COMP 652: Machine Learning

Lecture 14

Today

- Introduction to unsupervised learning
- Clustering problem
- K-means clustering
- Mixture models
- Expectation Maximization for mixture models

Unsupervised learning

- In supervised learning, we have data in the form of pairs $\langle \mathbf{x}, y \rangle$, where $y = f(\mathbf{x})$. The goal is to approximate f.
- 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:
  - Density estimation
    - Parametric or non-parametric, already seen
    - Mixture density estimation (sum of parametric densities)
  - Clustering: "Flat" clustering or partitioning, hierarchical clustering
  - Dimensionality reduction, for: visualization, compression, pre-processing
- The definition of "ground truth" is often missing (no clear error function, like in supervised learning, or at least many reasonable alternatives)

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

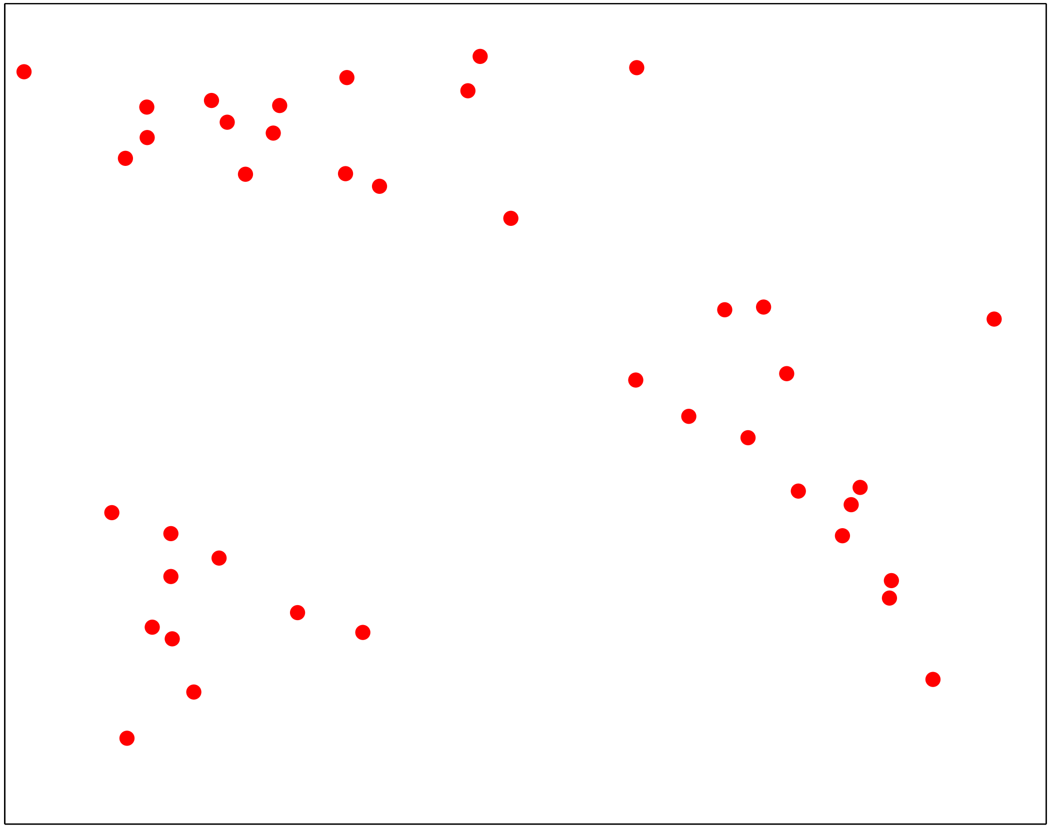
K-means clustering with real-valued data

- 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), $C : \{1, \ldots, m\} \mapsto \{1, \ldots, K\}$.

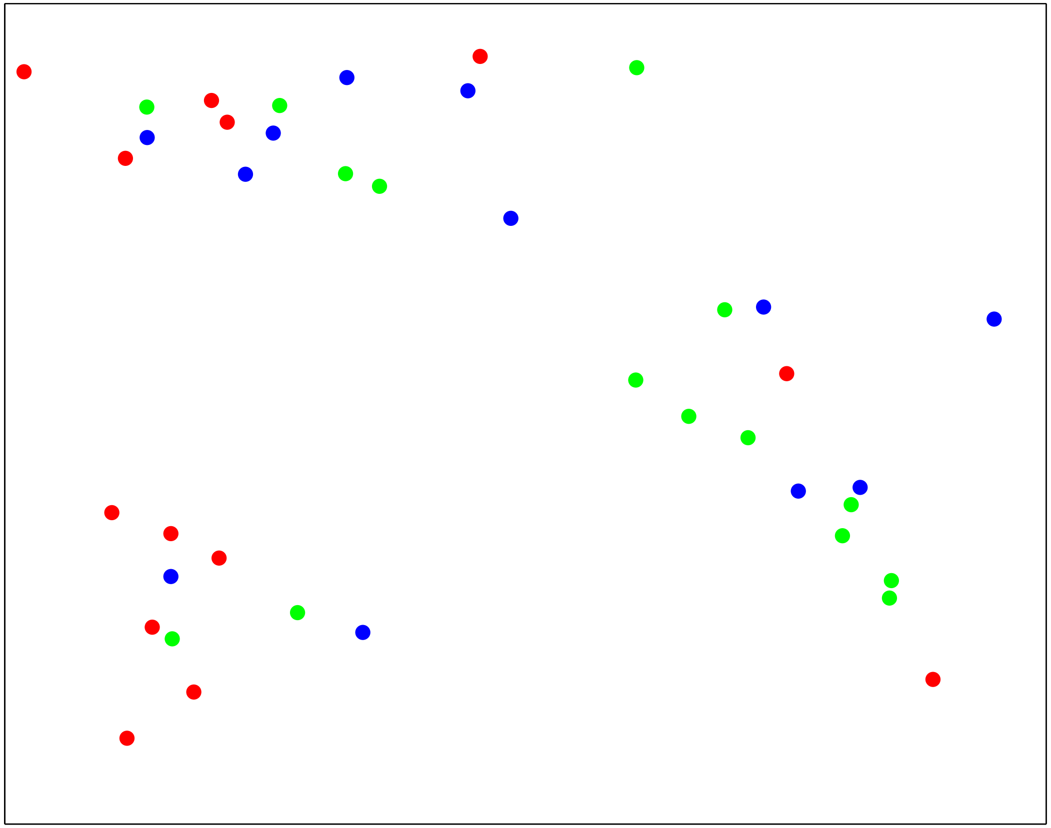
1. Initialize C randomly.
2. Repeat
   (a) Compute the *centroid* of each cluster (which is mean of all the instances in the cluster)
   (b) If any instance is closer to a different cluster centroid than its current one, reassign it to the cluster with closest centroid.

   Until C stops changing.

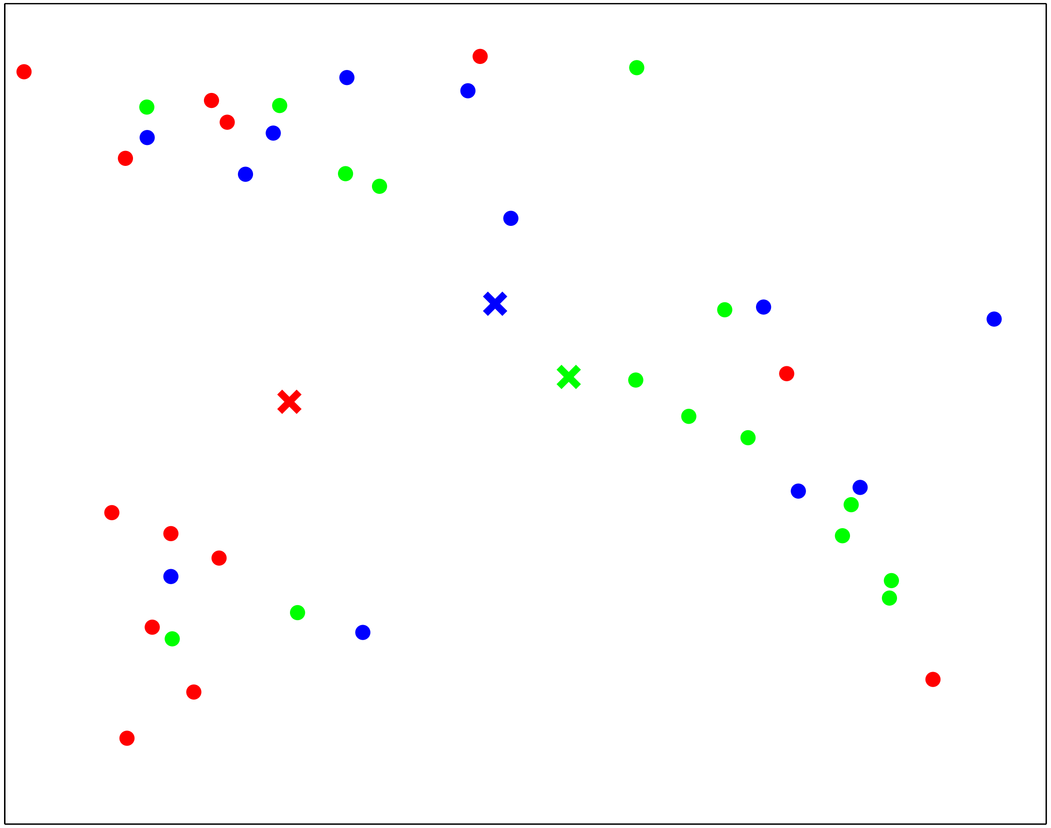
Example: initial data



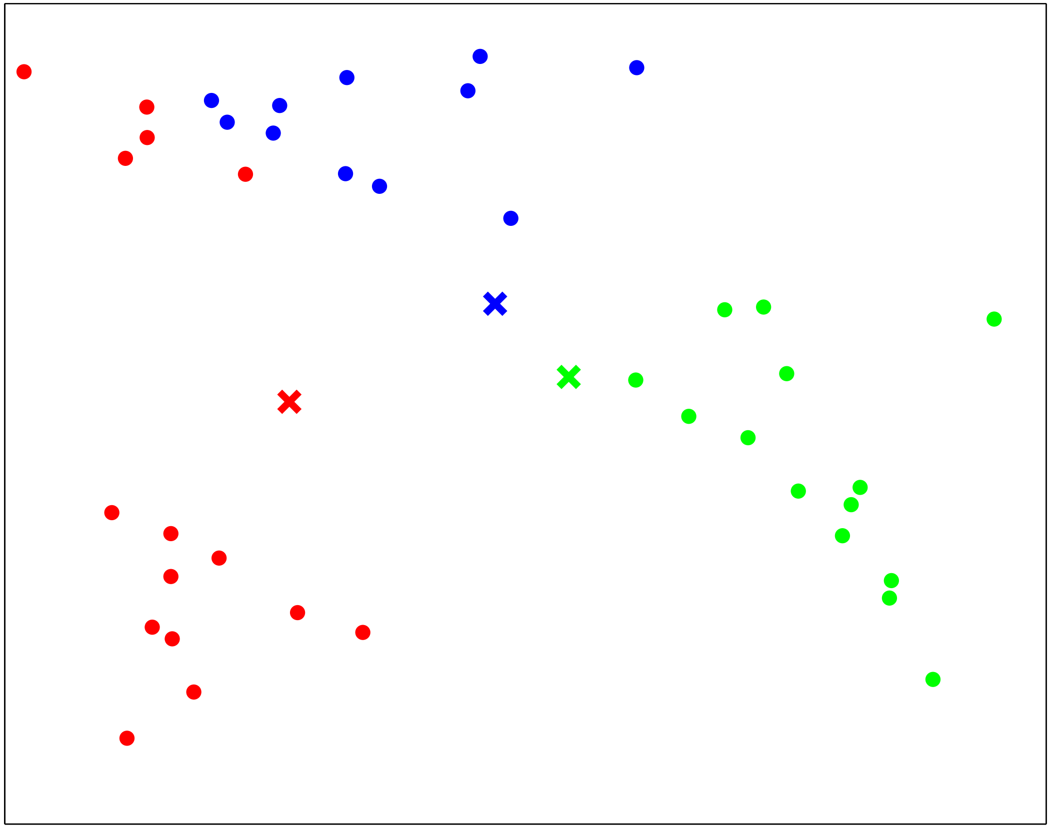
Example: assign into 3 clusters randomly



