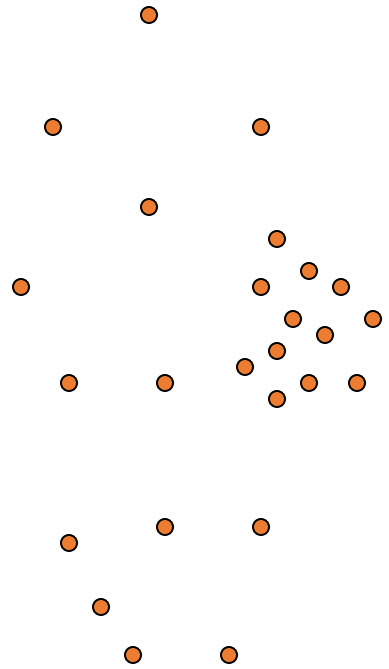
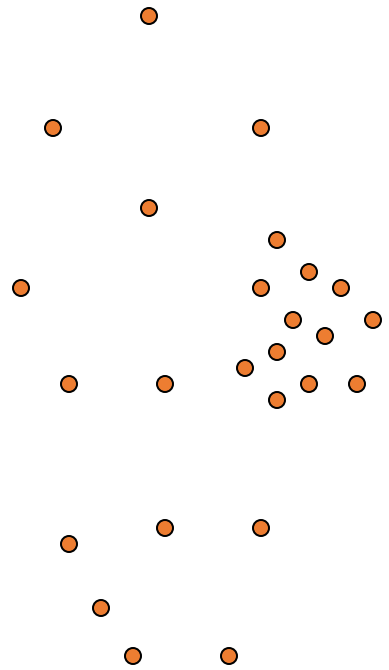
10701
Gaussian Mixture Models and Top Down Clustering

(One) bad case for K-means



- Clusters may overlap
- Some clusters may be “wider” than others
- Clusters may not be linearly separable

(One) bad case for K-means



- Clusters may overlap
- Some clusters may be “wider” than others
- Clusters may not be linearly separable

Partitioning Algorithms

- K-means
  - **hard assignment**: each object belongs to only one cluster
- Mixture modeling
  - **soft assignment**: probability that an object belongs to a cluster

Generative approach: think of each cluster as a component distribution, and any data point is drawn from a “mixture” of multiple component distributions

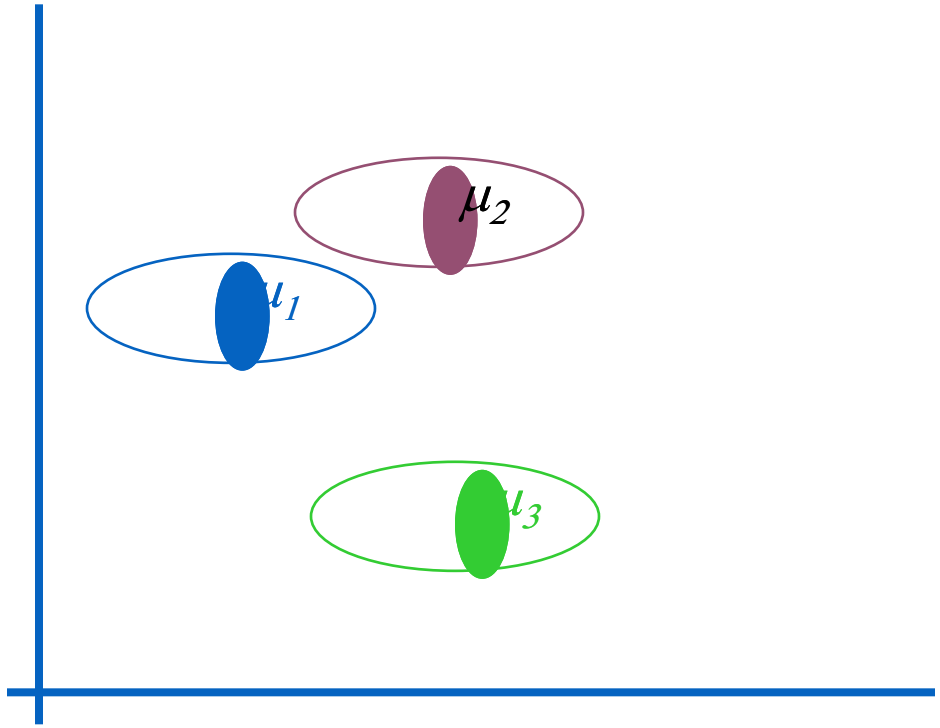
Gaussian Mixture Model

Mixture of K Gaussian distributions: (Multi-modal distribution)

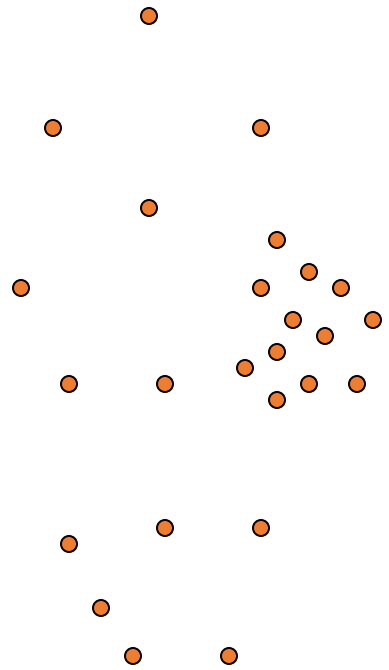
$$p(x|y=i) \sim \mathrm{N}(\mu_i, \sigma^2 \boldsymbol{I})$$

$$p(x) = \sum_i p(x|y=i)\, P(y=i)$$

Mixture component **Mixture proportion**



(One) bad case for K-means



- Clusters may overlap
- Some clusters may be “wider” than others
- Clusters may not be linearly separable

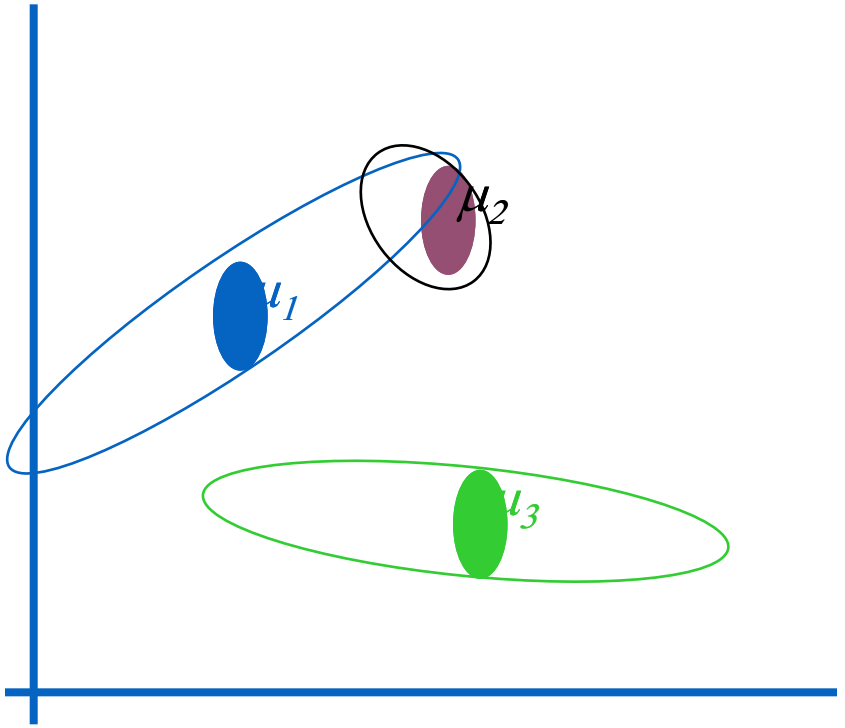
General GMM

GMM – Gaussian Mixture Model (Multi-modal distribution)

$$p(x|y=i) \sim \mathrm{N}(\mu_i, \Sigma_i)$$

$$p(x) = \sum_i p(x|y=i)\, P(y=i)$$

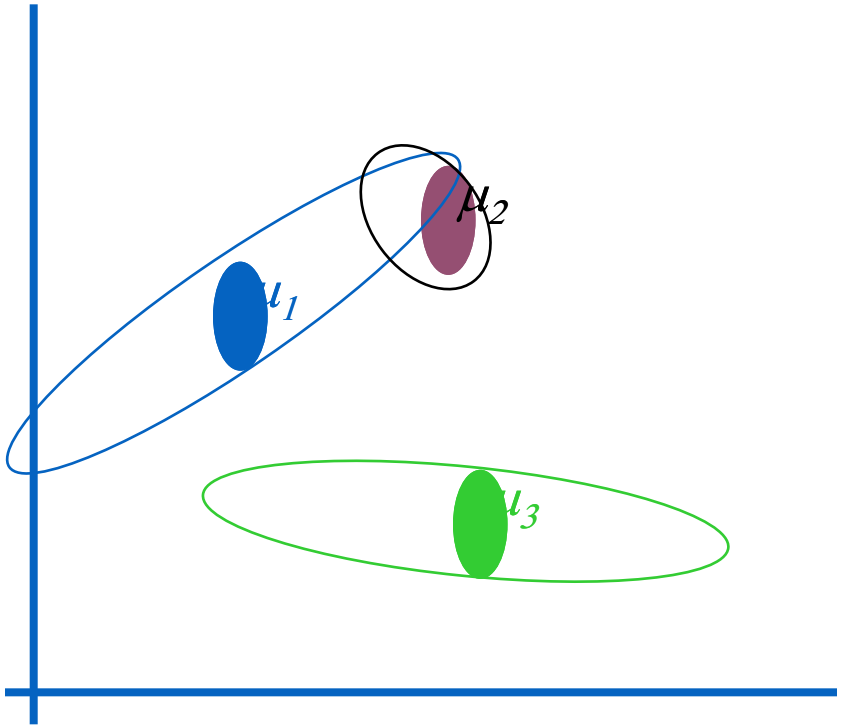
Mixture component — **Mixture proportion**



General GMM

GMM – Gaussian Mixture Model (Multi-modal distribution)

