COMP 6321
Machine Learning

Instructor: Adam Krzyżak
Email: krzyzak@cs.concordia.ca

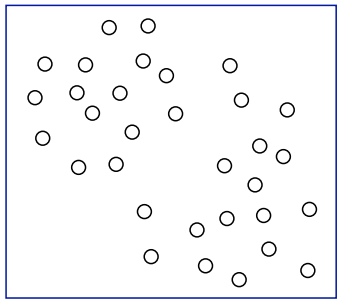
Lecture 7: Learning. Clustering. EM

- Active learning scenarios
- Semi-supervised learning
- Introduction to unsupervised learning
- K-means clustering
- Hierarchical clustering
- Mixture models
- Expectation Maximization for mixture models

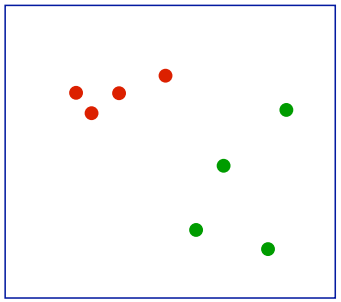
Recall: Passive supervised learning

- The environment provides labeled data in the form of pairs $(\mathbf{x}, y)$
- We can process the examples either as a batch or one at a time, with the goal of producing a predictor of y as a function of $\mathbf{x}$
- We assume that there is an underlying distribution P generating the examples
- Each example is drawn i.i.d. from P
- What if instead we are allowed to *ask for particular examples*?
- Intuitively, if we are allowed to ask questions, and if we are smart about what we want to know, fewer examples may be necessary

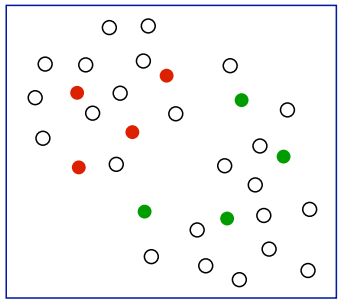
Semi-Supervised and Active Learning



Unlabeled points



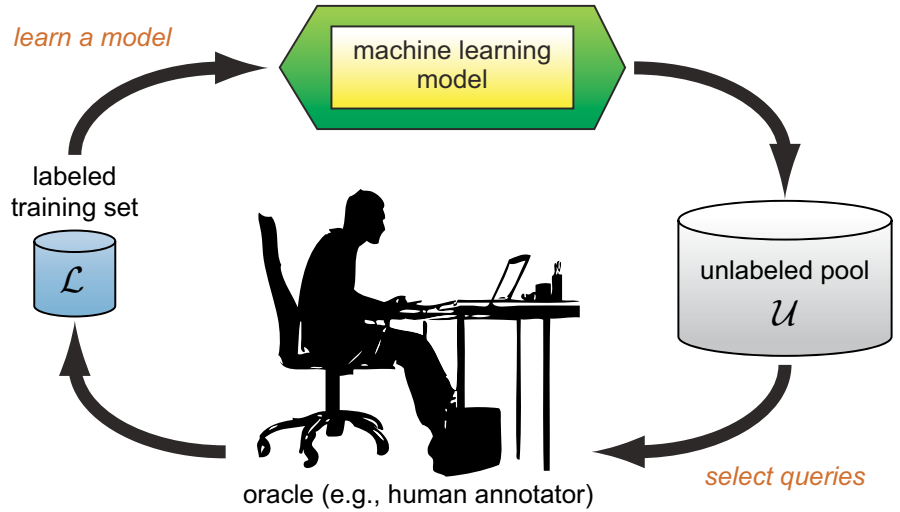
Supervised learning



Semisupervised and active learning

- Suppose you had access to a lot of unlabeled data
  E.g. all the documents on the web
  E.g. all the pictures on Picasa
- You can also get some labelled data, but not much
- How can we take advantage of the unlabeled data to improve supervised learning performance?

Active Learning



- The learner can query an “expert” for a label on any example
- The expert could be a person or a fancy automated program
- Queries are expensive or slow
- What examples should we ask for next?

Applications

- Document classification
- Document tagging (e.g. determining parts-of-speech, semantic objects like places, names, ..)
- Image classification
- Image tagging (e.g. tag all people in a picture)
- Chemistry
- Biomedical applications (labels are obtained by asking a doctor)
- Robotics: what is the true position and velocity of the robot?

Semi-supervised learning

- Label examples with current classifier
- Add the ones with highest confidence to the current labeled set, and re-fit the classifier
- Similar to active learning, but where the learner itself pays the role of the "expert" as well
- Blum and Mitchell: co-training (1998), a version in which two separate sources of information are used to train each other

  E.g. a web page can be classifed based on its text, or based on its link structure

Unsupervised learning

- In supervised learning, data is in the form of pairs $\langle \mathbf{x}, y \rangle$, where $y = f(\mathbf{x})$, and the goal is to approximate f well.
- In *unsupervised learning*, the data just contains $\mathbf{x}$!
- Goal is to "summarize" or find "patterns" or "structure" in the data
- A variety of problems and uses:
  - Clustering: "Flat" clustering or partitioning, hierarchical clustering
  - Density estimation
  - Dimensionality reduction, for: visualization, compression, pre-processing
- The definition of "ground truth" is often missing: no clear error function, or at least many reasonable alternatives
- Often useful in exploratory data analysis, and as a pre-processing step for supervised learning

What is clustering?

- Clustering is grouping similar objects together.
  - To establish prototypes, or detect outliers.
  - To simplify data for further analysis/learning.
  - To visualize data (in conjunction with dimensionality reduction)
- Clusterings are usually not "right" or "wrong" – different clusterings/clustering criteria can reveal different things about the data.
- Some clustering criteria/algorithms have natural probabilistic interpretations
- Clustering algorithms:
  - Employ some notion of distance between objects
  - Have an explicit or implicit criterion defining what a good cluster is
  - Heuristically optimize that criterion to determine the clustering

K-means clustering

- One of the most commonly-used clustering algorithms, because it is easy to implement and quick to run.
- Assumes the objects (instances) to be clustered are n-dimensional vectors, $\mathbf{x}_i$.
- Uses Euclidean distance
- The goal is to *partition* the data into K disjoint subsets

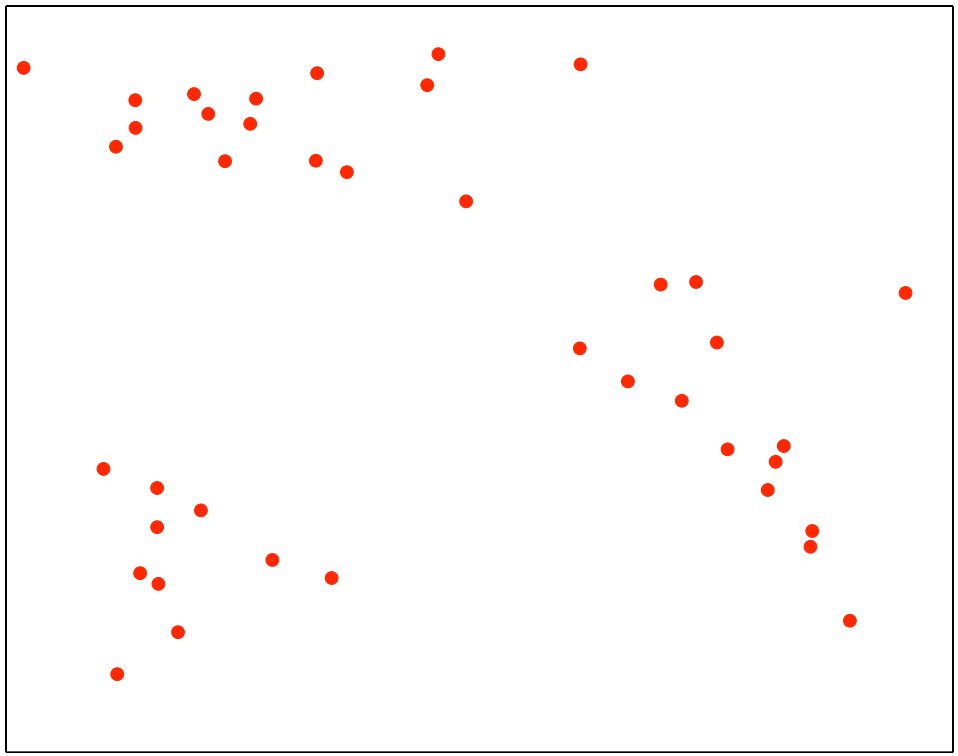
K-means clustering with real-valued data

- Inputs:
  - A set of n-dimensional real vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$.
  - K, the desired number of clusters.
- Output: A mapping of the vectors into K clusters (disjoint subsets), $C : \{1, \ldots, m\} \mapsto \{1, \ldots, K\}$.

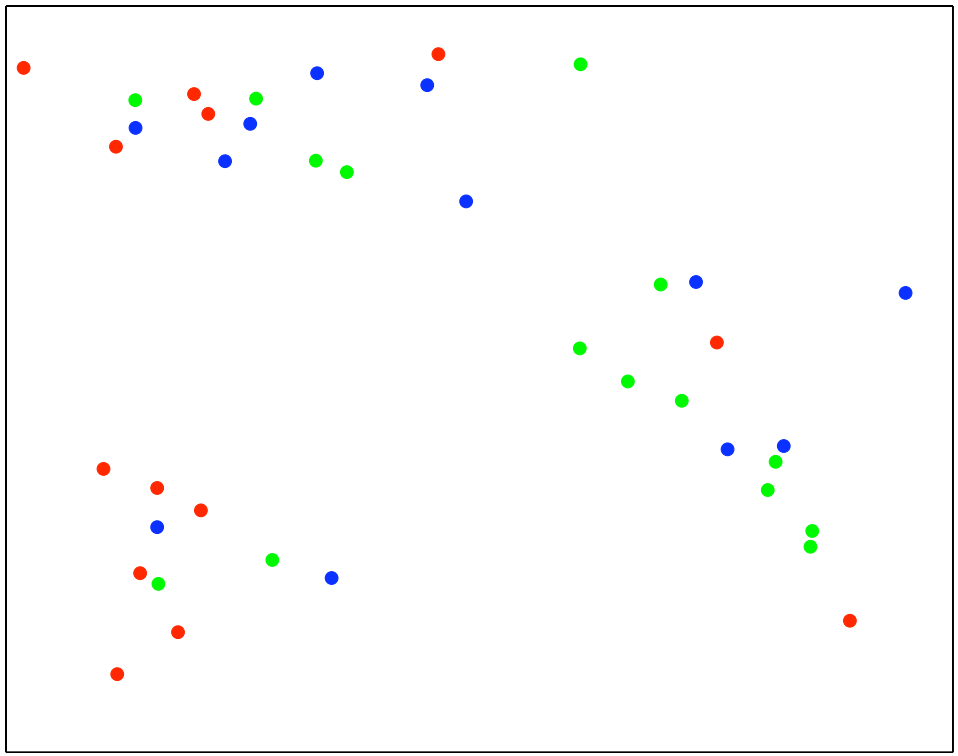
1. Initialize C randomly.
2. Repeat
   (a) Compute the *centroid* of each cluster (the mean of all the instances in the cluster)
   (b) Reassign each instance to the cluster with closest centroid

   until C stops changing.

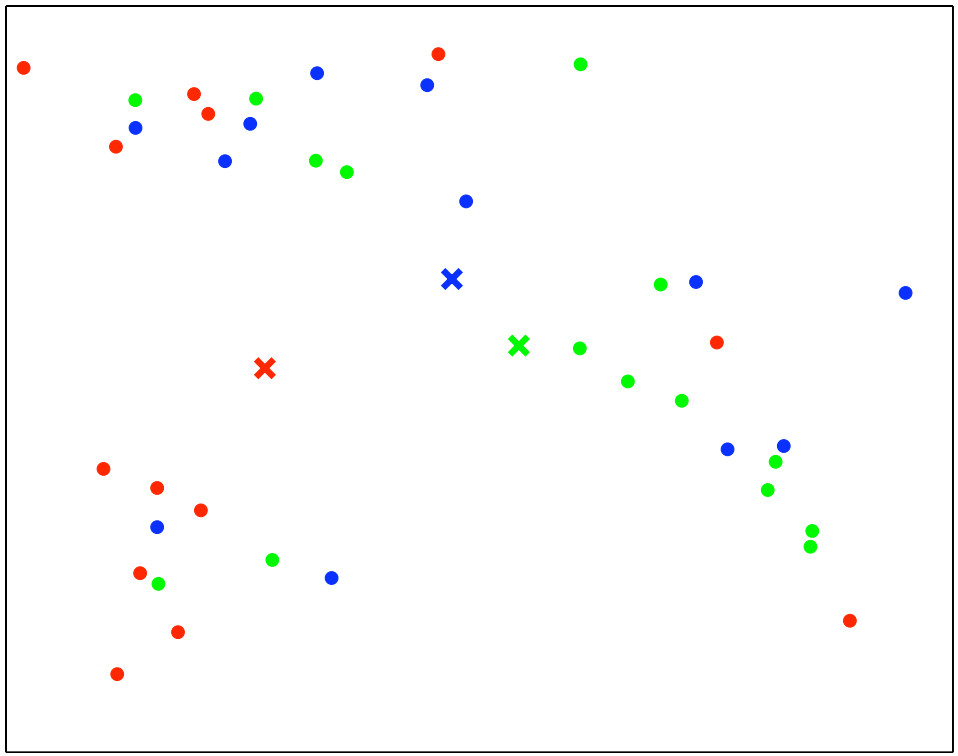
Example: initial data



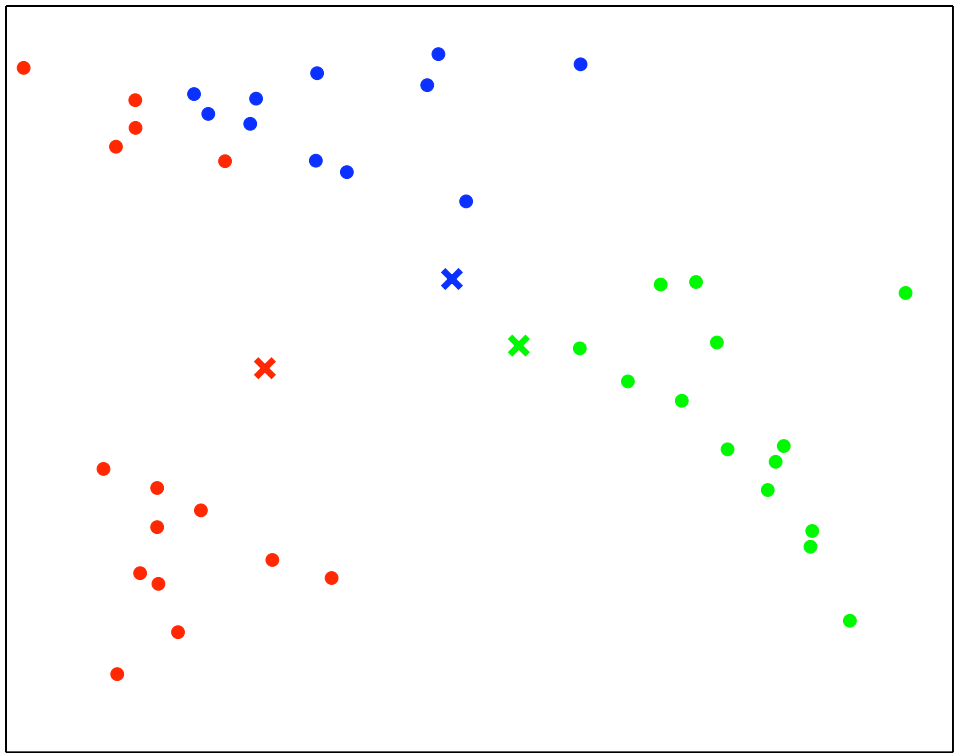
Example: assign into 3 clusters randomly



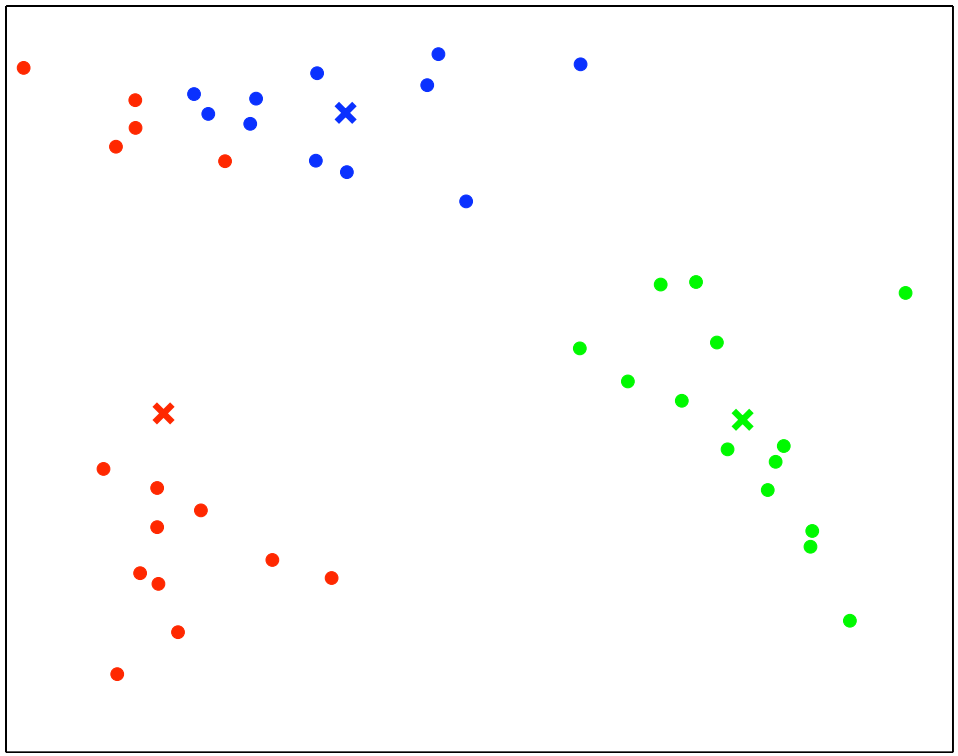
Example: compute centroids



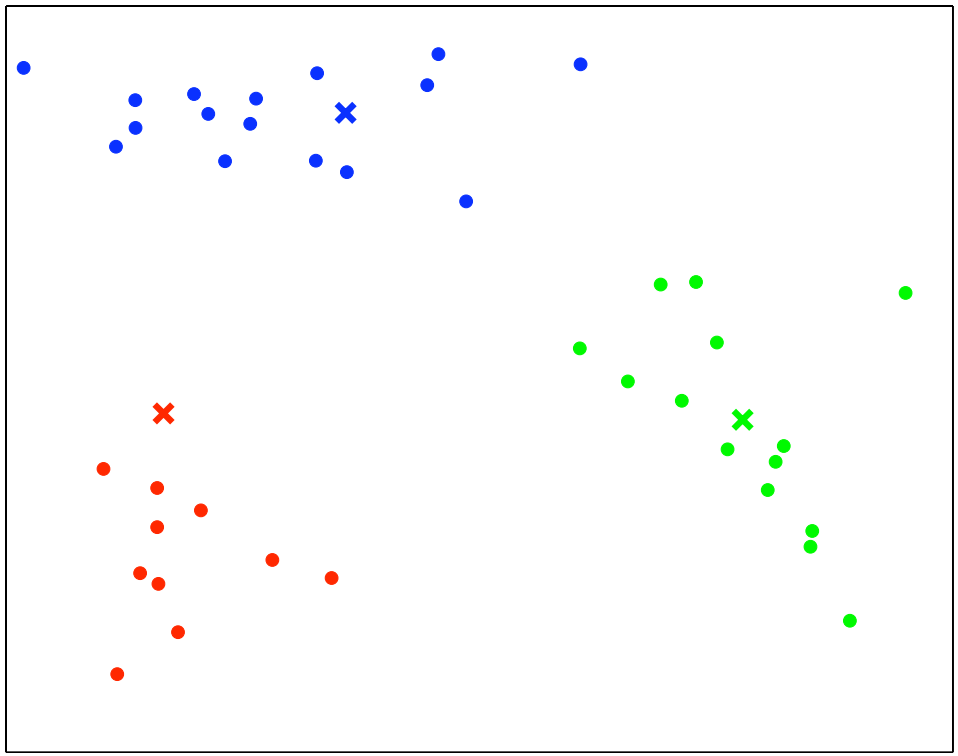
Example: reassign clusters



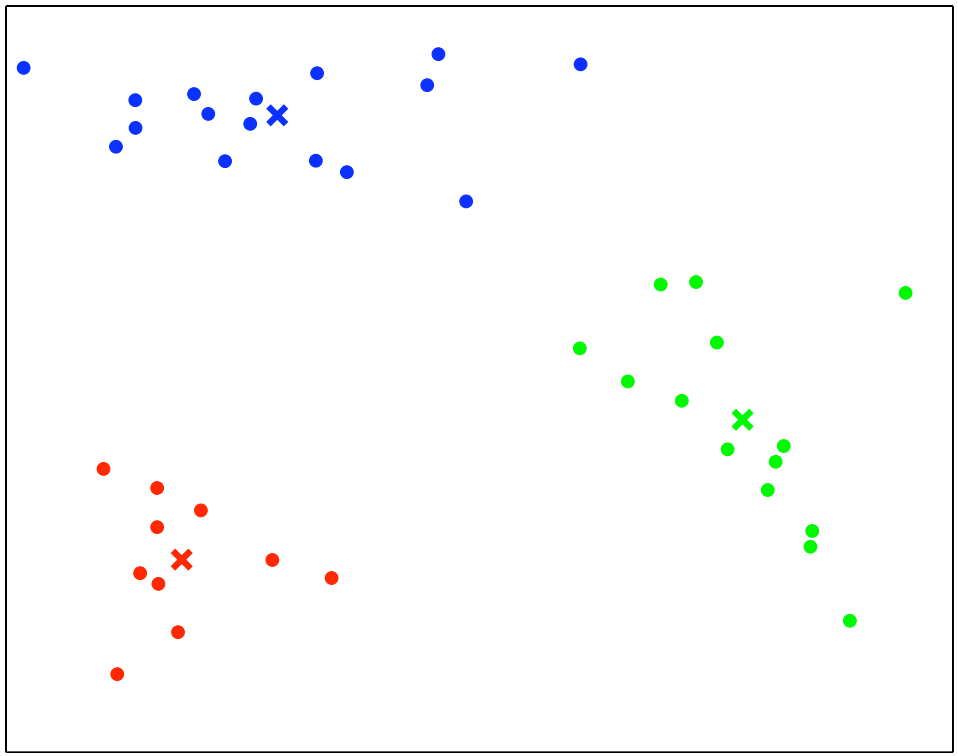
Example: recompute centroids



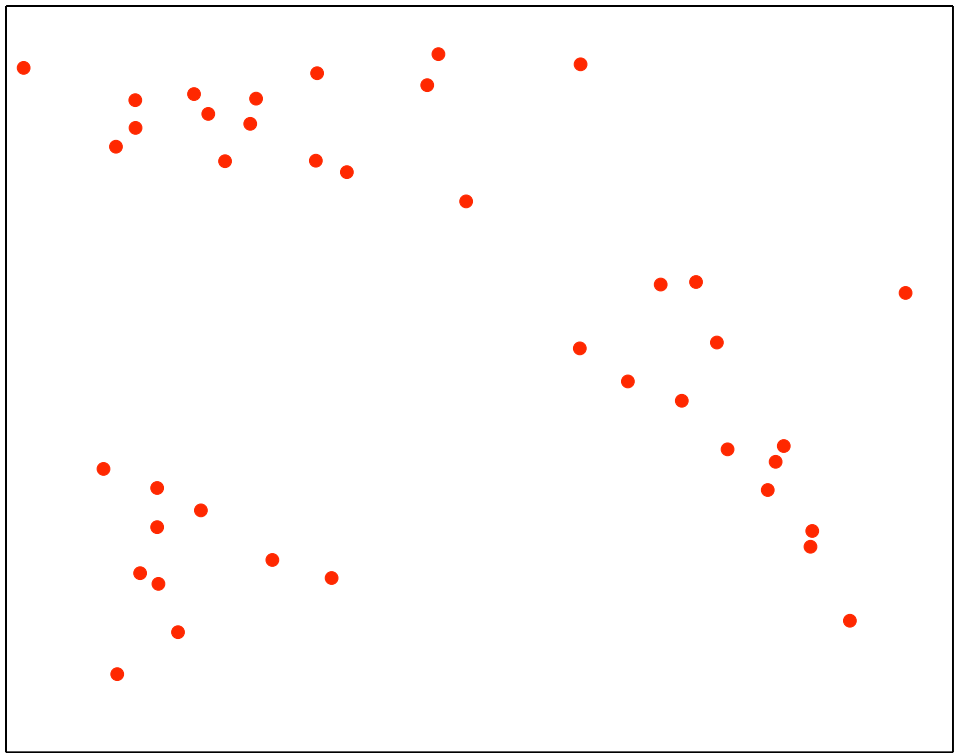
Example: reassign clusters



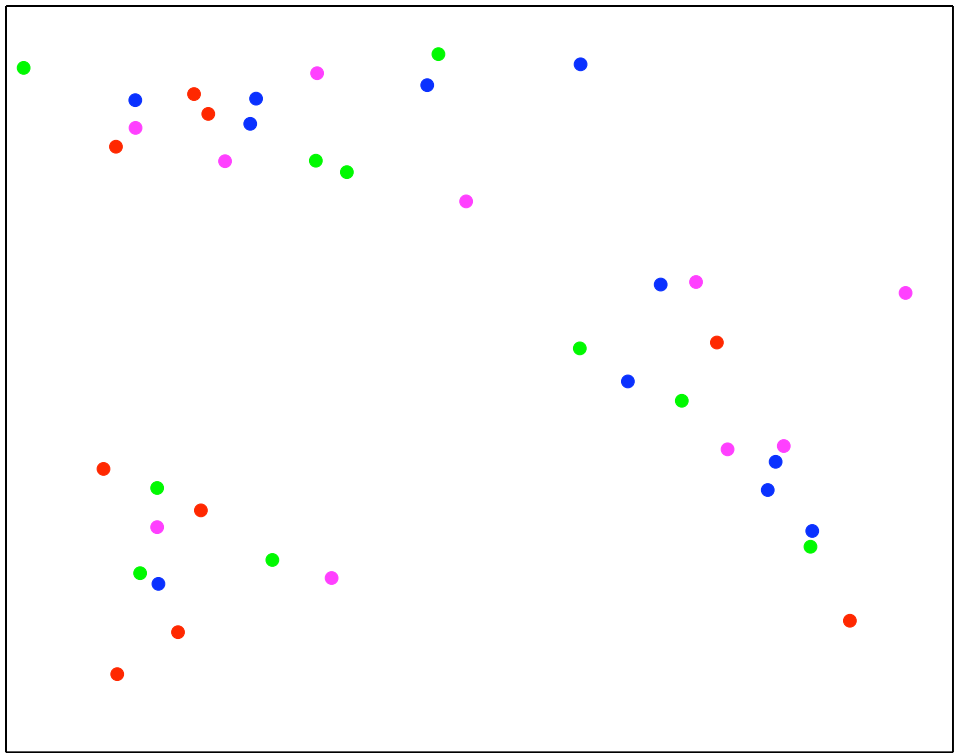
Example: recompute centroids – done!