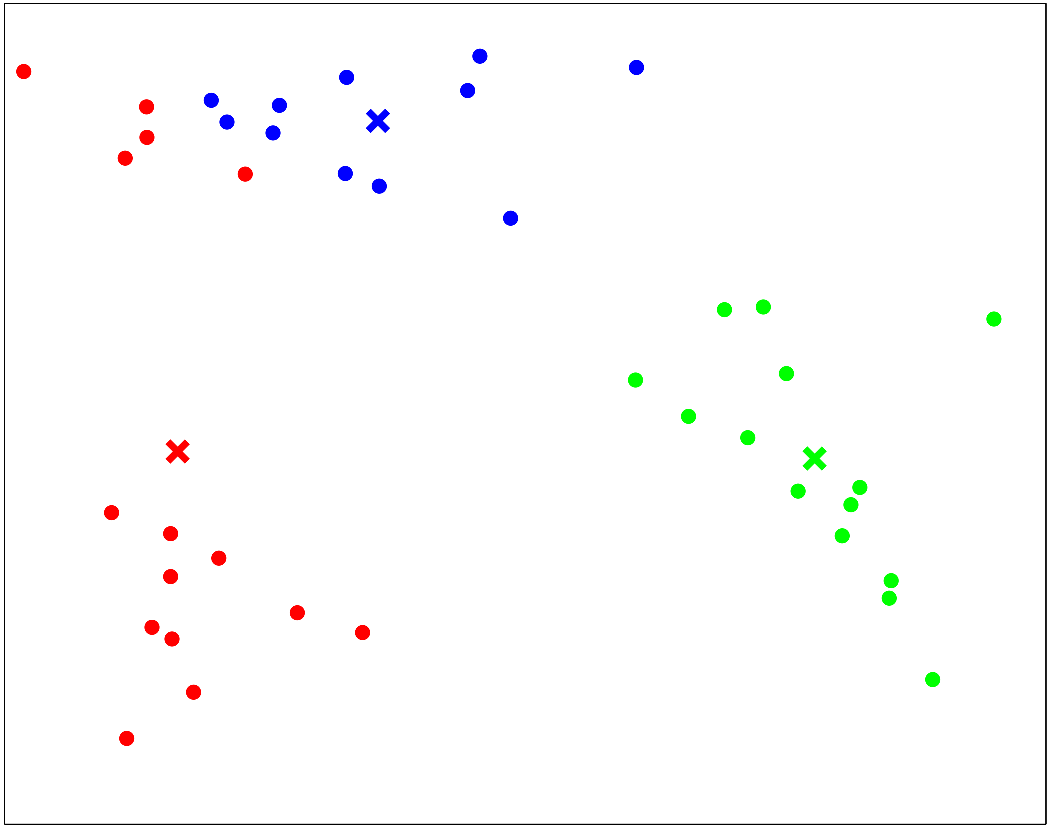
Example: compute centroids



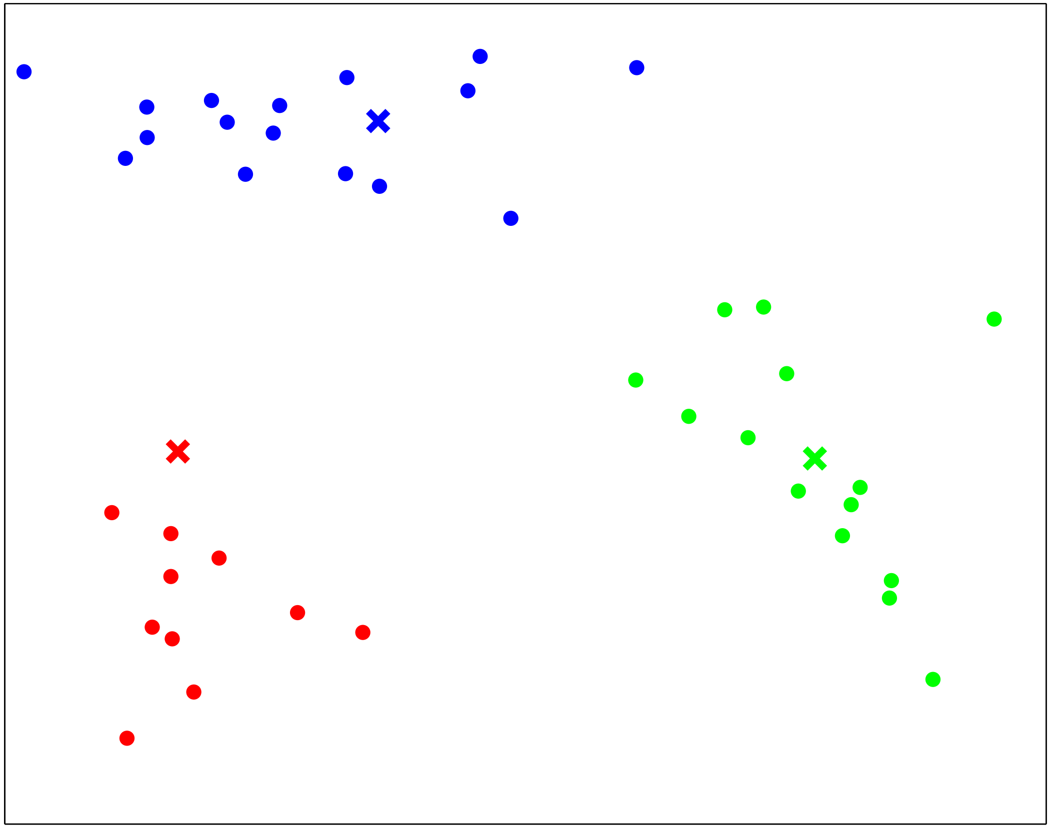
Example: reassign clusters



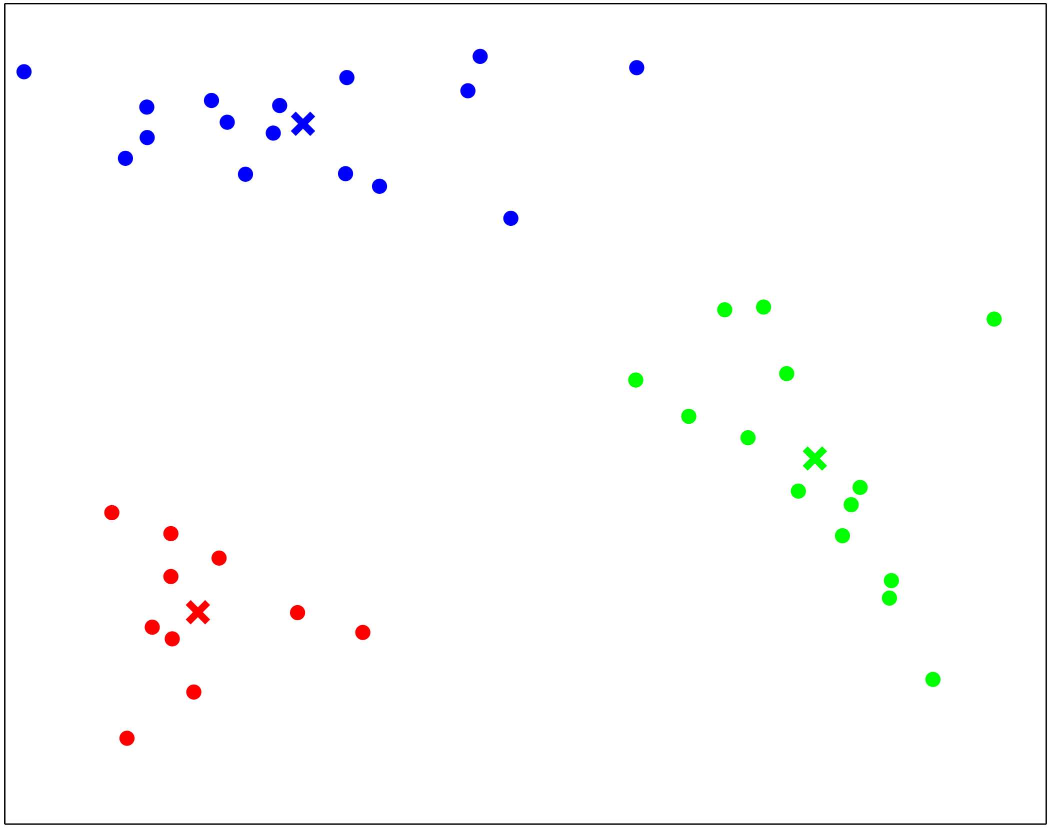
Example: recompute centroids



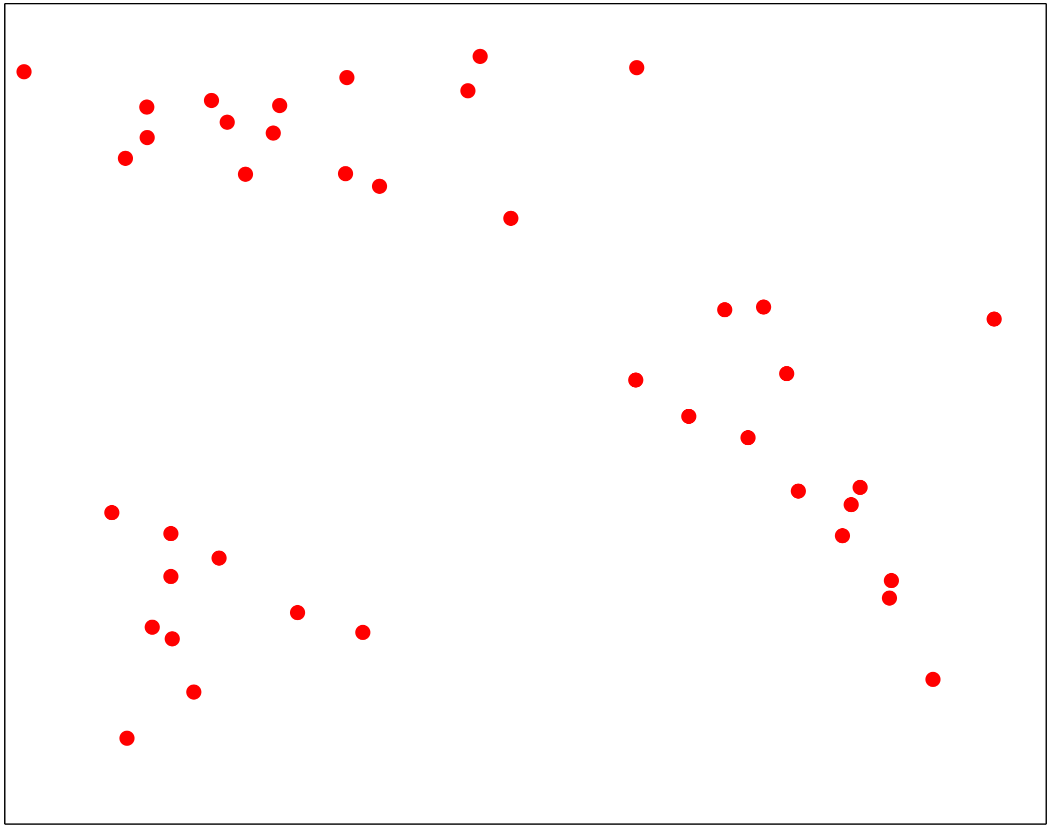
Example: reassign clusters



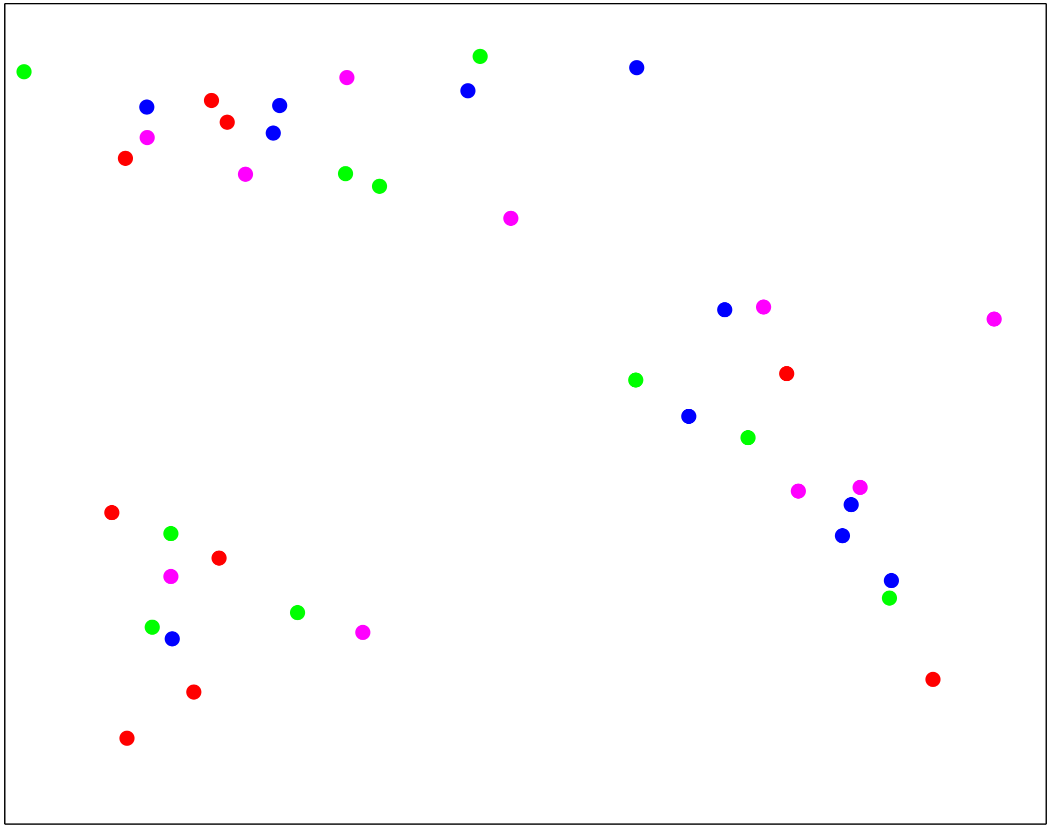
Example: recompute centroids – done!



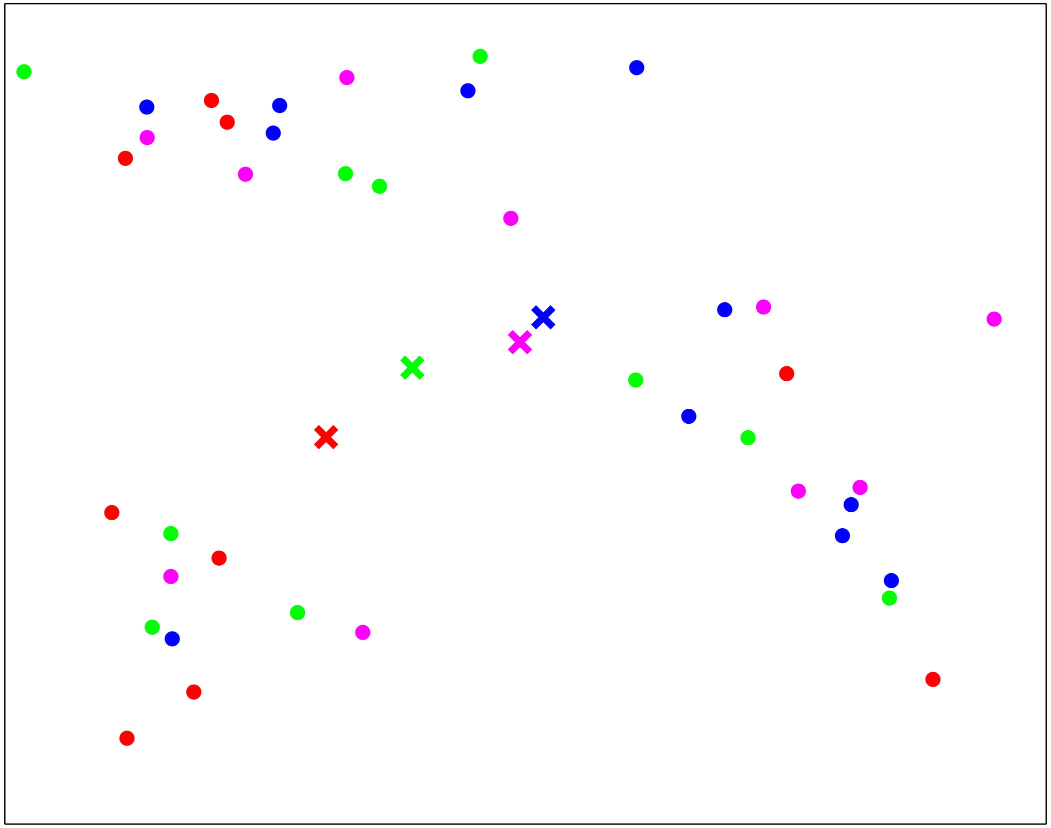
What if we do not know the right number of clusters?



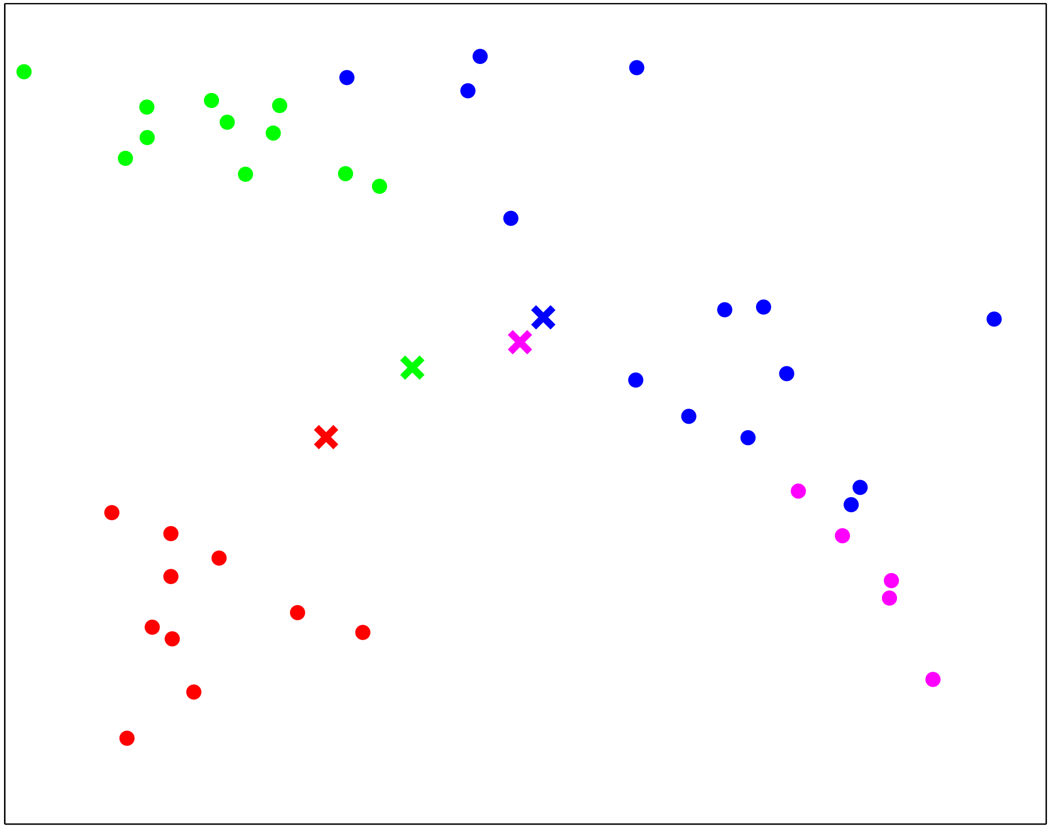
Example: assign into 4 clusters randomly