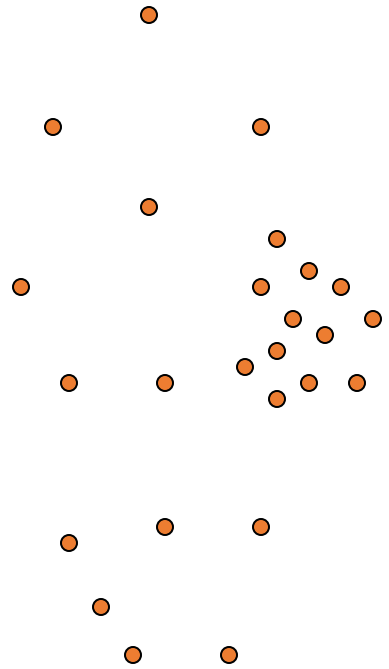
- There are k components
- Component *i* has an associated mean vector μ_i
- Each component generates data from a Gaussian with mean μ_i and covariance matrix Σ_i



Each data point is generated according to the following recipe:

1) Pick a component at random: Choose component i with probability *P(y=i)*
2) Data-point x ~ $\mathrm{N}(\mu_i, \Sigma_i)$

(One) bad case for K-means



- Clusters may overlap
- Some clusters may be “wider” than others
- Clusters may not be linearly separable

General GMM

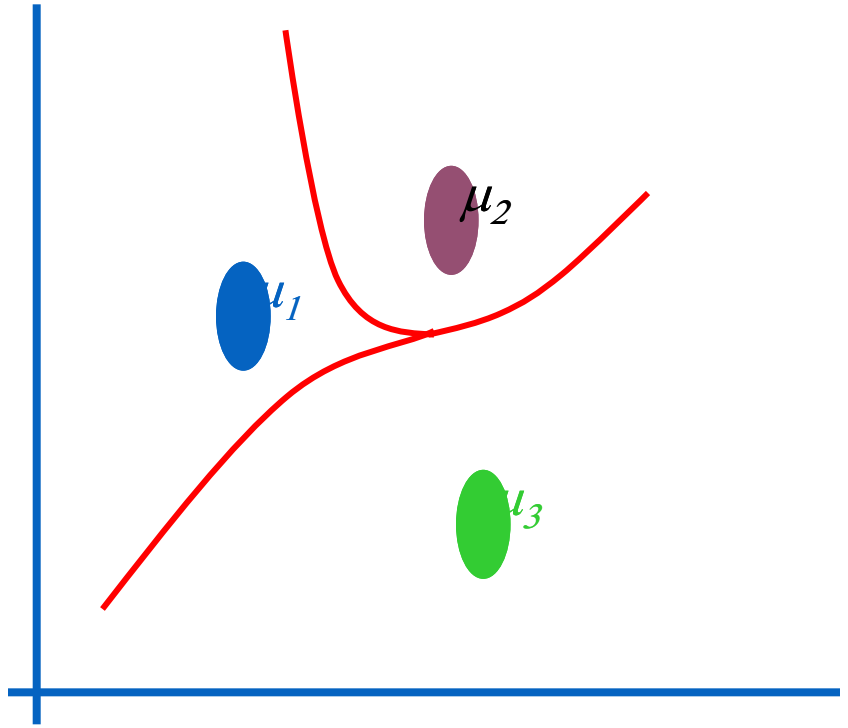
GMM – Gaussian Mixture Model (Multi-modal distribution)

$$p(x|y=i) \sim \mathrm{N}(\mu_i, \Sigma_i)$$

Gaussian Bayes Classifier:

$$\log \frac{\mathrm{P}(\mathrm{y}=\mathrm{i}\,|\,\mathrm{x})}{\mathrm{P}(\mathrm{y}=\mathrm{j}\,|\,\mathrm{x})}$$

$$= \log \frac{\mathrm{p}(\mathrm{x}\,|\,\mathrm{y}=\mathrm{i})\mathrm{P}(\mathrm{y}=\mathrm{i})}{\mathrm{p}(\mathrm{x}\,|\,\mathrm{y}=\mathrm{j})\mathrm{P}(\mathrm{y}=\mathrm{j})}$$

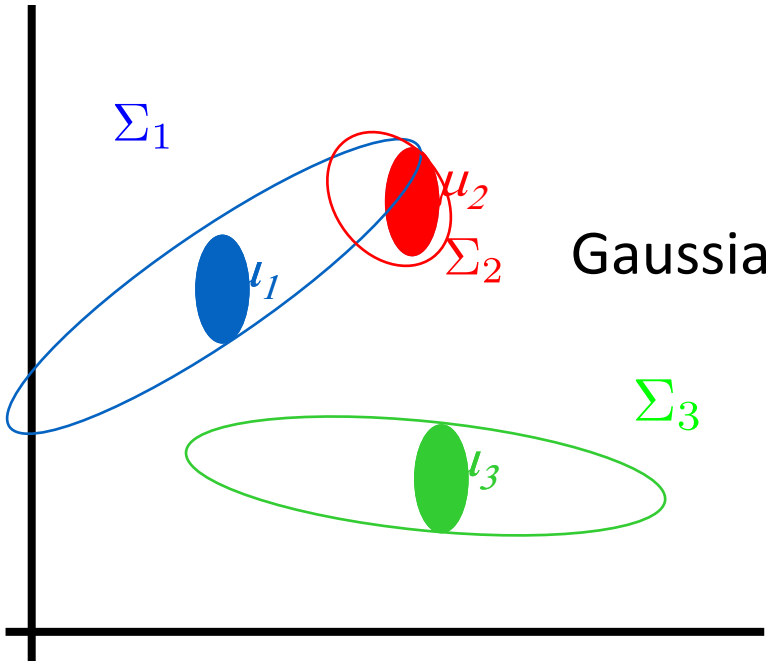


"Quadratic Decision boundary" – Variance / Co-Variance add quadratic decision boundaries

Learning General GMM

$$x_1, \ldots, x_m \sim p(x) = \sum_{i=1}^{k} p(x|Y=i)P(Y=i)$$

Mixture component ↓ $p(x|Y=i)$ ↓ **Mixture proportion, p_i**



Gaussian mixture model

$$p(x|Y=i) \sim \mathcal{N}(\mu_i, \Sigma_i)$$

Parameters: $\{p_i, \mu_i, \Sigma_i\}_{i=1}^{K}$

- How to estimate parameters? Maximum Likelihood
  But don't know labels Y (recall Gaussian Bayes classifier)

Learning General GMM

Maximize marginal likelihood:

$$\text{argmax} \prod_j P(x_j) = \text{argmax} \prod_j \sum_{i=1}^{K} P(y_j=i, x_j) \qquad \text{... marginalizing } y_j$$

$$= \text{argmax} \prod_j \sum_{i=1}^{K} P(y_j=i) p(x_j \mid y_j=i)$$

$P(y_j=i) = P(y=i)$ Mixture component i is chosen with prob P(y = i)

$$= \arg\max \prod_{j=1}^{m} \sum_{i=1}^{k} P(y=i) \frac{1}{\sqrt{\det(\Sigma_i)}} \exp\left[-\frac{1}{2} (x_j - \mu_i)^T \Sigma_i (x_j - \mu_i) \right]$$

How do we find the $\boldsymbol{\mu}_i$'s and P(y=i)s which give max. marginal likelihood?

* Set $\frac{\partial}{\partial \mu_i}$ log Prob (....) = 0 and solve for μ_i's. Non-linear non-analytically solvable

* Use gradient descent: Doable, but often slow

GMM vs. k-means

Maximize marginal likelihood:

$$\text{argmax} \prod_j P(x_j) = \text{argmax} \prod_j \sum_{i=1}^{K} P(y_j=i, x_j)$$
$$= \text{argmax} \prod_j \sum_{i=1}^{K} P(y_j=i) p(x_j | y_j=i)$$

- What happens if we assume **Hard assignment**?

$$P(y_j = i) = 1 \text{ if } i = C(j)$$
$$= 0 \text{ otherwise}$$

Same as k-means (if we assume equal covariance matrix)!

$$\text{argmax} \prod_j P(x_j) = \text{argmax} \prod_j p(x_j | y_j=C(j))$$
$$= \text{argmax} \prod_{j=1}^{n} \exp\left(\frac{-1}{2\sigma^2} \|x_j - \mu_{C(j)}\|^2\right)$$
$$= \text{argmin} \sum_{j=1}^{n} \|x_j - \mu_{C(j)}\|^2) = \arg\min_{\mu, C} F(\mu, C)$$

Expectation-Maximization (EM)

A general algorithm to deal with hidden data, but we will study it in the context of unsupervised learning (hidden labels) first

- No need to choose step size as in Gradient methods.
- EM is an Iterative algorithm with two linked steps:

  E-step: fill-in hidden data (Y) using inference

  M-step: apply standard MLE/MAP method to estimate parameters

  $\{p_i, \mu_i, \Sigma_i\}_{i=1}^{k}$

- We will see that this procedure monotonically improves the likelihood (or leaves it unchanged). Thus it always converges to a local optimum of the likelihood.

EM for spherical, same variance GMMs

E-step

Compute "expected" classes of all datapoints for each class

$$P(y = i | x_j, \mu_1 \ldots \mu_k) \propto \exp\left(-\frac{1}{2\sigma^2} \| x_j - \mu_i \|^2 \right) P(y = i)$$

In K-means "E-step" we do hard assignment

EM does soft assignment

M-step

Compute MLE for p_i, μ and Σ given our data's class membership distributions (weights)

Similar to K-means, but with weighted data

Iterate.