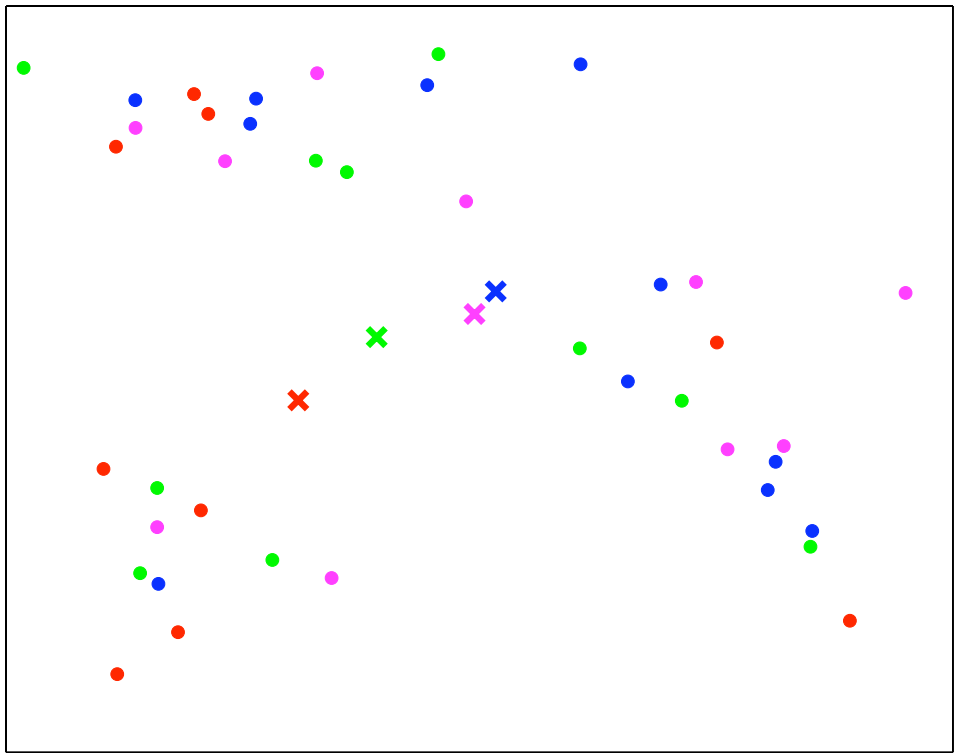
What if we do not know the right number of clusters?



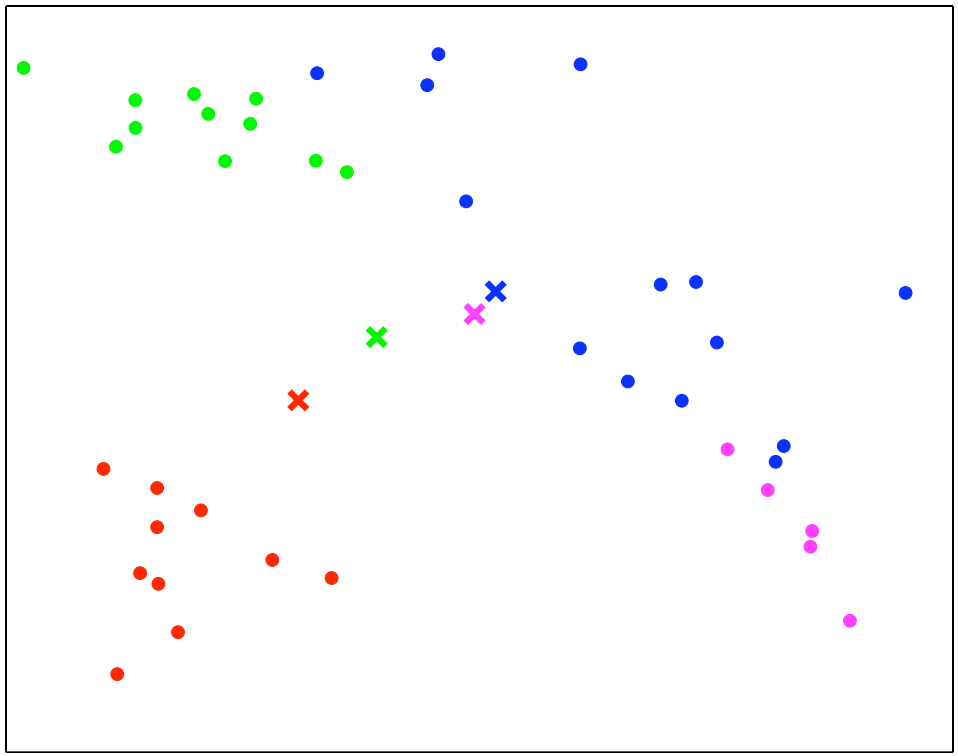
Example: assign into 4 clusters randomly



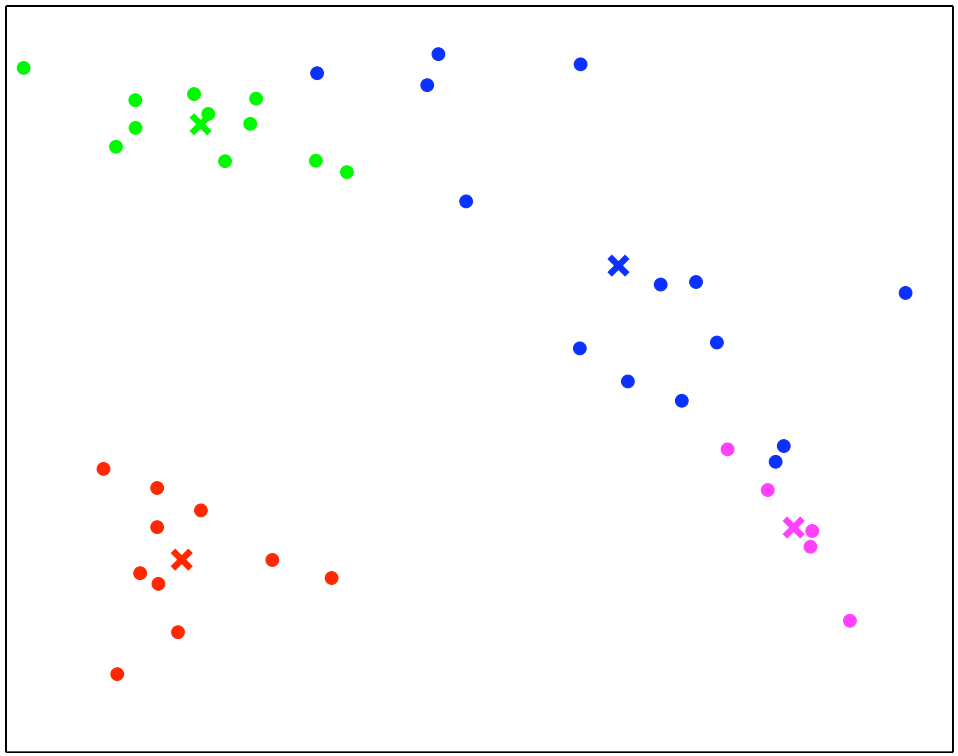
Example: compute centroids



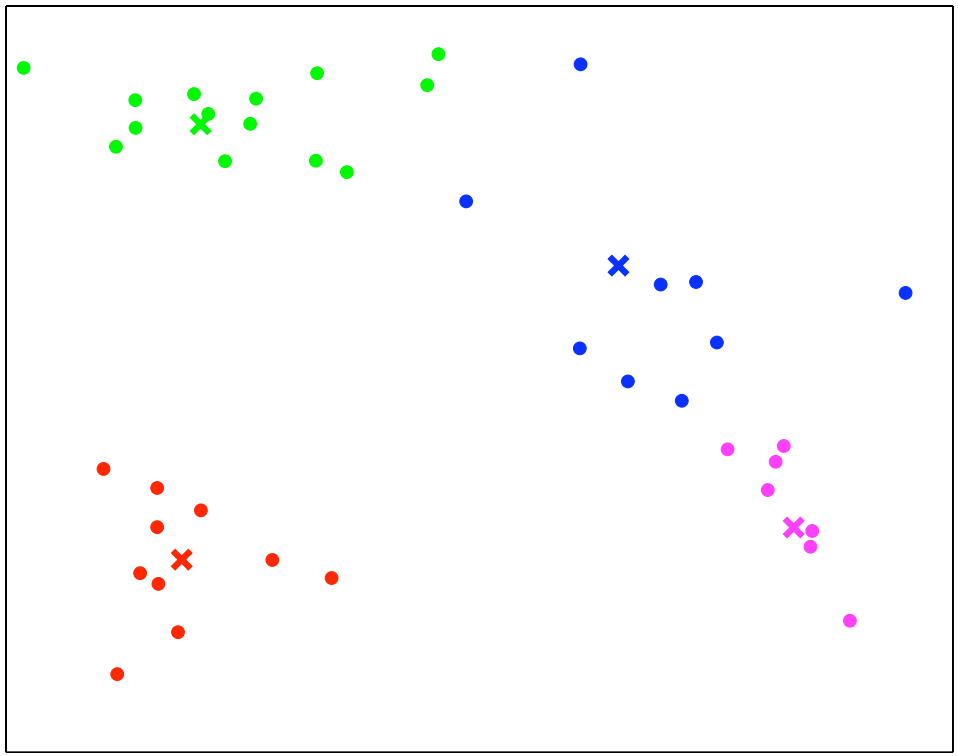
Example: reassign clusters



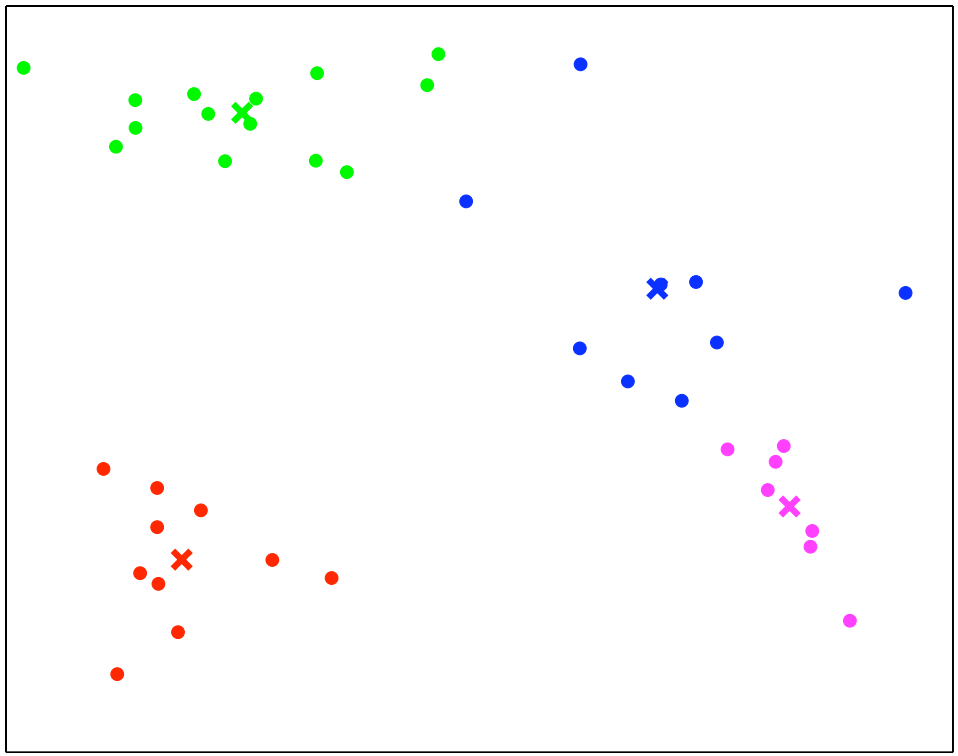
Example: recompute centroids



Example: reassign clusters



Example: recompute centroids – done!



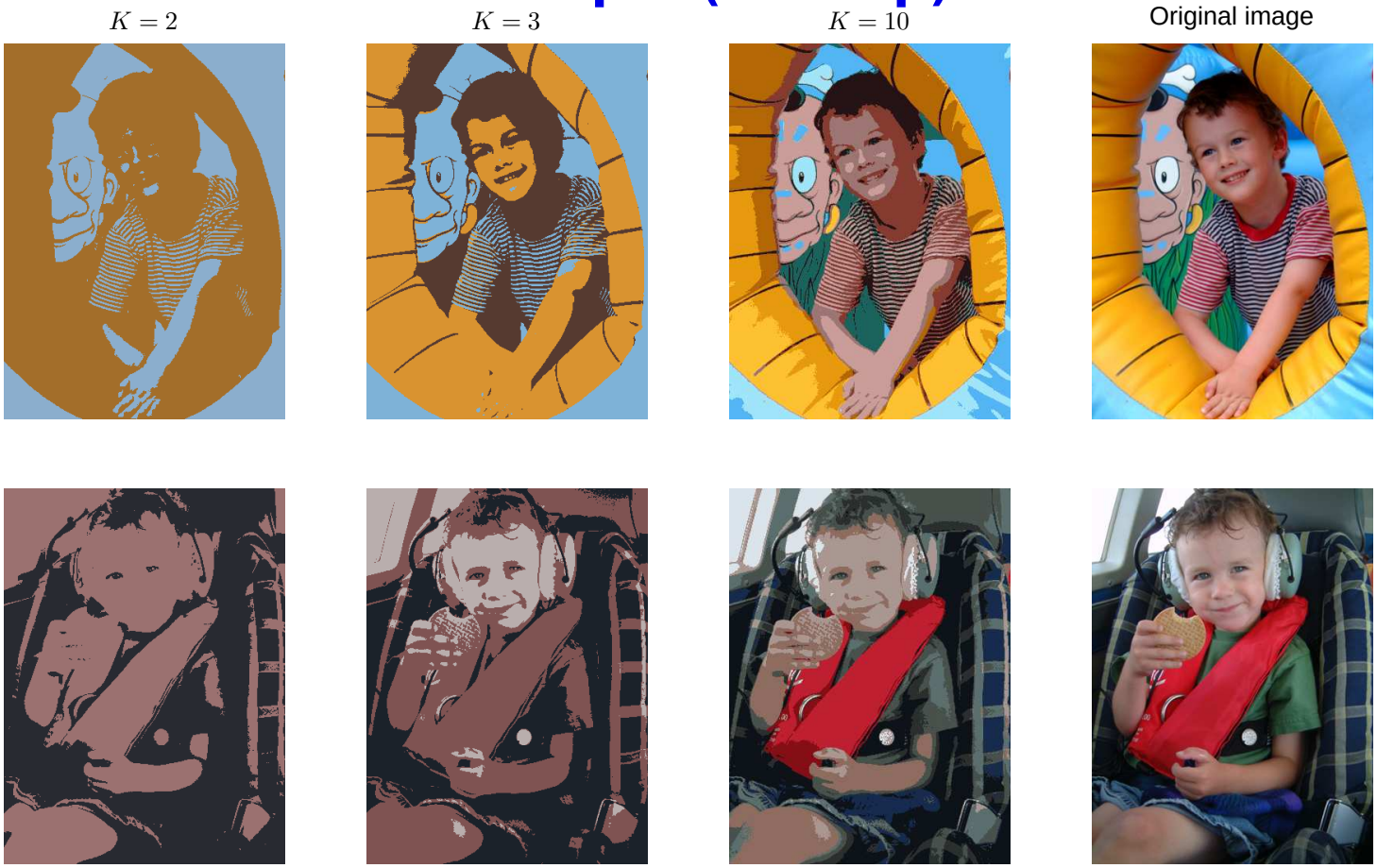
Example application: Color quantization

- Suppose you have an image stored with 24 bits per pixel
- You want to compress it so that you use only 8 bits per pixel (256 colors)
- You want the compressed image to look *as similar as possible* to the original image

$\Rightarrow$ Perform $K-$means clustering on the original set of color vectors with $K = 256$ colors.

  - Cluster centers (rounded to integer intensities) form the entries in the 256-color colormap
  - Each pixel represented by 8-bit index into colormap

Example (Bishop)



More generally: Vector quantization with Euclidean loss

- Suppose we want to send all the instances over a communication channel
- In order to compress the message, we cluster the data and *encode each instance as the center of the cluster* to which it belongs
- The *reconstruction error* for real-valued data can be measured as Euclidian distance between the true value and its encoding
- An optimal K-means clustering minimizes the reconstruction error among all possible codings of the same type

Questions

- What is K-means trying to optimize?
- Will it terminate?
- Will it always find the same answer?
- How should we choose the initial cluster centers?
- Can we automatically choose the number of centers?

Does K-means clustering terminate?

- For given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ and a clustering C, consider the sum of the squared Euclidian distance between each vector and the center of its cluster:

$$J = \sum_{i=1}^{m} \|\mathbf{x}_i - \mu_{C(i)}\|^2 ,$$

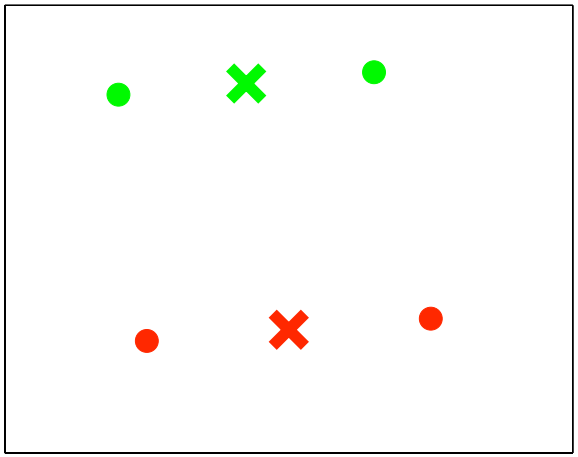
  where $\mu_{C(i)}$ denotes the centroid of the cluster containing $\mathbf{x}_i$.

- There are finitely many possible clusterings: at most K^m.
- Each time we reassign a vector to a cluster with a nearer centroid, J decreases.
- Each time we recompute the centroids of each cluster, J decreases (or stays the same.)
- Thus, the algorithm must terminate.

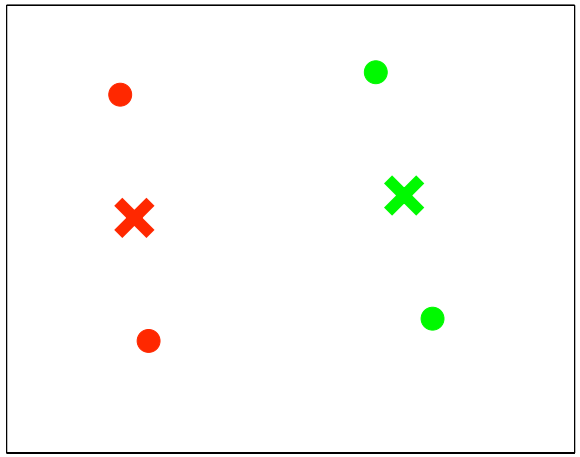
Does K-means always find the same answer?

