll assume:

z_i is multinomial (think rolling a die)

$p(\mathbf{x}_i \mid z_i = k) \sim \mathcal{N}(\mu_k, \Sigma_k)$ (given a k, $\mathbf{x}_i$ is Multivariate Guassian)

Gaussian Mixture Models

$$p(\mathbf{x}_i \mid z_i = k) \sim \mathcal{N}(\mu_k, \Sigma_k)$$

μ_k is a mean vector (just like in K-Means)

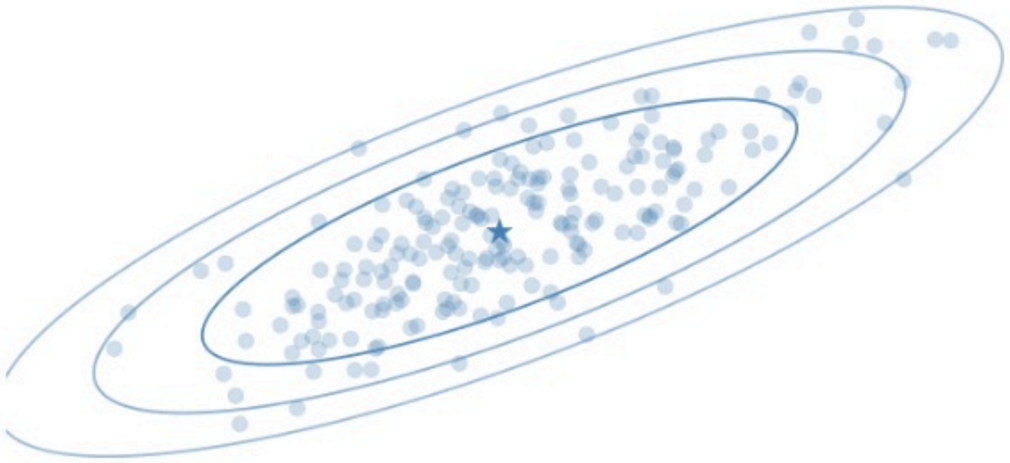
Σ_k is a covariance matrix

Gaussian Mixture Models

$$p(\mathbf{x}_i \mid z_i = k) \sim \mathcal{N}(\mu_k, \Sigma_k)$$

μ_k is a mean vector (just like in K-Means)

Σ_k is a covariance matrix



Gaussian Mixture Models

$$p(\mathbf{x}_i \mid z_i = k) \sim \mathcal{N}(\mu_k, \Sigma_k)$$

μ_k is a mean vector (just like in K-Means)

Σ_k is a covariance matrix

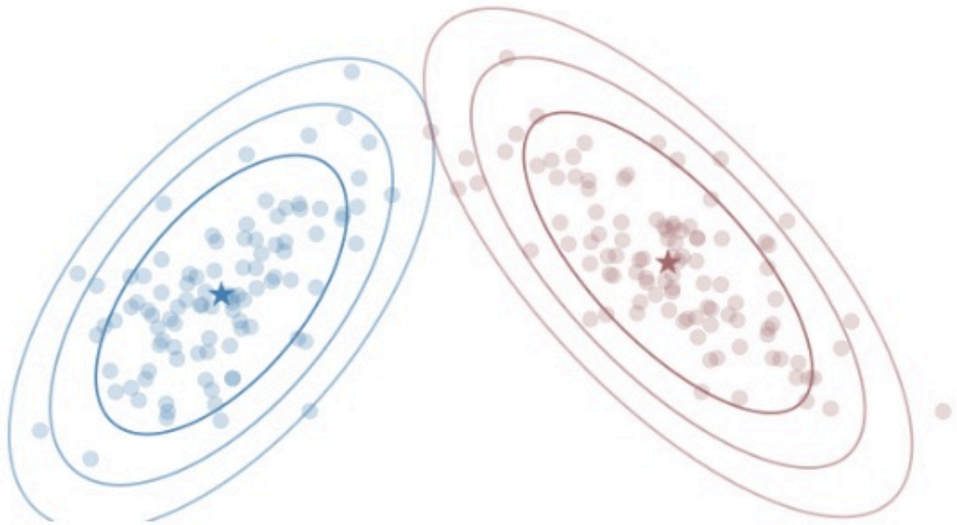
Density function for $\mathbf{x} \in \mathbb{R}^n$ and cluster k is given by

$$p(\mathbf{x} \mid z_i = k) = \frac{1}{(2\pi)^{n/2}|\Sigma_k|^{1/2}} \exp\left\{ -\frac{1}{2}(\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) \right\}$$