EM for general GMMs

Iterate. On iteration t let our estimates be

$\lambda_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \dots \mu_k^{(t)}, \Sigma_1^{(t)}, \Sigma_2^{(t)} \dots \Sigma_k^{(t)}, p_1^{(t)}, p_2^{(t)} \dots p_k^{(t)} \}$

$p_i^{(t)}$ is shorthand for estimate of $P(y=i)$ on t'th iteration

E-step

Compute "expected" classes of all datapoints for each class

$$P(y = i | x_j, \lambda_t) \propto p_i^{(t)} p(x_j | \mu_i^{(t)}, \Sigma_i^{(t)})$$

Just evaluate a Gaussian at x_j

M-step

Compute MLEs given our data's class membership distributions (weights)

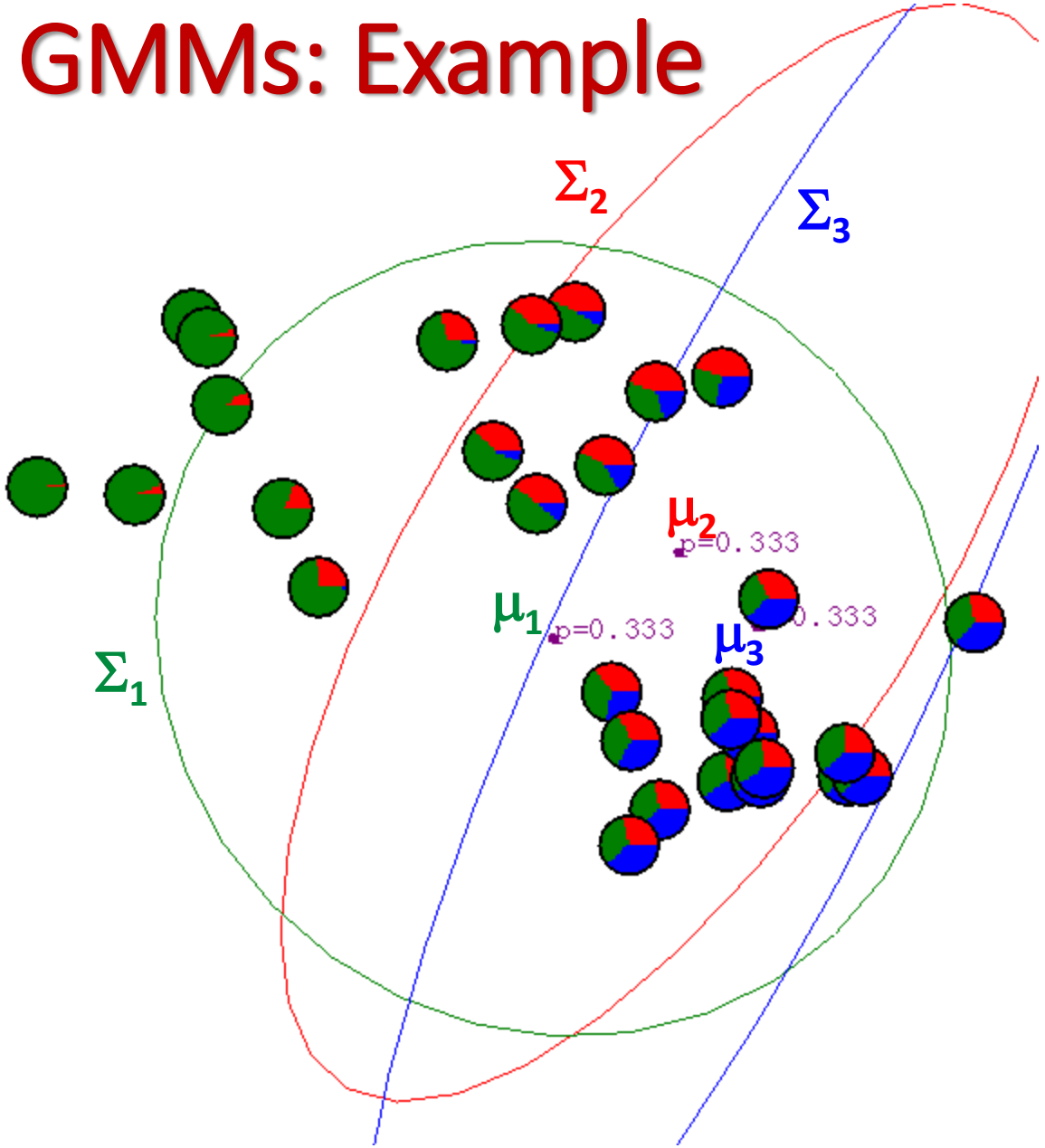
$$\mu_i^{(t+1)} = \frac{\sum_j P(y = i | x_j, \lambda_t) x_j}{\sum_j P(y = i | x_j, \lambda_t)}$$

$$\Sigma_i^{(t+1)} = \frac{\sum_j P(y = i | x_j, \lambda_t)(x_j - \mu_i^{(t+1)})(x_j - \mu_i^{(t+1)})^T}{\sum_j P(y = i | x_j, \lambda_t)}$$

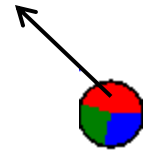
$$p_i^{(t+1)} = \frac{\sum_j P(y = i | x_j, \lambda_t)}{m}$$

m = #data points

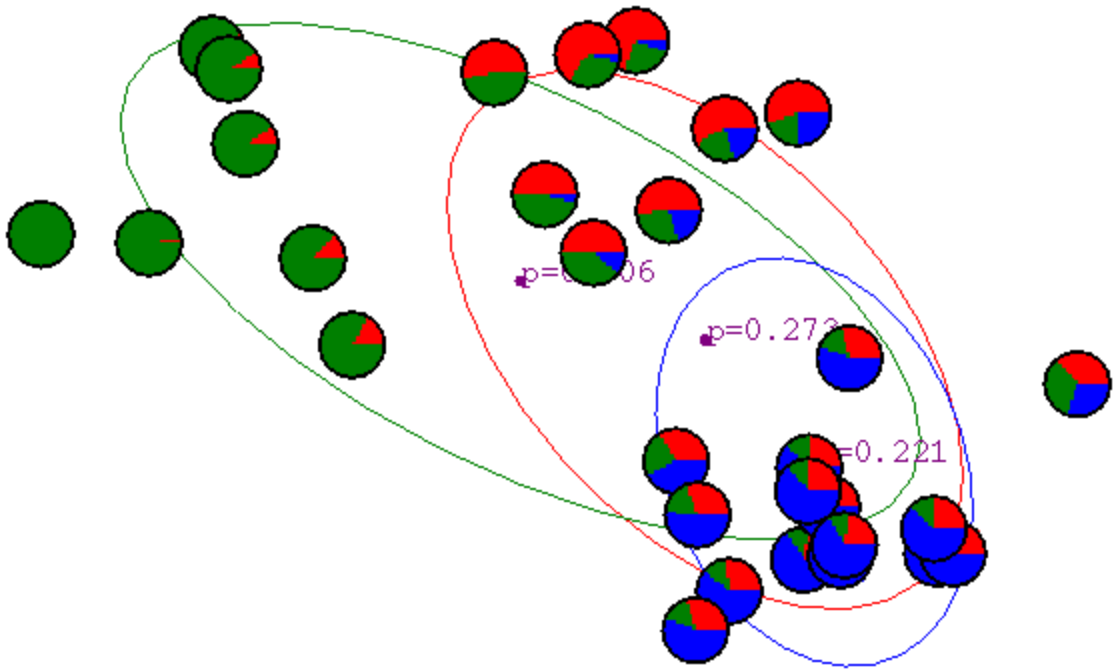
EM for general GMMs: Example



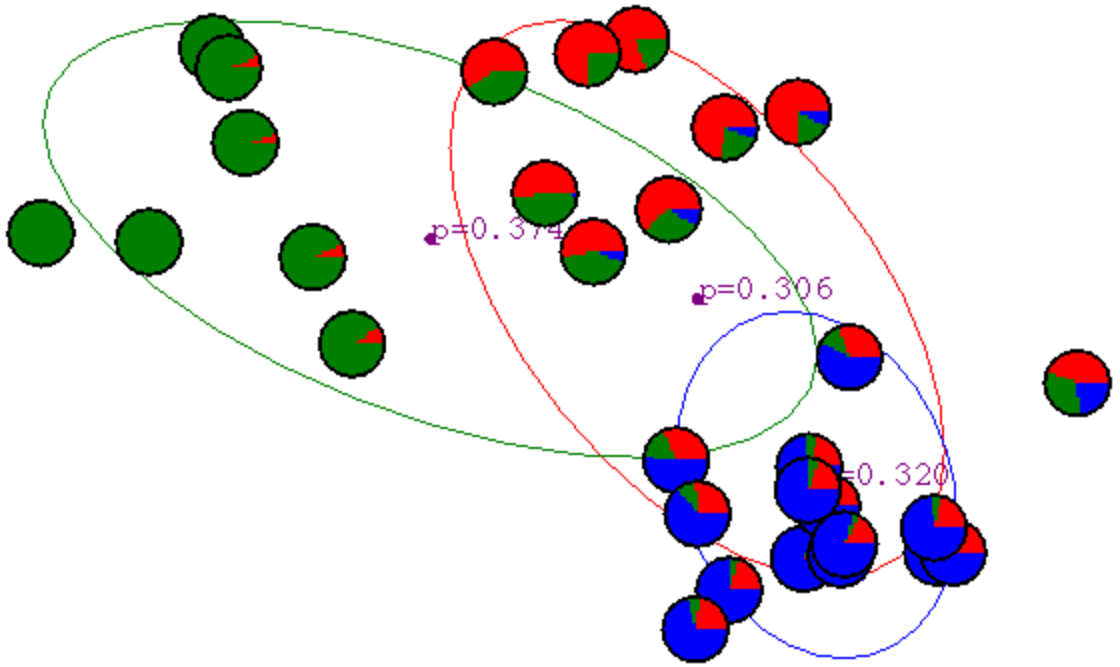
$P(y = \bullet | x_j, \mu_1, \mu_2, \mu_3, \Sigma_1, \Sigma_2, \Sigma_3, p_1, p_2, p_3)$