- K-means is a version of coordinate descent, where the parameters are the cluster center coordinates, and the assignments of points to clusters.
- It minimizes the sum of squared Euclidean distances from vectors to their cluster centroid.
- This error function has many local minima!
- The solution found is *locally optimal*, but *not globally optimal*
- Because the solution depends on the initial assignment of instances to clusters, random restarts will give different solutions

Example



$J = 0.22870$



$J = 0.3088$

Finding good initial configurations

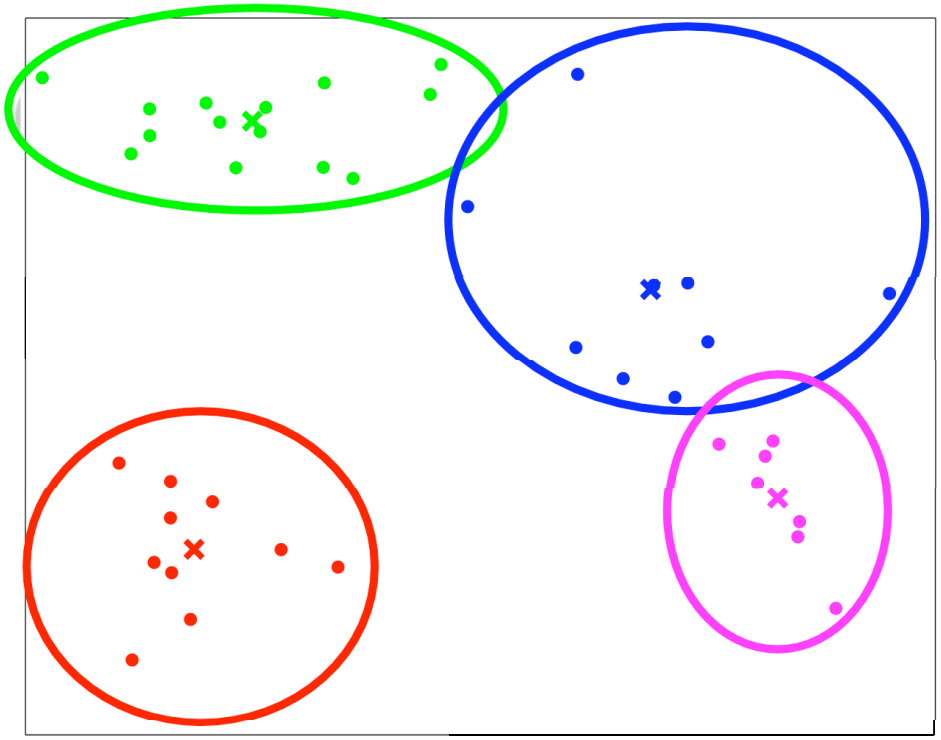
- The initial configuration can influence the final clustering
- Assigning each item to random cluster in $\{1, \ldots, K\}$ is unbiased. . . but typically results in cluster centroids near the centroid of all the data in the first round.
- A different heuristic tries to spread the initial centroids around as much as possible:
  - Place first center on top of a randomly chosen data point
  - Place second center on a data point as far away as possible from the first one
  - Place the i-th center as far away as possible from the closest of centers 1 through $i - 1$
- K-means clustering typically runs quickly. With a randomlized intialization step, you can run the algorithm multiple times and take the clustering with smallest J.

Choosing the number of clusters

- A difficult problem, ideas are floating around
- Delete clusters that cover too few points
- Split clusters that cover too many points
- Add extra clusters for “outliers”
- Minimum description length: minimize loss + complexity of the clustering
- Use a hierarchical method first (see in a bit)

Why the sum of squared Euclidean distances?

Subjective reason: It produces nice, round clusters.



Why the sum of squared Euclidean distances?

Objective reason: Maximum Likelihood Principle

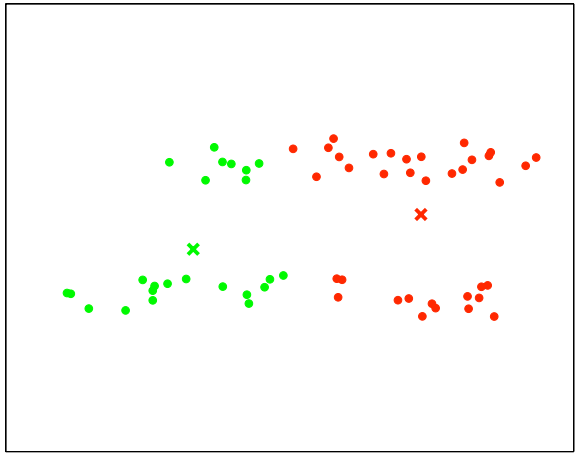
- Suppose the data really does divide into K clusters.
- Suppose the data in each cluster is generated by independent samples from a multivariate Gaussian distribution, where:
  - The mean of the Gaussian is the centroid of the cluster
  - The covariance matrix is of the form $\sigma^2 I$
- Then the probability of the data is highest when the sum of squared Euclidean distances is smallest.

Derivation: similar to MSE motivation in supervised learning

$$
\begin{aligned}
L(\mathbf{x}_1, \ldots, \mathbf{x}_m | C(i), \mu_j) &= \prod_{i=1}^{m} L(\mathbf{x} | (C(i), \mu_j) \\
&= \prod_{i=1}^{m} \frac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mu_{C(i)}\|^2\right) \\
\log L(\mathbf{x}_1, \ldots, \mathbf{x}_m | C(i), \mu_j) &\propto -\sum_{i=1}^{m} \|\mathbf{x}_i - \mu_{C(i)}\|^2 = J
\end{aligned}
$$

Why *not* the sum of squared Euclidean distances?

1. It produces nice round clusters!



2. Differently scaled axes can dramatically affect results.
3. There may be symbolic attributes, which have to be treated differently

K-means-like clustering in general

- Given a set of objects (need not be real vectors),
  - Choose a notion of pairwise distance / similarity between the objects.
  - Choose a scoring function for the clustering
  - Optimize the scoring function, to find a good clustering.
- For most choices, the optimization problem will be intractable. Local optimization is often necessary.

Distance metrics

- Euclidean distance
- Hamming distance (number of mismatches between two strings)
- Travel distance along a manifold (e.g. for geographic points)
- Tempo / rhythm similarity (for songs)
- Shared keywords (for web pages), or shared in-links
- . . .

Scoring functions

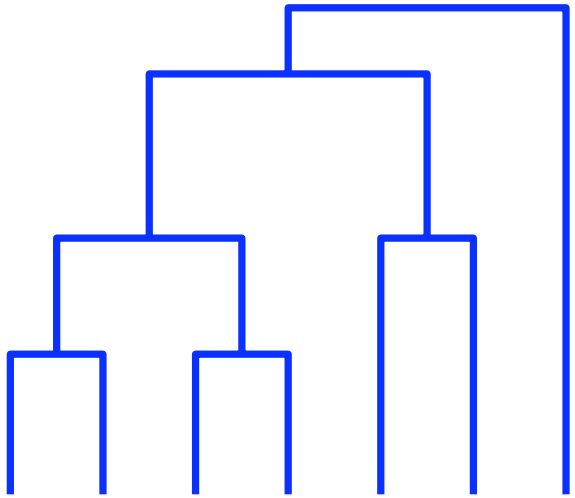
- Minimize: Summed distances between all pairs of objects in the same cluster. (Also known as ”within-cluster scatter.”)
- Minimize: Maximum distance between any two objects in the same cluster. (Can be hard to optimize.)
- Maximize: Minimum distance between any two objects in different clusters.

Common uses of K-means

- Often used in exploratory data analysis
- Often used as a pre-processing step before supervised learning
- In one-dimension, it is a good way to discretize real-valued variables into non-uniform buckets
- Used in speech understanding/recognition to convert wave forms into one of k categories (vector quantization)

Hierarchical clustering

- Organizes data instances into trees.
- For visualization, exploratory data analysis.
- *Agglomerative methods* build the tree bottom-up, successively grouping together the clusters deemed most similar.
- *Divisive methods* build the tree top-down, recursively partitioning the data.

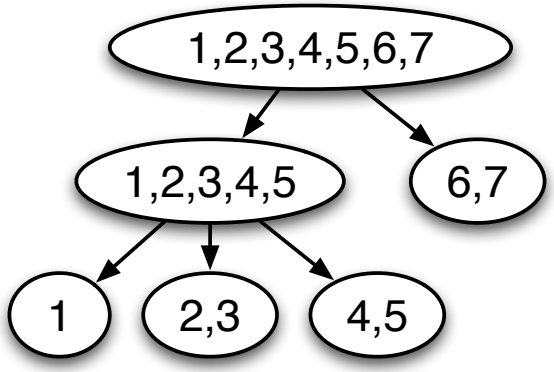


What is a hierarchical clustering?

- Given instances $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$.
- A hierarchical clustering is a set of subsets (clusters) of D, $C = \{C_1, \ldots, C_K\}$, where
  - Every element in D is in at least one set of C (the root)
  - The C_j can be assigned to the nodes of a tree such that the cluster at any node is precisely the union of the clusters at the node's children (if any).

Example of a hierarchical clustering

- Suppose $D = \{1, 2, 3, 4, 5, 6, 7\}$. A hierarchical clustering is $C = \{\{1\}, \{2, 3\}, \{4, 5\}, \{1, 2, 3, 4, 5\}, \{6, 7\}, \{1, 2, 3, 4, 5, 6, 7\}\}$.



- In this example:
  - Leaves of the tree need not correspond to single instances.
  - The branching factor of the tree is not limited.
- However, most hierarchical clustering algorithms produce binary trees, and take single instances as the smallest clusters.

Agglomerative clustering

- Input: Pairwise distances $d(\mathbf{x}, \mathbf{x}')$ between a set of data objects $\{\mathbf{x}_i\}$.
- Output: A hierarchical clustering
- Algorithm:
  1. Assign each instance as its own cluster on a working list W.
  2. Repeat
     (a) Find the two clusters in W that are most "similar".
     (b) Remove them from W.
     (c) Add their union to W.
     until W contains a single cluster with all the data objects.
  3. Return *all clusters* appearing in W at any stage of the algorithm.

How many clusters?

- How many clusters are generated by the agglomerative clustering algorithm?
- Answer: $2m - 1$, where m is the number of data objects.
- Why? A binary tree with m leaves has $m - 1$ internal nodes, thus $2m - 1$ nodes total.
- More explicitly:
  - The working list W starts with m singleton clusters
  - Each iteration removes two clusters from W and adds one new one
  - The algorithm stops when W has one cluster, which is after $m - 1$ iterations

How do we measure dissimilarity between clusters?

- Distance between nearest objects ("Single-linkage" agglomerative clustering, or "nearest neighbor"):

$$\min_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$

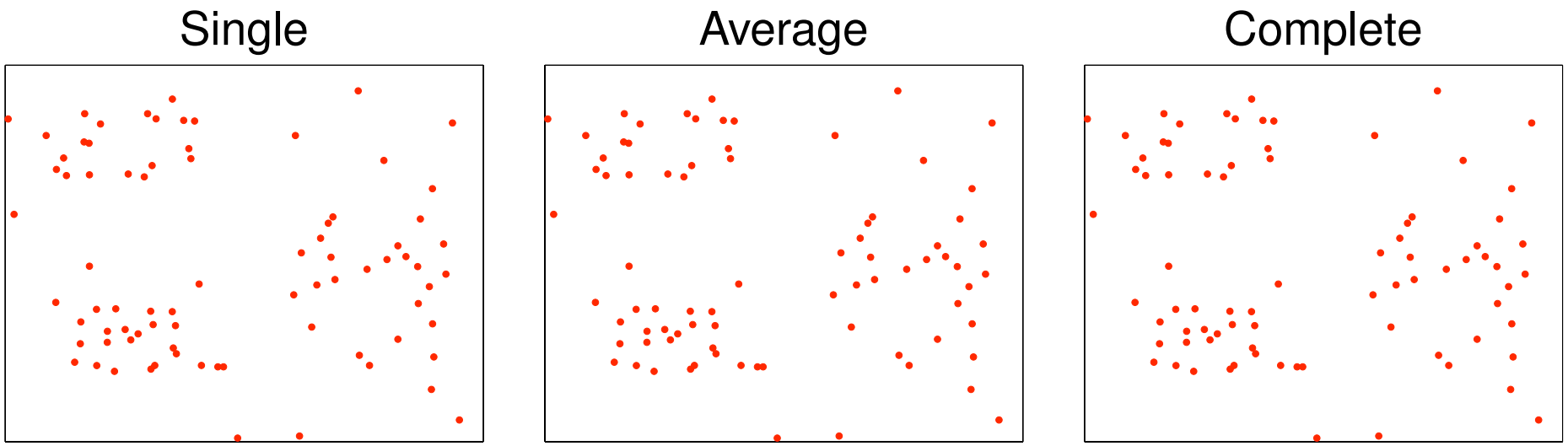
- Distance between farthest objects ("Complete-linkage" agglomerative clustering, or "furthest neighbor"):

$$\max_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$

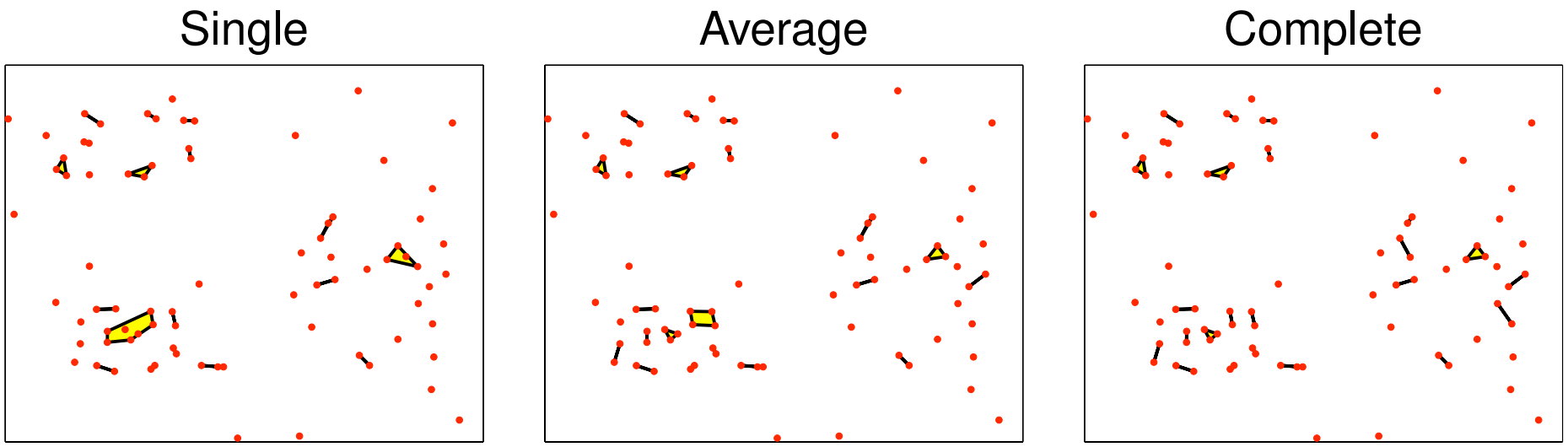
- Average distance between objects ("Group-average" agglomerative clustering):

$$\frac{1}{|C||C'|} \sum_{\mathbf{x} \in C, \mathbf{x}' \in C'} d(\mathbf{x}, \mathbf{x}')$$