Gaussian Mixture Models

Can generate data from model by marginalizing over k

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(\mathbf{x}, z = k) = \sum_{k=1}^{K} p(\mathbf{x} \mid z = k) p(z = k)$$



GMMs and the EM Algorithm

OK, but we're not trying to generate data

We're trying to cluster data

Our problem is, given our data $\{\mathbf{x}_i\}_{i=1}^m$, estimate the parameters in our model so we can say something about the z_i's

GMMs and the EM Algorithm

OK, but we're not trying to generate data

We're trying to cluster data

Our problem is, given our data $\{\mathbf{x}_i\}_{i=1}^m$, estimate the parameters in our model so we can say something about the z_i's

We know we need to estimate μ_k and Σ_k for each k

But we also need to model the Multinomial prior on z

Define $\pi = (\pi_1, \pi_2, \ldots, \pi_K)$ s.t. $\pi_k \geq 0$ and $\sum_{k=1}^K \pi_k = 1$

Estimate π_k, μ_k, Σ_k for all k

GMMs and the EM Algorithm

It'd be nice if we could do this by Maximum Likelihood Estimation

In that vein, let's define the log-likelihood as

$$\begin{aligned}\ell(\pi, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(\mathbf{x}_i \mid \pi, \mu, \Sigma) \\ &= \sum_{i=1}^{m} \log \sum_{k=1}^{K} p(\mathbf{x}_i \mid z_i = k, \pi, \mu, \Sigma) p(z_i = k \mid \pi)\end{aligned}$$

GMMs and the EM Algorithm

It'd be nice if we could do this by Maximum Likelihood Estimation

In that vein, let's define the log-likelihood as

$$\begin{aligned}\ell(\pi, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(\mathbf{x}_i \mid \pi, \mu, \Sigma) \\ &= \sum_{i=1}^{m} \log \sum_{k=1}^{K} p(\mathbf{x}_i \mid z_i = k, \pi, \mu, \Sigma) p(z_i = k \mid \pi)\end{aligned}$$

It'd be great if we could find MLE estimates in the usual way, by taking derivatives wrt parameters, setting to zero, and solving

But unfortunately, because we don't know the z's, this won't work

(go ahead and try!)

GMMs and the EM Algorithm

Suppose for a sec that we **did** know the z's

$$\begin{aligned} \ell(\pi, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(\mathbf{x}_i \mid \pi, \mu, \Sigma) \\ &= \sum_{i=1}^{m} \log p(\mathbf{x}_i \mid z_i, \pi, \mu, \Sigma) + \log p(z_i \mid \pi) \end{aligned}$$

GMMs and the EM Algorithm

Suppose for a sec that we **did** know the z's

$$\begin{aligned} \ell(\pi, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(\mathbf{x}_i \mid \pi, \mu, \Sigma) \\ &= \sum_{i=1}^{m} \log p(\mathbf{x}_i \mid z_i, \pi, \mu, \Sigma) + \log p(z_i \mid \pi) \end{aligned}$$

The MLE estimates for the parameters are then given by (EFY!)

$$\begin{aligned} \pi_k &= \frac{1}{m} \sum_{i=1}^{m} I\{z_i = k\} \\ \mu_k &= \frac{\sum_{i=1}^{m} I\{z_i = k\}\mathbf{x}_i}{\sum_{i=1}^{m} I\{z_i = k\}} \\ \Sigma_k &= \frac{\sum_{i=1}^{m} I\{z_i = k\}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T}{\sum_{i=1}^{m} I\{z_i = k\}} \end{aligned}$$

GMMs and the EM Algorithm

OK, but really we don't know the z's. So what should we do?

Maybe we could iterate?

GMMs and the EM Algorithm

OK, but really we don't know the z's. So what should we do?

Maybe we could iterate?

Estimate the probability that $\mathbf{x}_i$ belongs to each cluster k?

Hold the z's fixed and do the MLE estimate of the parameters?

Sounds kinda familiar...

GMMs and the EM Algorithm

OK, but really we don't know the z's. So what should we do?

Maybe we could iterate?

Estimate the probability that $\mathbf{x}_i$ belongs to each cluster k?

Hold the z's fixed and do the MLE estimate of the parameters?

Sounds kinda familiar...

Sounds a lot like K-Means!