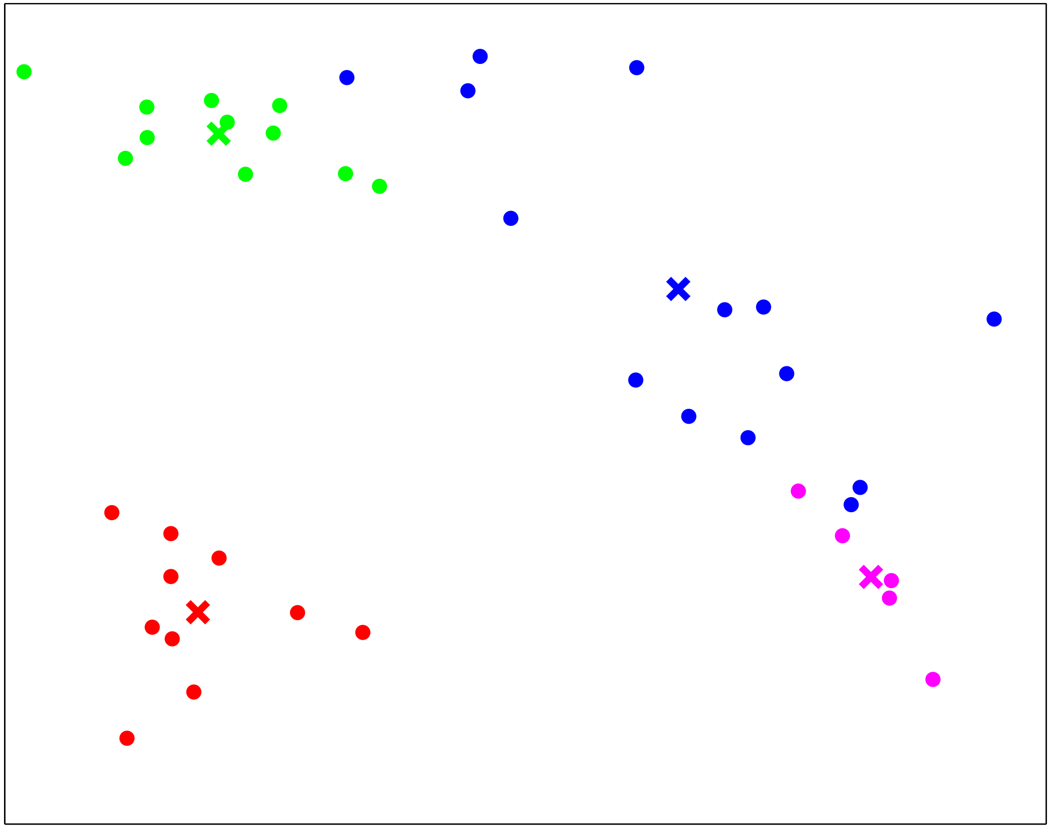
Example: compute centroids



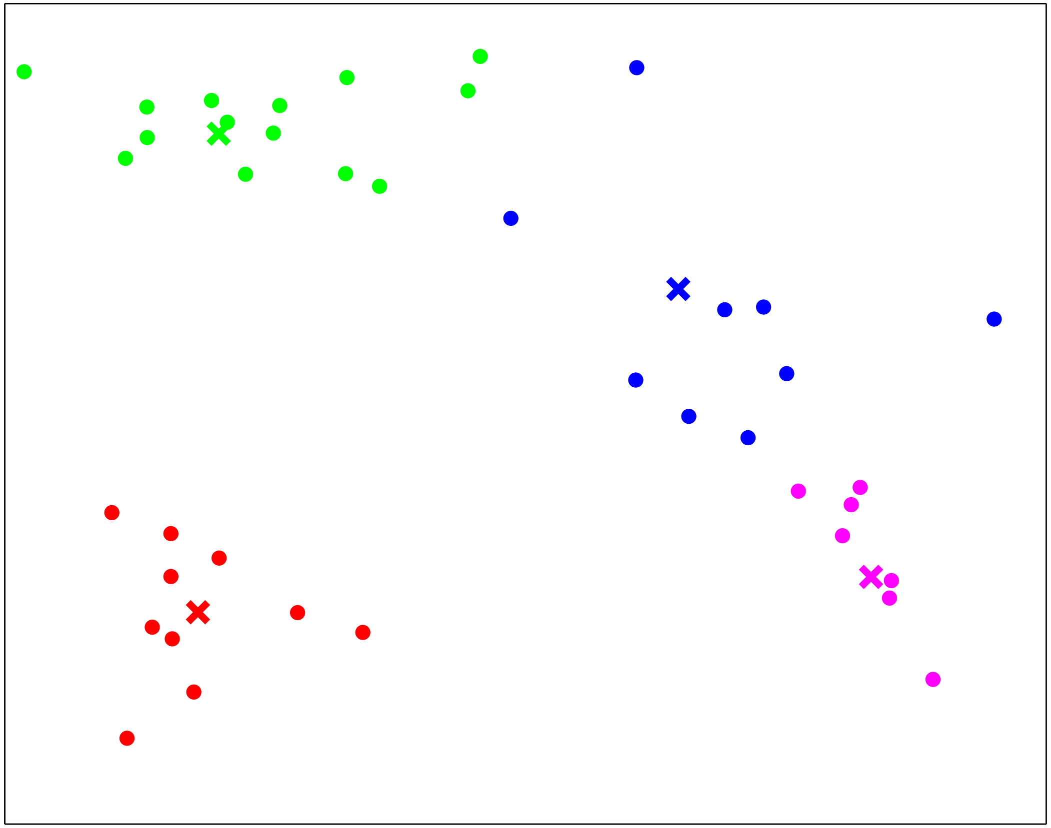
Example: reassign clusters



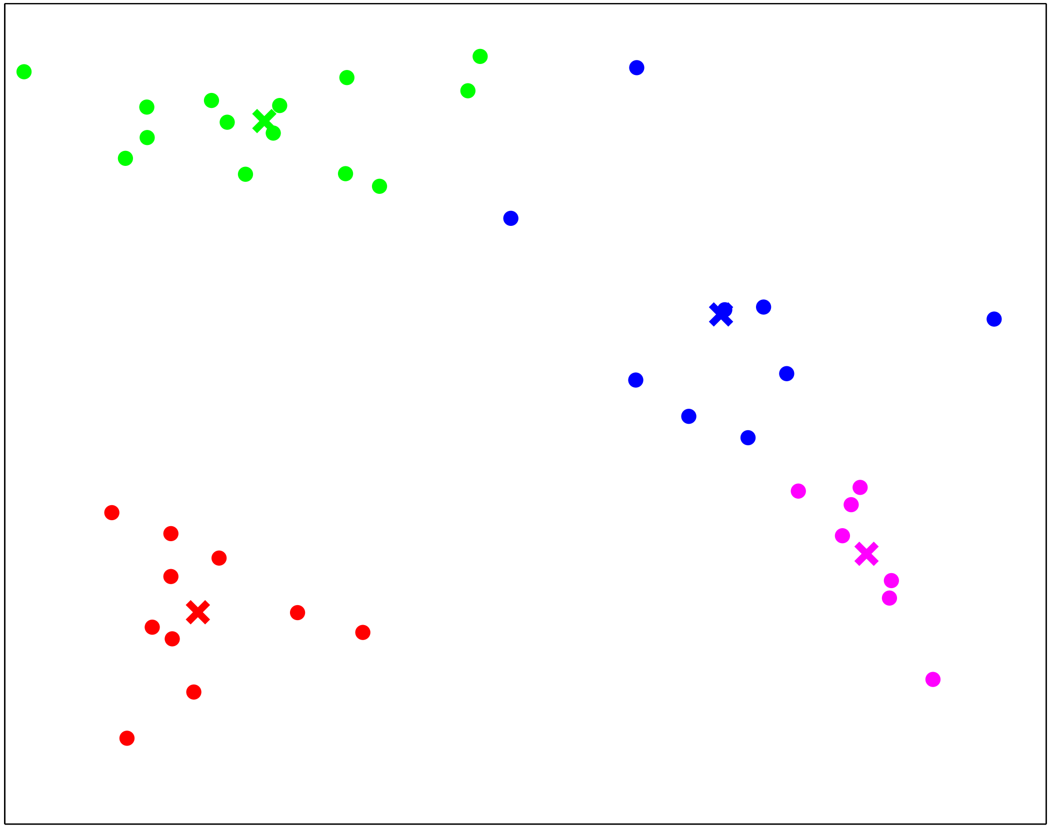
Example: recompute centroids



Example: reassign clusters



Example: recompute centroids – done!



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?

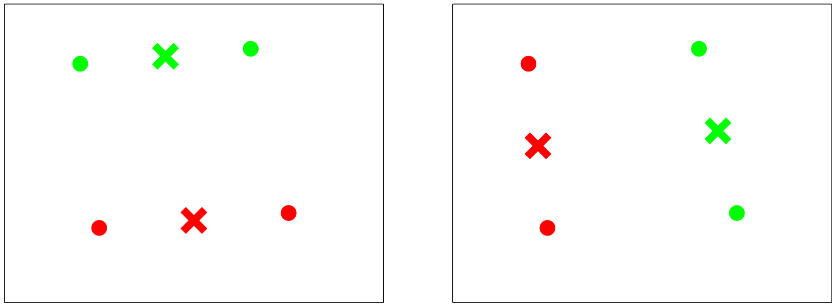
Does K-means clustering terminate?

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

Does K-means always find the same answer?

- K-means is an iterative procedure for minimizing 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 depends on the initial assignment of instances to clusters



$J = 0.22870$ $\qquad$ $J = 0.3088$

- There is also a trivial non-uniqueness in that renaming the categories $1, \ldots, K$ doesn't change the value of J.

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

⇒ 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 repesented by 8-bit index into colormap

More generally: Vector quantization with Euclidean loss

- In a similar way, 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 (J-minimizing) K-means clustering minimizes the reconstruction error among all possible codings of the same type

Finding good initial configurations

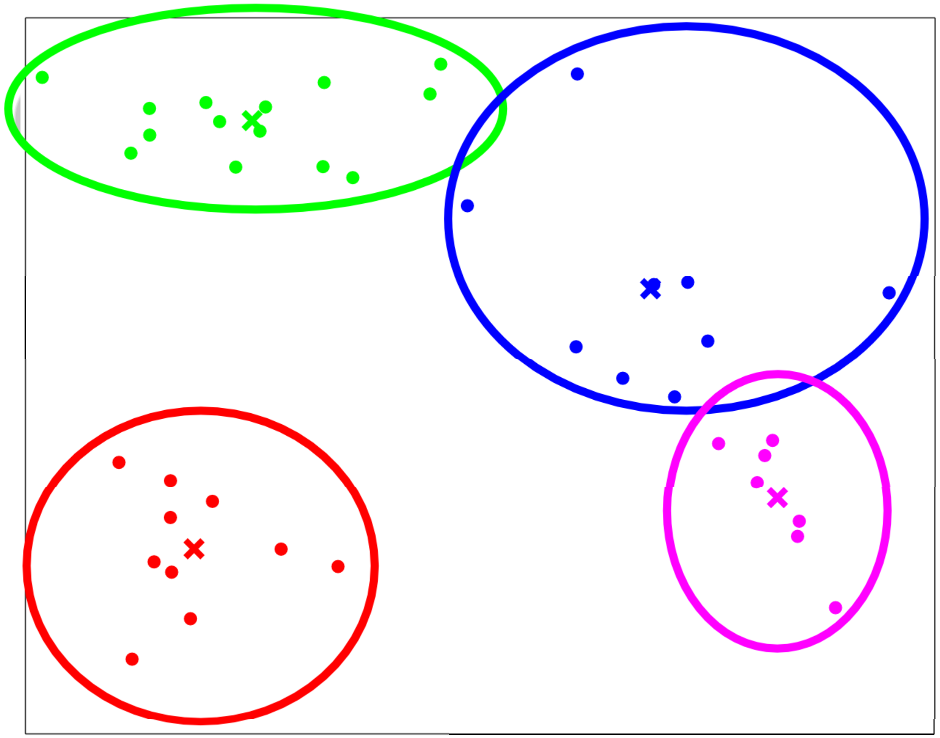
- The initial configuration can influence the final clustering
- Assigning each item to random cluster in $\{1, \ldots, K\}$ is unbiased... but typically results in cluster centroids near the centroid of all the data in the first round.
- A different heuristic tries to spread the initial centroids around 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
- ...

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. (See next slide.)

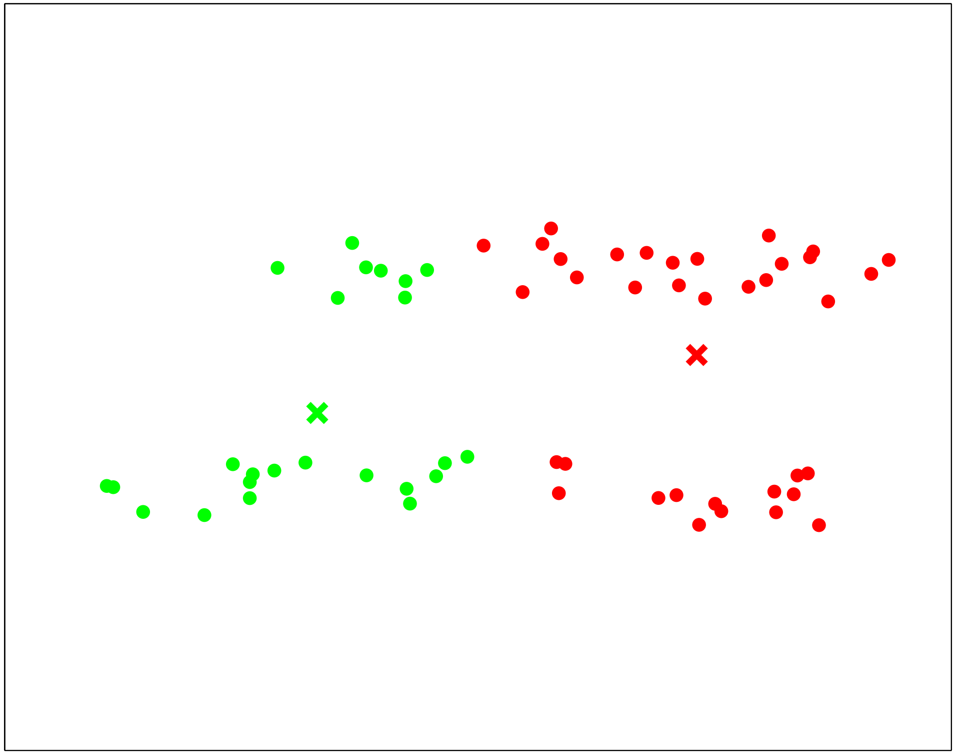
Maximum likelihood clustering

$$
\begin{array}{lcl}
 & & l(\mathbf{x}_1, \ldots, \mathbf{x}_m | C(i), \mu_j) \\
 & = & \Pi_{i=1}^{m} l(\mathbf{x_i} | C(i), \mu_j) \\
 & = & \Pi_{i=1}^{m} \dfrac{1}{(2\pi)^{n/2}\sigma^n} \exp\left(-\dfrac{1}{2\sigma^2} \|\mathbf{x_i} - \mu_{C(i)}\|^2 \right) \\
\Rightarrow & & \log l(\mathbf{x}_1, \ldots, \mathbf{x}_m | C(i), \mu_j) \\
 & \propto & -\|\mathbf{x_i} - \mu_{C(i)}\|^2 \\
 & = & J
\end{array}
$$

Why not the sum of squared Euclidean distances?

Why not the sum of squared Euclidean distances?

- Reason 1: It produces nice round clusters!



- Reason 2: Differently scaled axes can dramatically affect results.
- Reason 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.

Examples

Distance metrics:

- Euclidean distance
- Hamming distance (number of mismatches between two strings)
- Travel distance (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)

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
- Let us try a more general **mixture model**, using Gaussian distributions

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:

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

- 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 <u>*expectation*</u>
  - Then we compute new parameters, which <u>*maximize*</u> 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

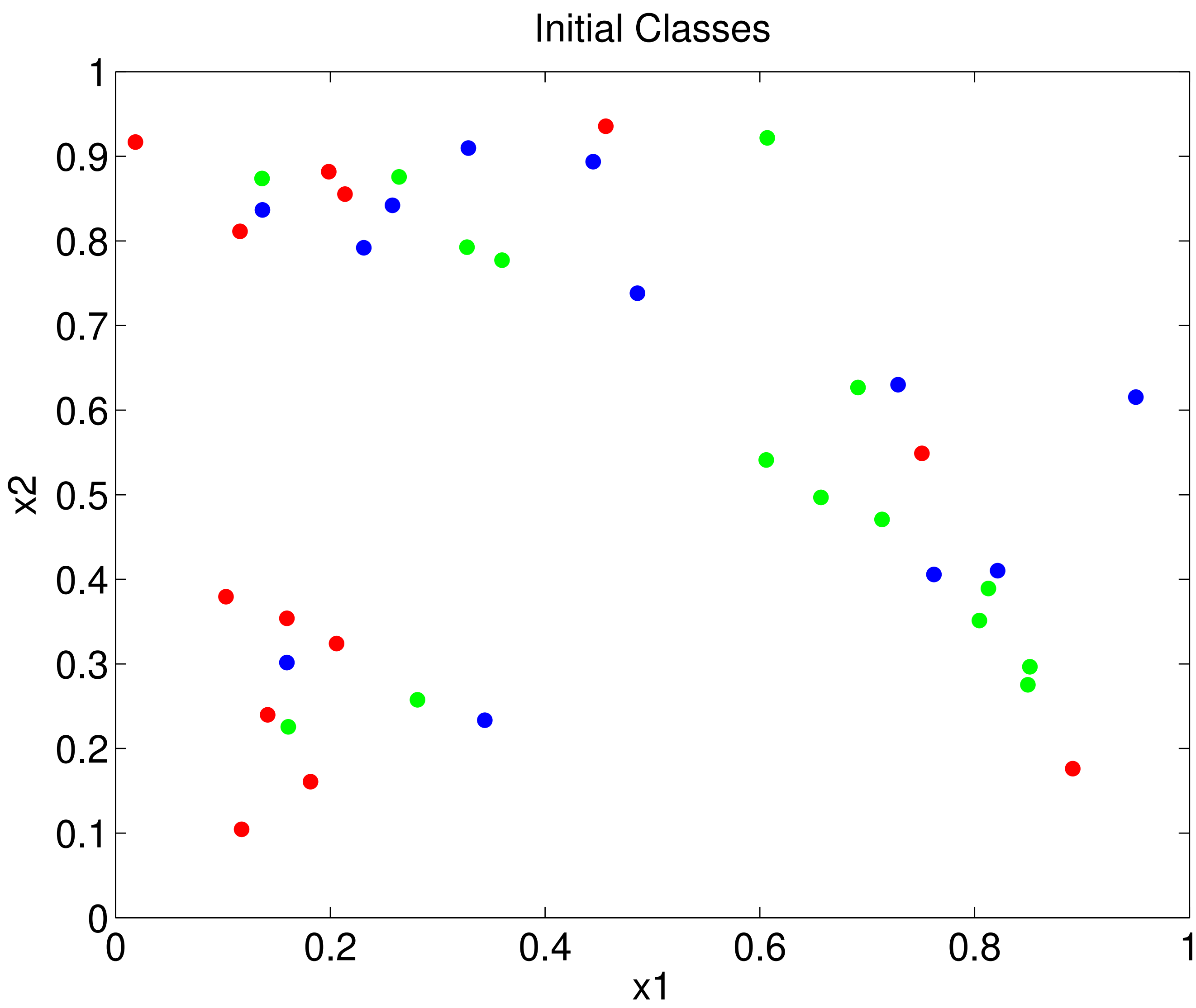
   $$p_j = \frac{1}{m}\sum_{i=1}^{m}\delta_{ij} \qquad \mu_j = \frac{\sum_{i=1}^{m}\delta_{ij}\mathbf{x}_i}{\sum_{i=1}^{m}\delta_{ij}}$$

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

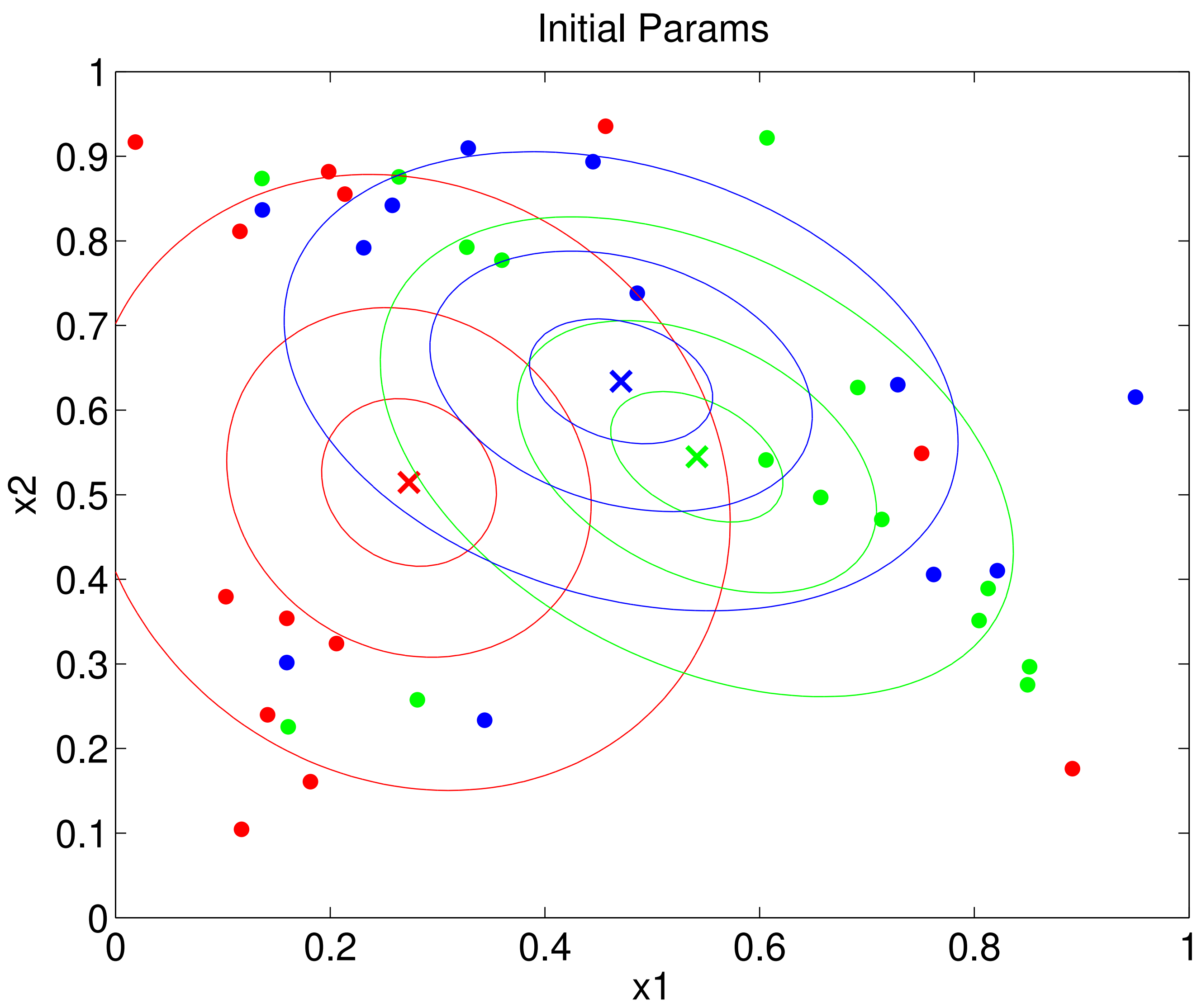
Hard EMf for Mixture of Gaussians: Example