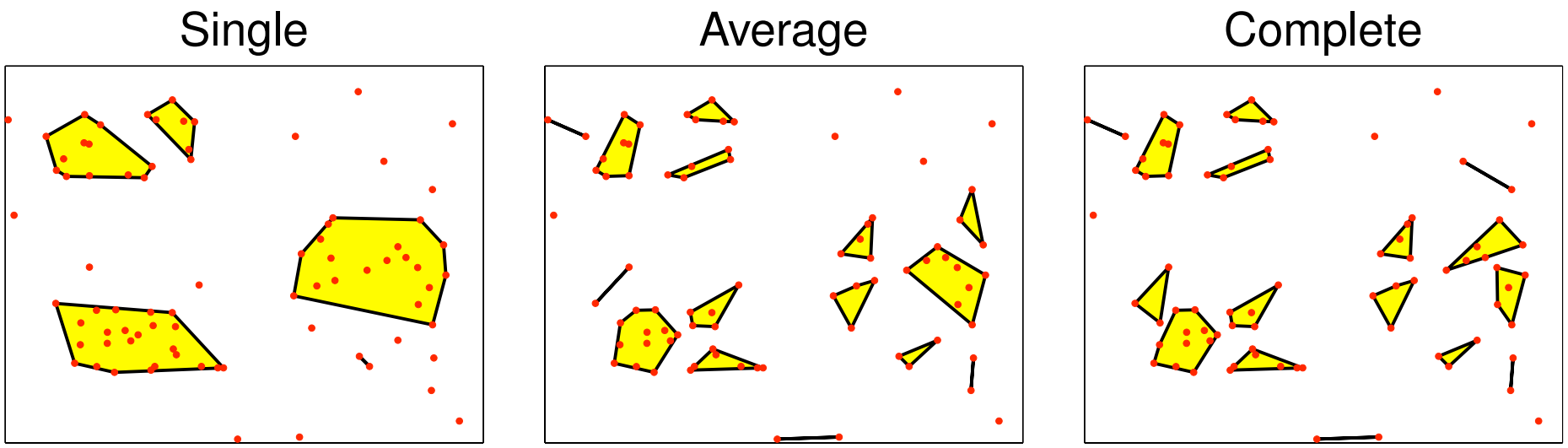
Example 1: Data



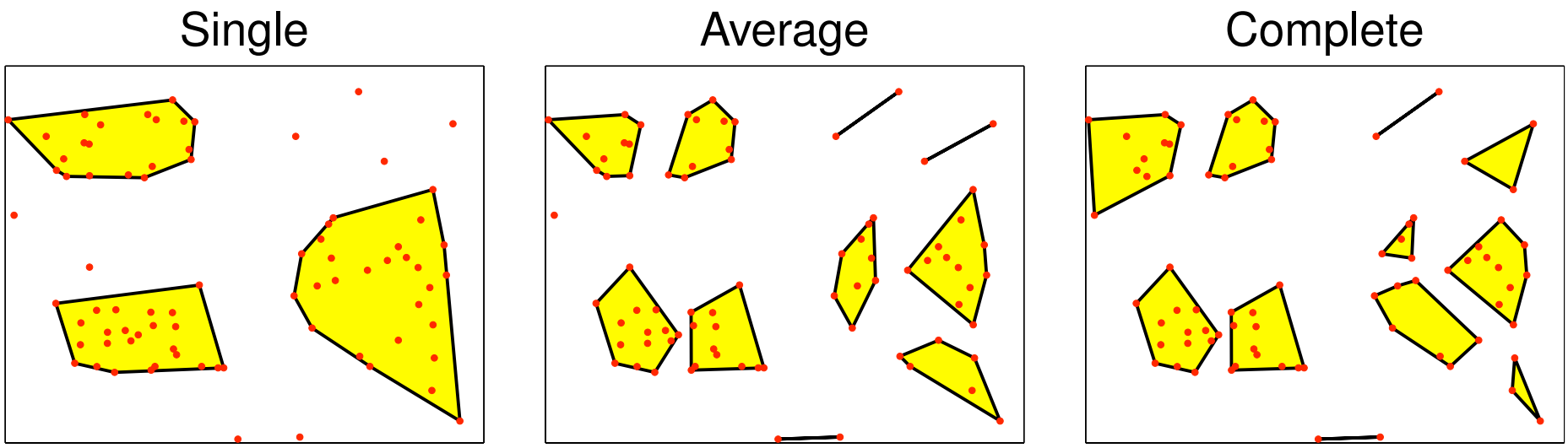
Example 1: Iteration 30



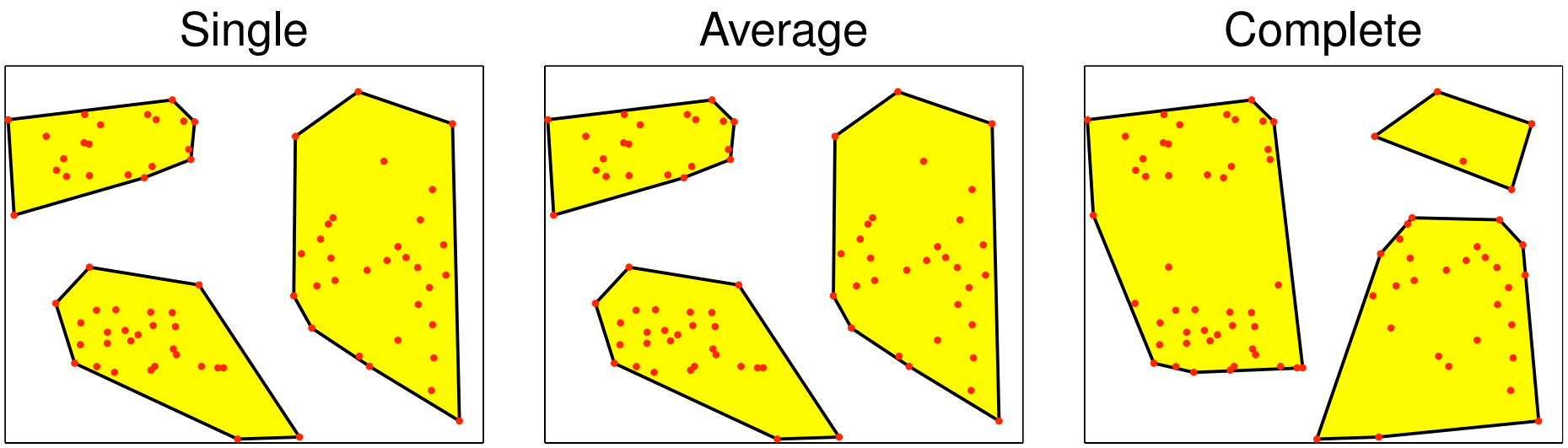
Example 1: Iteration 60



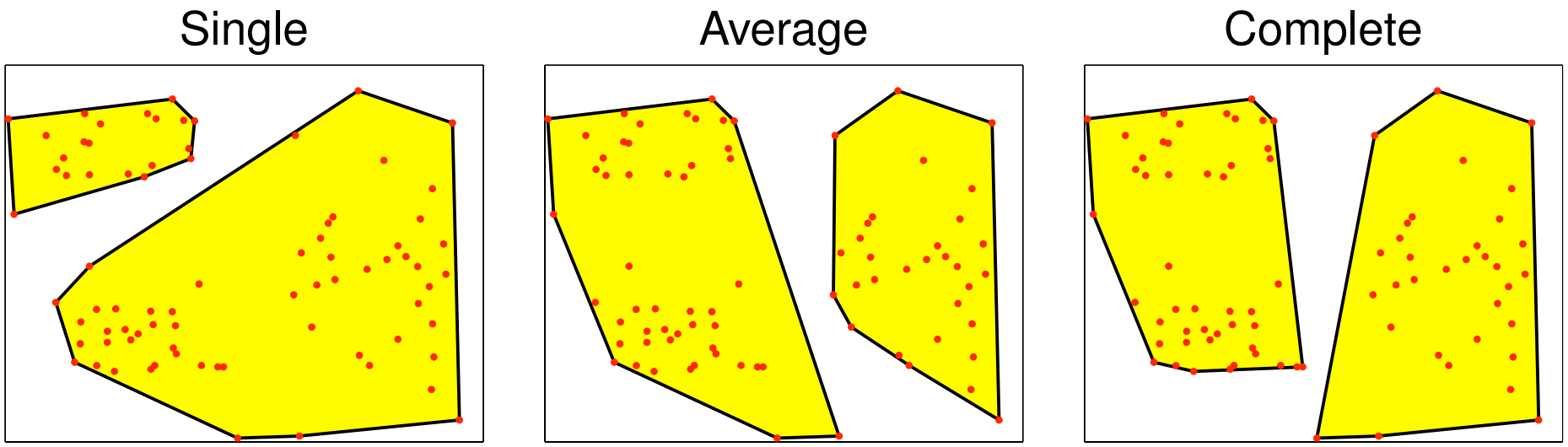
Example 1: Iteration 70



Example 1: Iteration 78

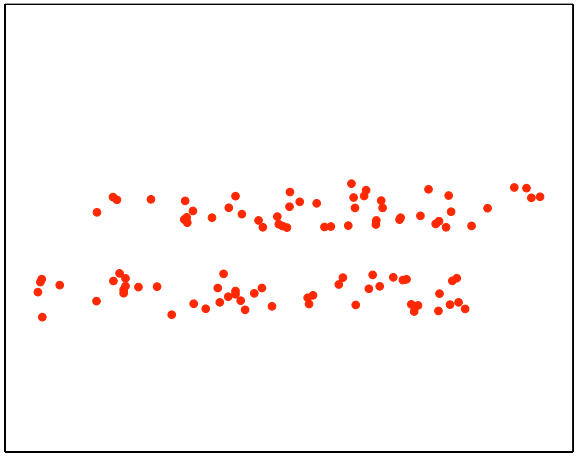


Example 1: Iteration 79

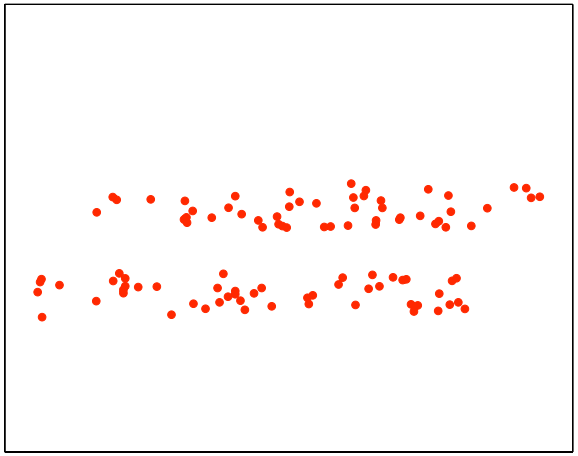


Example 2: Data

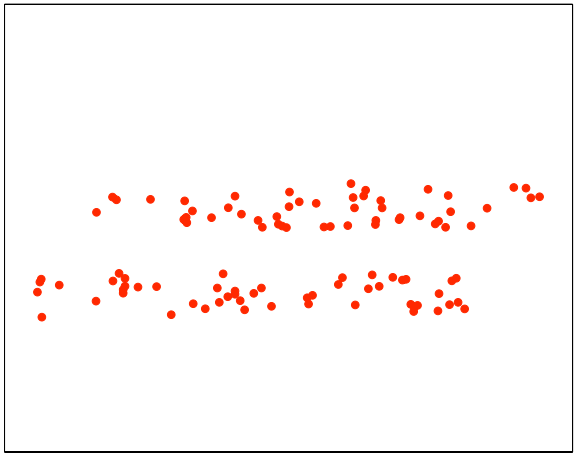
Single



Average

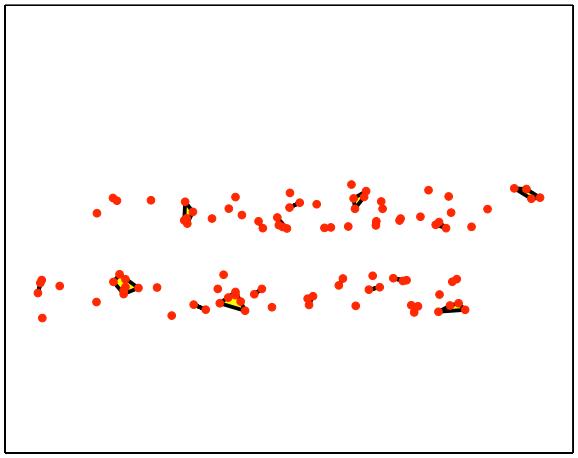


Complete

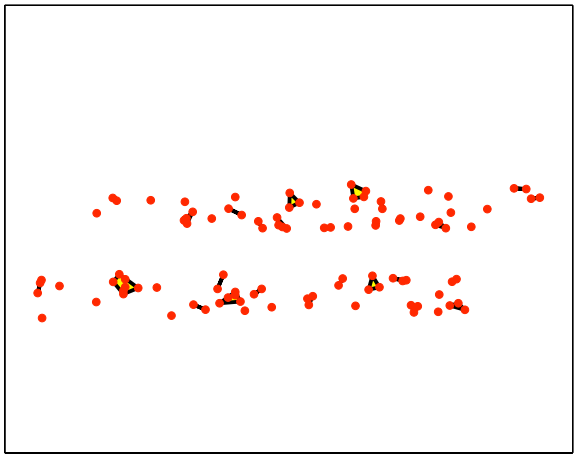


Example 2: Iteration 50

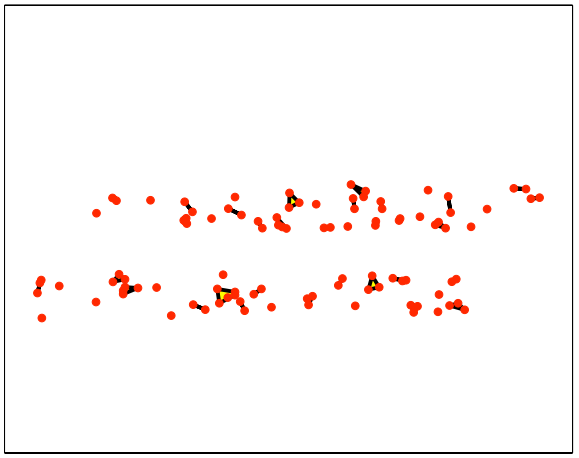
Single



Average

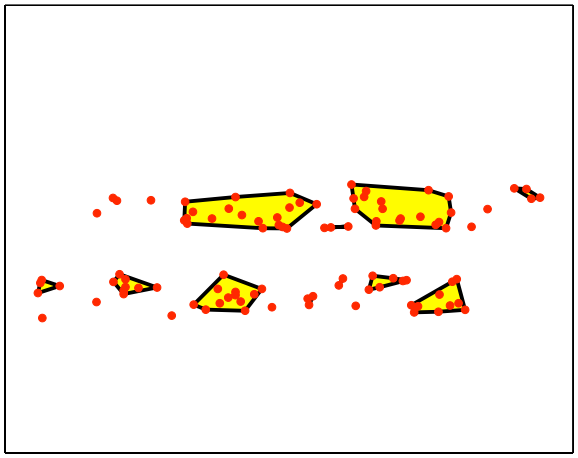


Complete

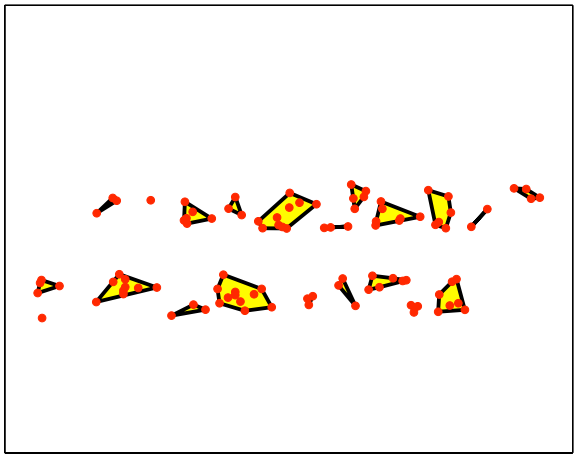


Example 2: Iteration 80

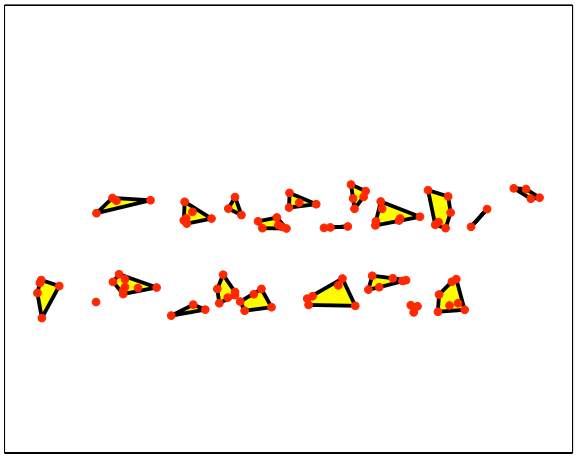
Single



Average

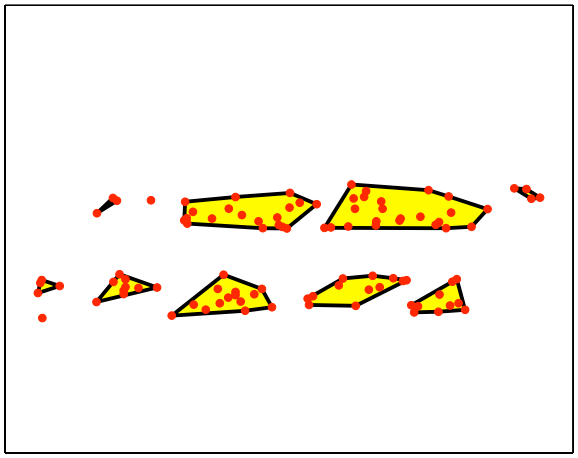


Complete

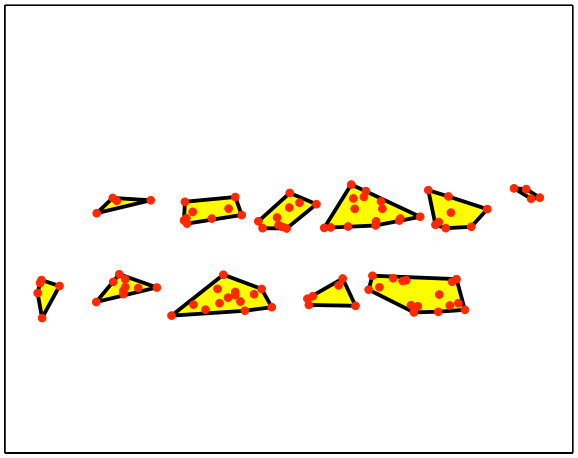


Example 2: Iteration 90

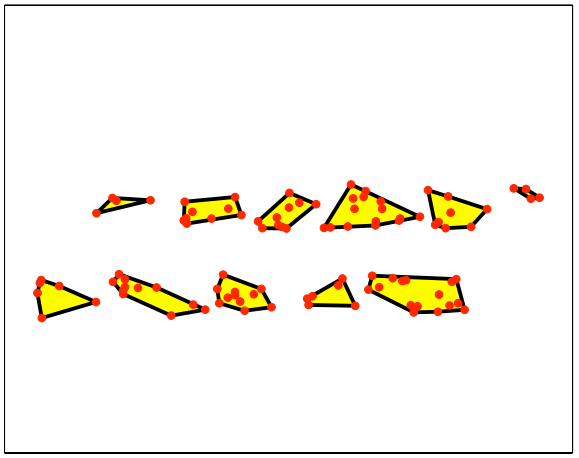
Single



Average

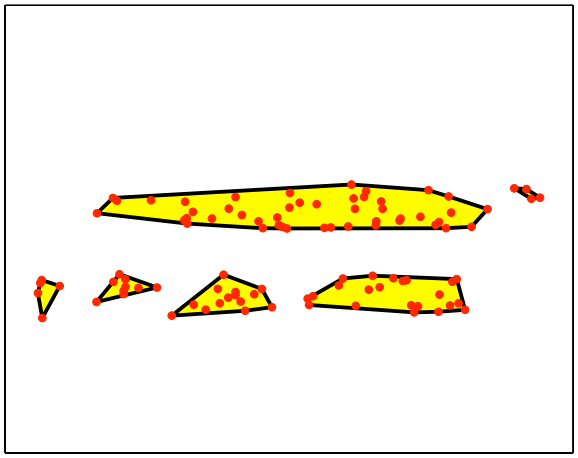


Complete

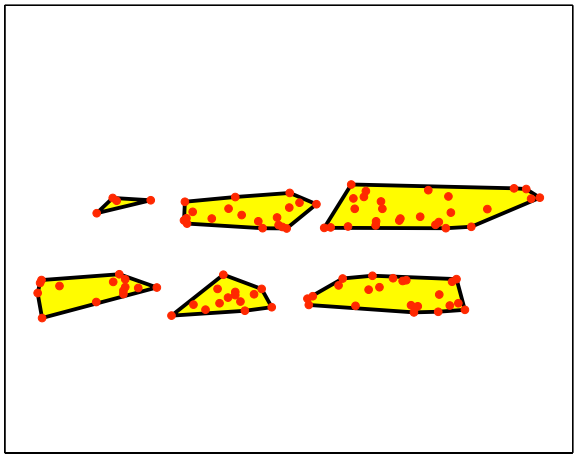


Example 2: Iteration 95

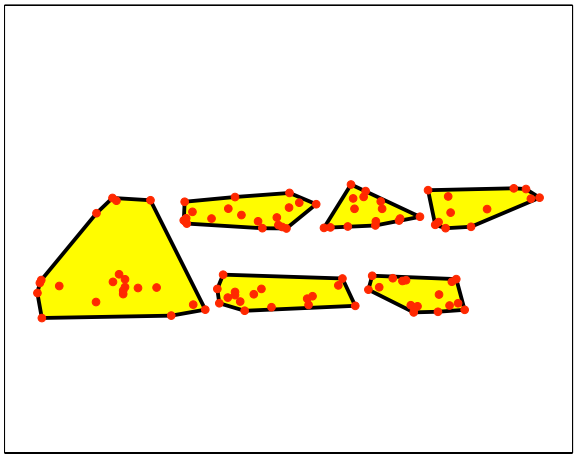
Single



Average

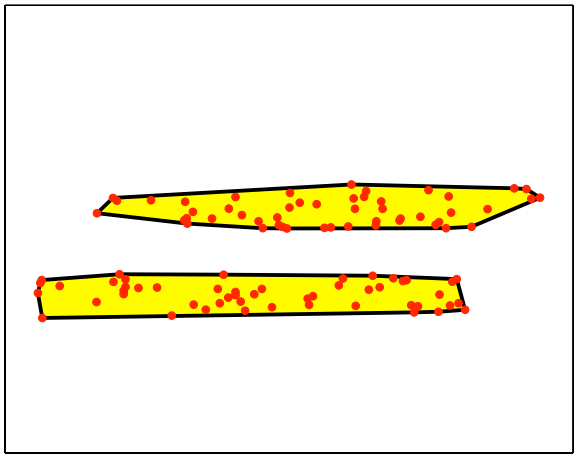


Complete

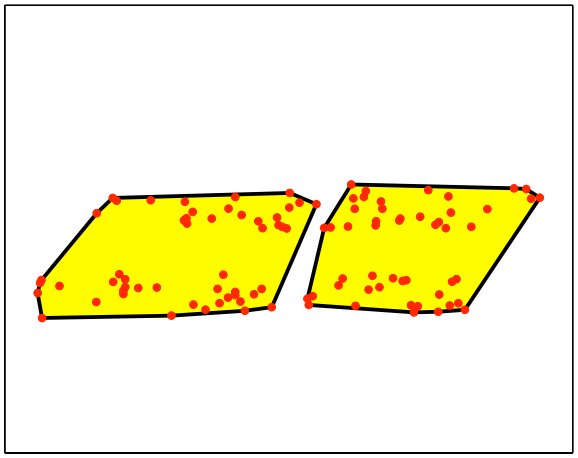


Example 2: Iteration 99

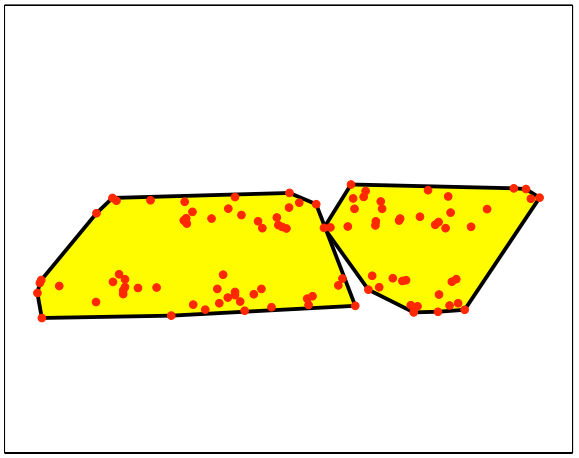
Single



Average



Complete

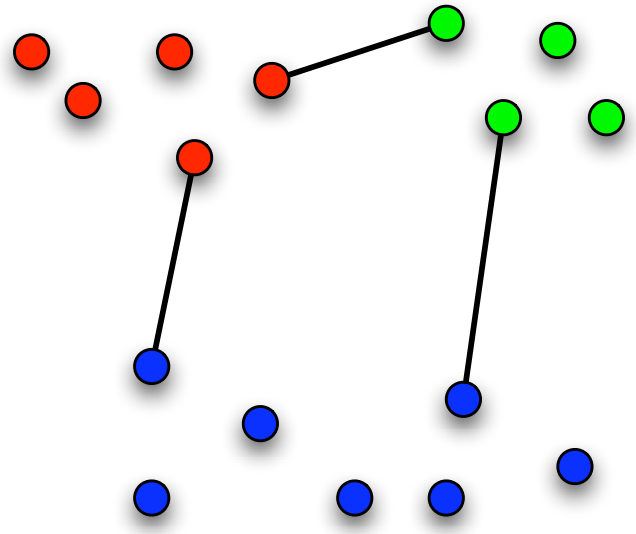


Intuitions about cluster similarity

- Single-linkage
  - Favors spatially-extended / filamentous clusters
  - Often leaves singleton clusters until near the end
- Complete-linkage favors compact clusters
- Average-linkage is somewhere in between

Monotonicity

- Let A, B, C be clusters.
- Let d be one of the dissimilarity measures: single-linkage (see below), average linkage or complete linkage
- If $d(A, B) \leq d(A, C)$ and $d(A, B) \leq d(B, C)$, then $d(A, B) \leq d(A \cup B, C)$.



Monotonicity of single-linkage criterion: Proof

- Suppose that that $d(A, B) \leq d(A, C)$ and $d(A, B) \leq d(B, C)$
- Then:

$$
\begin{aligned}
d(A \cup B, C) &= \min_{x \in A \cup B, x' \in C} d(x, x') \\
&= \min \left(\min_{x_a \in A, x' \in C} d(x_a, x'), \min_{x_b \in B, x' \in C} d(x_a, x') \right) \\
&= \min \left(d(A, C), d(B, C) \right) \\
&\geq \min \left(d(A, B), d(A, B) \right) \\
&= d(A, B)
\end{aligned}
$$

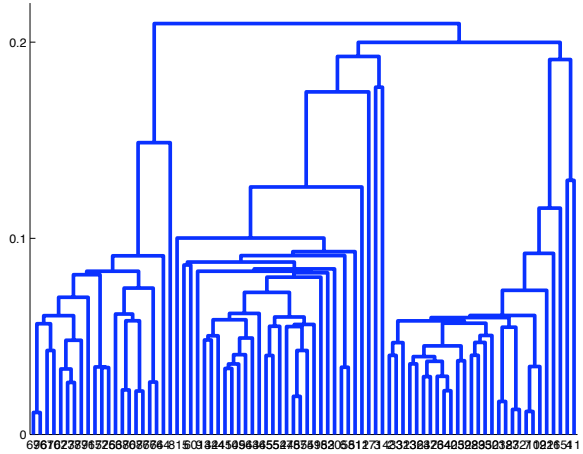
- Proofs for group-average and complete-linkage are similar.

Dendrograms

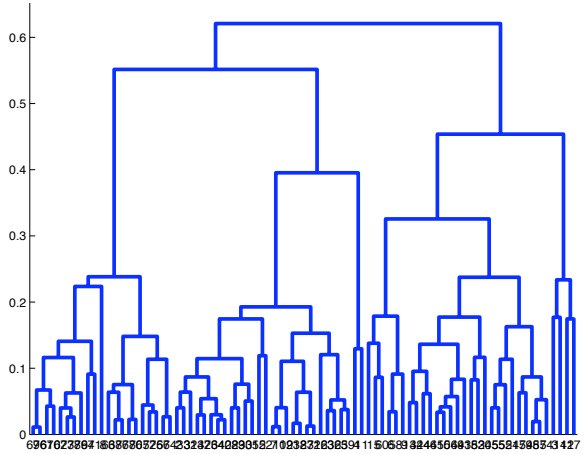
- The monotonicity property implies that every time agglomerative clustering merges two clusters, the dissimilarity of those clusters is $\geq$ the dissimilarity of all previous merges.
- Dendrograms (trees depicting hierarchical clusterings) are often drawn so that the height of a node corresponds to the dissimilarity of the merged clusters.
- We can form a flat clustering by cutting the tree at any height.
- Jumps in the height of the dendrogram can suggest natural cutoffs.

Dendrograms for Example 1

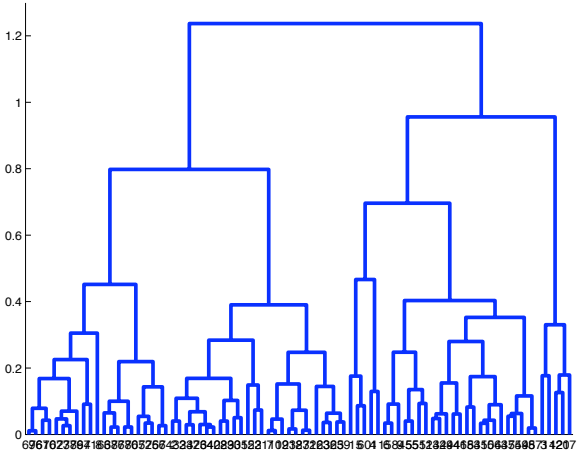
Single



Average

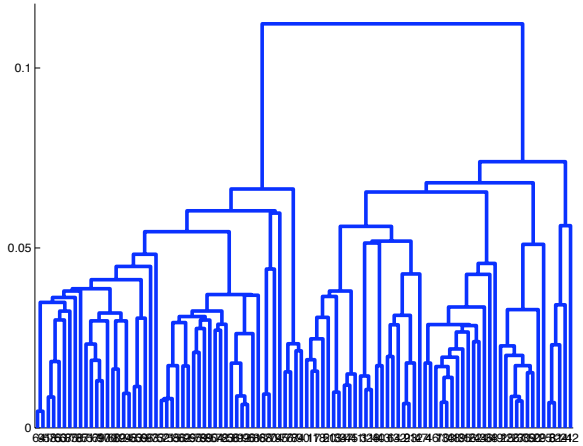


Complete

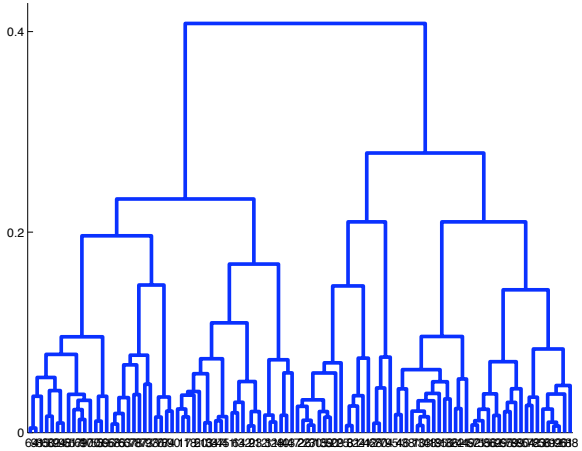


Dendrograms for Example 2

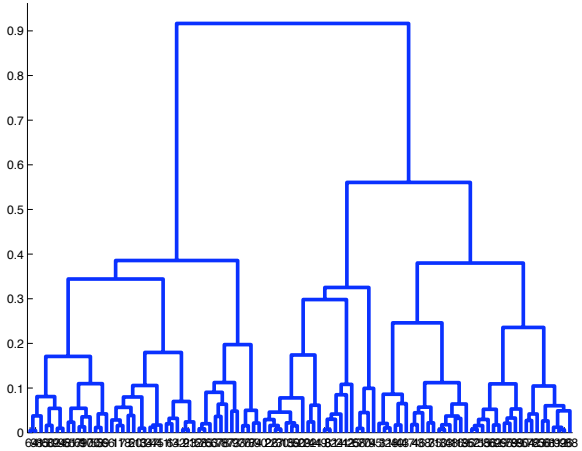
Single



Average



Complete



Divisive clustering

- Works by recursively partitioning the instances.
- How might you do that?
  - K-means?
  - Max weighted cut on graph where edges are weighted by pairwise distances?
  - Maximum margin?
- Many heuristics for partitioning the instances have been proposed . . . but many are computationally hard and/or violate monotonicity, making it hard to draw dendrograms.

Summary of clustering

- K-means
  - Fast way of partitioning data into K clusters
  - It minimizes the sum of squared Euclidean distances to the clusters centroids
  - Different clusterings can result from different initializations
  - Can be interpreted as fitting a mixture distribution
- Hierarchical clustering
  - Organizes data objects into a tree based on similarity.
  - Agglomerative (bottom-up) tree construction is most popular.
  - There are several choices of distance metric (linkage criterion)
  - Monotonicity allows us to draw dendrograms in which the height of a node corresponds to the dissimilarity of the clusters merged.
  - Trees can be cut off at some level, to generate a flat partitioning of the data.