This is the idea behind the EM algorithm

GMMs and the EM Algorithm

Do until convergence...

(E-step) For each i and k, set

$$r_{ik} = p(z_i = k \mid \mathbf{x}_i, \pi, \mu, \Sigma)$$

(M-step) Update the parameters:

$$\begin{aligned}
\pi_k &= \frac{1}{m} \sum_{i=1}^{m} r_{ik} \\
\mu_k &= \frac{\sum_{i=1}^{m} r_{ik} \mathbf{x}_i}{\sum_{i=1}^{m} r_{ik}} \\
\Sigma_k &= \frac{\sum_{i=1}^{m} r_{ik} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T}{\sum_{i=1}^{m} r_{ik}}
\end{aligned}$$

GMMs and the EM Algorithm

Do until convergence...

(E-step) For each i and k, set

$$r_{ik} = \frac{p(\mathbf{x}_i \mid z_i = k, \pi, \mu, \Sigma)\pi_k}{\sum_{k'} p(\mathbf{x}_i \mid z_i = k', \pi, \mu, \Sigma)\pi_{k'}}$$

(M-step) Update the parameters:

$$\pi_k = \frac{1}{m} \sum_{i=1}^{m} r_{ik}$$

$$\mu_k = \frac{\sum_{i=1}^{m} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{m} r_{ik}}$$

$$\Sigma_k = \frac{\sum_{i=1}^{m} r_{ik}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T}{\sum_{i=1}^{m} r_{ik}}$$

GMMs and the EM Algorithm

The EM in EM Algorithm stands for Expectation-Maximization

First estimate the Expectation of the z_i's

Then Maximize the likelihood of the parameters

GMMs and the EM Algorithm

The EM in EM Algorithm stands for Expectation-Maximization

First estimate the Expectation of the z_i's

Then Maximize the likelihood of the parameters

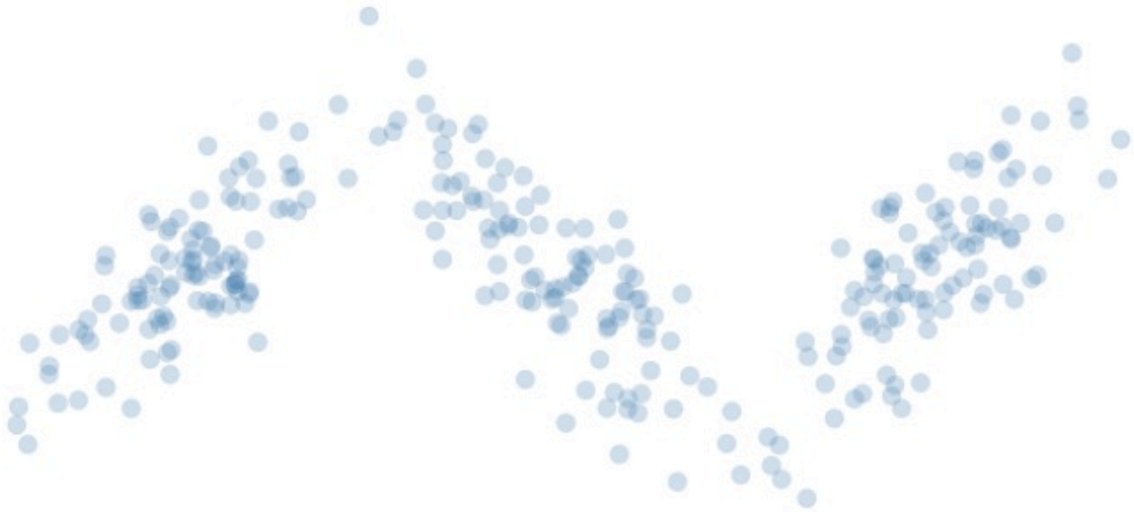
We'll look more closely at a simple example in class to figure out why this works so well

For now, let's look at an example ...