- Hard EM with $K = 3$
- Example data from start of lectures
- Initial classes assigned randomly
- Initial parameters estimated from initial classes

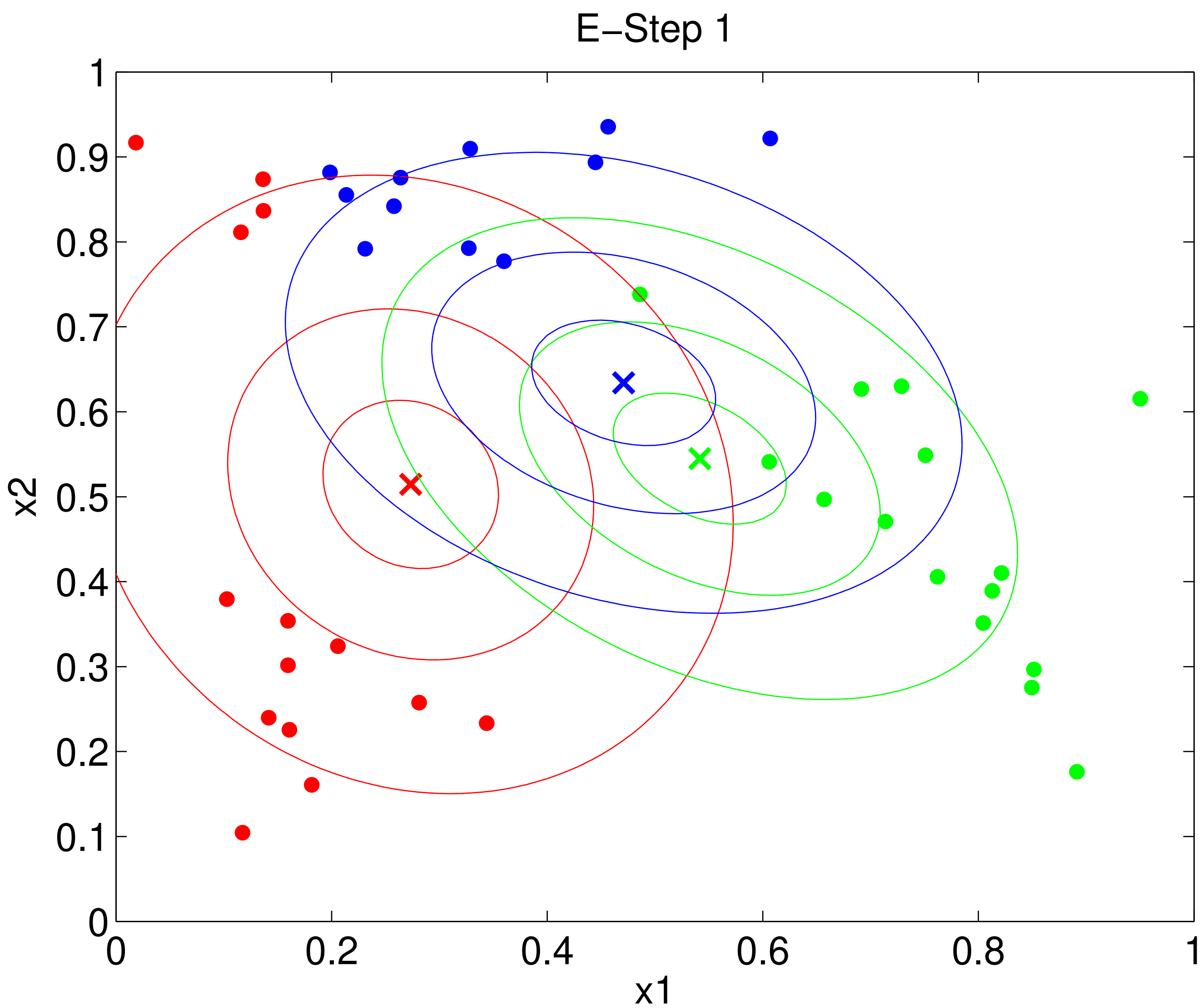
Hard EM for Mixture of Gaussians: Example



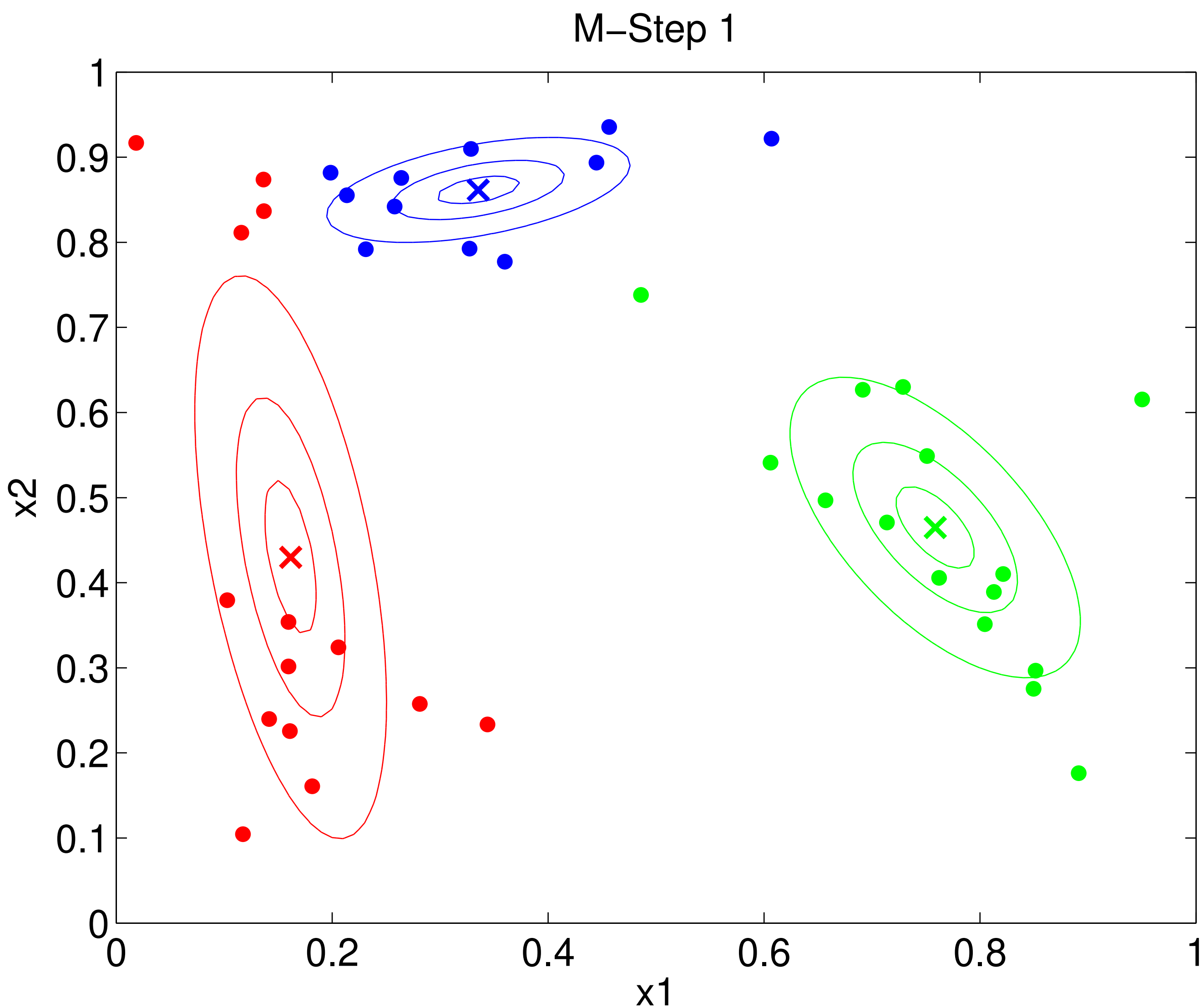
Hard EM for Mixture of Gaussians: Example



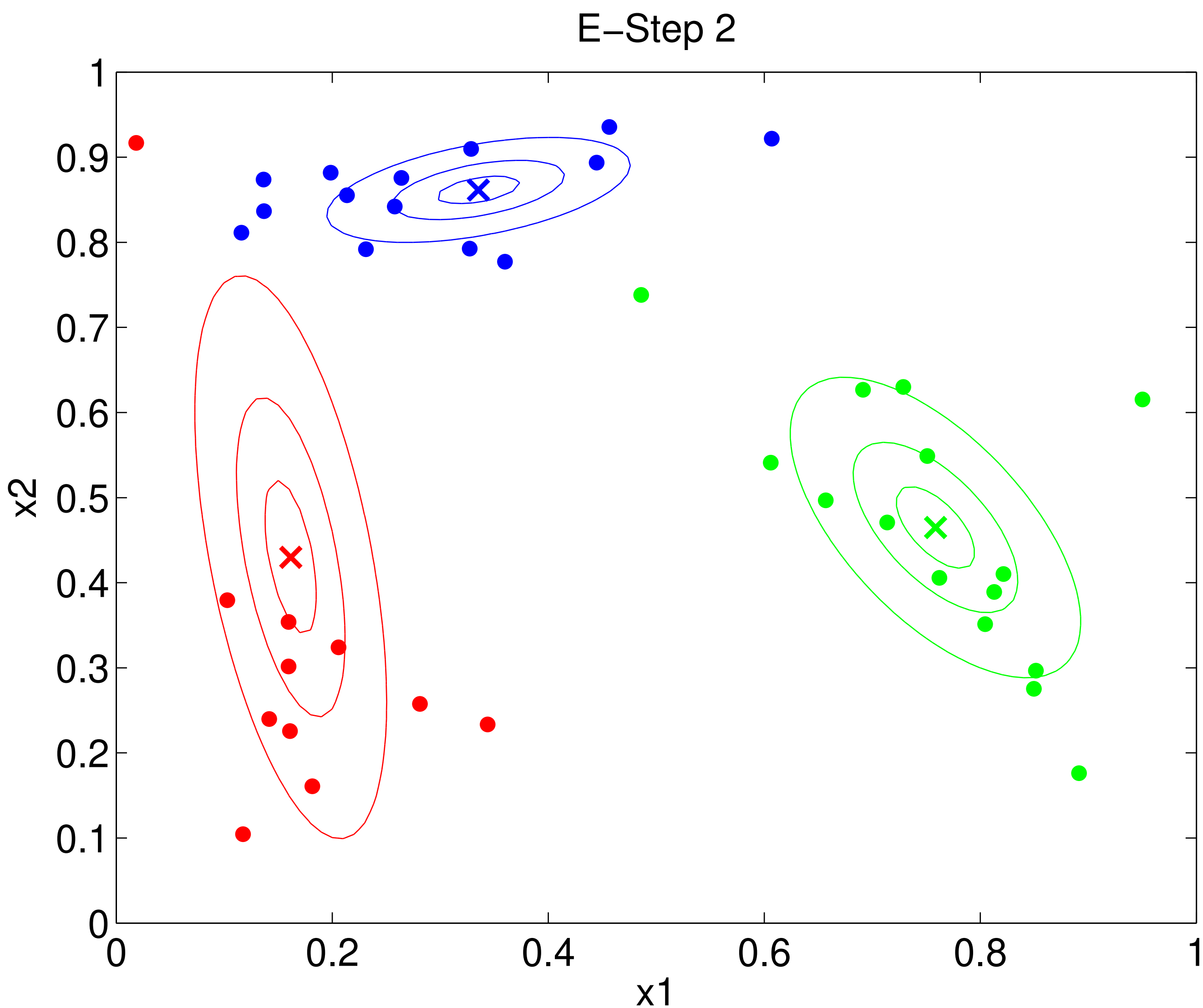
Hard EM for Mixture of Gaussians: Example



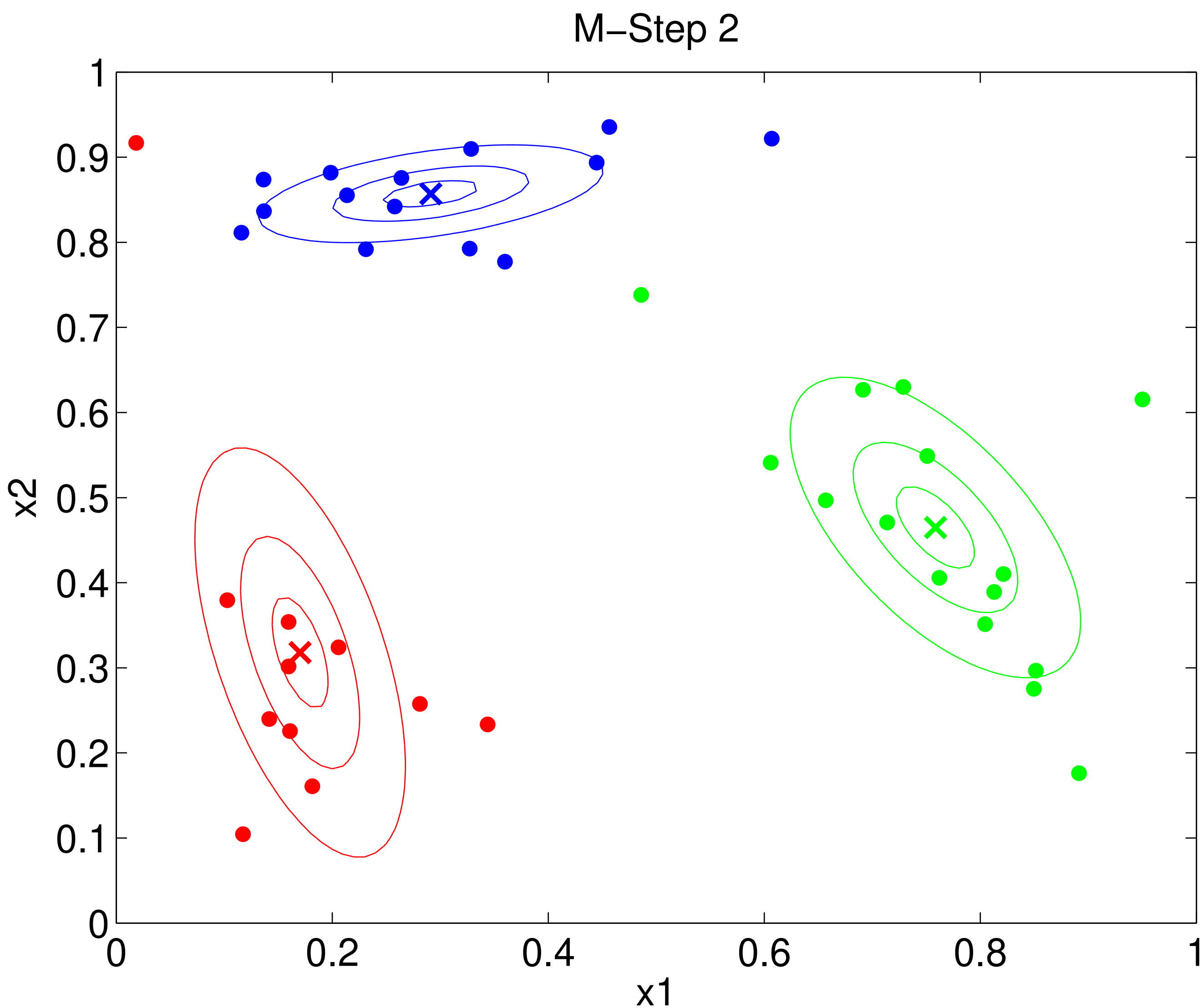
Hard EM for Mixture of Gaussians: Example