A different view of clustering

- Suppose you had a classification data set, with data coming from K classes
- But someone erased all the class labels!
- You would like to know to what class each example belongs
- This is exactly the problem solved by K-means!
- And as we saw, it gives a maximum likelihood solution, but under a very restrictive assumption
- Now we formalize this problem more generally

More generally: Missing values / semi-supervised learning

- Suppose we have a generative model of supervised learning data with parameters θ, but the values y are missing in some of the instances
- The likelihood of the data can be written as:

$$\log L(\theta) = \sum_{\text{complete data}} \log P(\mathbf{x}_i, y_i|\theta) + \sum_{\text{incomplete data}} \log P(\mathbf{x}_i|\theta)$$

- For the second term, we must consider *all possible values* for y:

$$\sum_{\text{incomplete data}} \log P(\mathbf{x}_i|\theta) = \sum_{\text{incomplete data}} \log \left(\sum_y P(\mathbf{x}_i, y|\theta) \right)$$

Data likelihood with missing values

- Likelihood of the data:

$$\log L(\theta) = \sum_{\text{complete data}} \log P(\mathbf{x}_i, y_i|\theta) + \sum_{\text{incomplete data}} \log \left(\sum_y P(\mathbf{x}_i, y|\theta) \right)$$

- In semi-supervised learning, some y's are observed some not. Even so, using the $\mathbf{x}$'s with unobserved y's can be helpful.
- In the clustering problem, y is *never* observed, so we only have the second term above.

Expectation Maximization (EM)

- A general purpose method for learning from incomplete data
- Main idea:
  - If we had complete data we could easily maximize the likelihood
  - But because the data is incomplete, we get a summation inside the $\log$, which makes the optimization much harder
  - So in the case of missing values, we will "fantasize" what they should be, based on the current parameter setting
  - In other words, we *fill in the missing values based on our current expectation*
  - Then we *compute new parameters, which maximize the likelihood* of the completed data
  - In terms of the previous slide, we estimate y given θ, then we reestimate θ given y, then we reestimate y given the new θ, ...

Recall: Gaussian discriminant analysis

- Inputs $\mathbf{x}_i$ are real-valued vectors, each with a class $y_i \in \{1, 2, \ldots, K\}$. (We saw $y_i \in \{0, 1\}$, previously.)
- The parameters of the model are:
  - The prior probabilities, $P(y = k)$.
  - Mean and covariance matrix, μ_k, Σ_k, defining a multivariate Gaussian distribution for examples in class k.
- All parameters chosen to maximize the log likelihood of the data:

$$\log L = \sum_i (\log P(y_i) + \log P(\mathbf{x}_i | y_i))$$

Maximum likelihood solution

- Let $\delta_{ik} = 1$ if $y_i = k$ and 0 otherwise
- The class probabilities are determined by the empirical frequency of examples in each class:

$$P(y = k) = p_k = \frac{\sum_i \delta_{ik}}{\sum_k \sum_i \delta_{ik}}$$

- The mean and covariance matrix for class k are the empirical mean and covariance of the examples in that class:

$$\begin{aligned} \mu_k &= \frac{\sum_i \delta_{ik} \mathbf{x}_i}{\sum_i \delta_{ik}} \\ \Sigma_k &= \frac{\sum_i \delta_{ik} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T}{\sum_i \delta_{ik}} \end{aligned}$$

EM for Mixture of Gaussians

- We start with an initial guess for the parameters P_k, μ_k, Σ_k
- We will alternate an:
  - *expectation step (E-step)*, in which we "complete" the data—estimating the y_i
  - *maximization step (M-step)*, in which we re-compute the parameters P_k, μ_k, Σ_k
- In the *hard EM* version, completing the data means that each data point is assumed to be generated by *exactly one Gaussian*—taken to be the most likely assignment.
  (This is roughly equivalent to the setting of K-means clustering.)
- In the *soft EM* version (also usually known as EM), we assume that each data point could have been generated from *any component*
  - We estimate probabilities $P(y_i = k) = P(\delta_{ik} = 1) = E(\delta_{ik})$
  - Each $\mathbf{x}_i$ contributes to the mean and variance estimate of each component.

Hard EM for Mixture of Gaussians

1. Guess initial parameters p_k, μ_k, Σ_k for each class k
2. Repeat until convergence:
   (a) *E-step:* For each instance i and class j, assign each instance to most likely class:

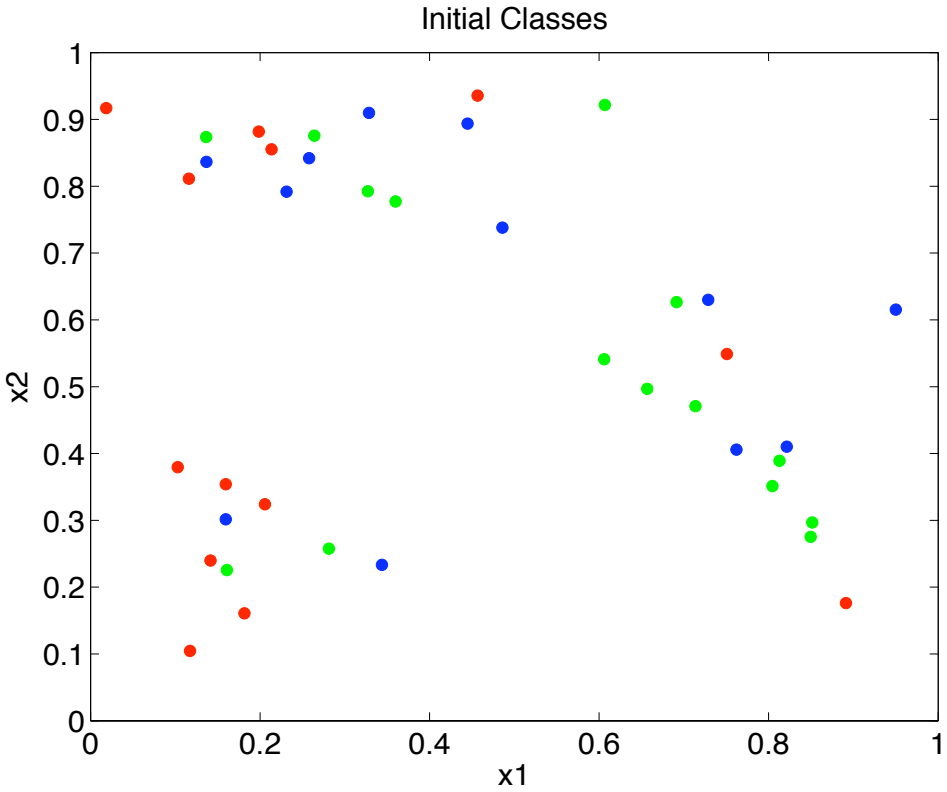
$$y_i = \arg\max_k P(y_i = k|\mathbf{x}_i) = \frac{P(\mathbf{x}_i|y_i)P(y_i)}{P(\mathbf{x}_i)}$$

   (b) *M-step:* Update the parameters of the model to maximize the likelihood of the data

$$p_j = \frac{1}{m}\sum_{i=1}^{m}\delta_{ij} \qquad \mu_j = \frac{\sum_{i=1}^{m}\delta_{ij}\mathbf{x}_i}{\sum_{i=1}^{m}\delta_{ij}}$$

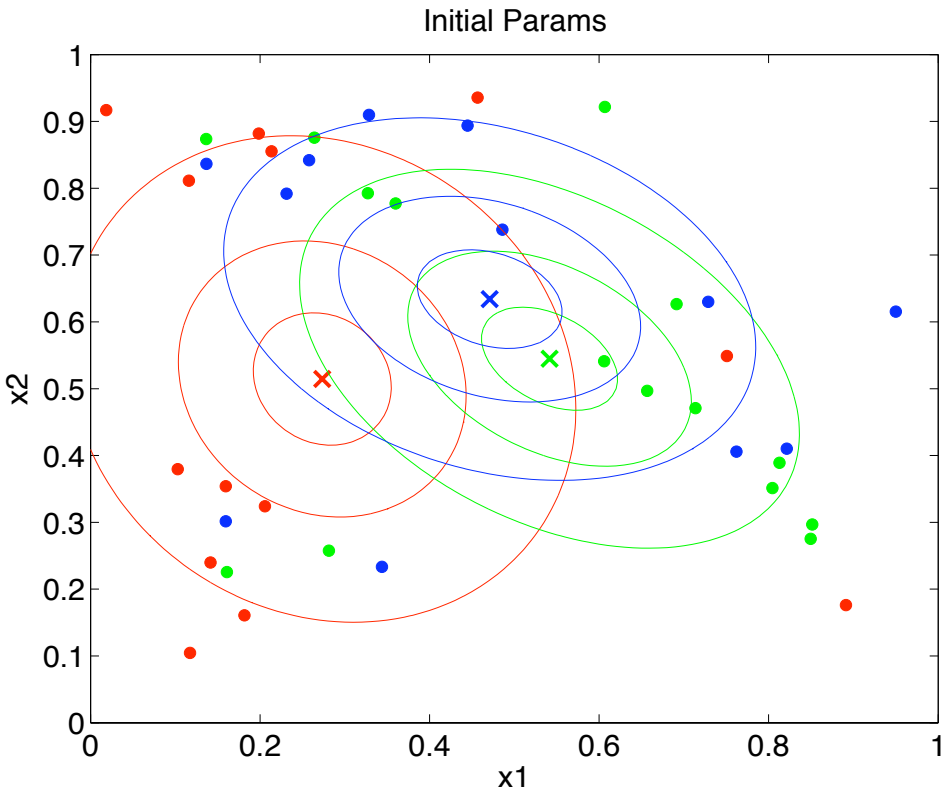
$$\Sigma_j = \frac{\sum_{i=1}^{m}\delta_{ij}\left(\mathbf{x}_i - \mu_j\right)\left(\mathbf{x}_i - \mu_j\right)^T}{\sum_{i=1}^{m}\delta_{ij}}$$

Hard EM for Mixture of Gaussians: Example



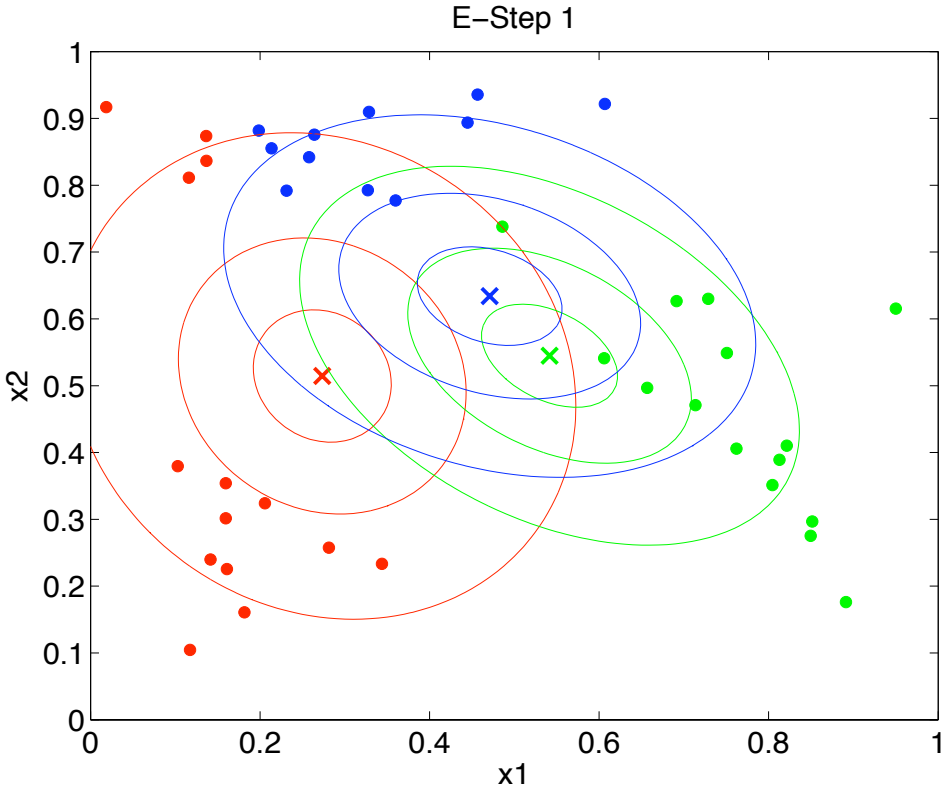
$K = 3$, initial assignment of points to components is random

Hard EM for Mixture of Gaussians: Example

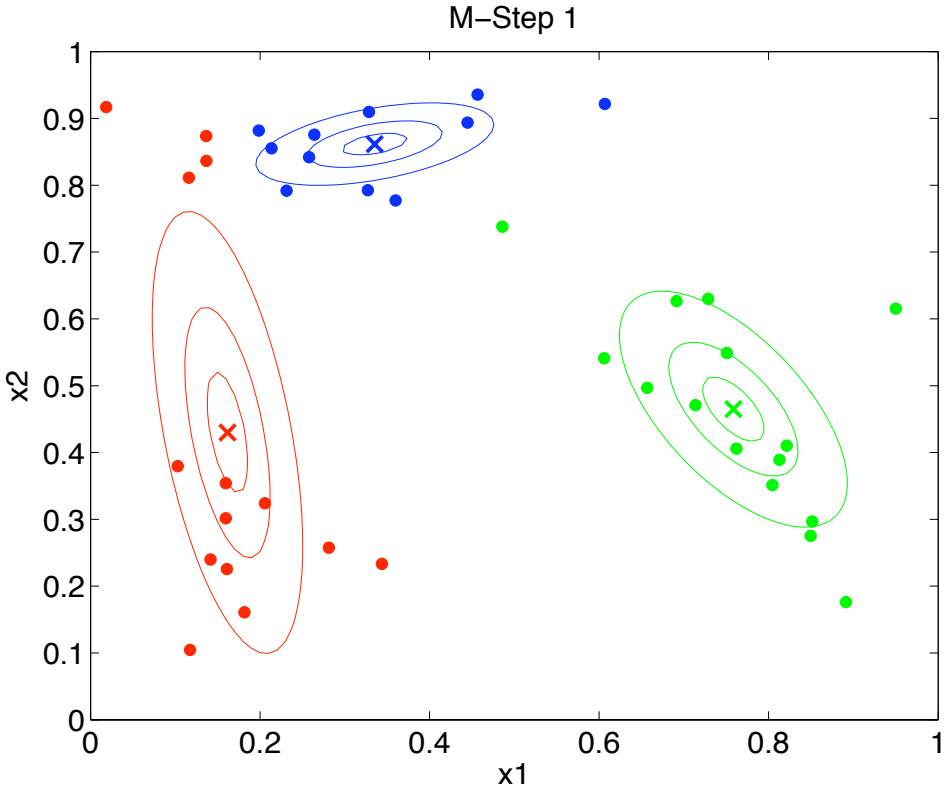


Initial parameters (means and variances) computed from initial assignments

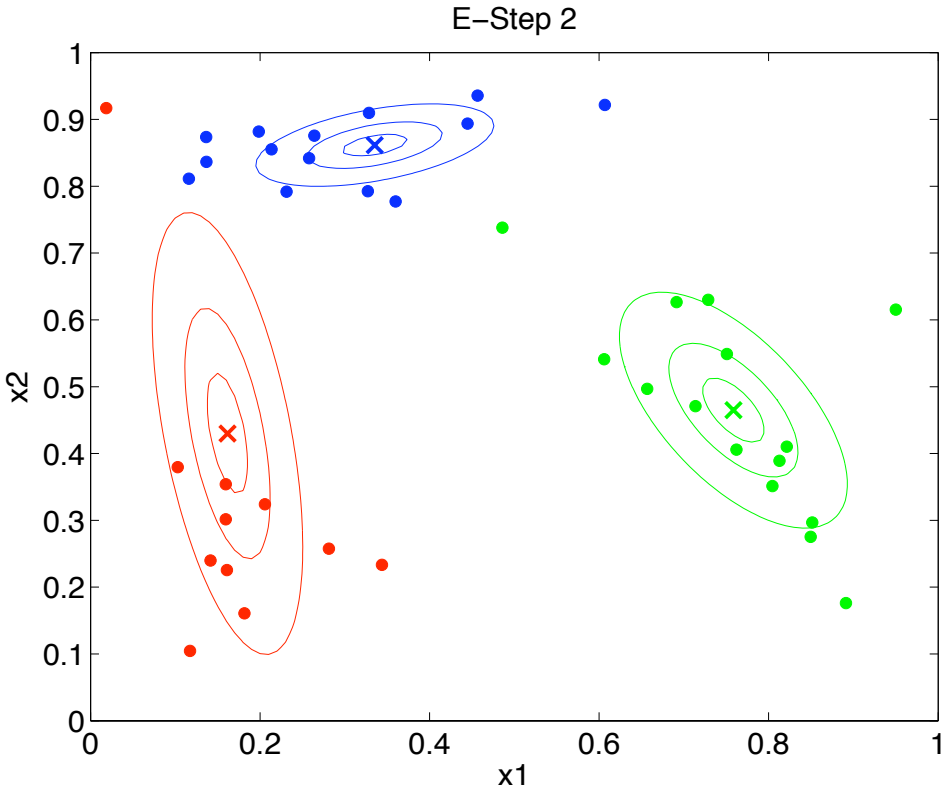
Hard EM for Mixture of Gaussians: Example



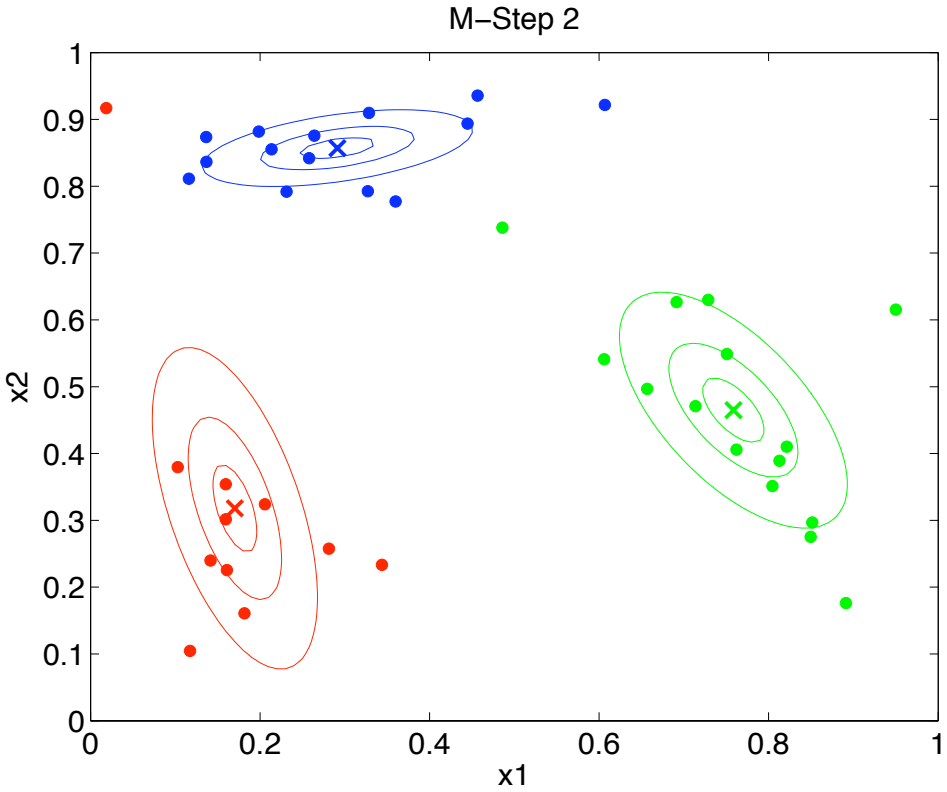
Hard EM for Mixture of Gaussians: Example



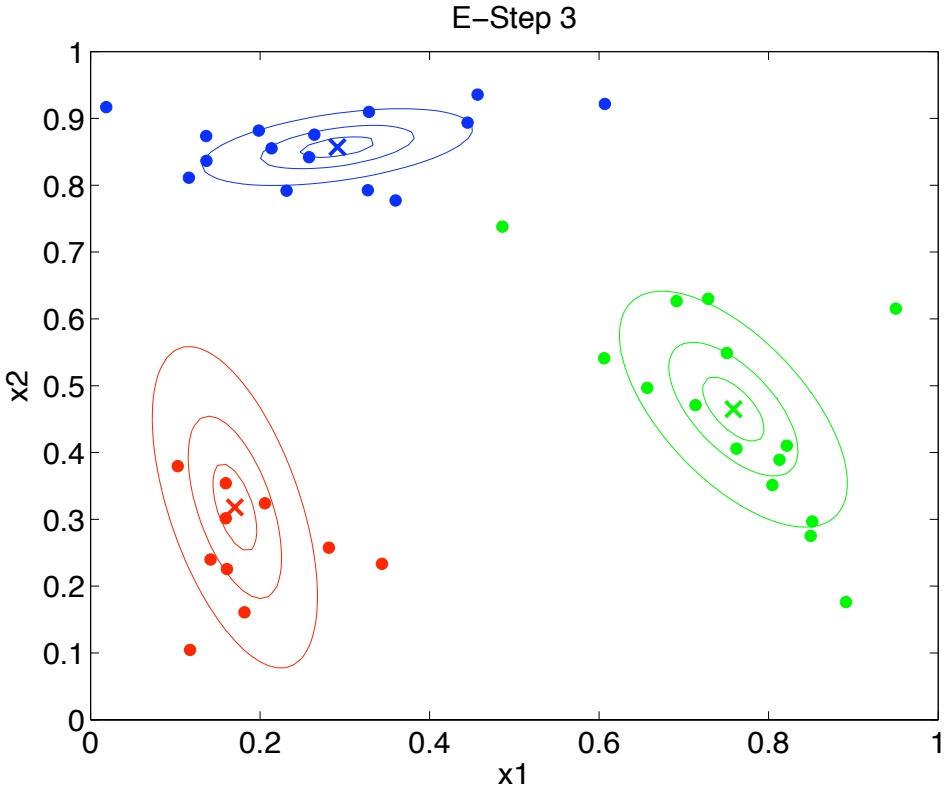
Hard EM for Mixture of Gaussians: Example



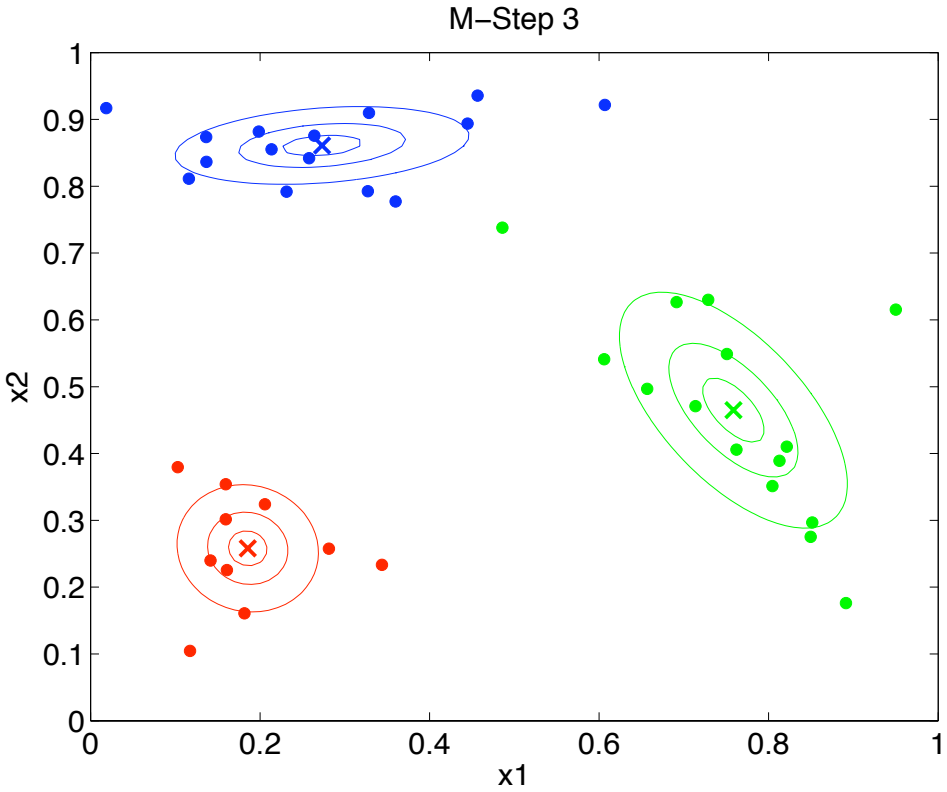
Hard EM for Mixture of Gaussians: Example