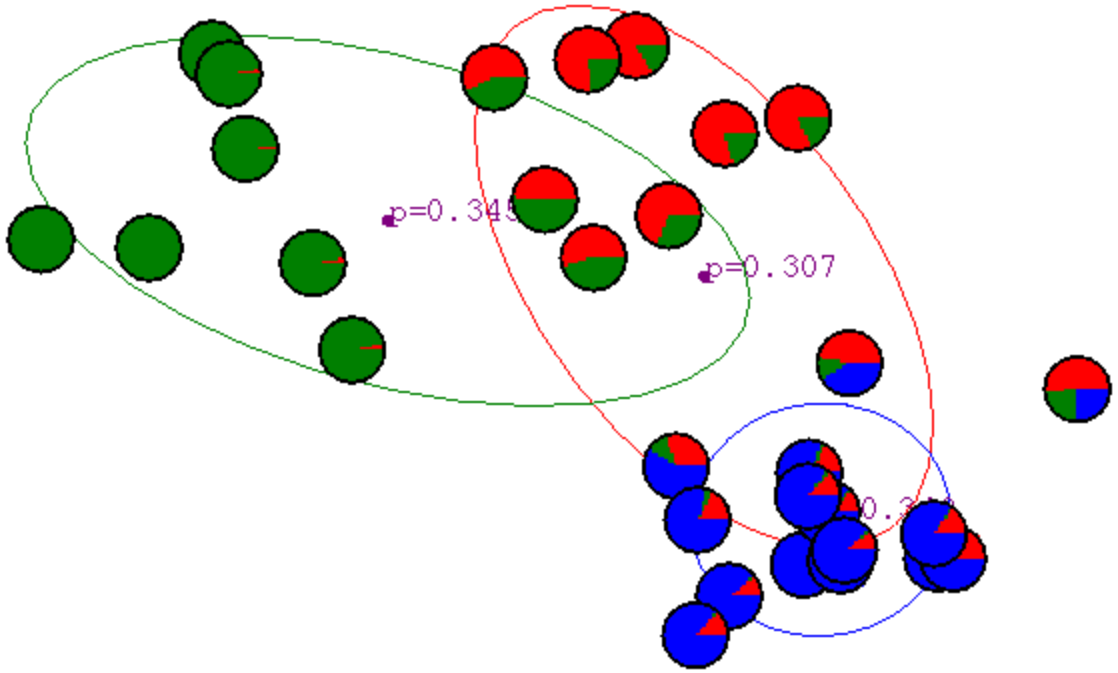
After 1st iteration



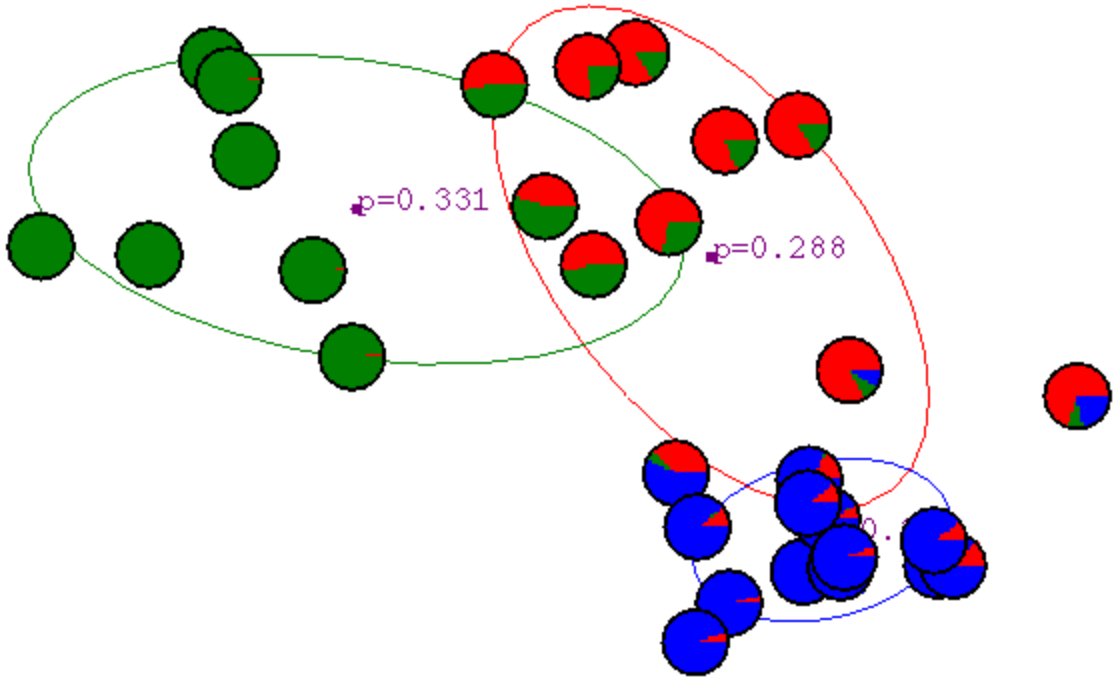
After 2nd iteration



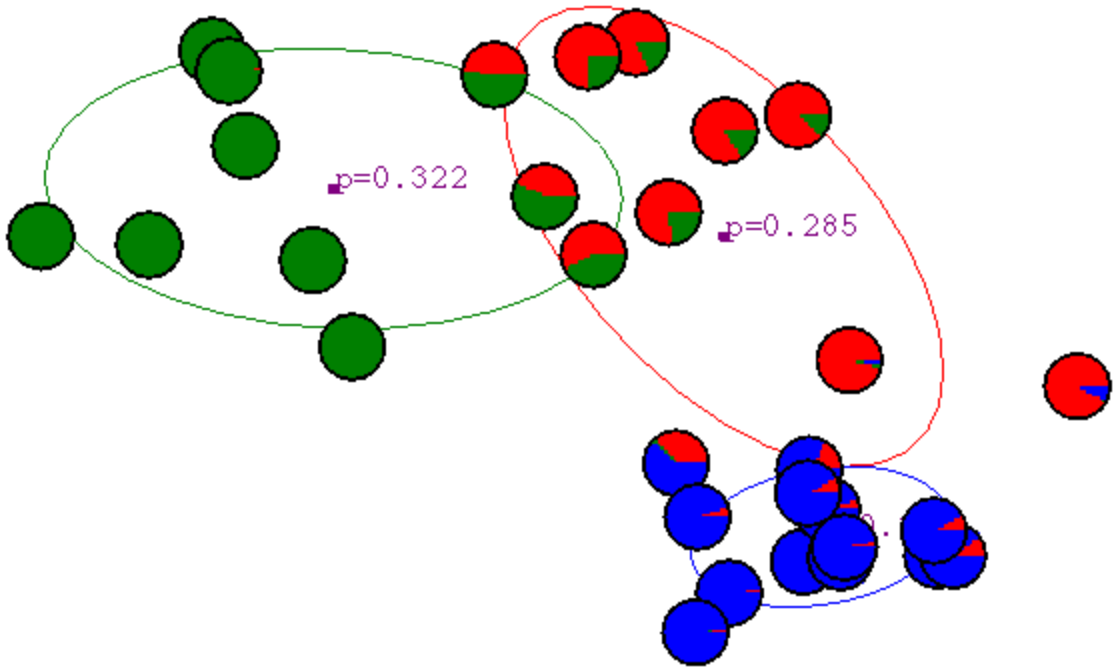
After 3rd iteration



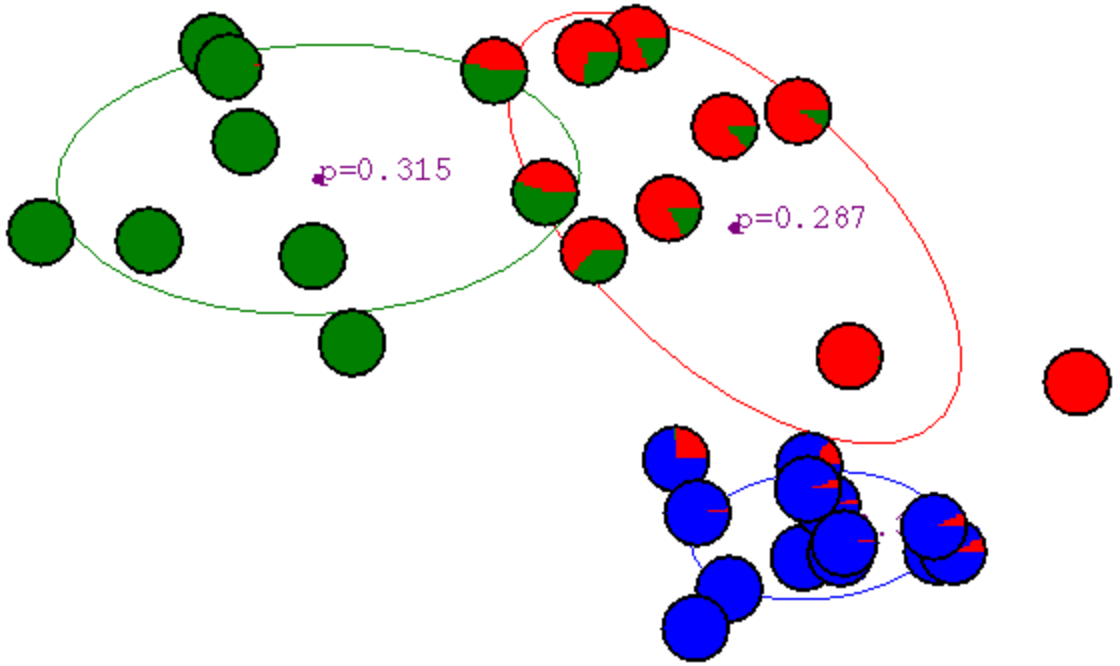
After 4th iteration



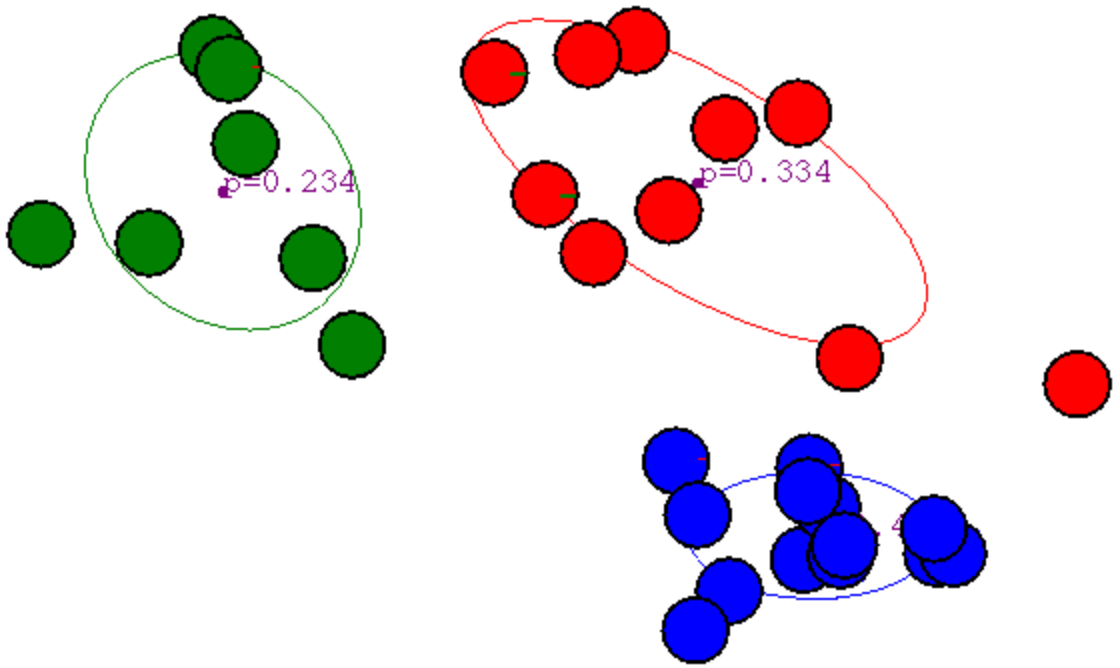
After 5th iteration



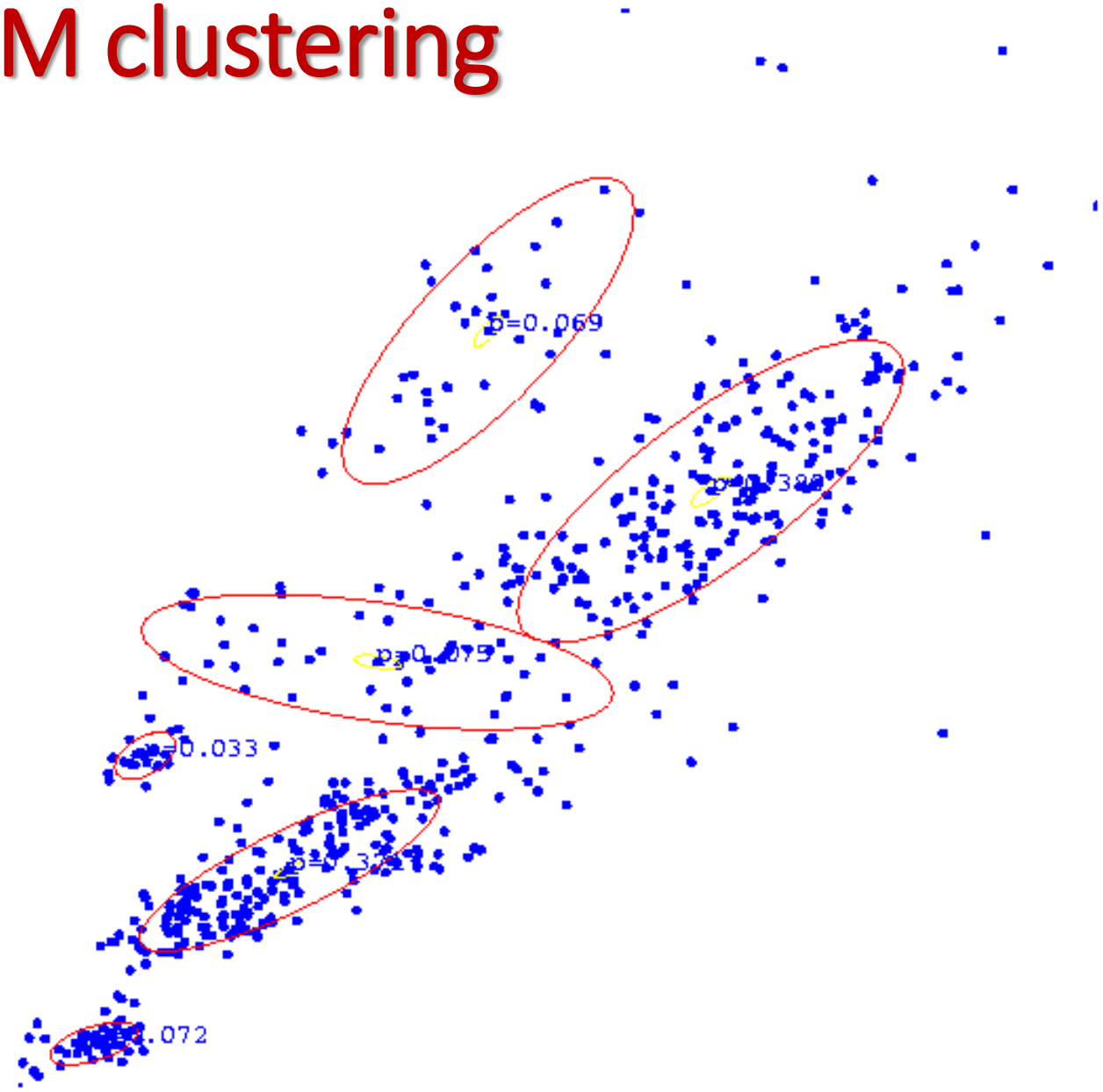
After 6th iteration



After 20th iteration



Example: GMM clustering



General EM algorithm

Marginal likelihood – **x** is observed, **z** is missing:

$$\log \mathrm{P}(\mathrm{D};\theta) = \log \prod_{j=1}^{m} P(\mathbf{x}_j \mid \theta)$$
$$= \sum_{j=1}^{m} \log P(\mathbf{x}_j \mid \theta)$$
$$= \sum_{j=1}^{m} \log \sum_{\mathbf{z}} P(\mathbf{x}_j, \mathbf{z} \mid \theta)$$

$$\mathrm{D} = \{\mathrm{x}_j\}_{j=1}^{m}$$

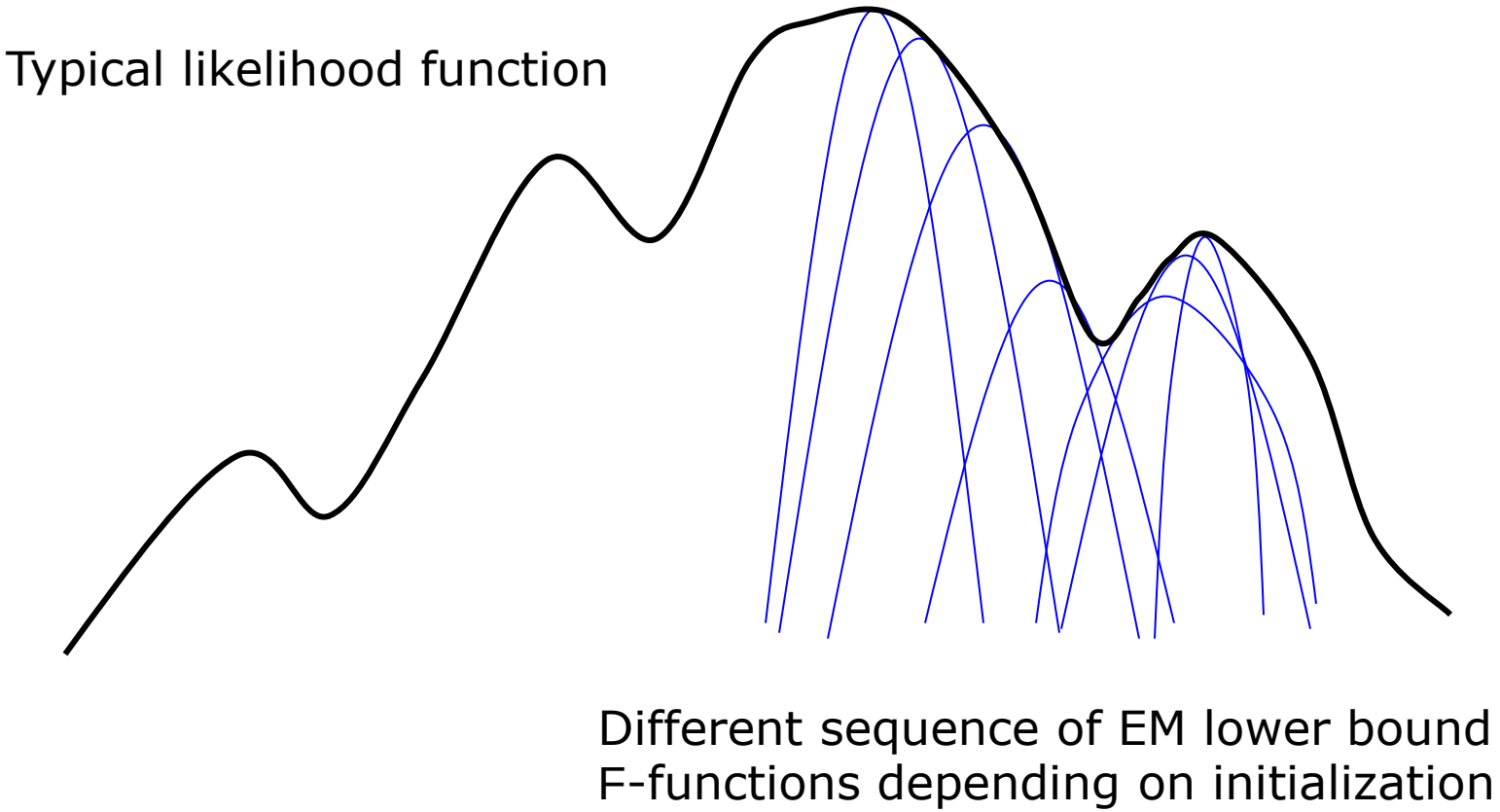
θ - model parameter(s)

Convergence of EM

Sequence of EM lower bound F-functions

EM monotonically converges to a local maximum of likelihood !

EM & Local Maxima



Use multiple, randomized initializations in practice

Summary: EM Algorithm

- A way of maximizing likelihood function for hidden variable models. Finds MLE of parameters when the original (hard) problem can be broken up into two (easy) pieces:
  1. Estimate some "missing" or "unobserved" data from observed data and current parameters.
  2. Using this "complete" data, find the maximum likelihood parameter estimates.

- Alternate between filling in the latent variables using the best guess (posterior) and updating the parameters based on this guess:

  1. E-step: $Q^{t+1} = \arg\max_{Q} F(\theta^t, Q)$
  2. M-step: $\theta^{t+1} = \arg\max_{\theta} F(\theta, Q^{t+1})$

- In the M-step we optimize a lower bound on the likelihood. In the E-step we close the gap, making bound=likelihood.

- EM performs coordinate ascent on F, but can get stuck in local minima.

- Extremely popular and useful in practice.