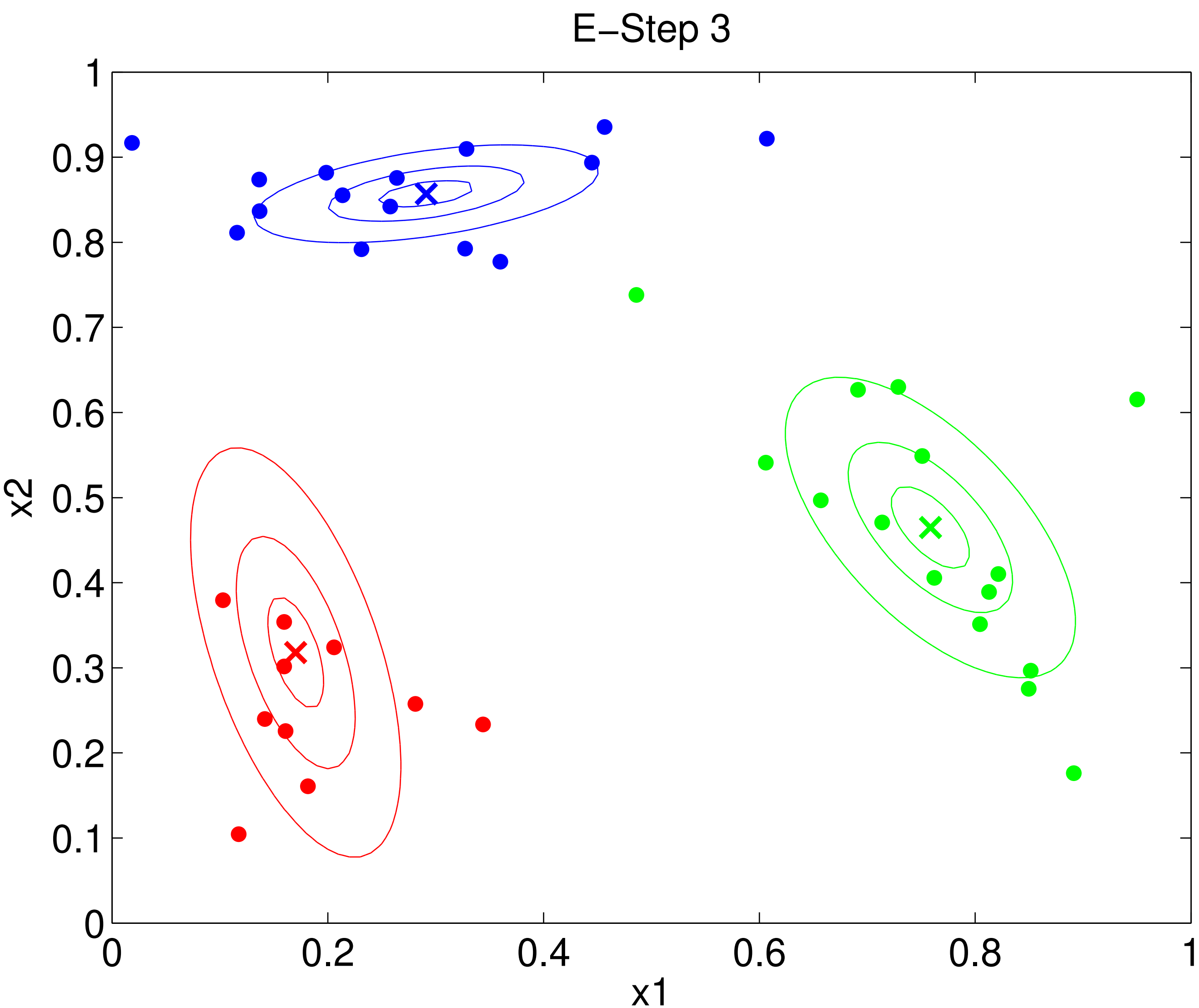
Hard EM for Mixture of Gaussians: Example



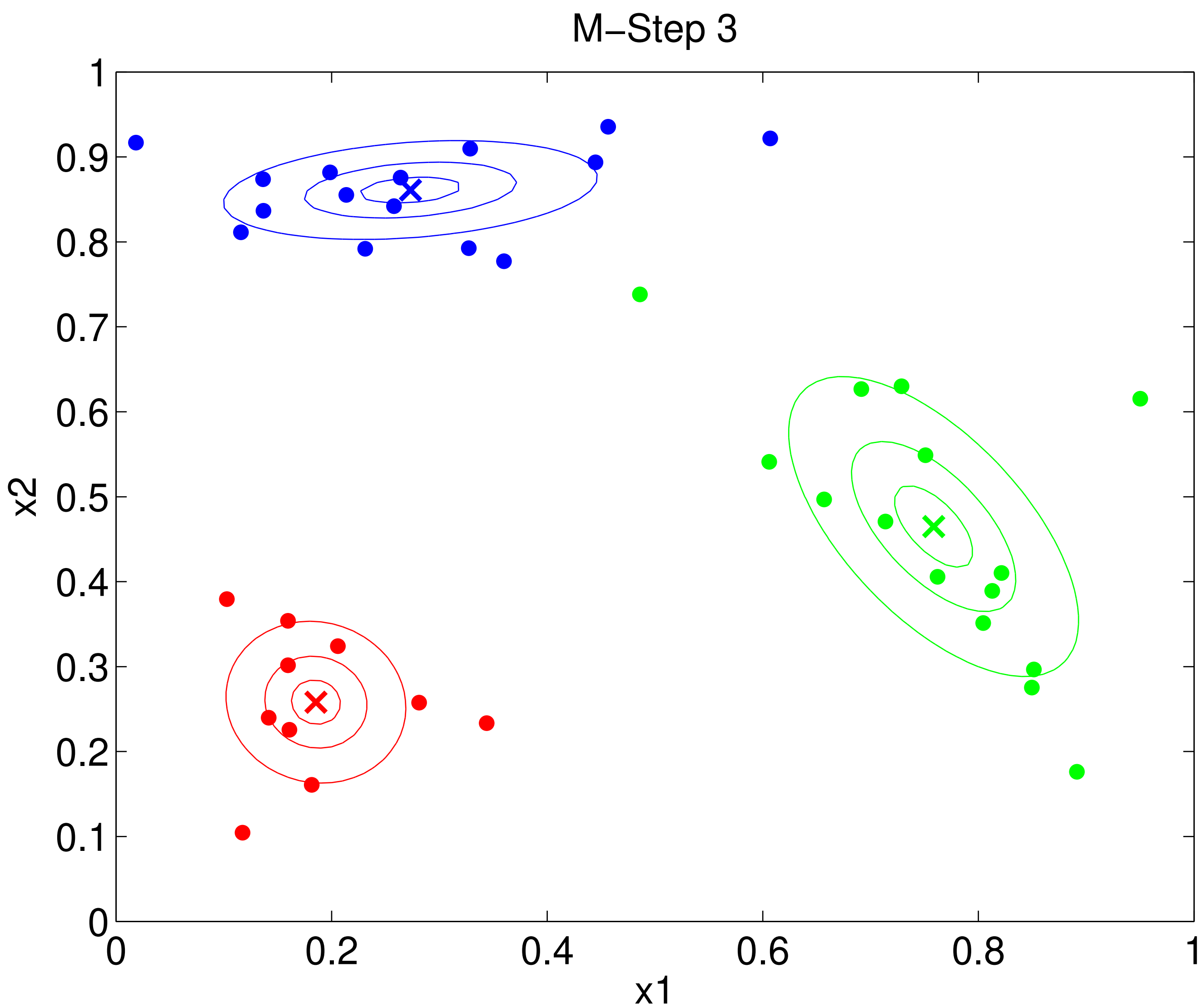
Hard EM for Mixture of Gaussians: Example



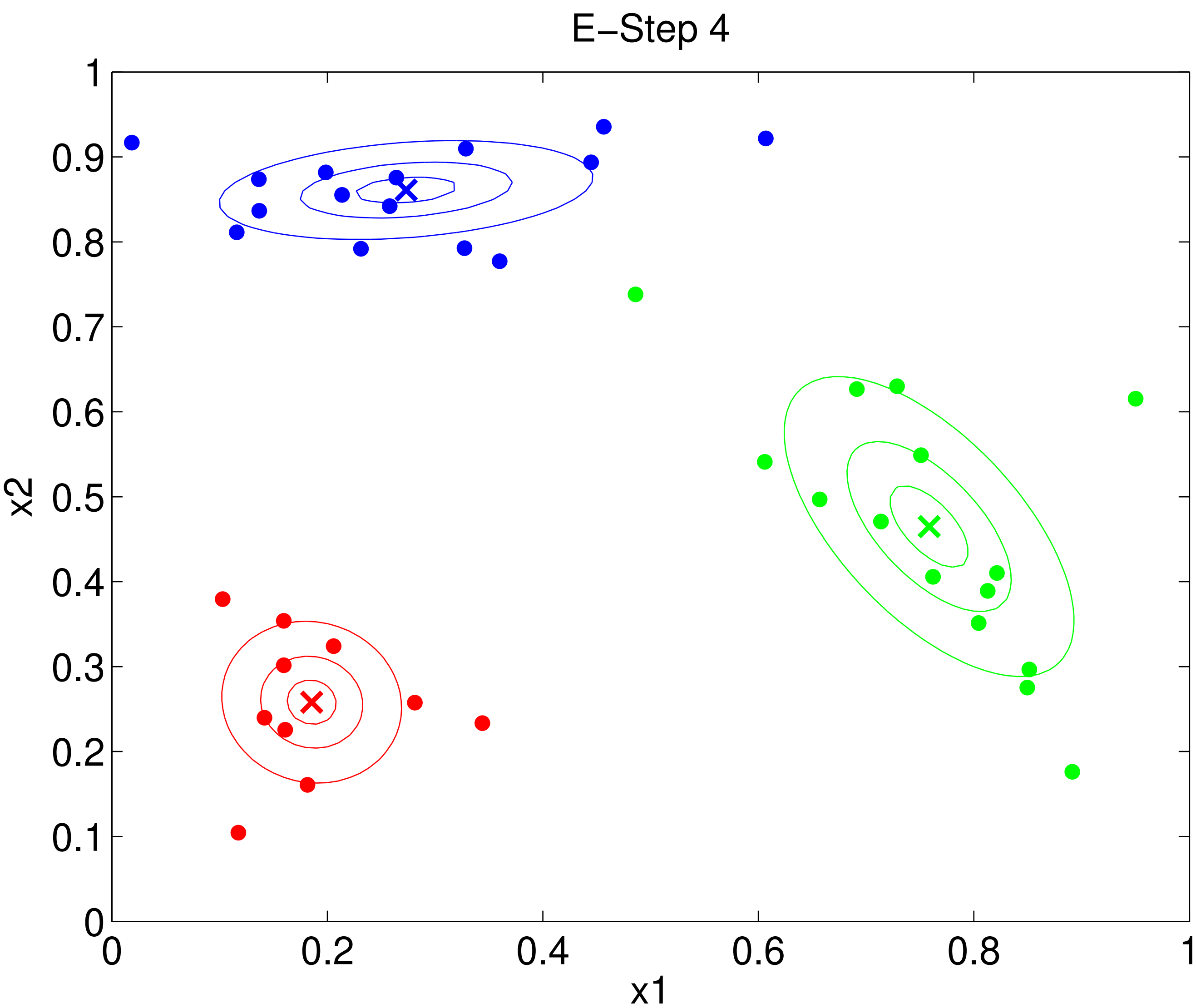
Hard EM for Mixture of Gaussians: Example



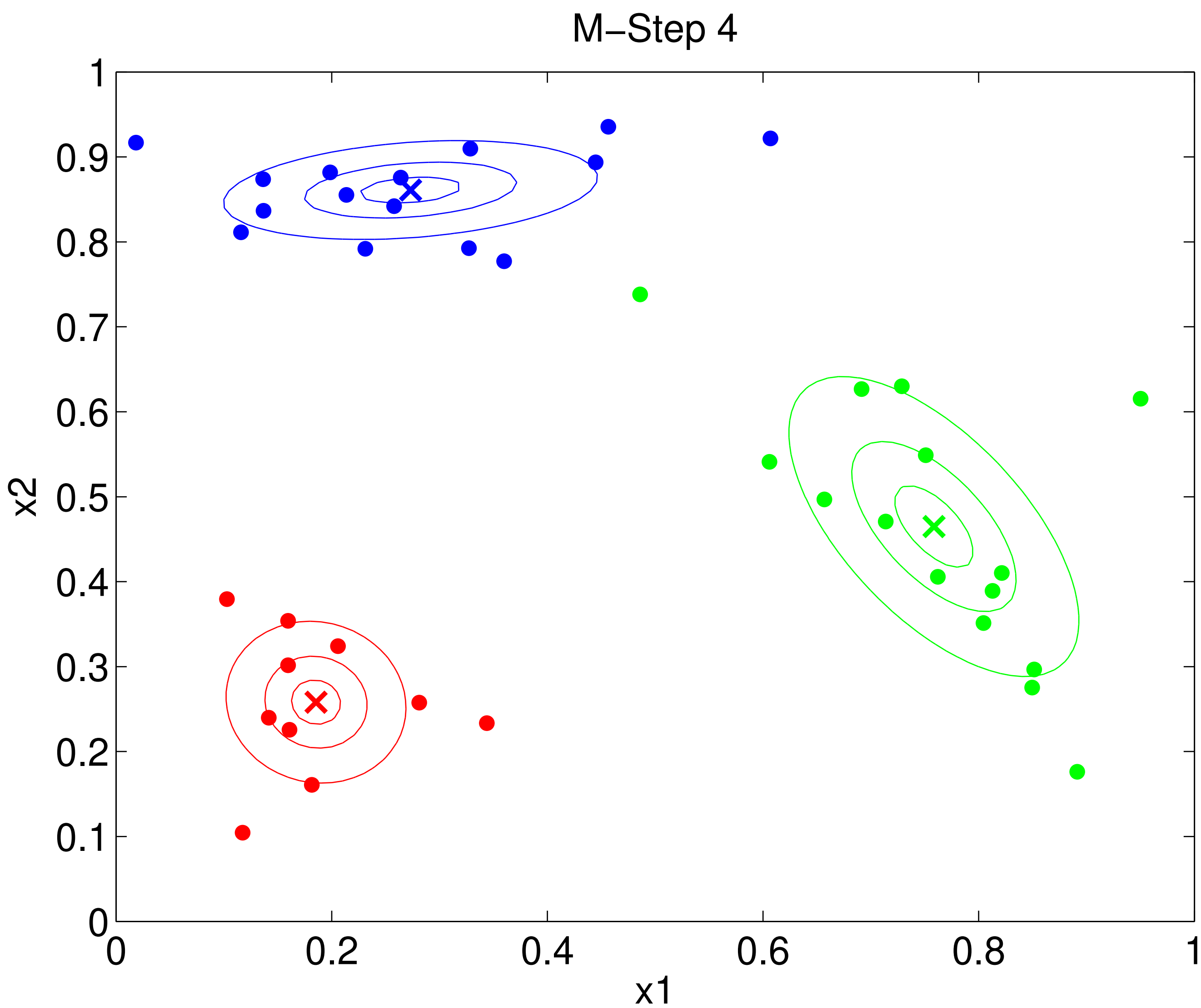
Hard EM for Mixture of Gaussians: Example



Hard EM for Mixture of Gaussians: Example



Hard EM for Mixture of Gaussians: Example



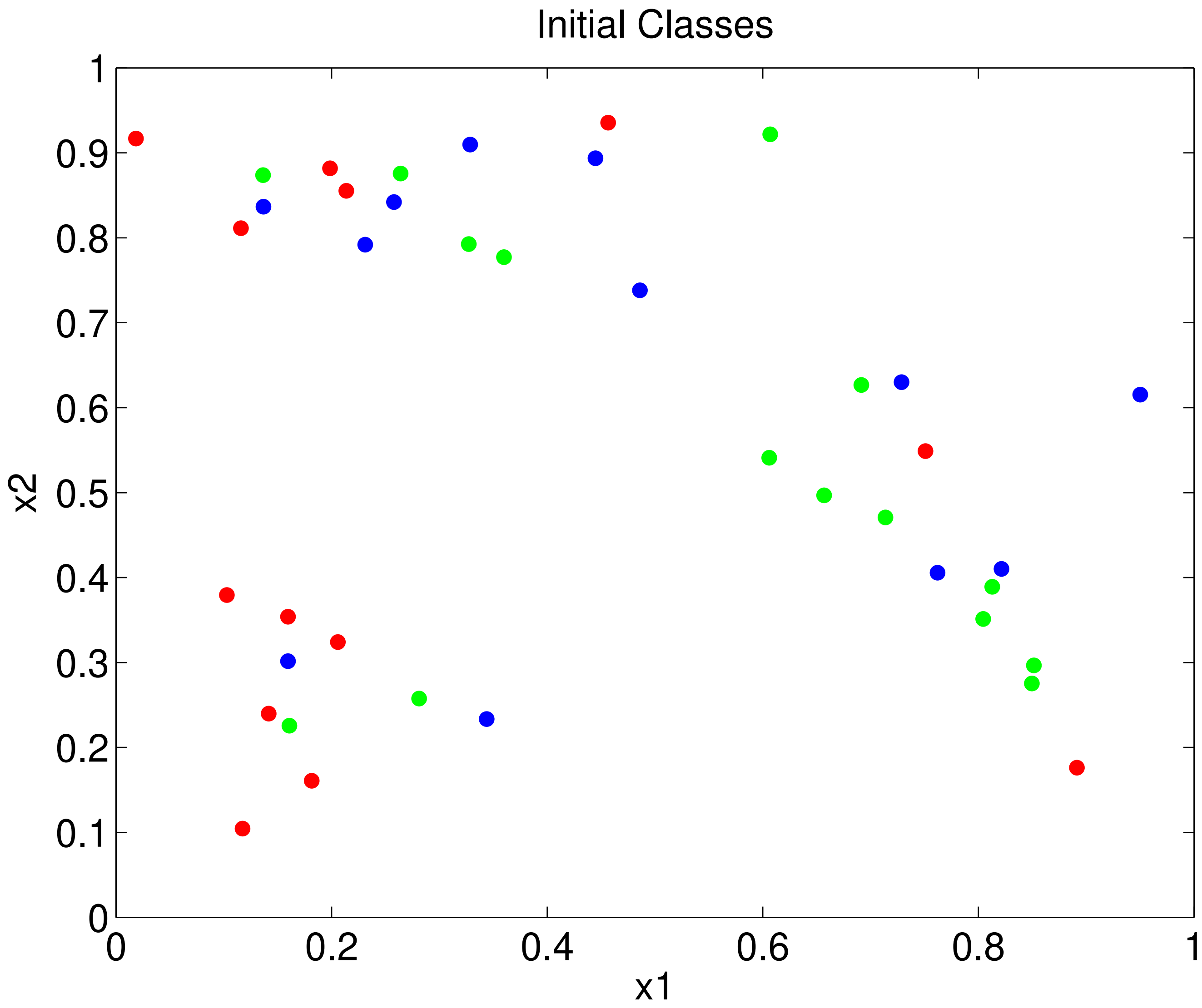
Soft EM for Mixture of Gaussians

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:
   $$w_{ij} = P(y_i = k|\mathbf{x_i}) = E(\delta_{ij} = \frac{P(\mathbf{x_i}|y_i = j)P(y_i = j)}{P(\mathbf{x_i})}$$
   (b) *M-step:* Update the parameters of the model to maximize the likelihood of the data
   $$p_j = \frac{1}{m}\sum_{i=1}^{m} w_{ij} \qquad \mu_j = \frac{\sum_{i=1}^{m} w_{ij}\mathbf{x}_i}{\sum_{i=1}^{m} w_{ij}}$$
   $$\Sigma_j = \frac{\sum_{i=1}^{m} w_{ij}\left(\mathbf{x}_i - \mu_j\right)\left(\mathbf{x}_i - \mu_j\right)^T}{\sum_{i=1}^{m} w_{ij}}$$