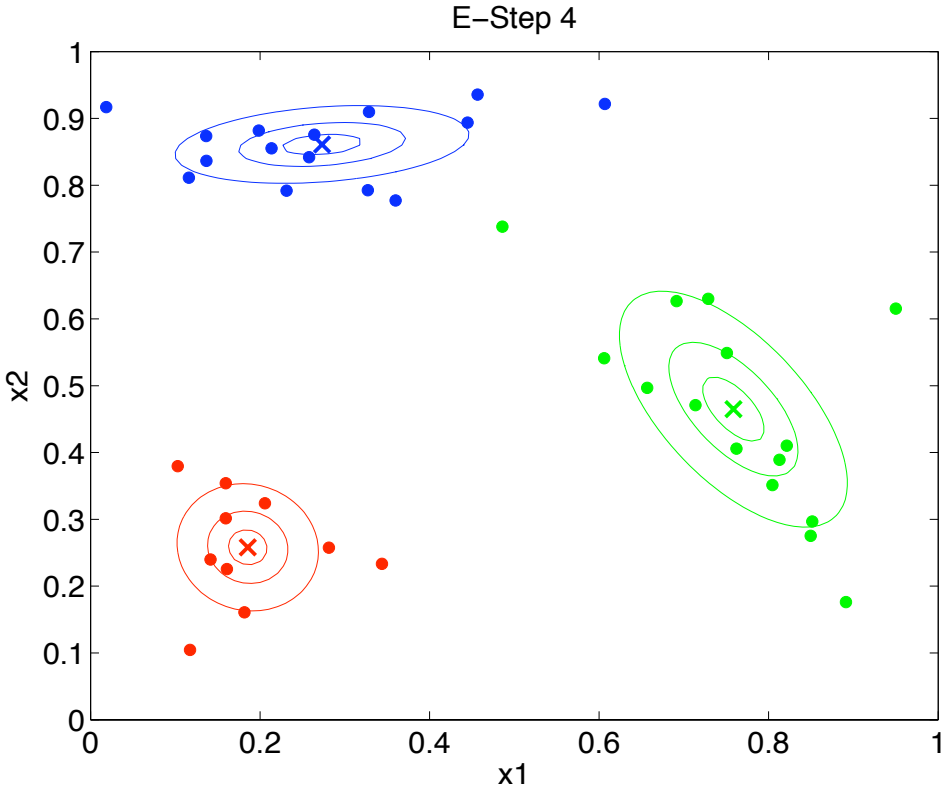
Hard EM for Mixture of Gaussians: Example



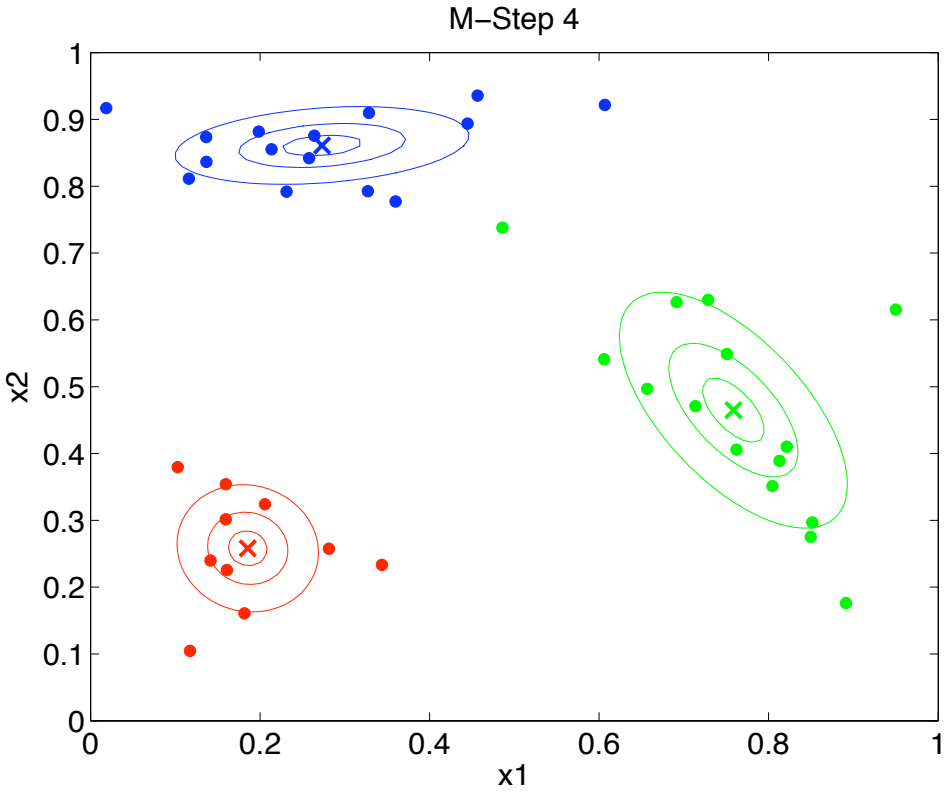
Hard EM for Mixture of Gaussians: Example



Hard EM for Mixture of Gaussians: Example



Hard EM for Mixture of Gaussians: Example



Soft EM for Mixture of Gaussians

1. Guess initial parameters p_k, μ_k, Σ_k for each class k
2. Repeat until convergence:
   (a) *E-step:* For each instance i and class j, compute the probabilities of class membership:

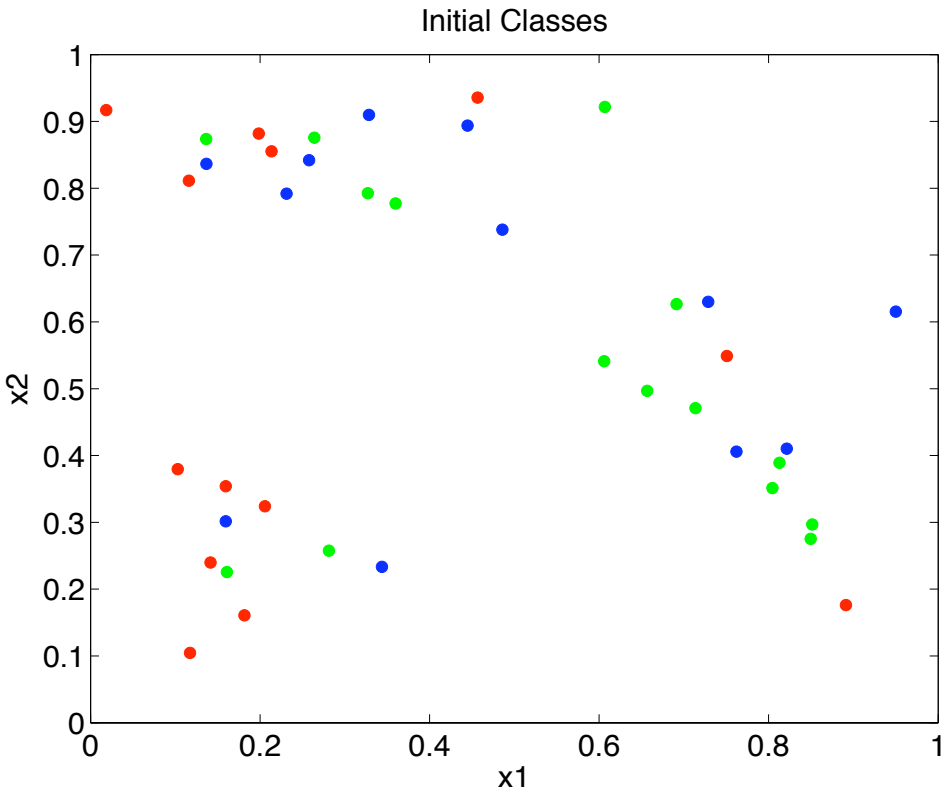
$$w_{ij} = P(y_i = j|\mathbf{x}_i) = E(\delta_{ij}) = \frac{P(\mathbf{x}_i|y_i = j)P(y_i = j)}{P(\mathbf{x}_i)}$$

   (b) *M-step:* Update the parameters of the model to maximize the likelihood of the data

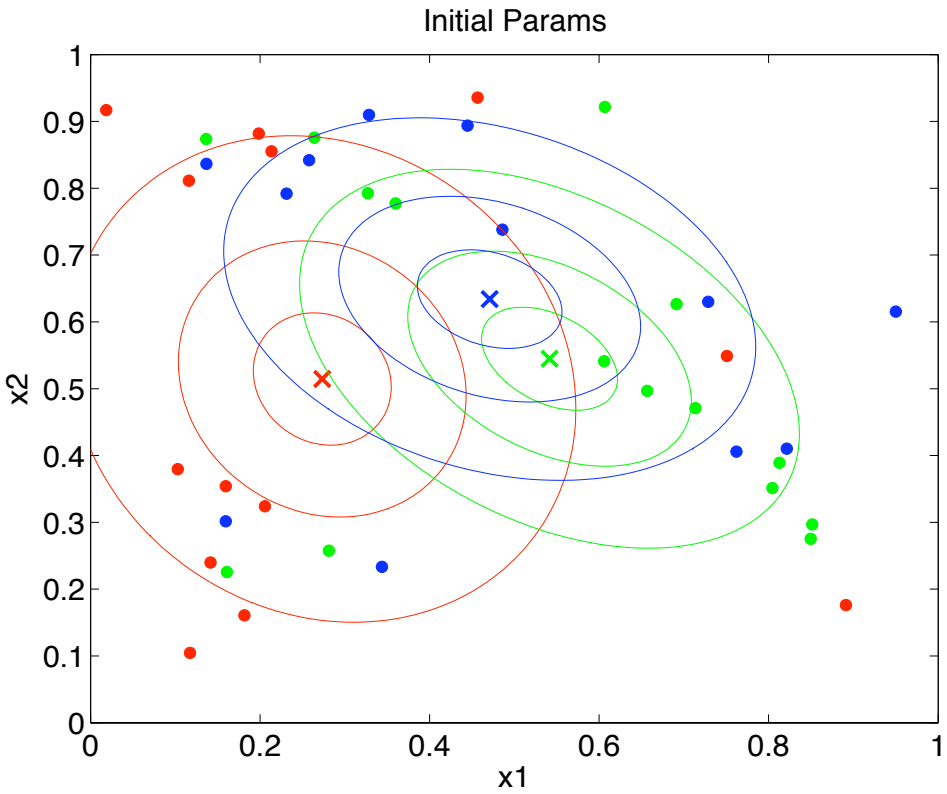
$$p_j = \frac{1}{m}\sum_{i=1}^{m} w_{ij} \qquad \mu_j = \frac{\sum_{i=1}^{m} w_{ij}\mathbf{x}_i}{\sum_{i=1}^{m} w_{ij}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} w_{ij}\left(\mathbf{x}_i - \mu_j\right)\left(\mathbf{x}_i - \mu_j\right)^T}{\sum_{i=1}^{m} w_{ij}}$$

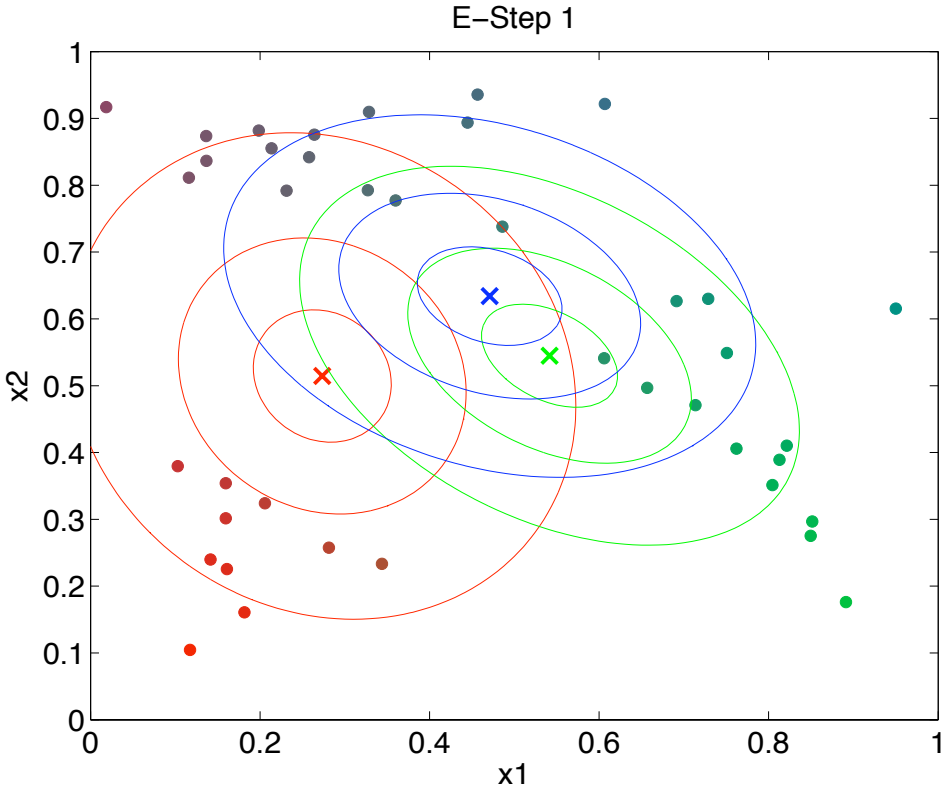
Soft EM for Mixture of Gaussians: Example



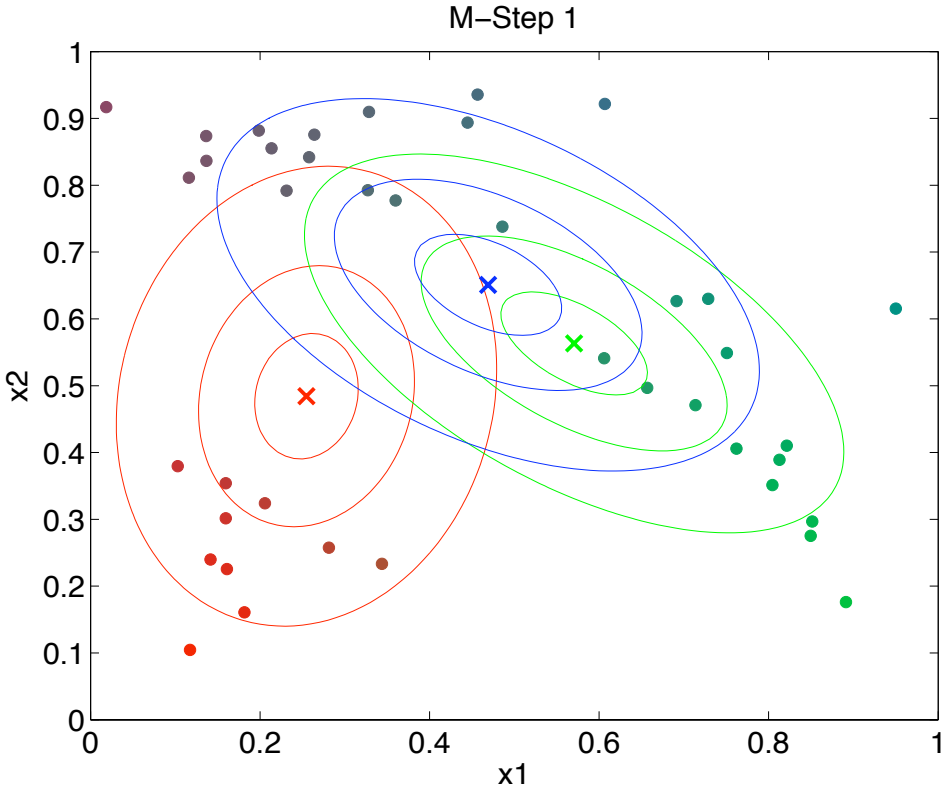
Soft EM for Mixture of Gaussians: Example



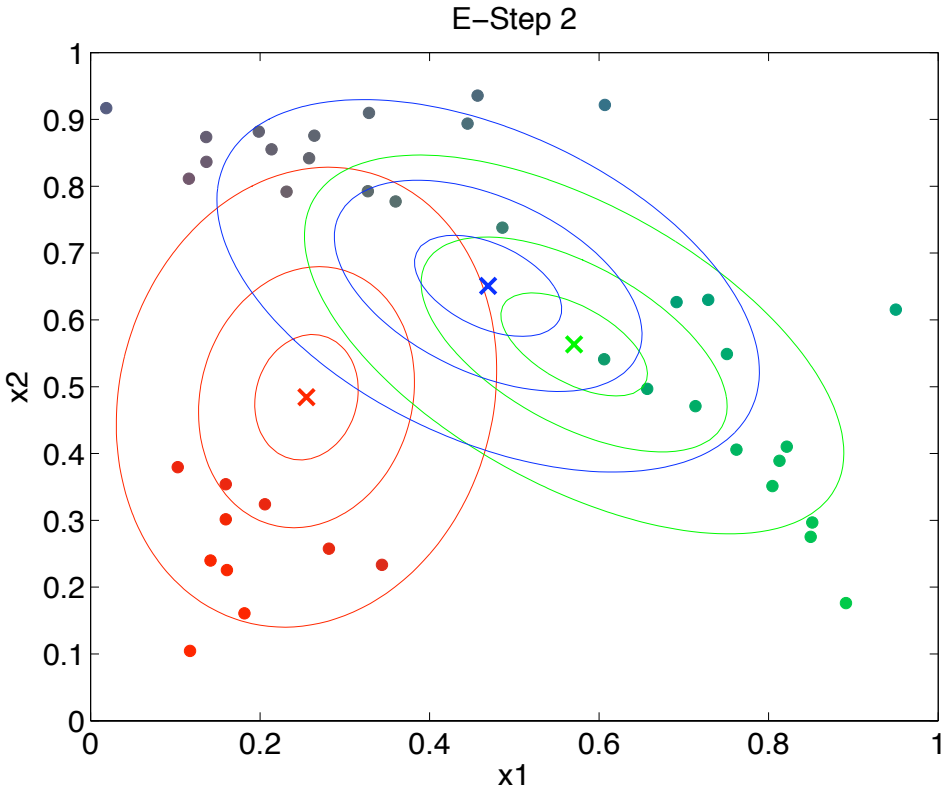
Soft EM for Mixture of Gaussians: Example



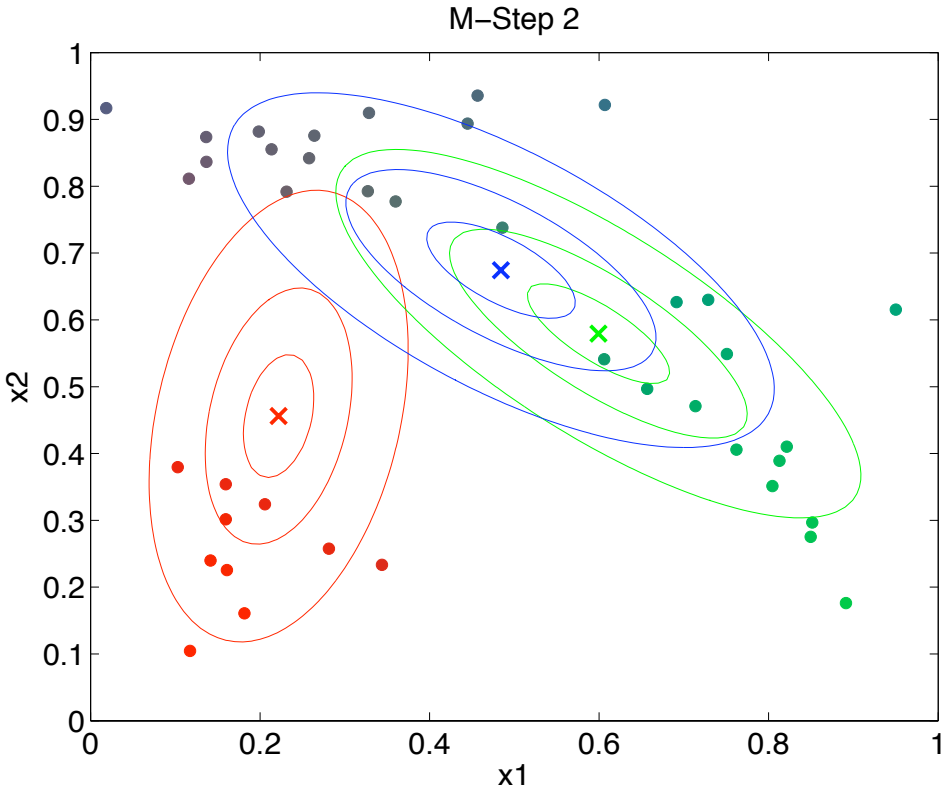
Soft EM for Mixture of Gaussians: Example



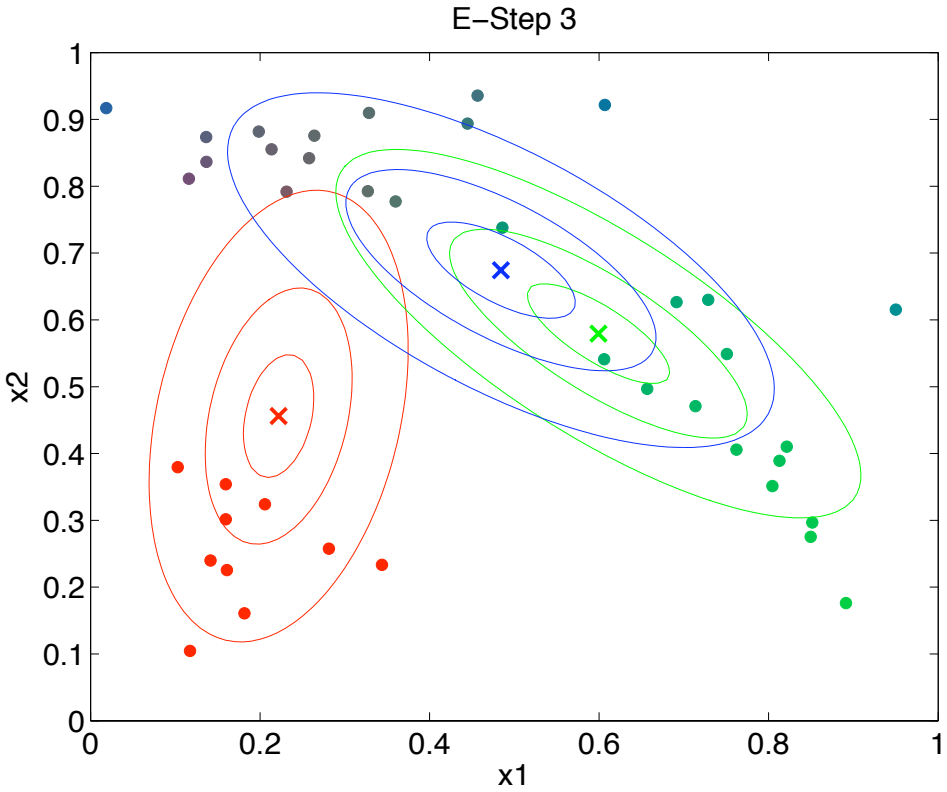
Soft EM for Mixture of Gaussians: Example