Strength of Gaussian Mixture Models

- *Interpretability*: learns a generative model of each cluster
  - you can generate new data based on the learned model
- *Relatively efficient*: $O(tkn)$, where n is # objects, k is # clusters, and t is # iterations. Normally, $k, t << n$.
- Intuitive (?) objective function: optimizes data likelihood

Weakness of Gaussian Mixture Models

- Often terminates at a *local optimum*. Initialization is important.
- Need to specify *K,* the *number* of clusters, in advance
- Not suitable to discover clusters with *non-convex shapes*

- Summary
  - To learn Gaussian mixture, assign probabilistic membership based on current parameters, and re-estimate parameters based on current membership

Algorithm: K-means and GMM

1. Decide on a value for *K*, the number of clusters.

2. Initialize the *K* cluster centers / parameters (randomly).

K-means

3. Decide the class memberships of the *N* objects by assigning them to the nearest cluster center.

4. Re-estimate the *K* cluster centers, by assuming the memberships found above are correct.

GMM

3. E-step: assign *probabilistic* membership

4. M-step: re-estimate parameters based on *probabilistic* membership

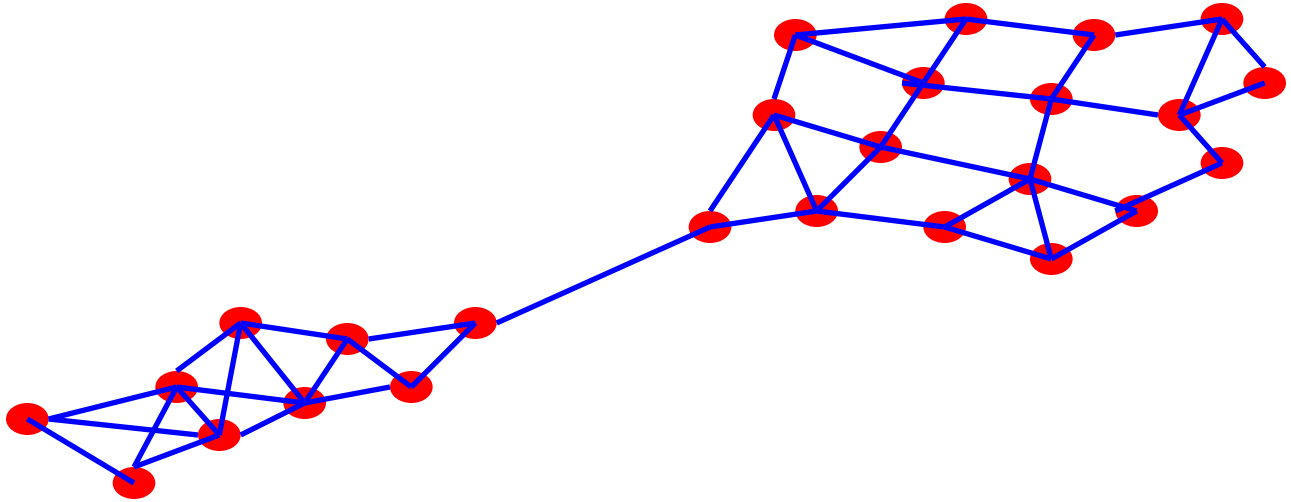
5. Repeat 3 and 4 until parameters do not change.

Clustering methods: Comparison

	Hierarchical	**K-means**	**GMM**
Running time	naively, $O(N^3)$	fastest (each iteration is linear)	fast (each iteration is linear)
Assumptions	requires a similarity / distance measure	strong assumptions	strongest assumptions
Input parameters	none	*K* (number of clusters)	*K* (number of clusters)
Clusters	subjective (only a tree is returned)	exactly *K* clusters	exactly *K* clusters

Top down: Graph based clustering

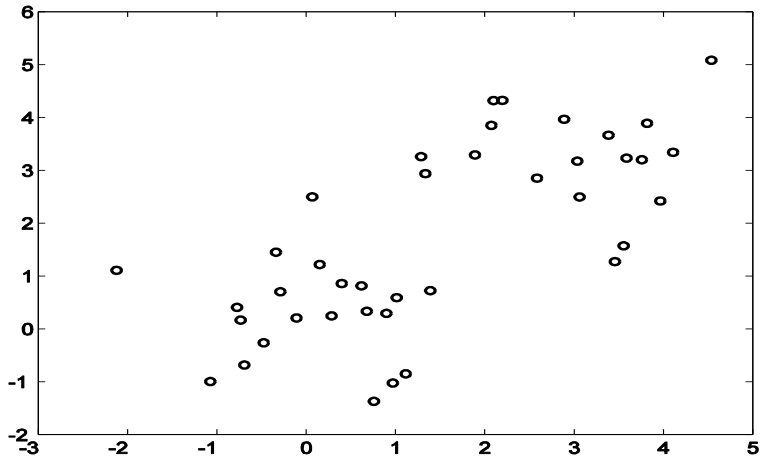
- Many top down clustering algorithms work by first constructing a neighborhood graph and then trying to infer some sort of connected components in that graph

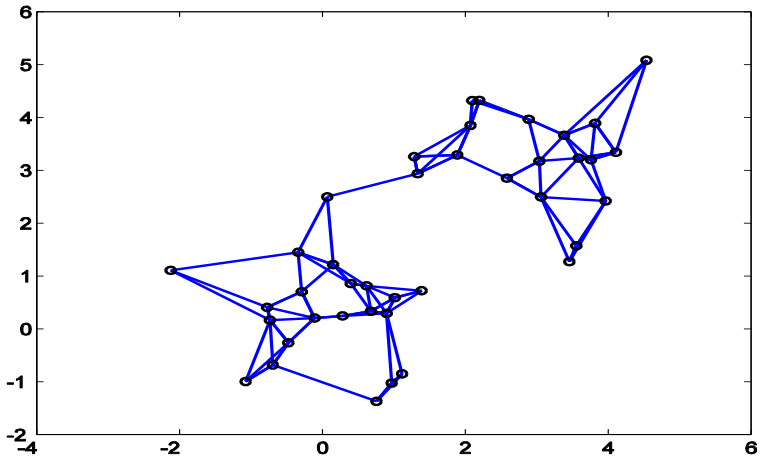


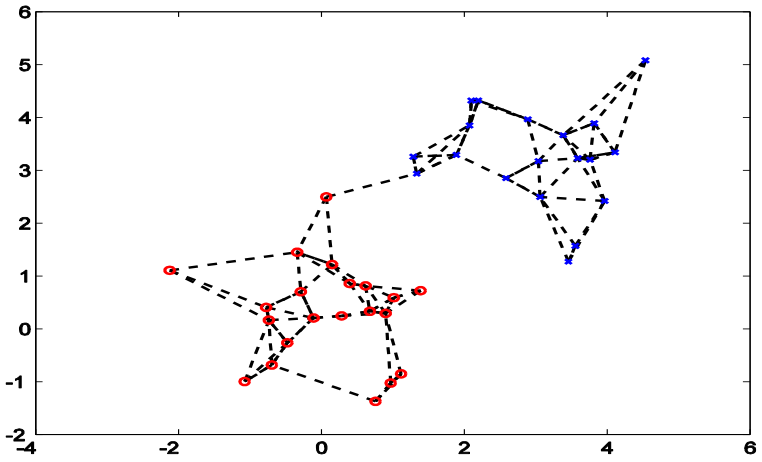
Graph based clustering

- We need to clarify how to perform the following three steps:
    1. construct the neighborhood graph
    2. assign weights to the edges (similarity)
    3. partition the nodes using the graph structure