Gaussian Mixture Models

Example: Consider our toy data set again

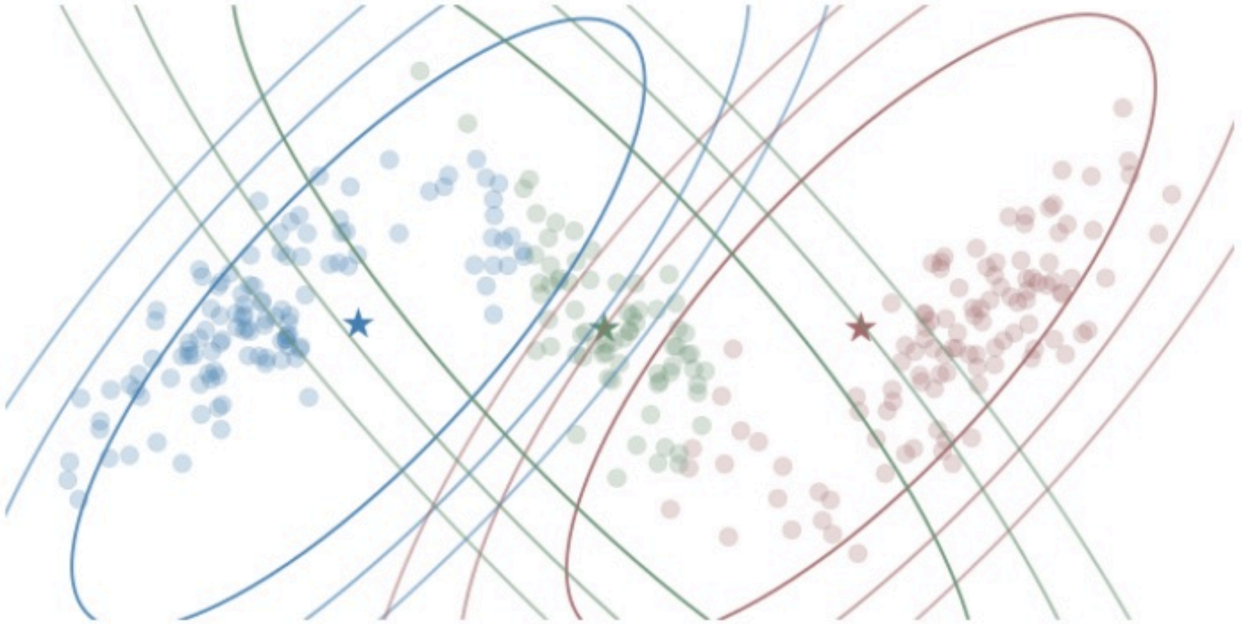
Initial distributions



Gaussian Mixture Models

Example: Consider our toy data set again

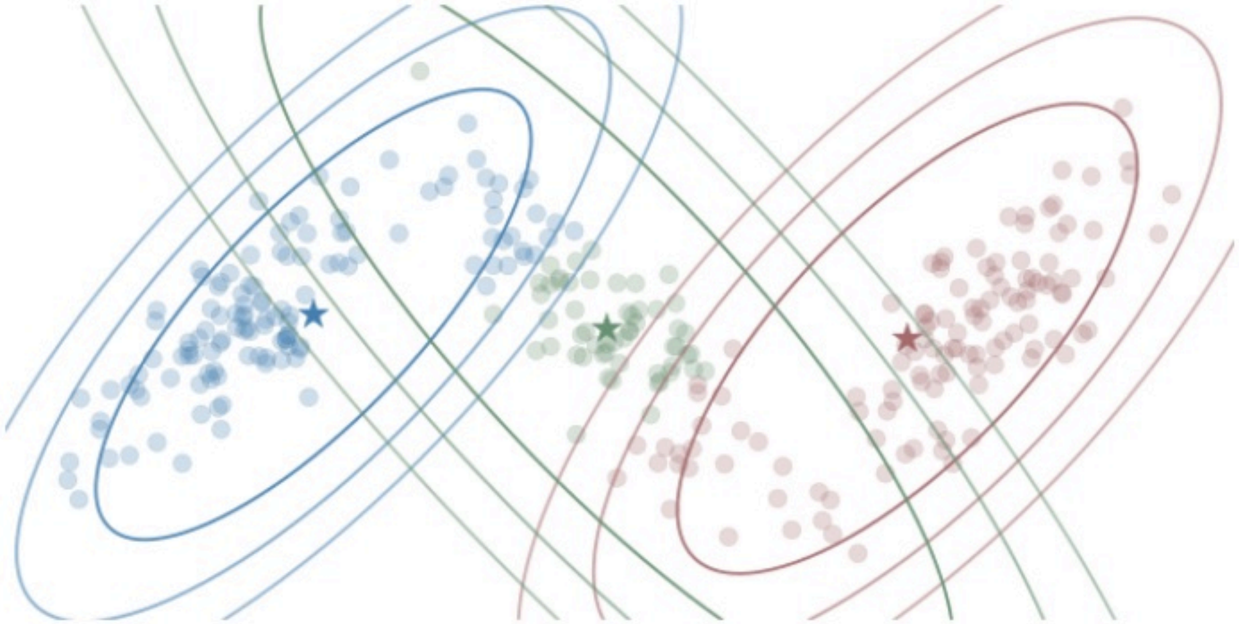
After random initialization of EM algorithm