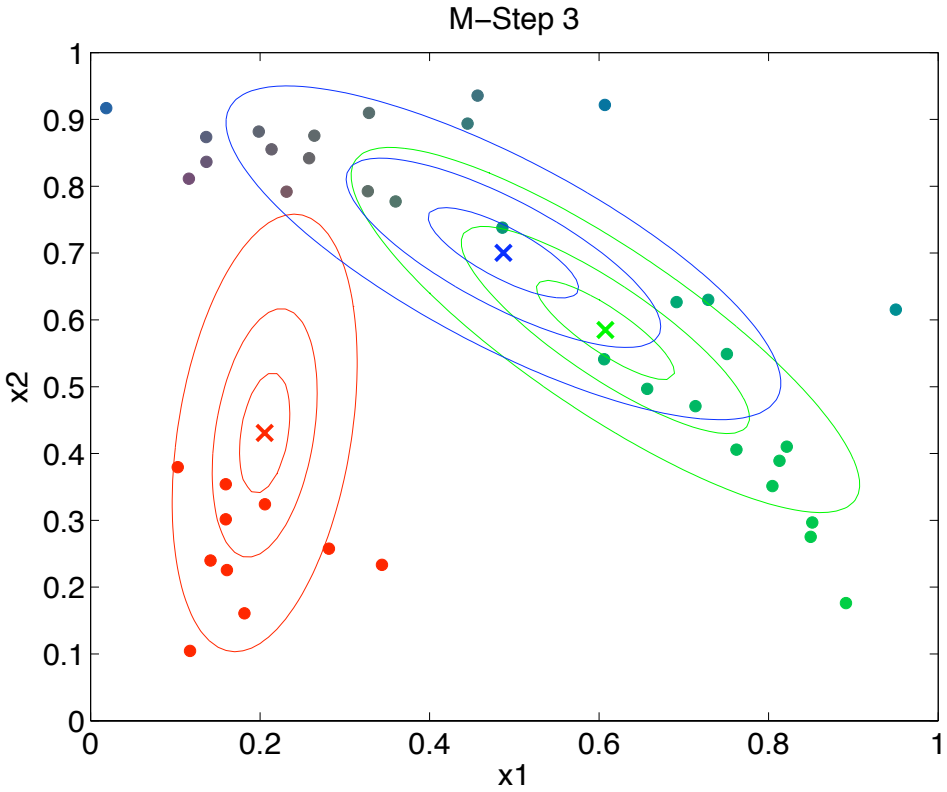
Soft EM for Mixture of Gaussians: Example



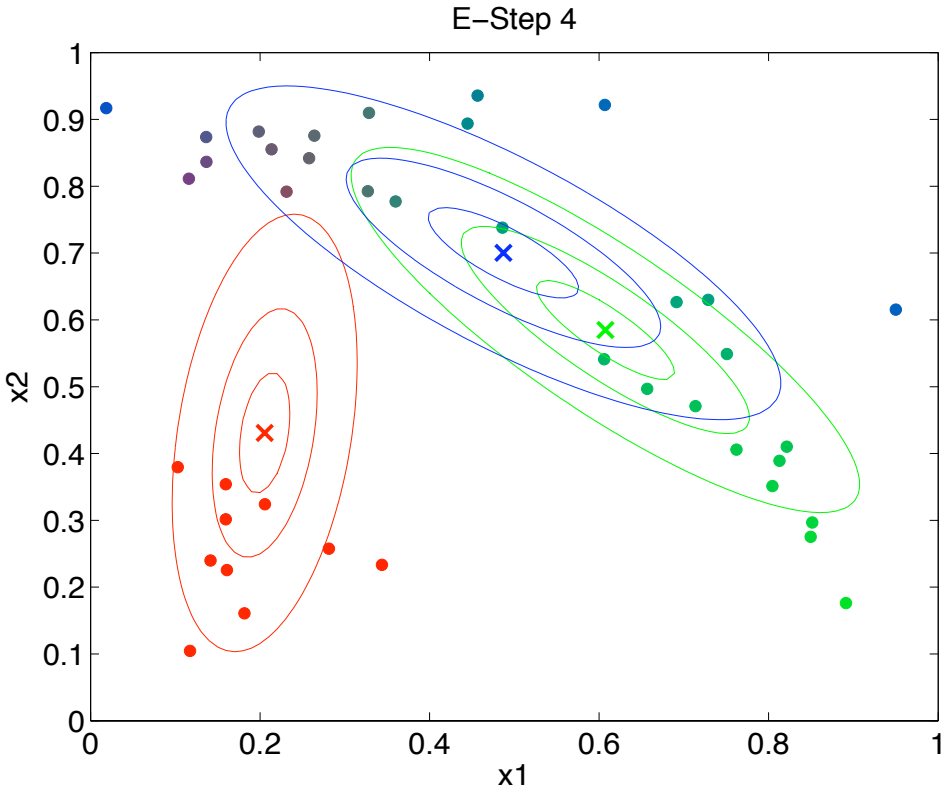
Soft EM for Mixture of Gaussians: Example



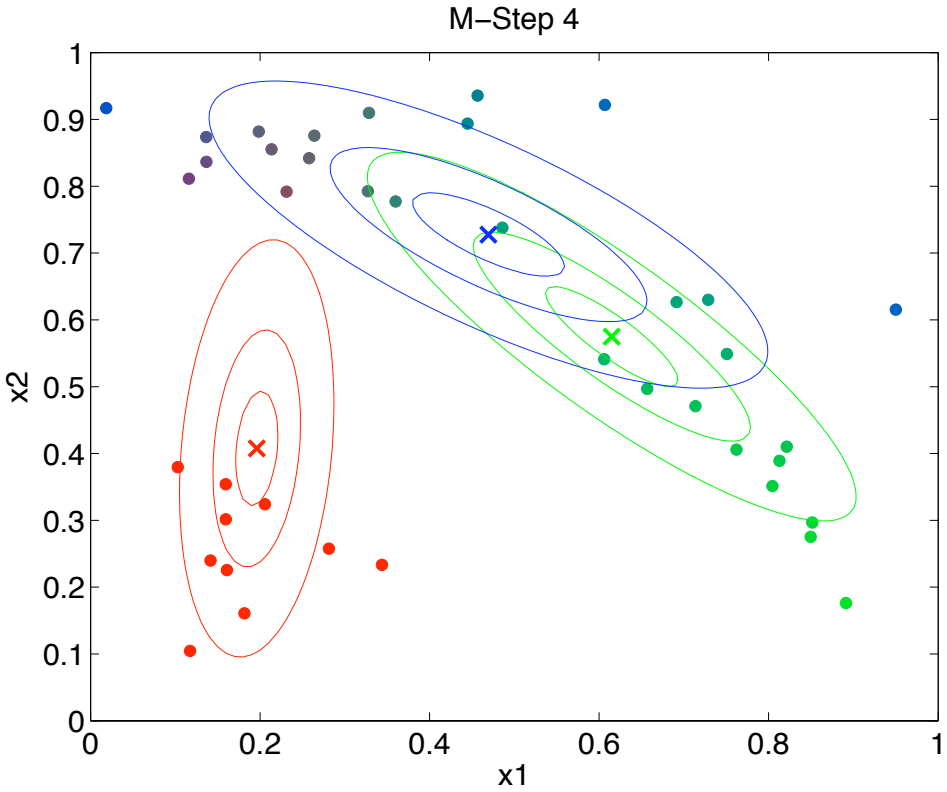
Soft EM for Mixture of Gaussians: Example



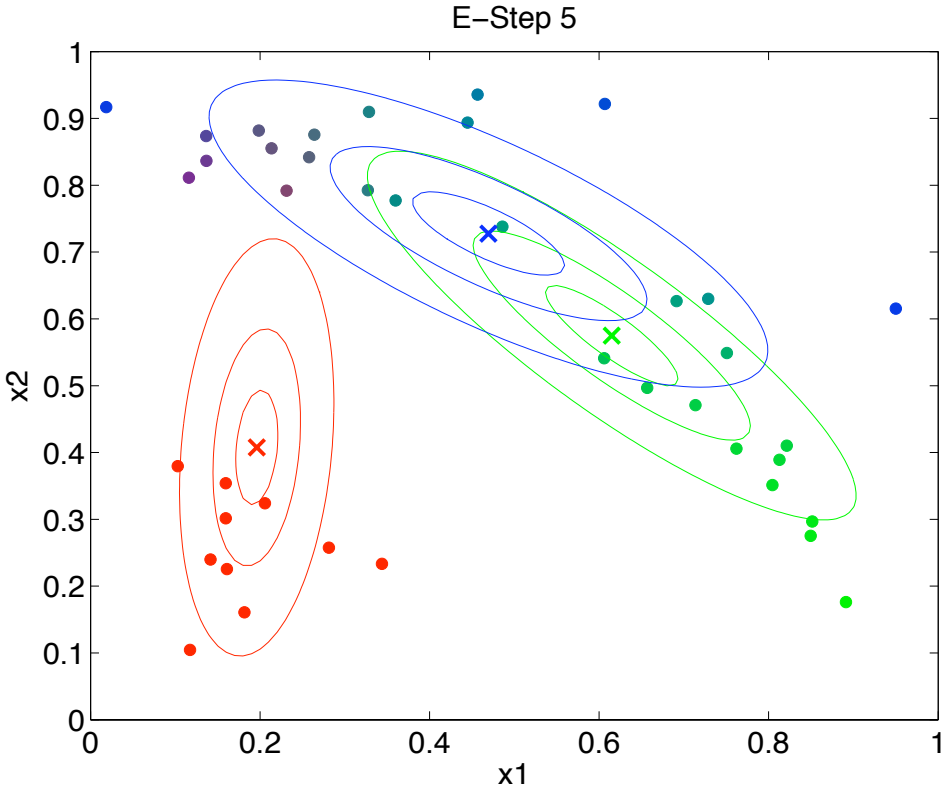
Soft EM for Mixture of Gaussians: Example



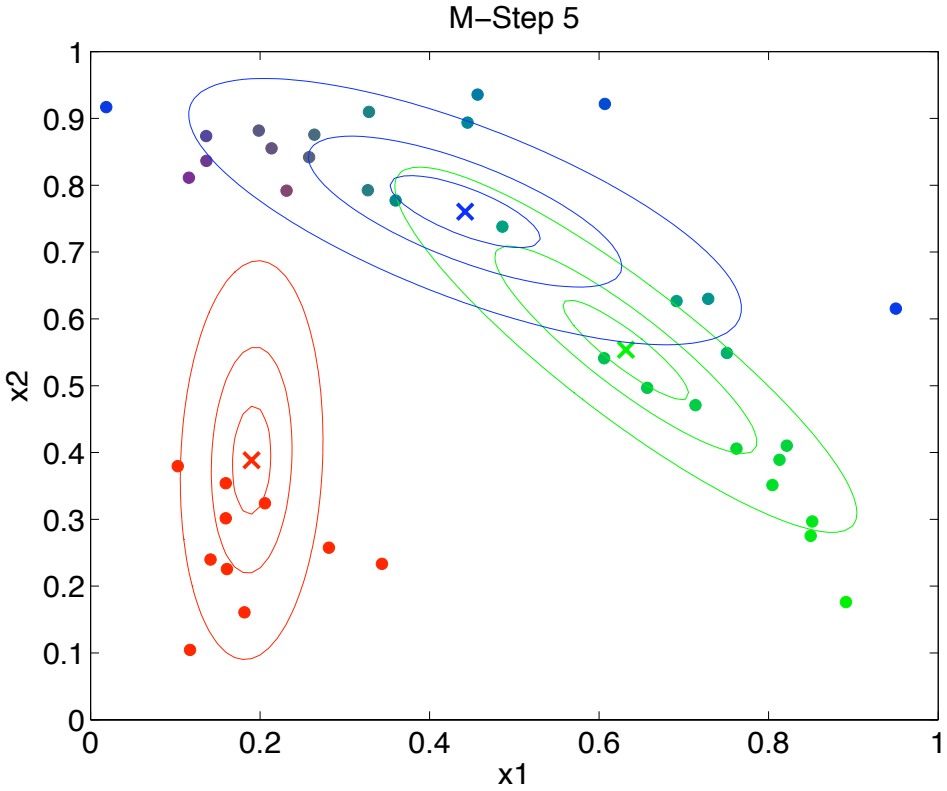
Soft EM for Mixture of Gaussians: Example



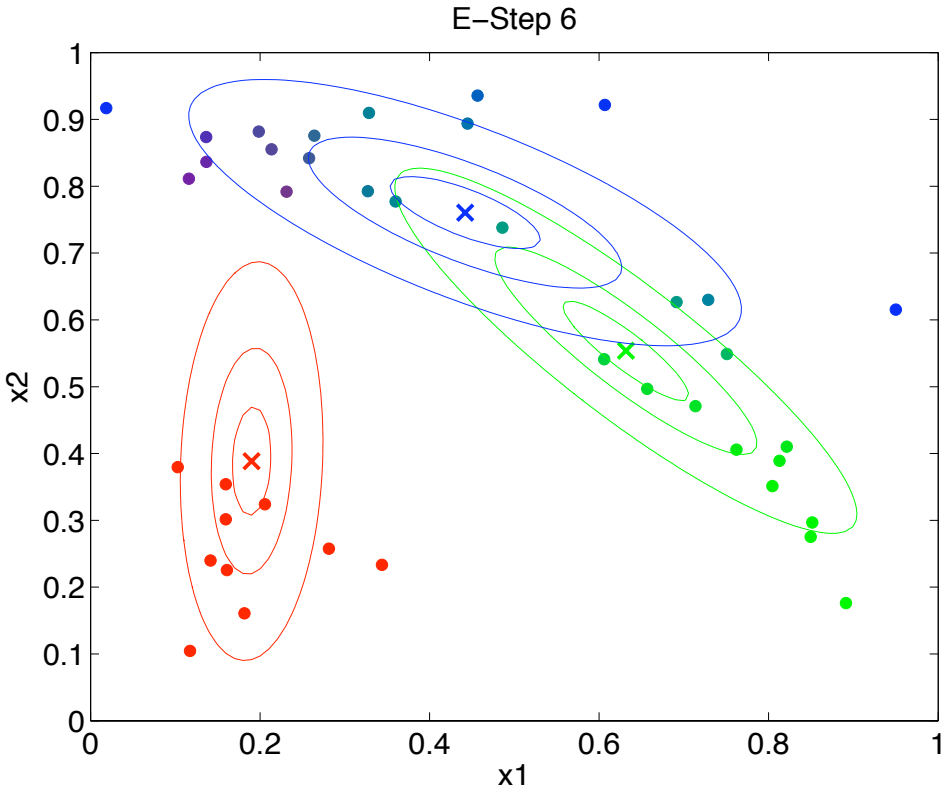
Soft EM for Mixture of Gaussians: Example



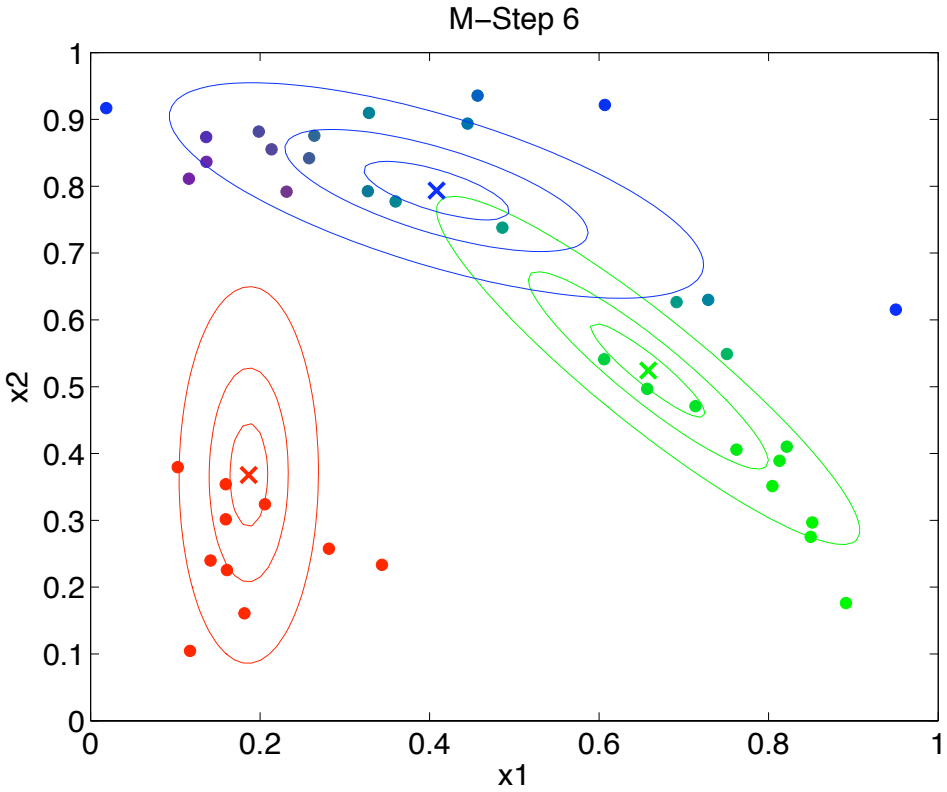
Soft EM for Mixture of Gaussians: Example



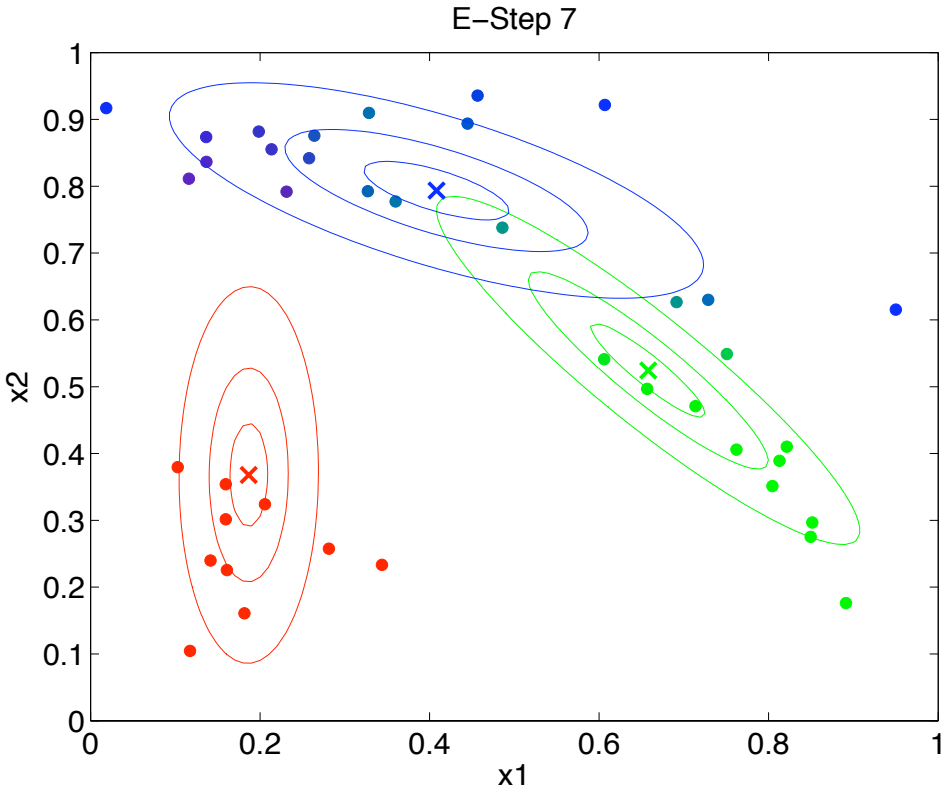
Soft EM for Mixture of Gaussians: Example



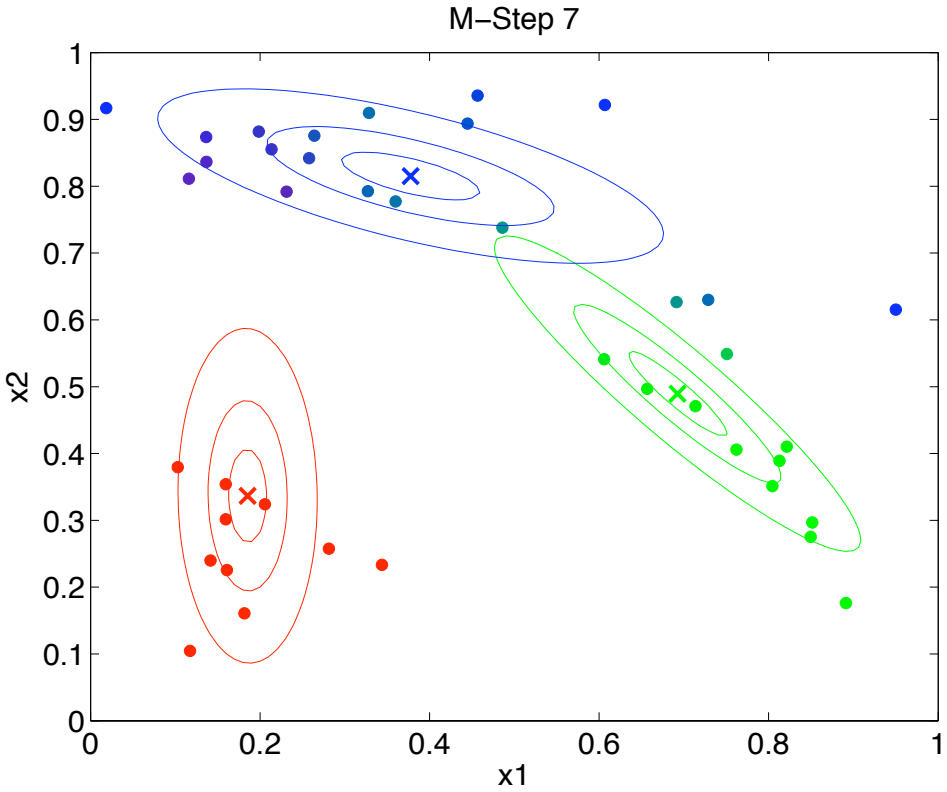
Soft EM for Mixture of Gaussians: Example



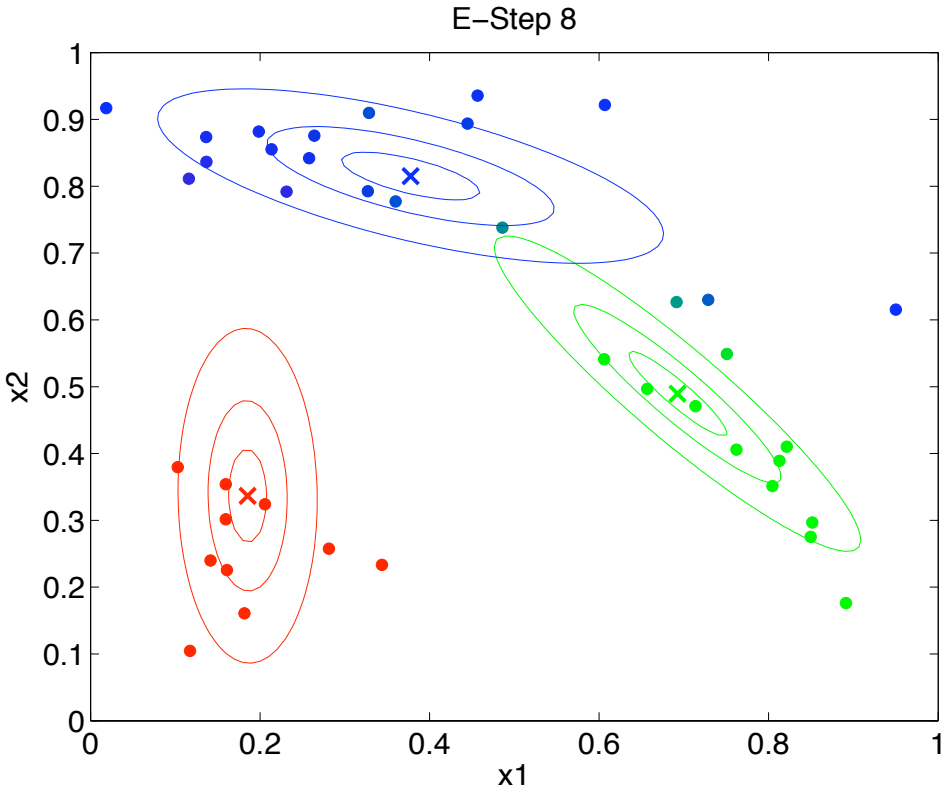
Soft EM for Mixture of Gaussians: Example