Example

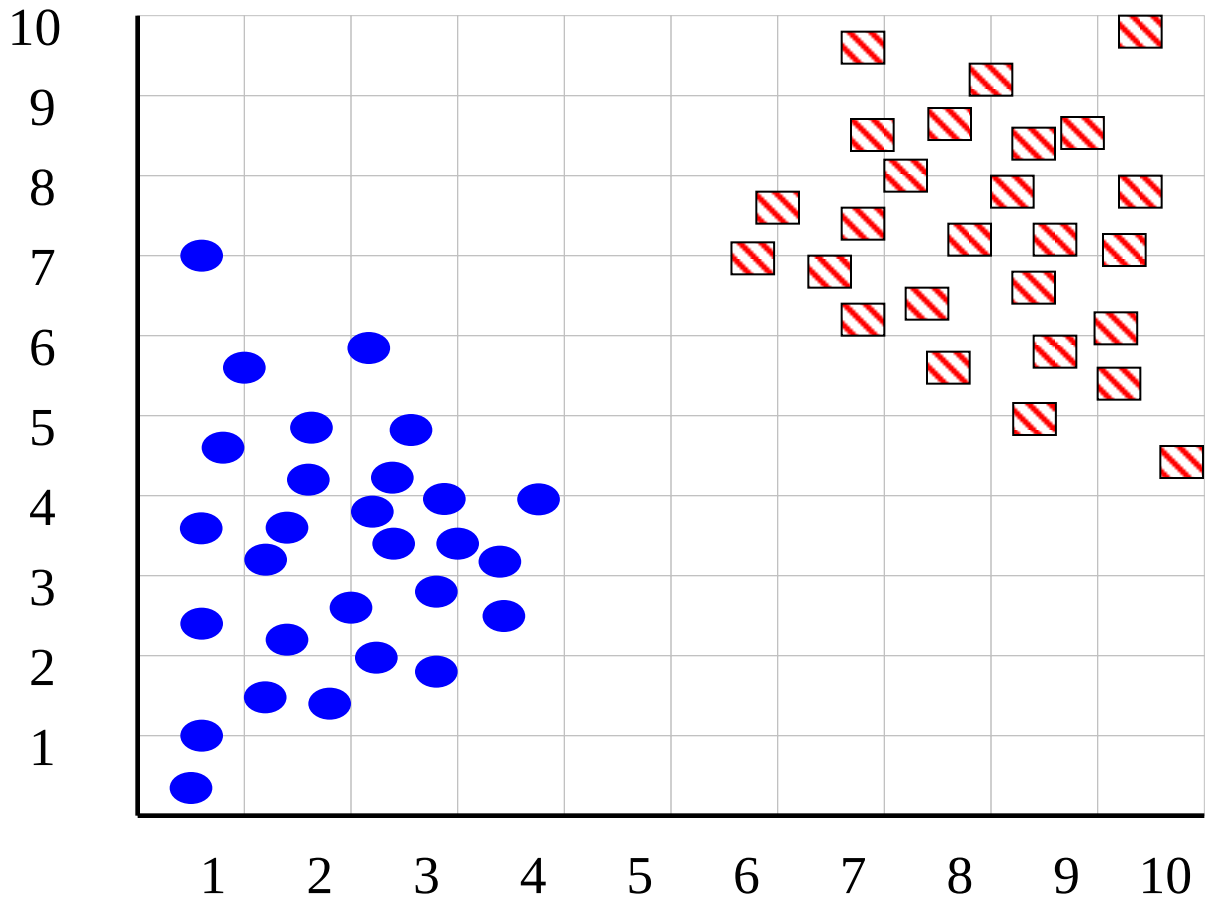




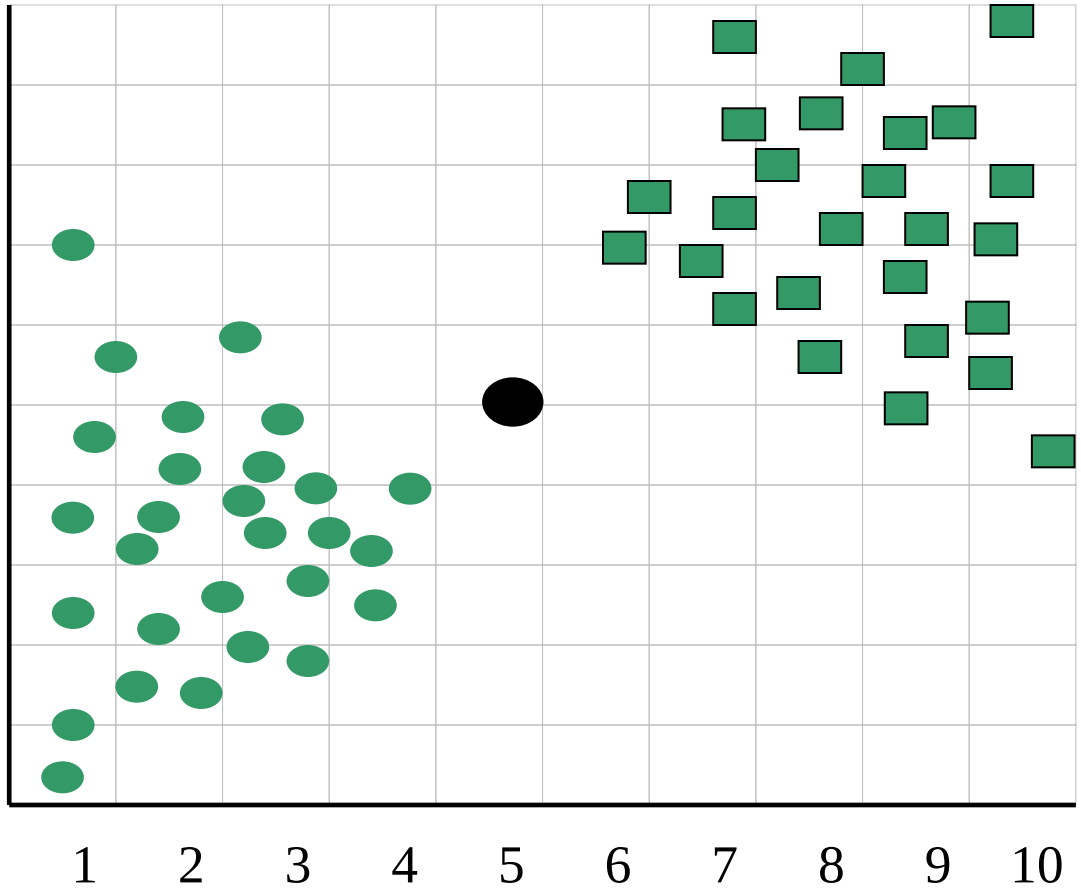


How can we tell the *right* number of clusters?

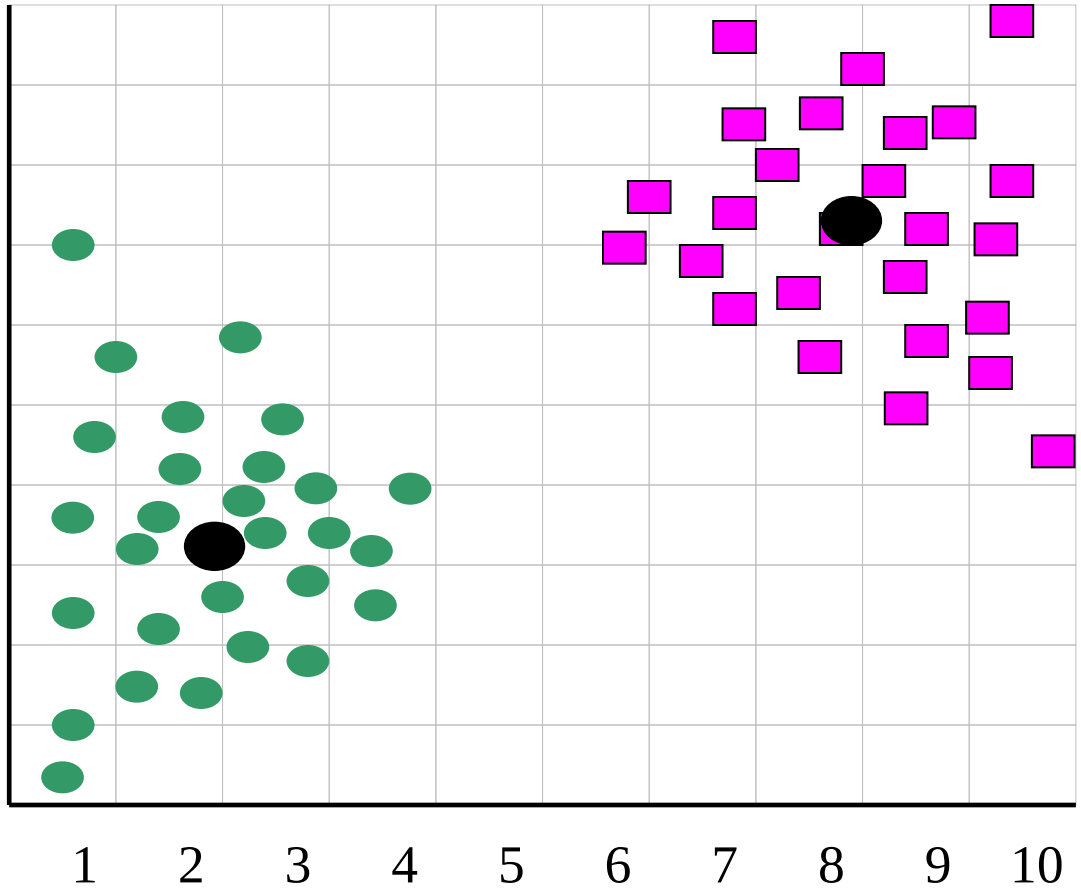
In general, this is a unsolved problem. However there are many approximate methods. In the next few slides we will see an example.



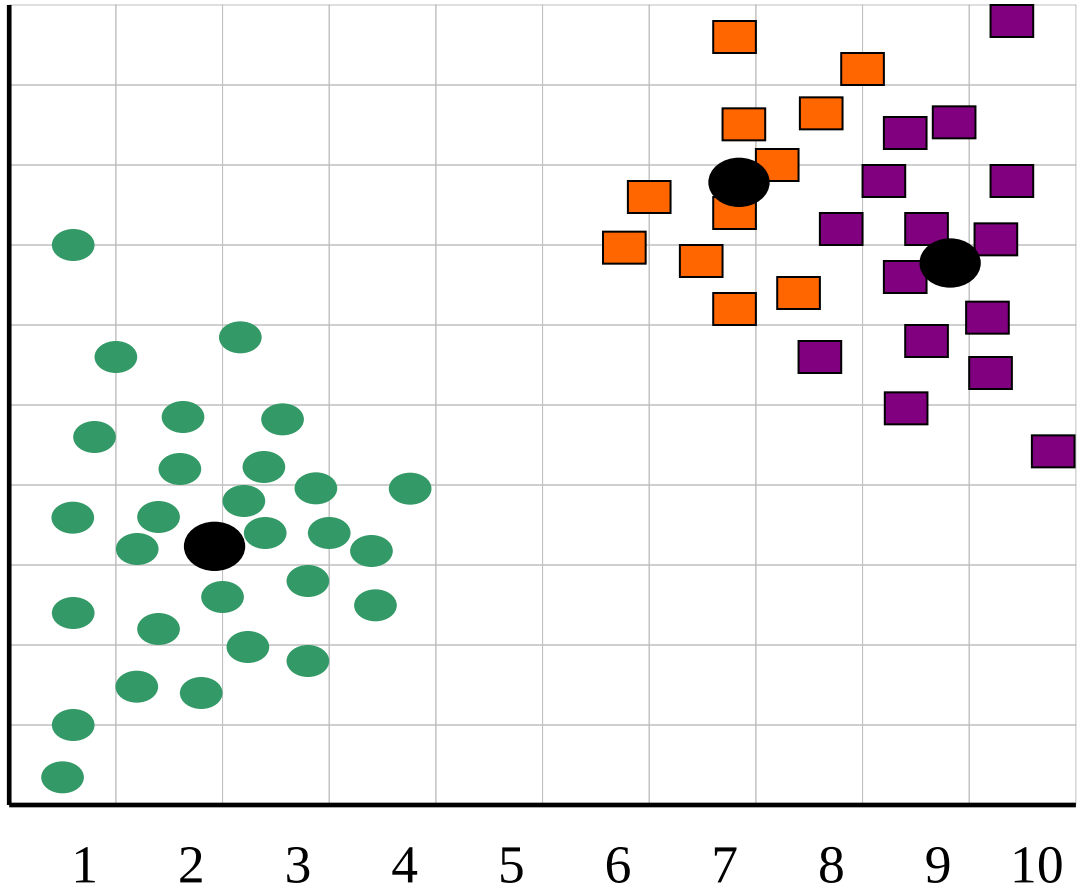
When k = 1, the objective function is 873.0