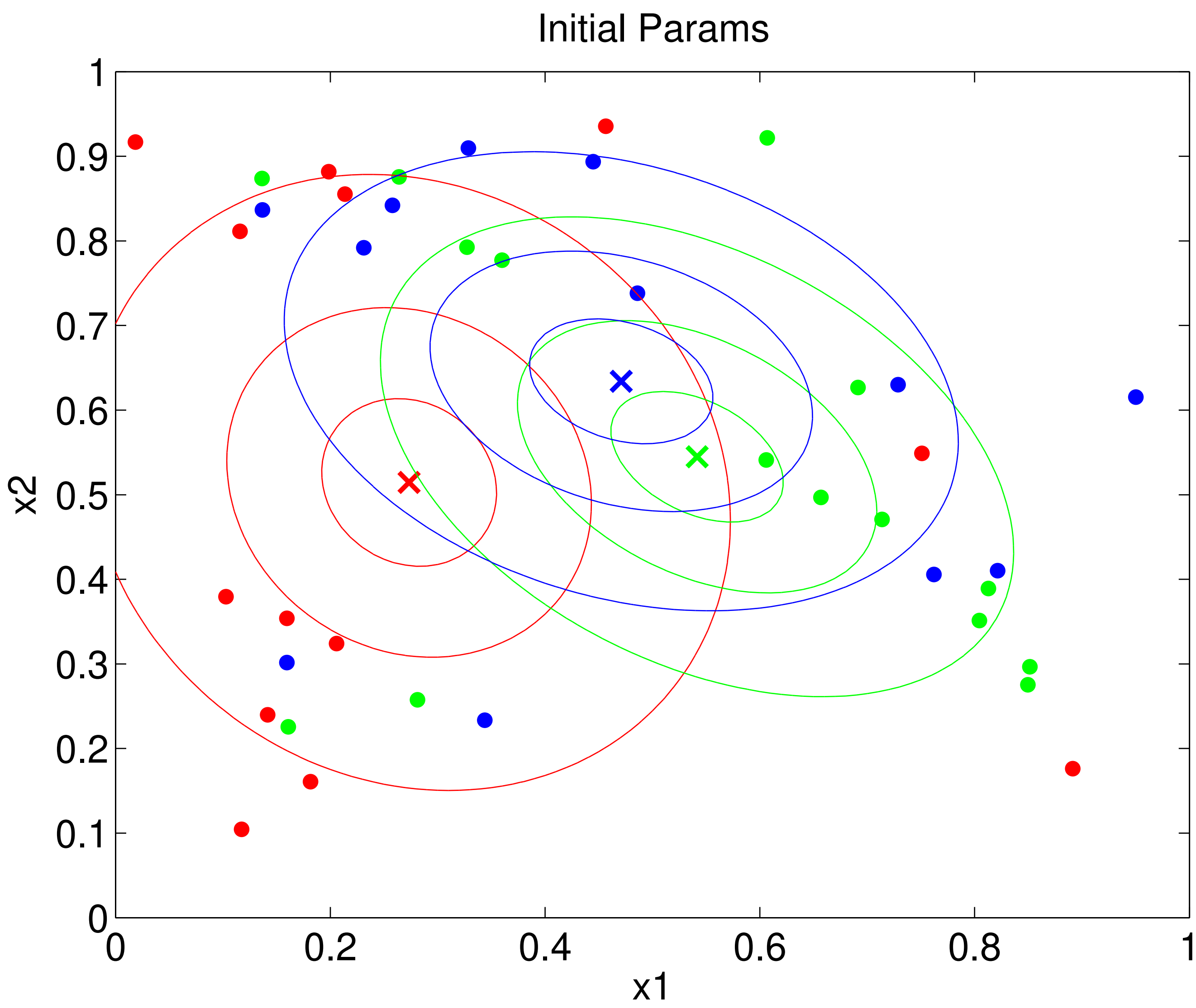
Soft EM for Mixture of Gaussians: Example

- Soft EM with $K = 3$
- Example data from start of lectures
- Initial classes assigned randomly
- Initial parameters estimated from initial classes

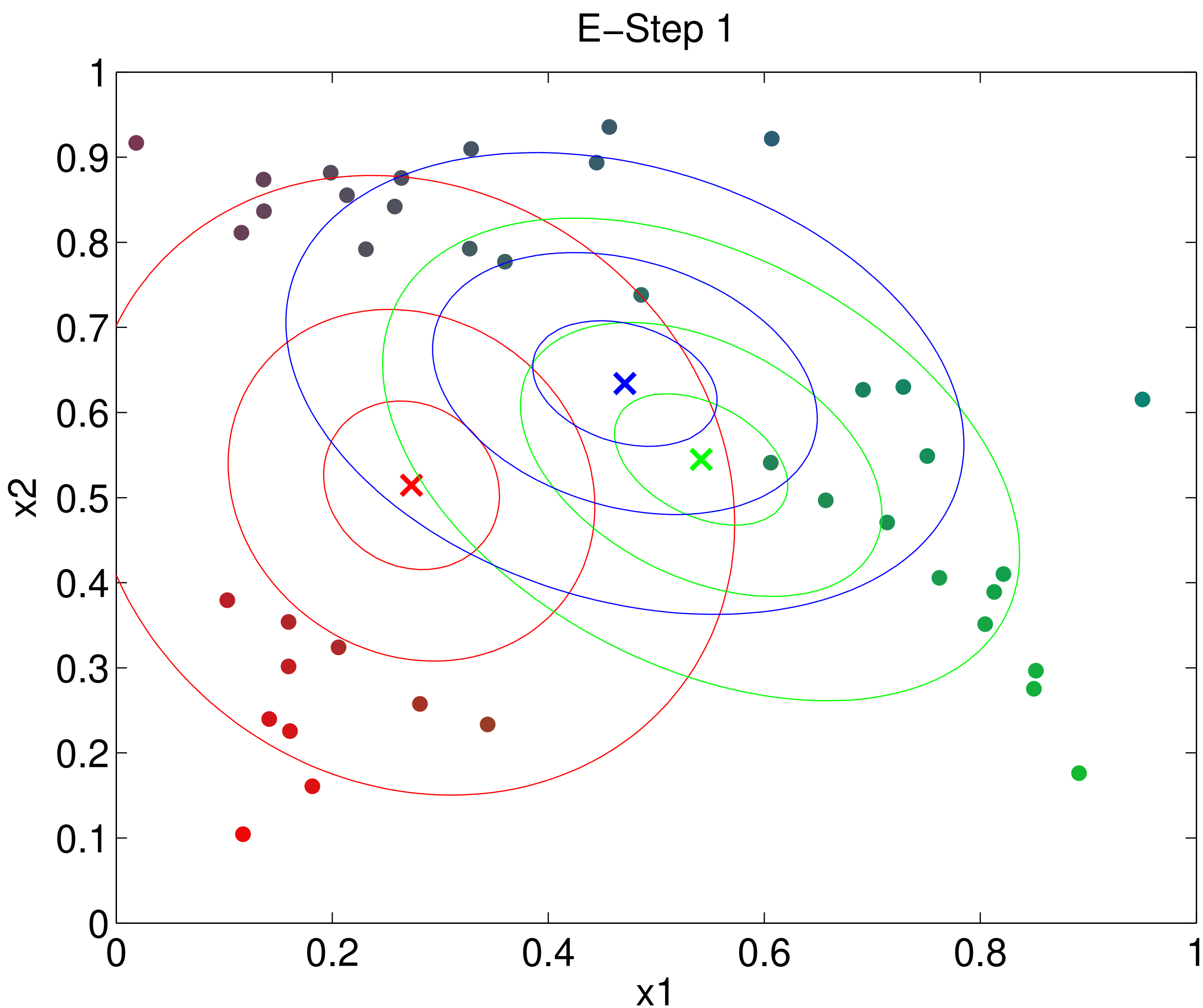
Soft EM for Mixture of Gaussians: Example



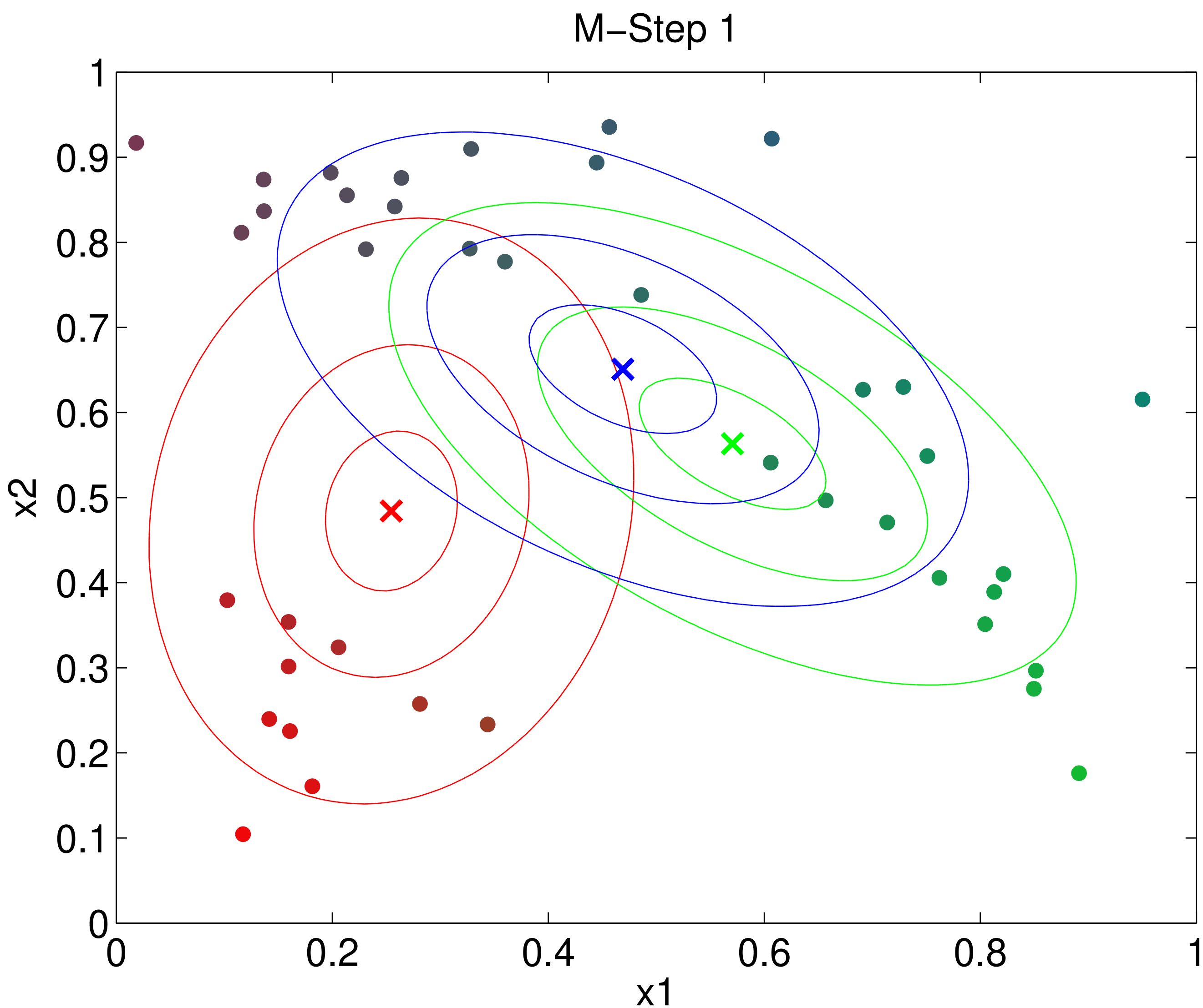
Soft EM for Mixture of Gaussians: Example



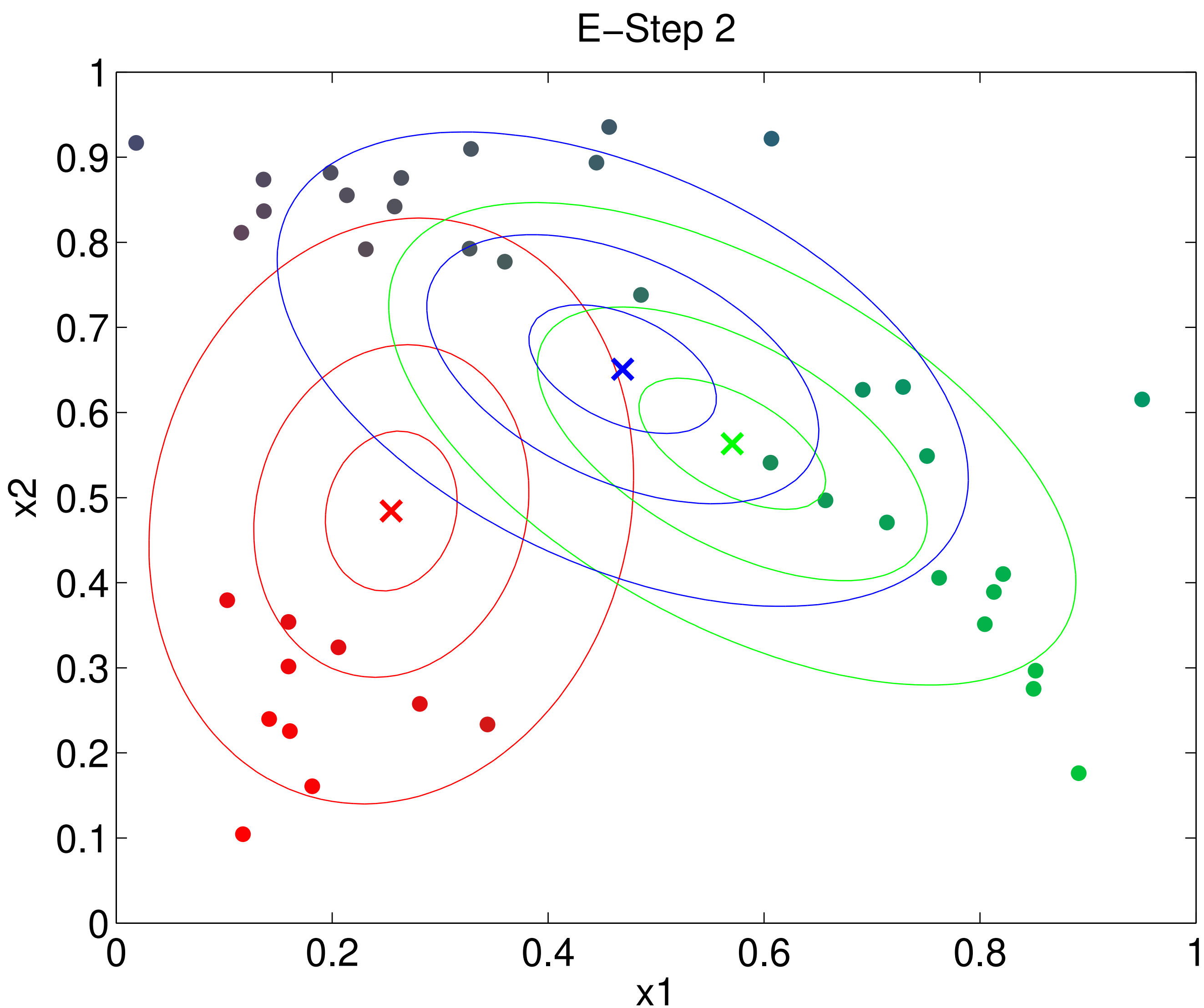
Soft EM for Mixture of Gaussians: Example



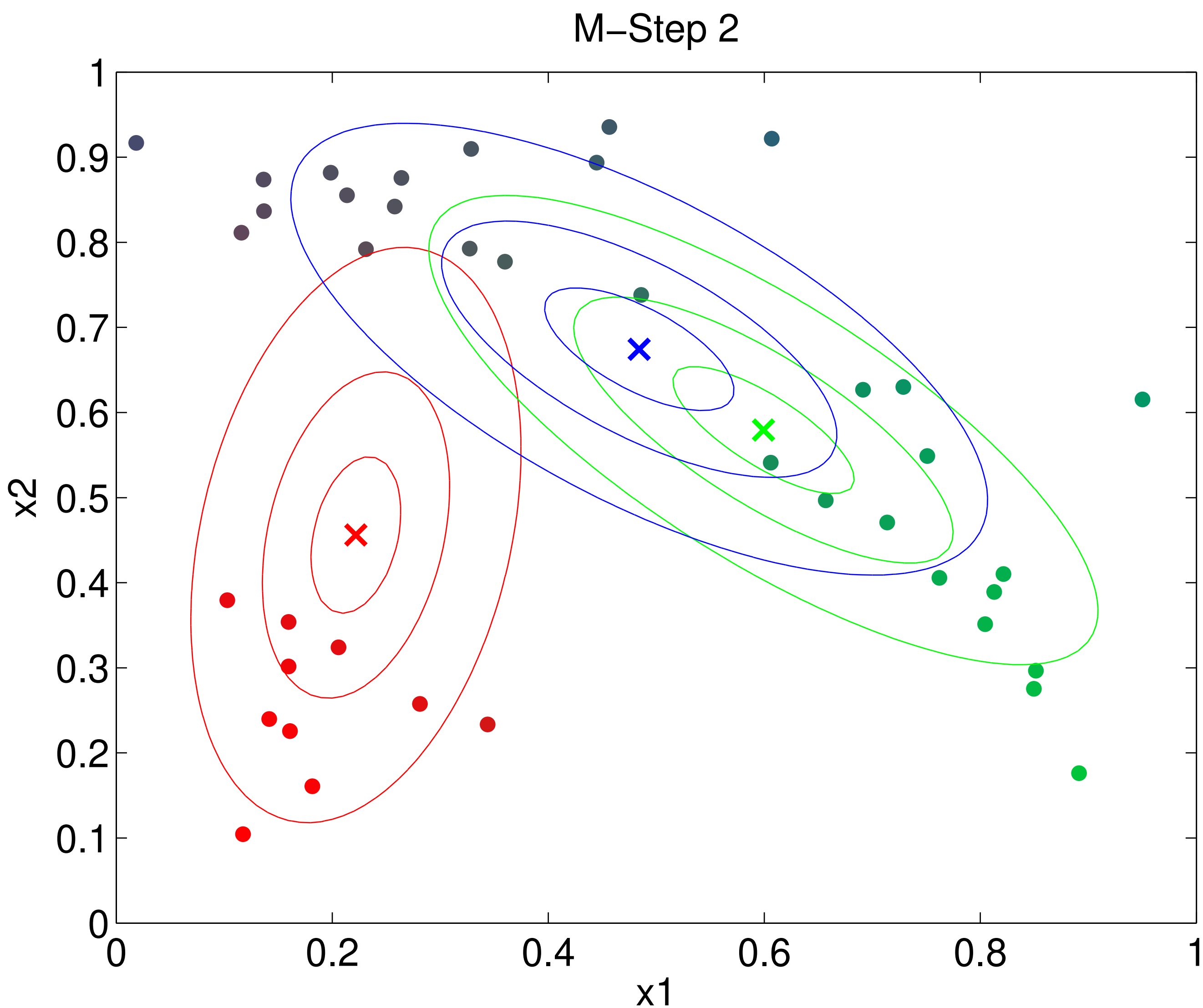
Soft EM for Mixture of Gaussians: Example