Gaussian Mixture Models

Example: Consider our toy data set again

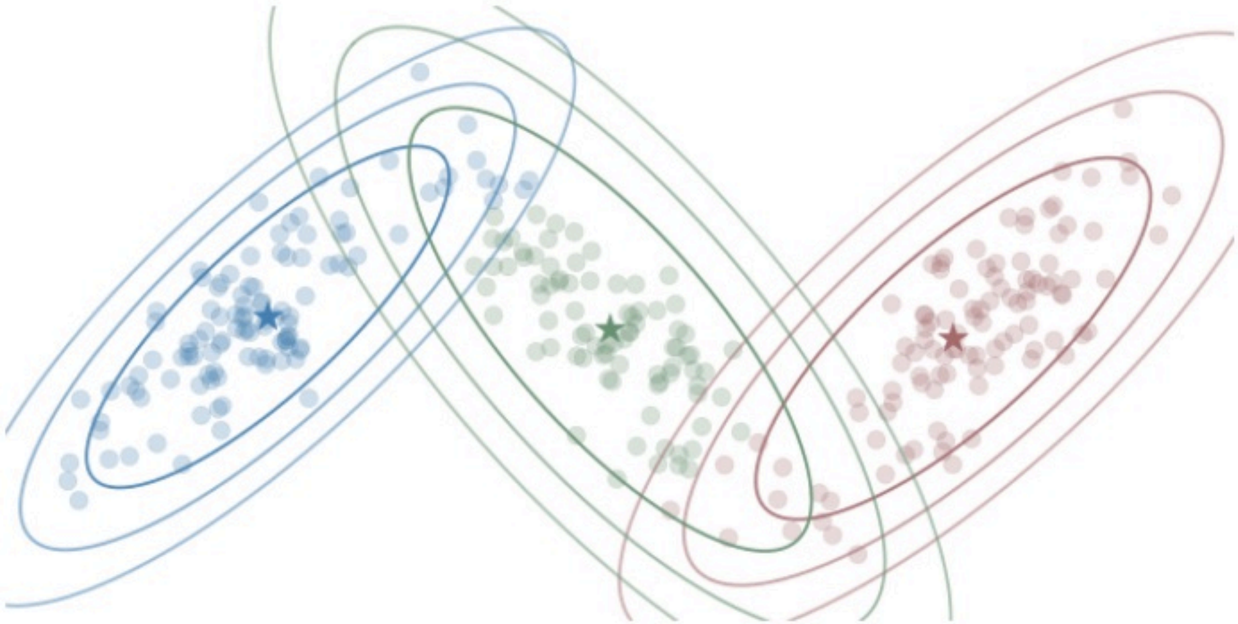
After 1 EM iteration



Gaussian Mixture Models

Example: Consider our toy data set again

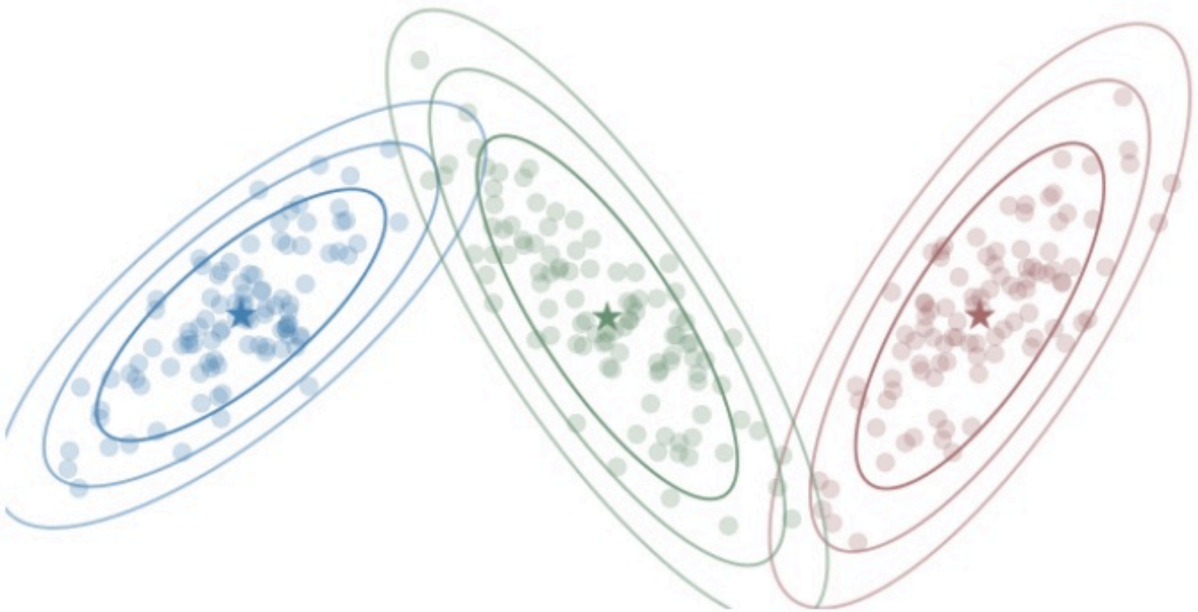
After 3 EM iterations



Gaussian Mixture Models

Example: Consider our toy data set again

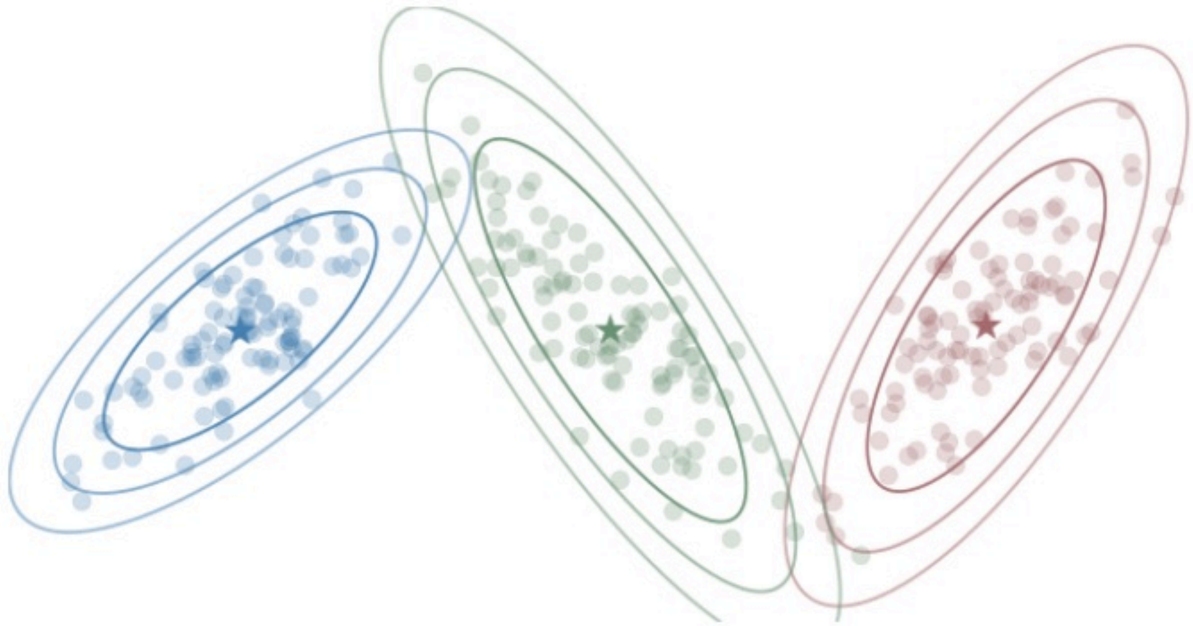
After 6 EM iterations



Gaussian Mixture Models

Example: Consider our toy data set again

After 9 EM iterations



GMMs and the EM Algorithm

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

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

But, it also has certain advantages

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

K-Means and the EM Algorithm

But wait, didn't we say that GMMs and K-Means were related?

It turns out that GMM with EM gives you exactly K-Means if you make the assumption that the covariance matrices are diagonal and the variances are known

$$\Sigma_k = \begin{bmatrix} \sigma^2 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma^2 \end{bmatrix} \quad \text{for each } k = 1, \ldots, K$$

We'll work this out in class, because it's kinda neat

Acknowledgements

Many of the slides in this lecture were adopted from Jordan Boyd-Graber

In Class

In Class

In Class

In Class