When k = 2, the objective function is 173.1

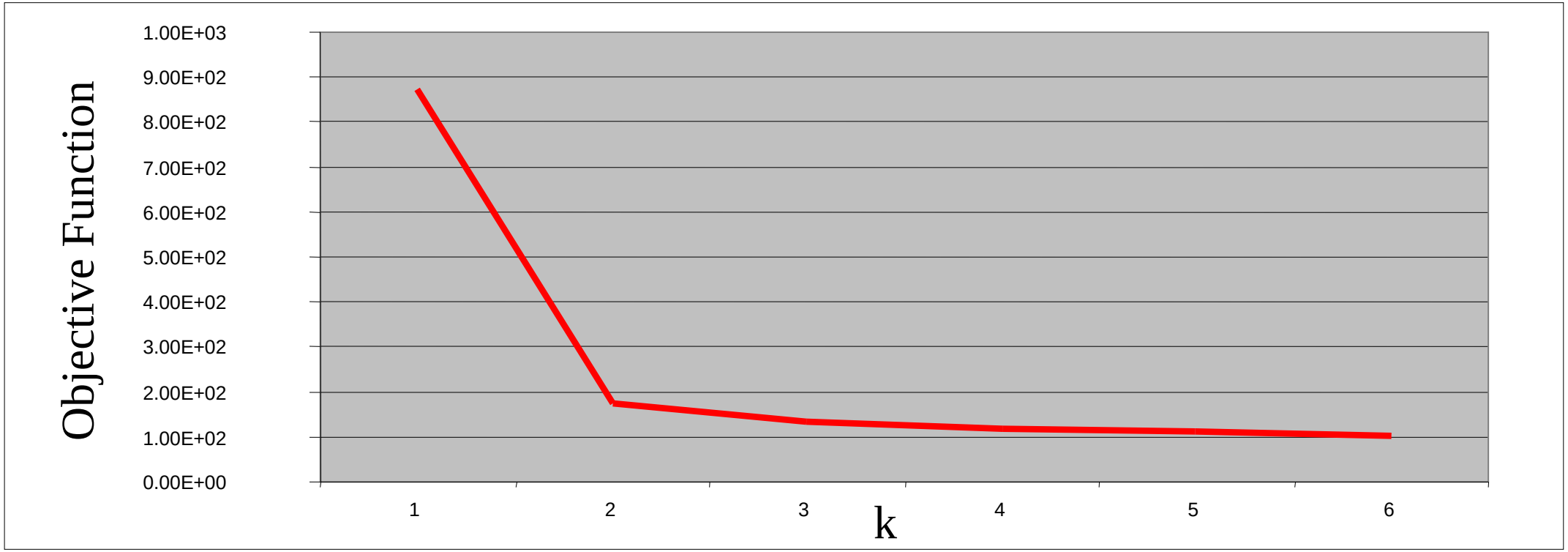


When k = 3, the objective function is 133.6



We can plot the objective function values for k equals 1 to 6…

The abrupt change at k = 2, is highly suggestive of two clusters in the data. This technique for determining the number of clusters is known as “knee finding” or “elbow finding”.

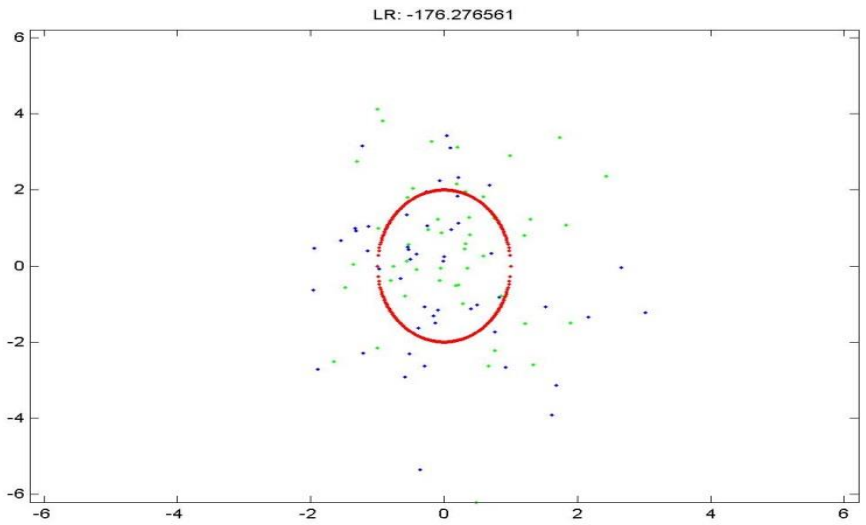


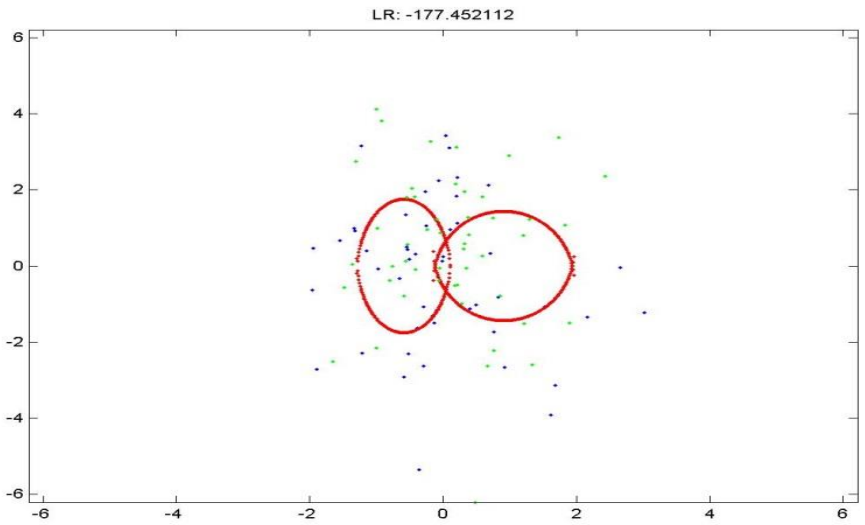
Note that the results are not always as clear cut as in this toy example

Cross validation

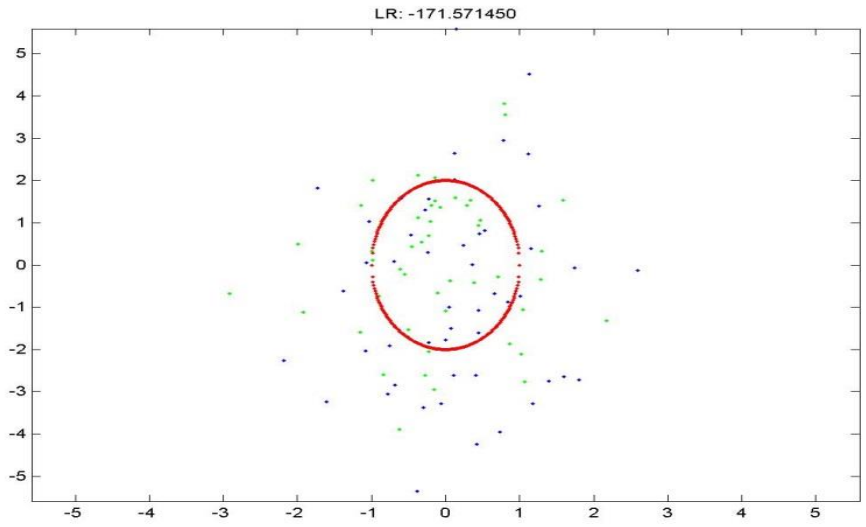
- We can also use cross validation to determine the correct number of classes
- Recall that GMMs is a generative model. We can compute the likelihood of the left out data to determine which model (number of clusters) is more accurate

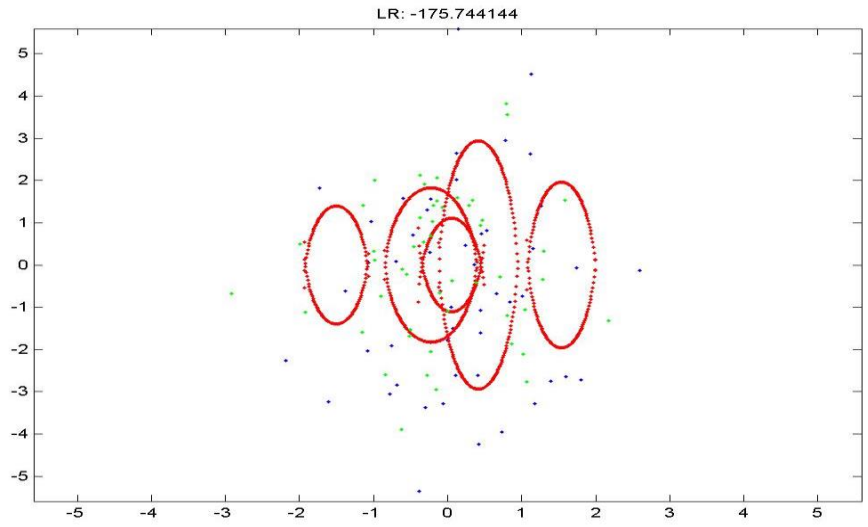
$$p(x_1 \cdots x_n \mid \theta) = \prod_{j=1}^{n} \left(\sum_{i=1}^{k} p(x_j \mid C = i) w_i \right)$$





Cross validation





Cluster validation

- We wish to determine whether the clusters are real or compare different clustering methods.
  - internal validation (stability, coherence)
  - external validation (match to known categories)

Internal validation: Coherence

- A simple method is to compare clustering algorithm based on the coherence of their results
- We compute the average inter-cluster similarity and the average intra-cluster similarity
- Requires the definition of the similarity / distance metric

Internal validation: Stability

- If the clusters capture real structure in the data they should be stable to minor perturbation (e.g., subsampling) of the data.
- To characterize stability we need a measure of similarity between any two k-clusterings.
- For any set of clusters C we define L(C) as the matrix of 0/1 labels such that $L(C)_{ij}$ =1 if objects i and j belong to the same cluster and zero otherwise.
- We can compare any two k clusterings C and C' by comparing the corresponding label matrices L(C) and L(C').

Validation by subsampling

- C is the set of k clusters based on all the objects
- C' denotes the set of k clusters resulting from a randomly chosen subset (80-90%) of objects
- We have high confidence in the original clustering if Sim(L(C),L(C')) approaches 1 with high probability, where the comparison is done over the objects common to both

External validation

- For this we need an external source that contains related, but usually not identical information.
- For example, assume we are clustering web pages based on the car pictures they contain.
- We have independently grouped these pages based on the text description they contain.
- Can we use the text based grouping to determine how well our clustering works?

External validation

- Suppose we have generated k clusters $C_1,\ldots,C_k$. How do we assess the significance of their relation to m known (potentially overlapping) categories $G_1,\ldots,G_m$?
- Let's start by comparing a single cluster C with a single category G_j. The p-value for such a match is based on the hyper-geometric distribution.
- Board.
- This is the probability that a randomly chosen $|C_i|$ elements out of n would have l elements in common with G_j.

P-value (cont.)

- If the observed overlap between the sets (cluster and category) is l elements (genes), then the p-value is

$$p = prob(l \geq \hat{l}) = \sum_{j=l}^{\min(c,m)} prob(exactly - j - matches)$$

- Since the categories $G_1,\ldots,G_m$ typically overlap we cannot assume that each cluster-category pair represents an independent comparison
- In addition, we have to account for the multiple hypothesis we are testing.
- Solution ?

External validation: Example