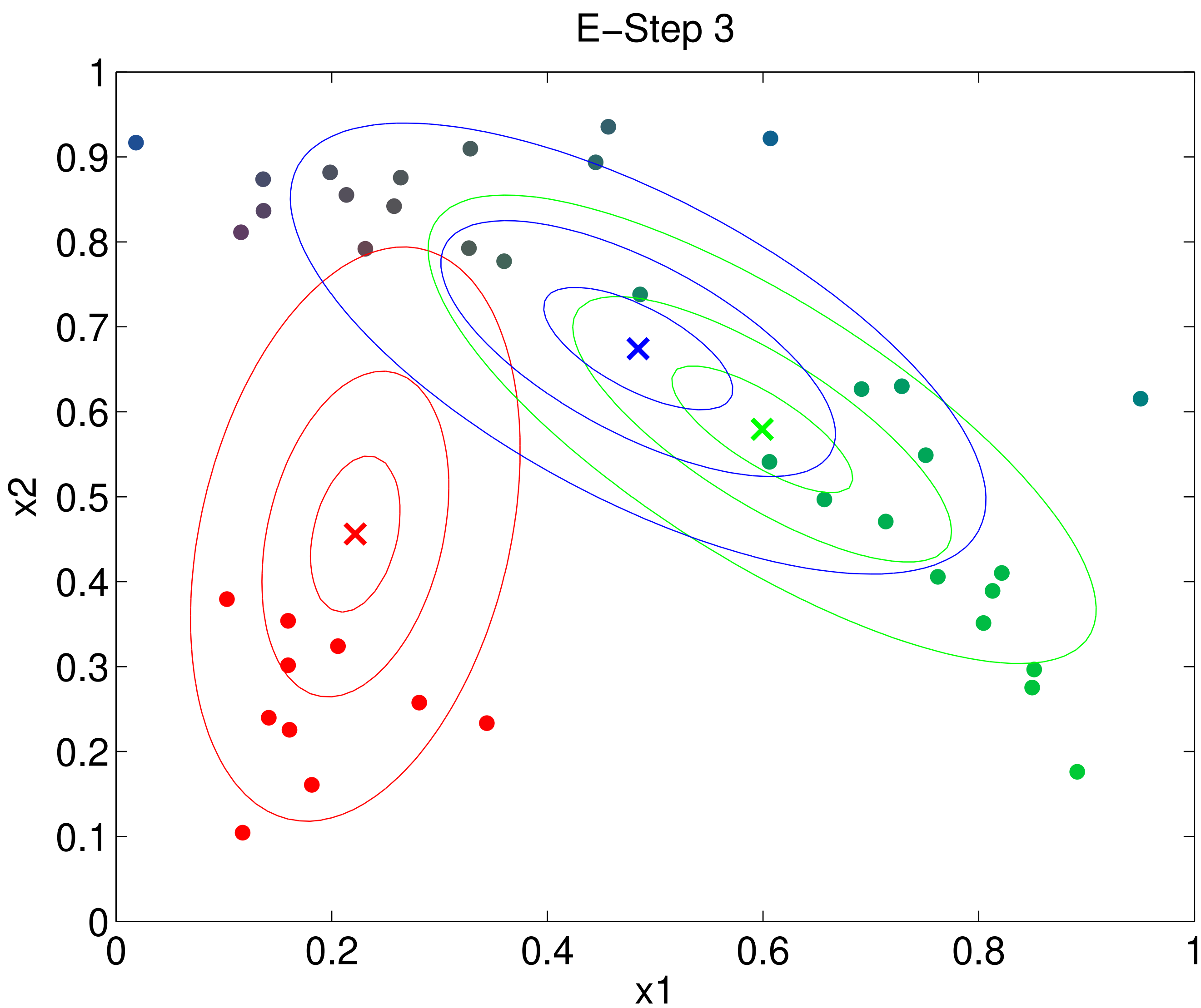
Soft EM for Mixture of Gaussians: Example



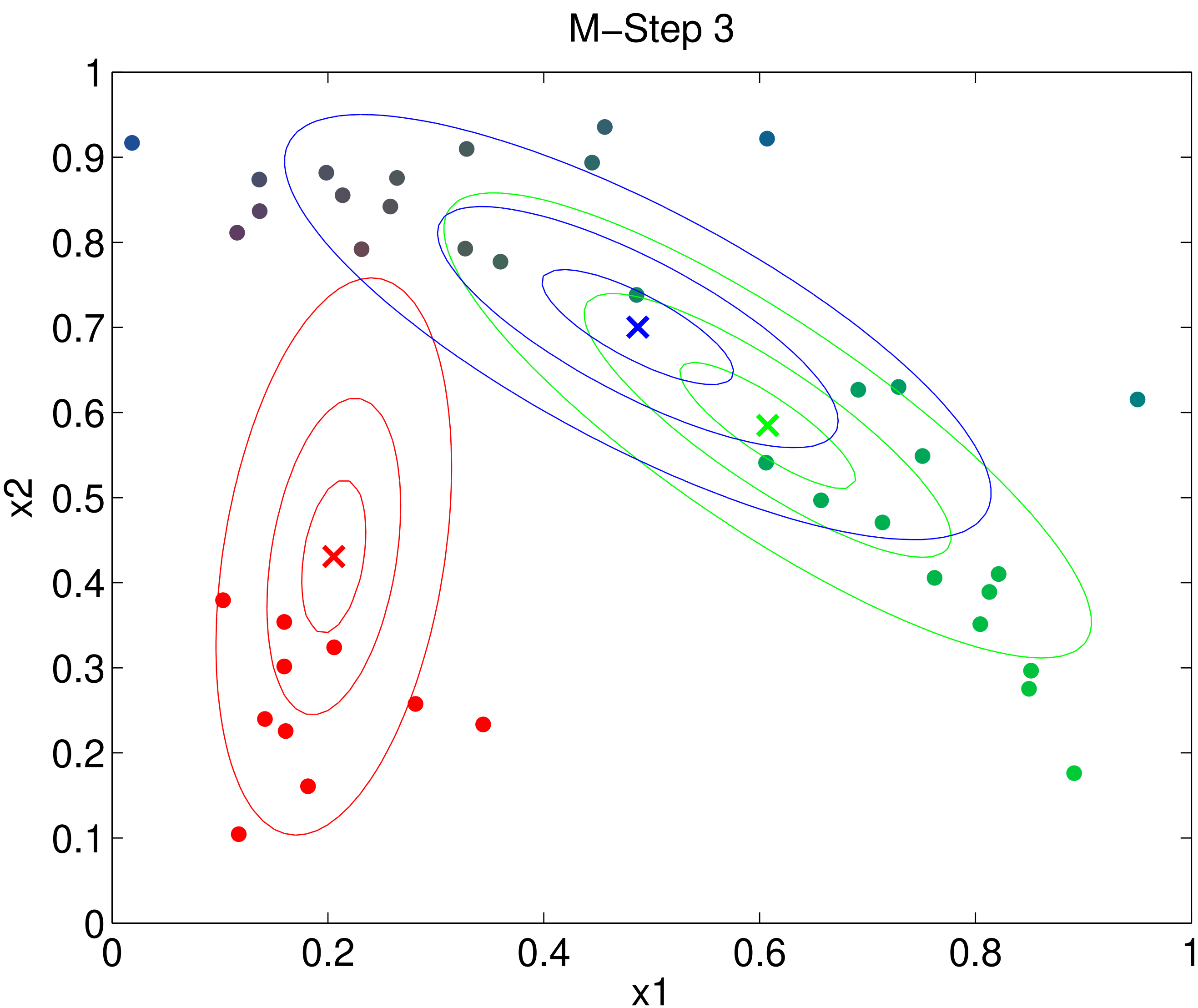
Soft EM for Mixture of Gaussians: Example



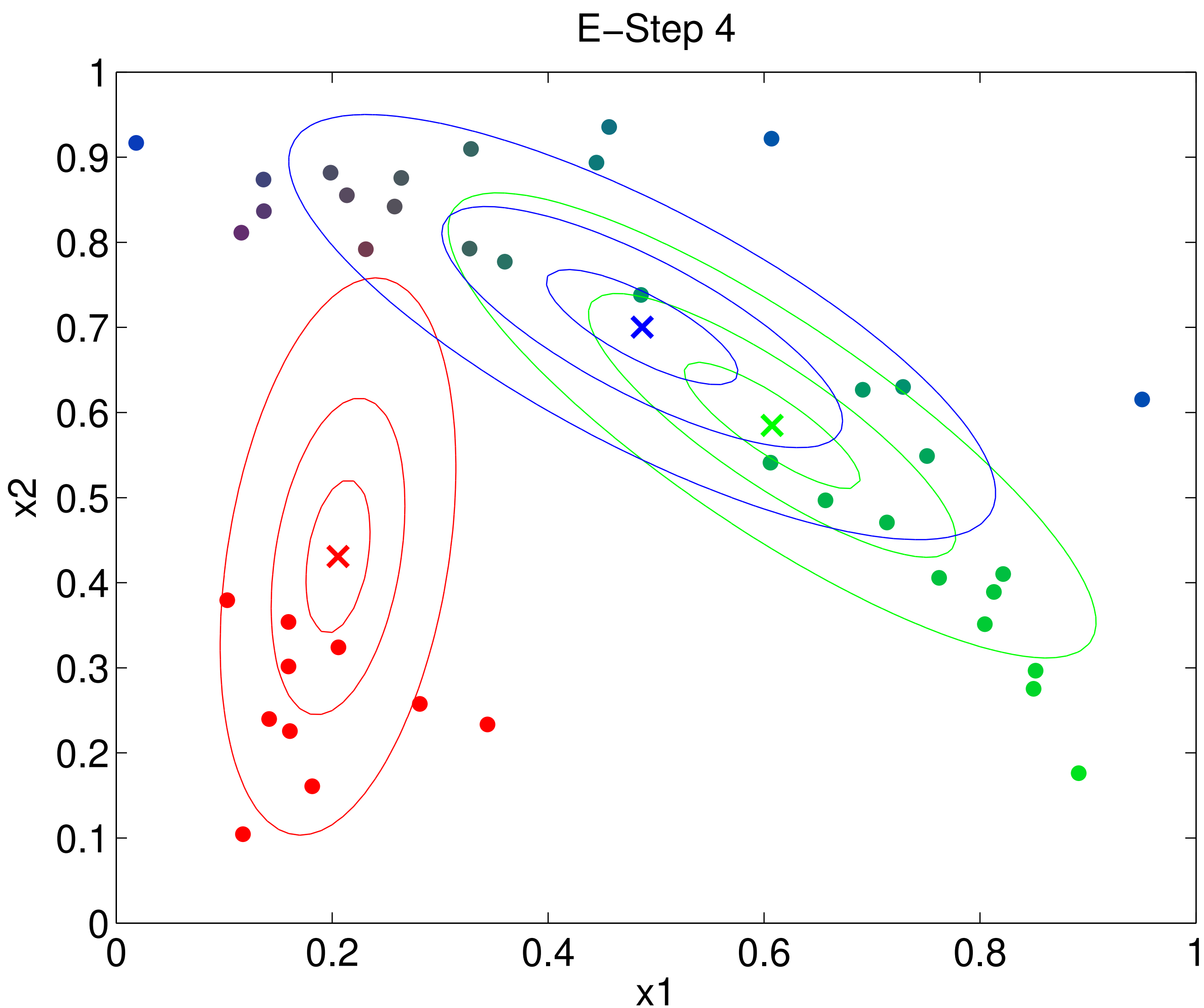
Soft EM for Mixture of Gaussians: Example



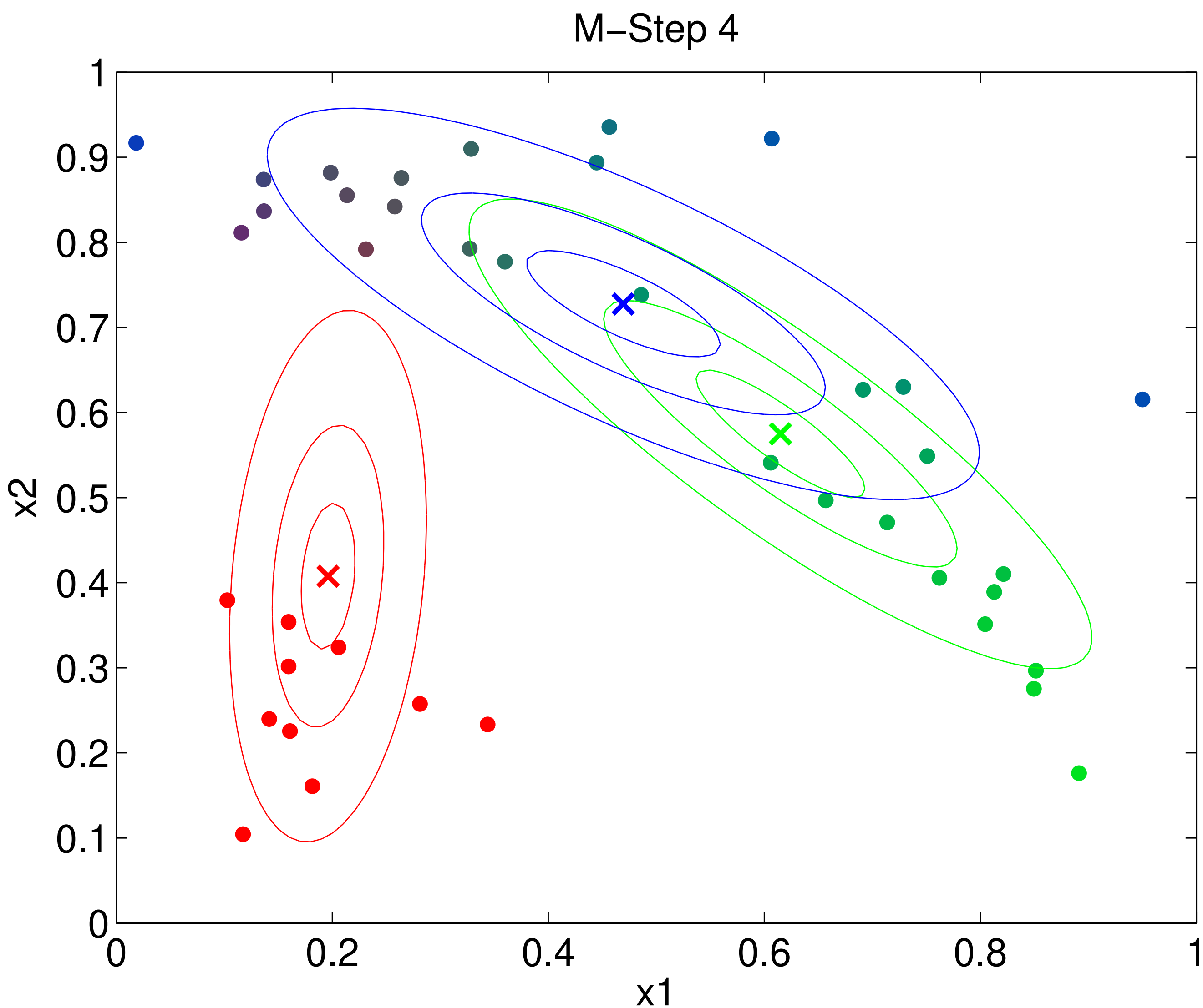
Soft EM for Mixture of Gaussians: Example