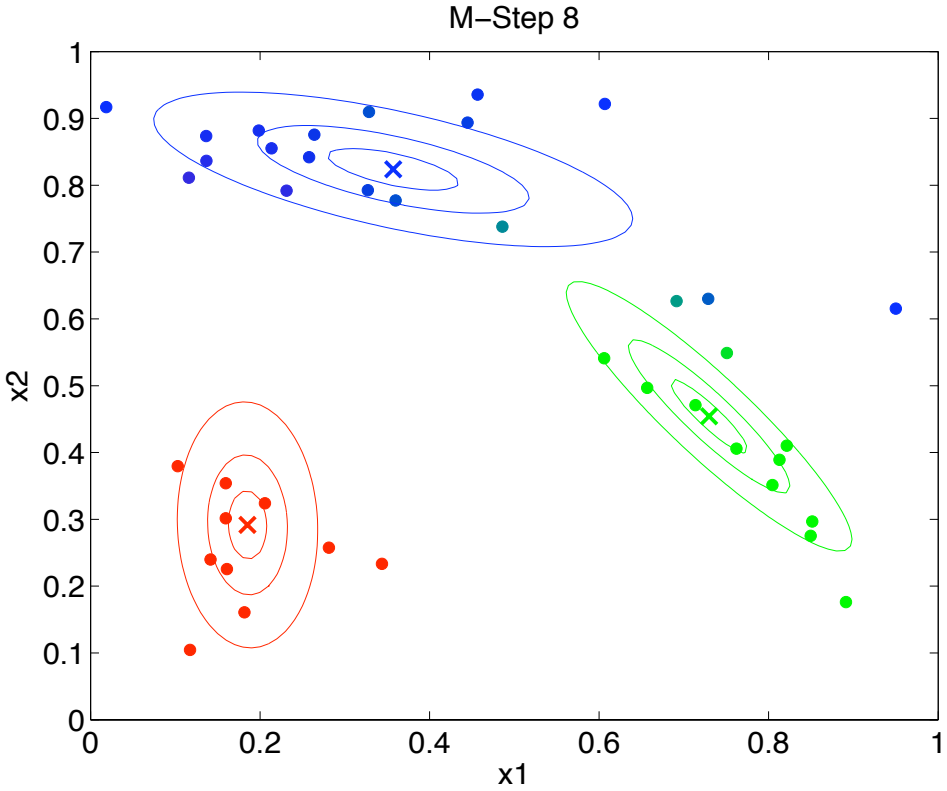
Soft EM for Mixture of Gaussians: Example



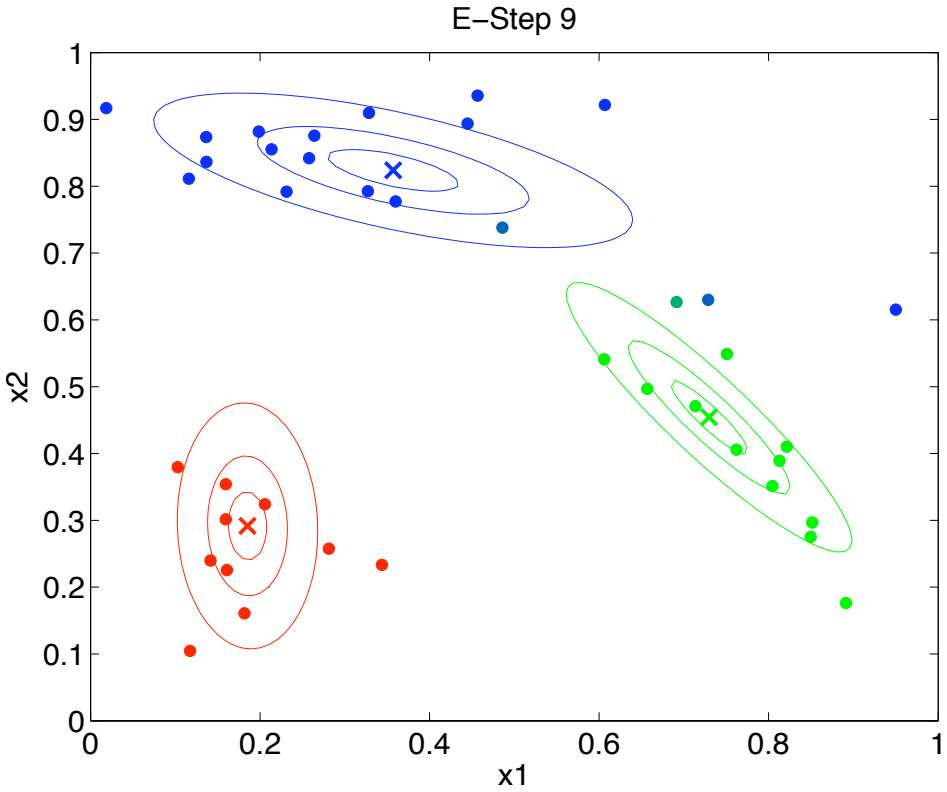
Soft EM for Mixture of Gaussians: Example



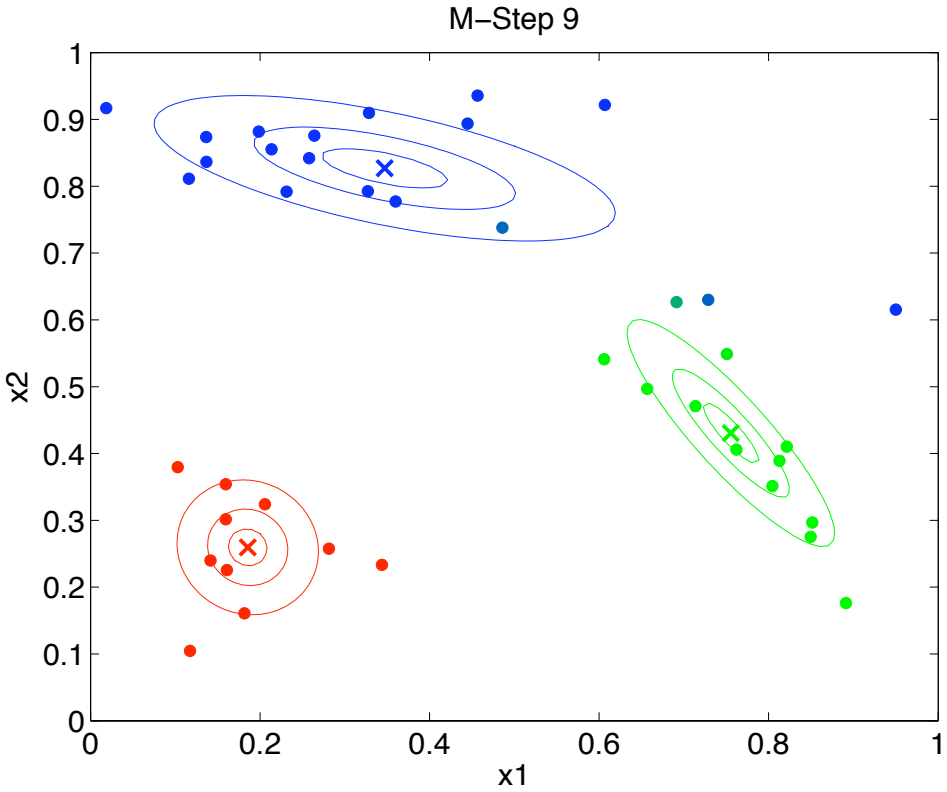
Soft EM for Mixture of Gaussians: Example



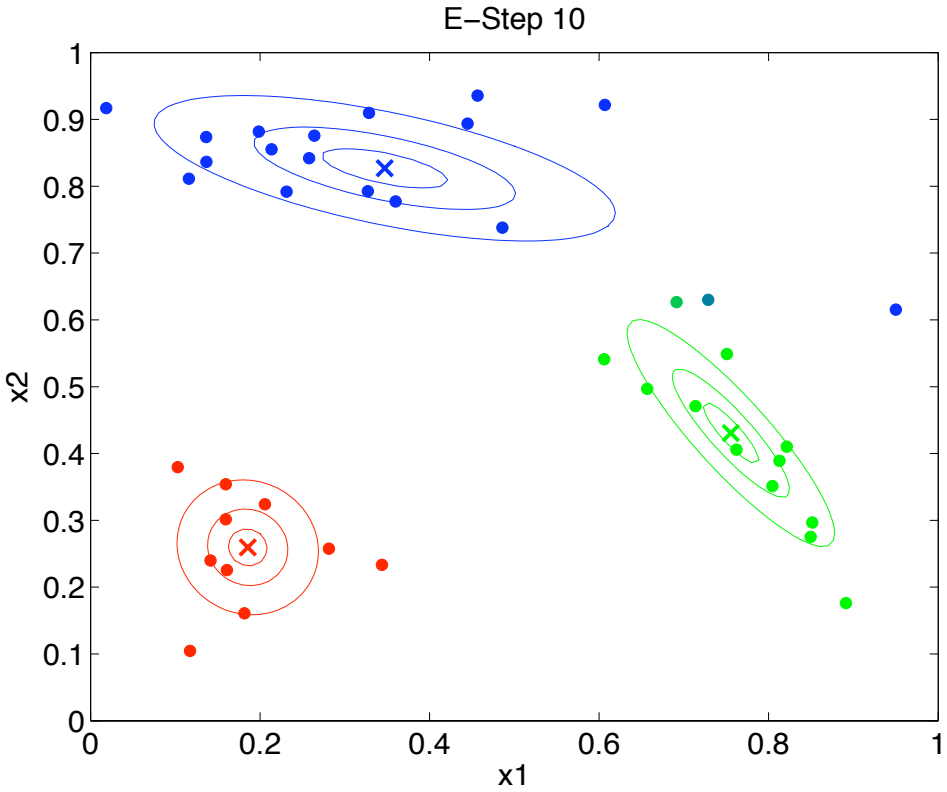
Soft EM for Mixture of Gaussians: Example



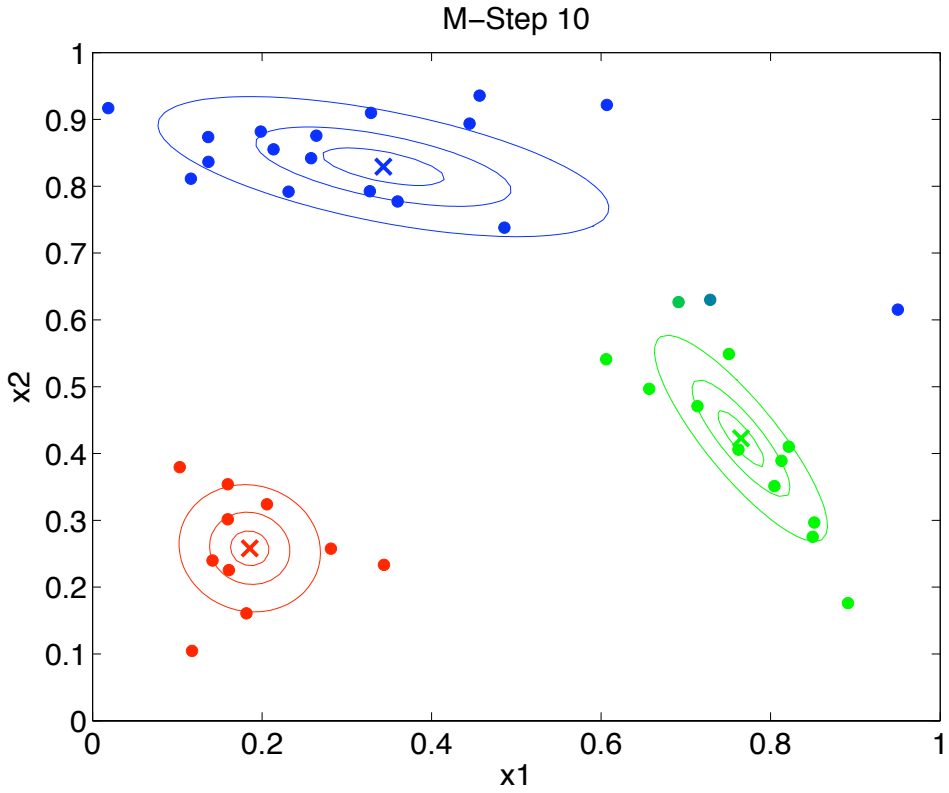
Soft EM for Mixture of Gaussians: Example



Soft EM for Mixture of Gaussians: Example



Soft EM for Mixture of Gaussians: Example



Comparison of hard EM and soft EM

- Soft EM does not commit to a particular value of the missing item. Instead, it considers all possible values, with some probability
- This is a pleasing property, given the uncertainty in the value
- Soft EM is almost always the method of choice (and often when people say “EM”, they mean the soft version)
- The complexity of each iteration of the two versions is pretty much the same. How many iterations does each take?

Theoretical properties of EM

- Each iteration improves the likelihood of the data given the class assignments, p_j, μ_j, and Σ_j.
  - Straightforward for Hard EM.
  - Less obvious for Soft EM.
- The algorithm works by making a convex approximation to the log-likelihood (by filling in the data)
- If the parameters do not change in one iteration, then the gradient of the log-likelihood function is zero

Mixture components converging to a point

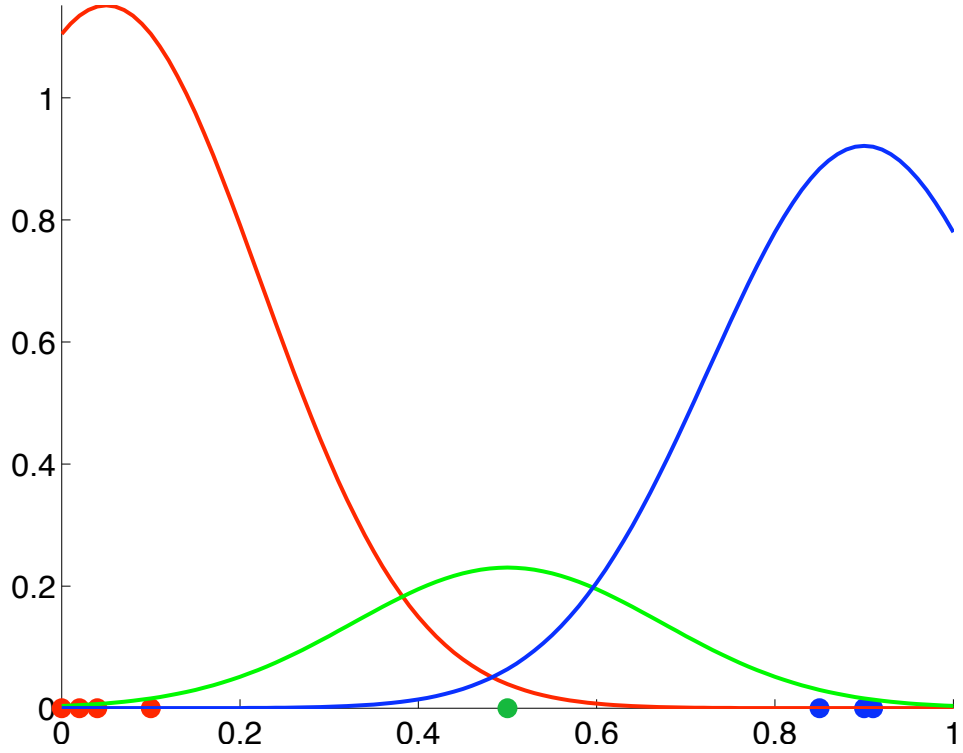
- What happens in Hard EM if a class contains a single point?

$\Rightarrow$ The covariance matrix is not defined!

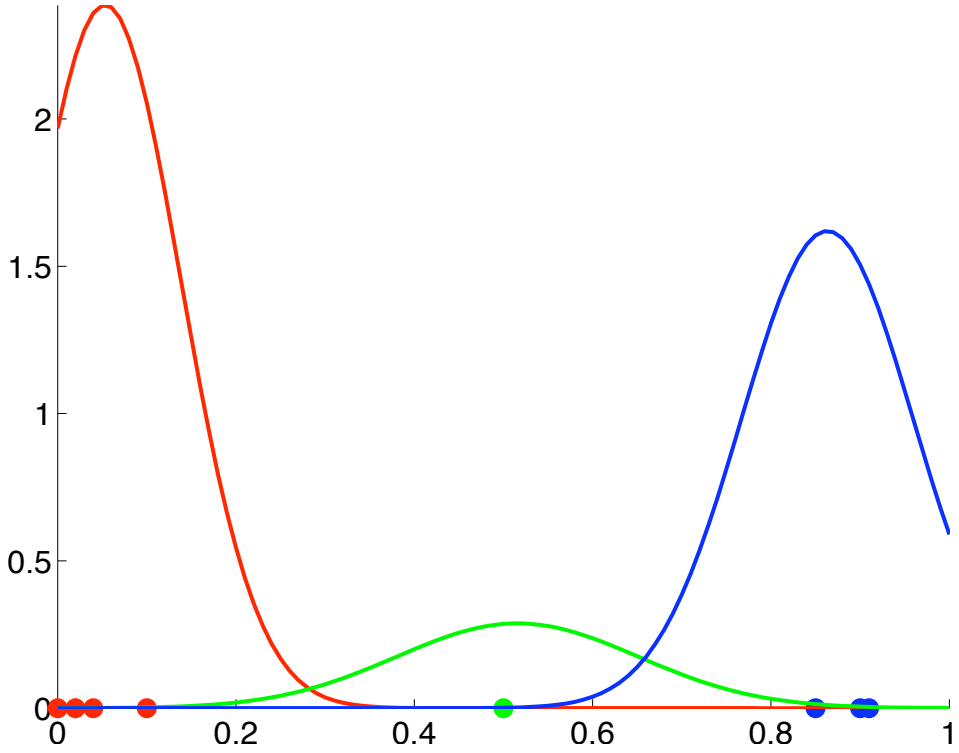
- Similarly, what happens in Soft EM if a class focusses more and more on a single point over iterations?

$\Rightarrow$ The covariance matrix goes to zero! And the likelihood of the data goes to $+\infty$! (See following slides.)

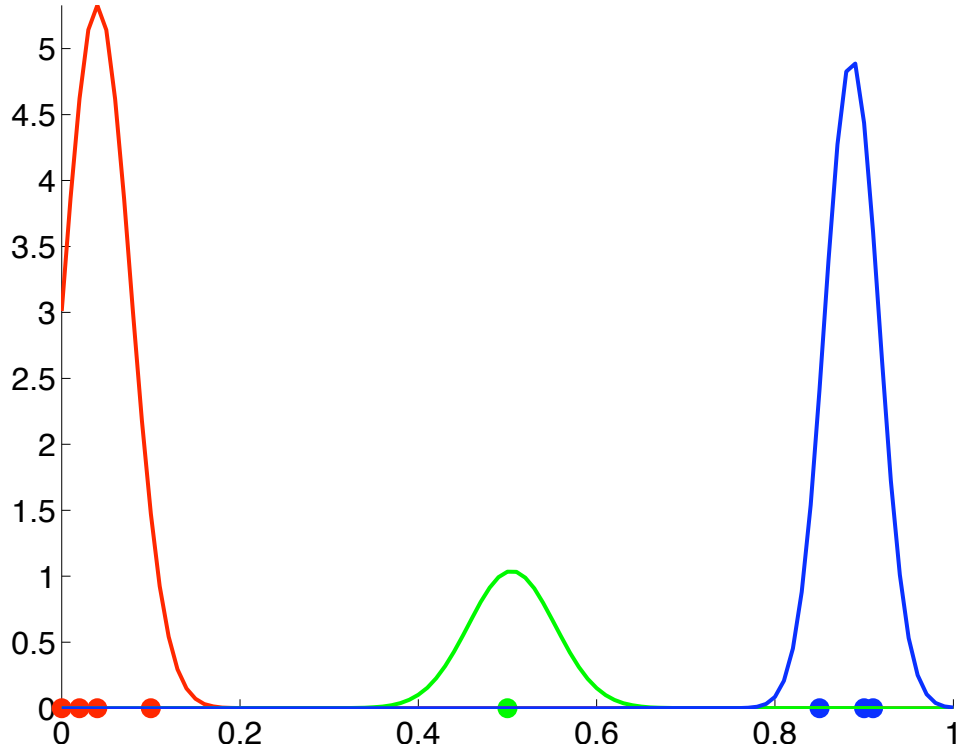
Mixture converging to a point: Example



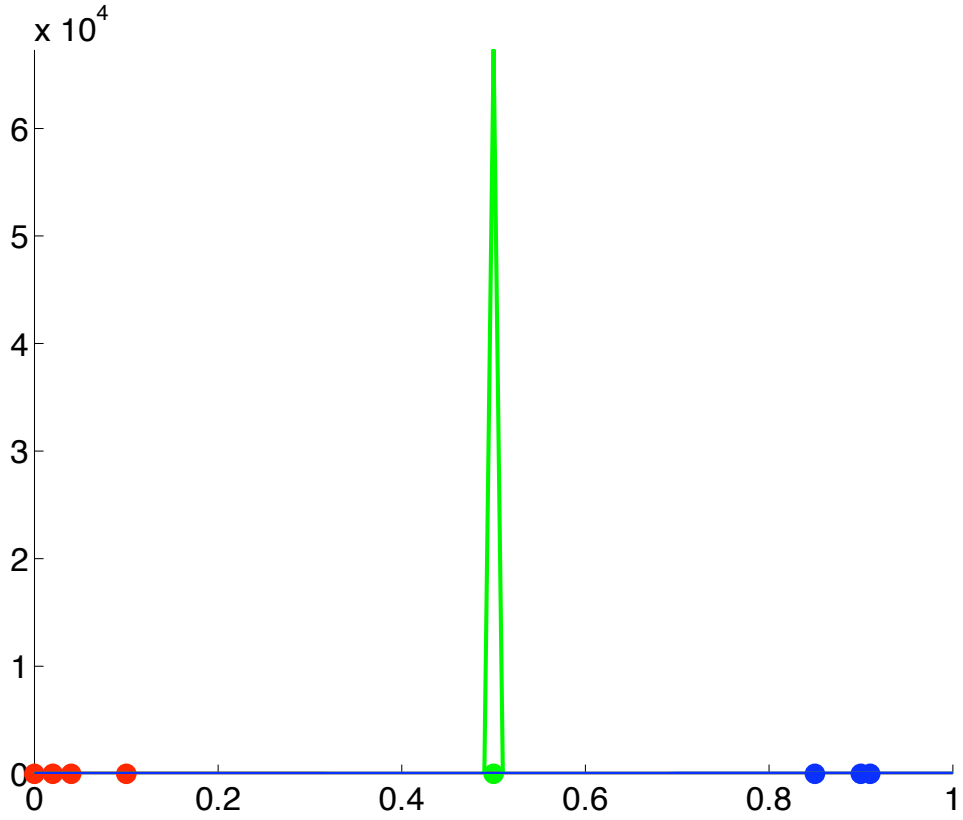
Mixture converging to a point: Example



Mixture converging to a point: Example



Mixture converging to a point: Example



Variations

- If only some of the data is incomplete, the likelihood will have one component based on the complete instances and another ones based on incomplete instances
- Sparse EM: Only compute probability at a few data points (most values will be close to 0 anyway)
- EM can be used if you have labeled data but which is possibly unreliable, in order to get better labels.
- Instead of a complete M-step, just improve the likelihood a bit
- Note that EM can be stuck in *local minima*, so it has to be restarted!
- It works very well for low-dimensional problems, but can have problems if θ is high-dimensional.