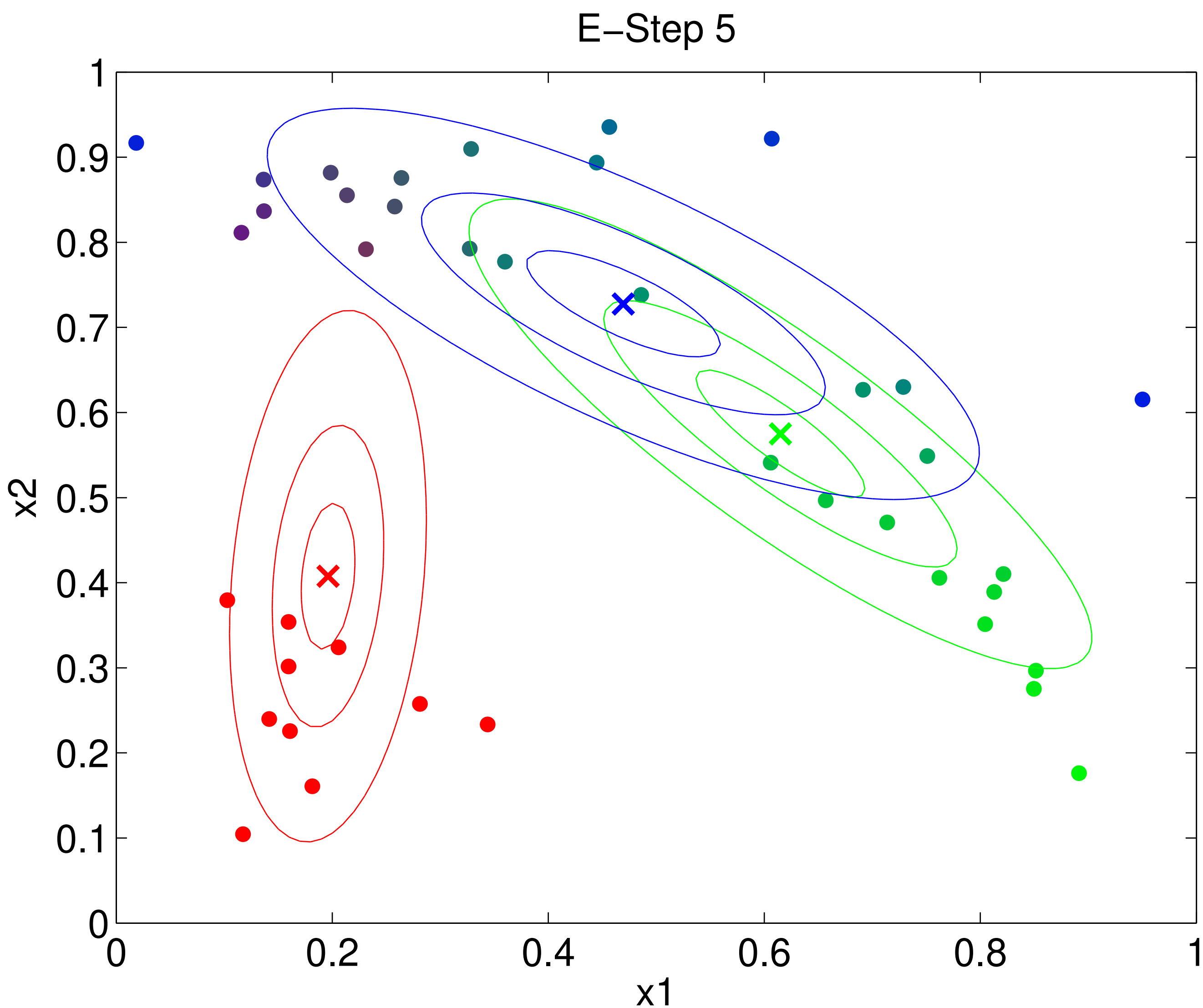
Soft EM for Mixture of Gaussians: Example



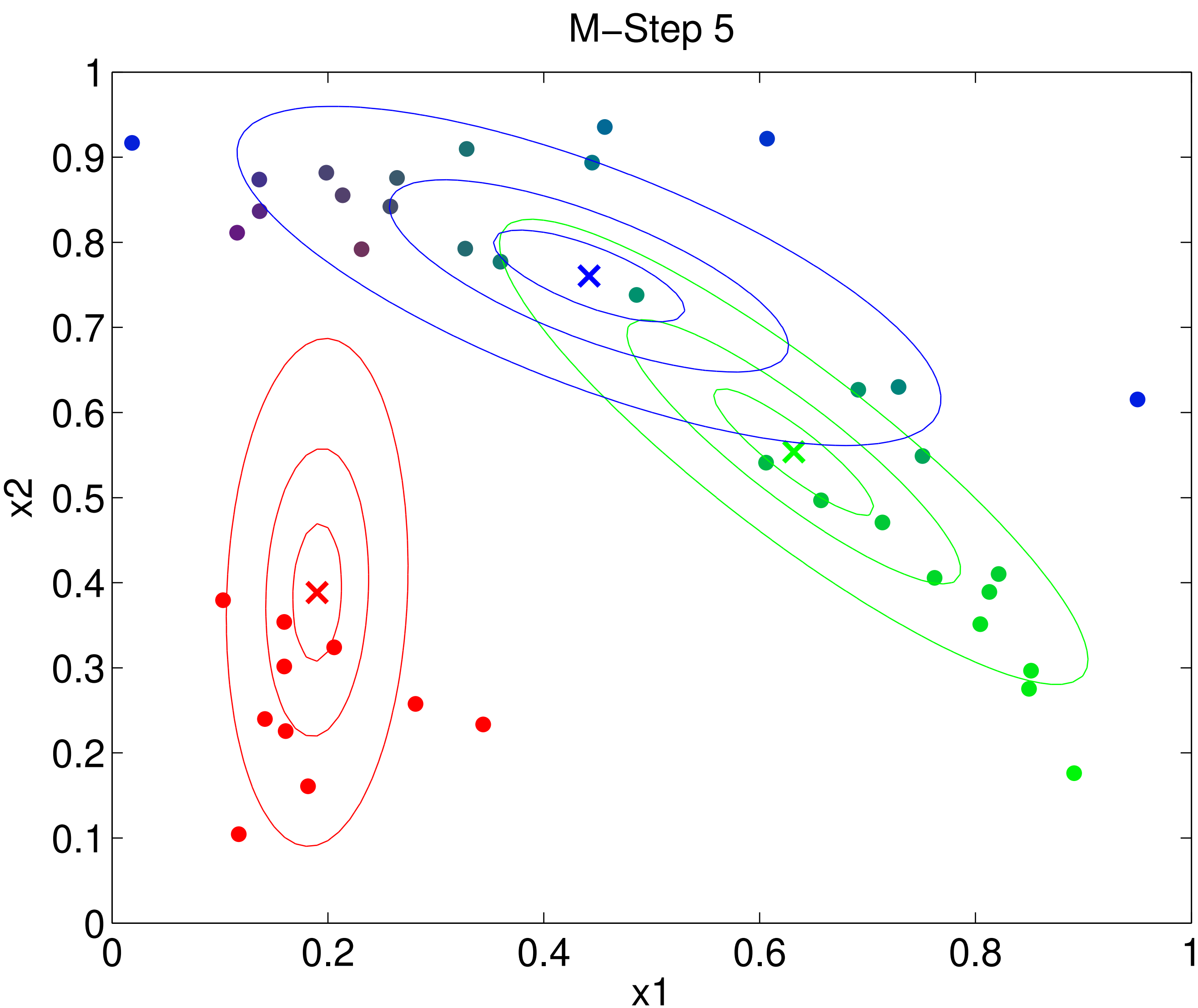
Soft EM for Mixture of Gaussians: Example



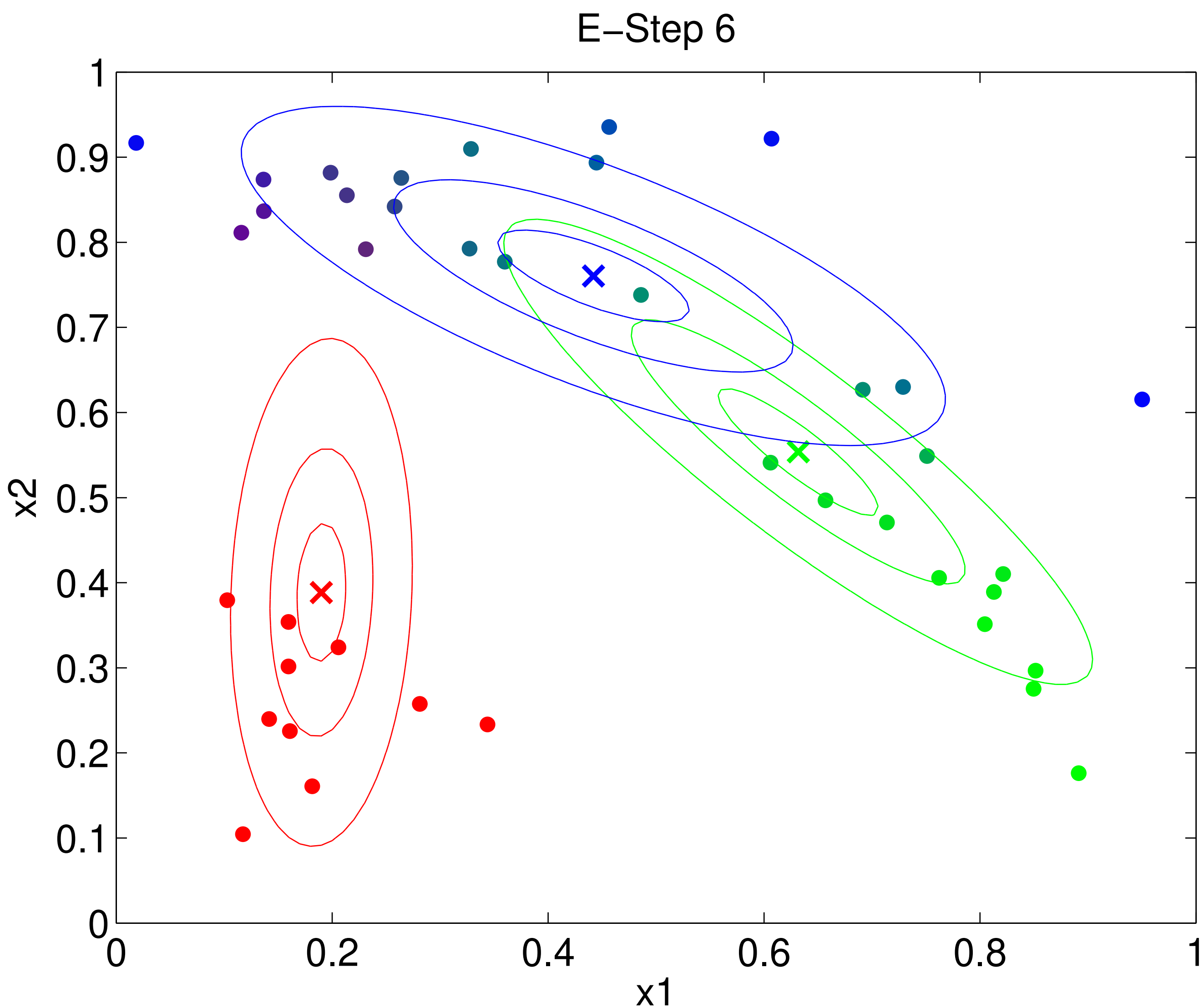
Soft EM for Mixture of Gaussians: Example



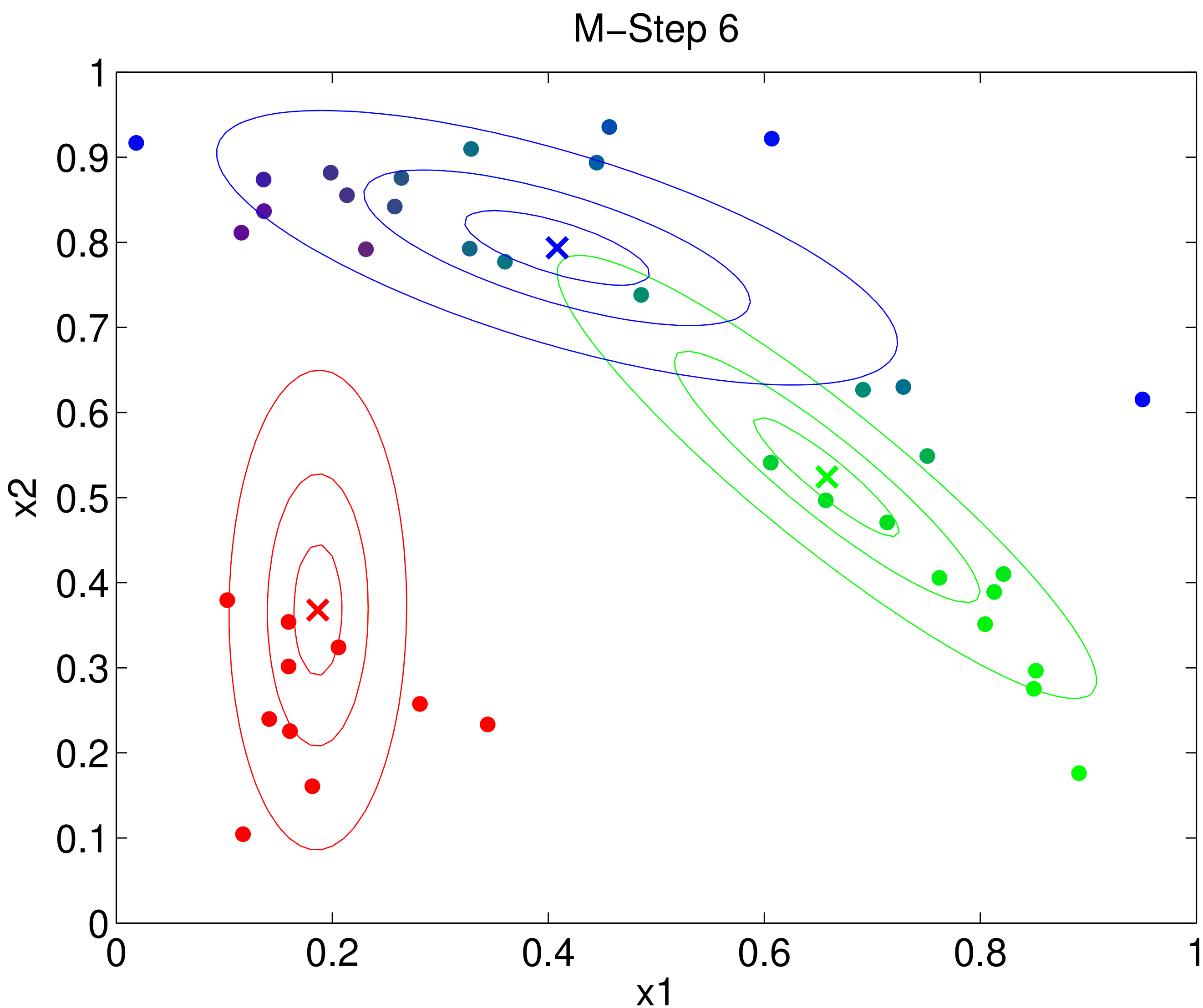
Soft EM for Mixture of Gaussians: Example



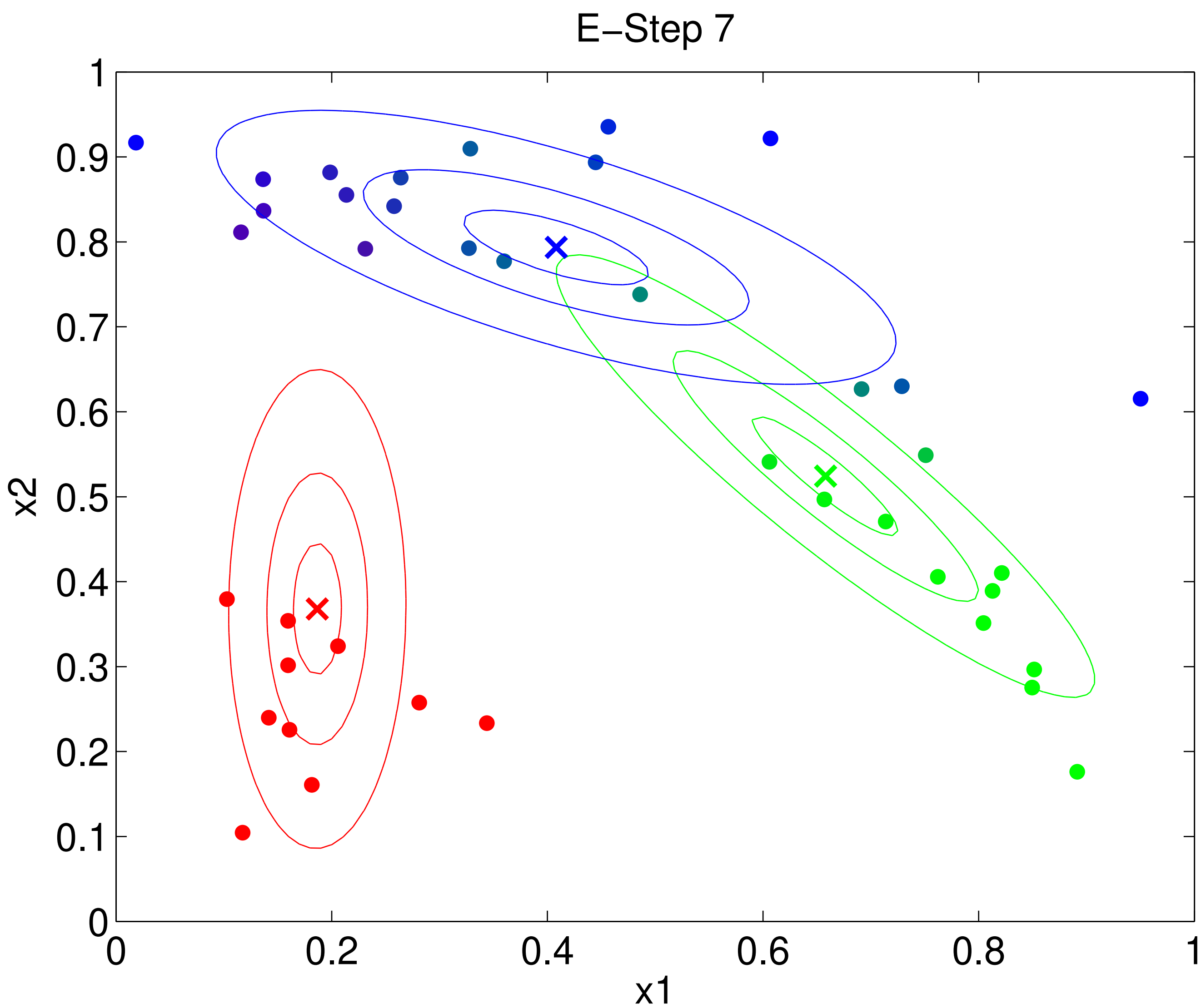
Soft EM for Mixture of Gaussians: Example



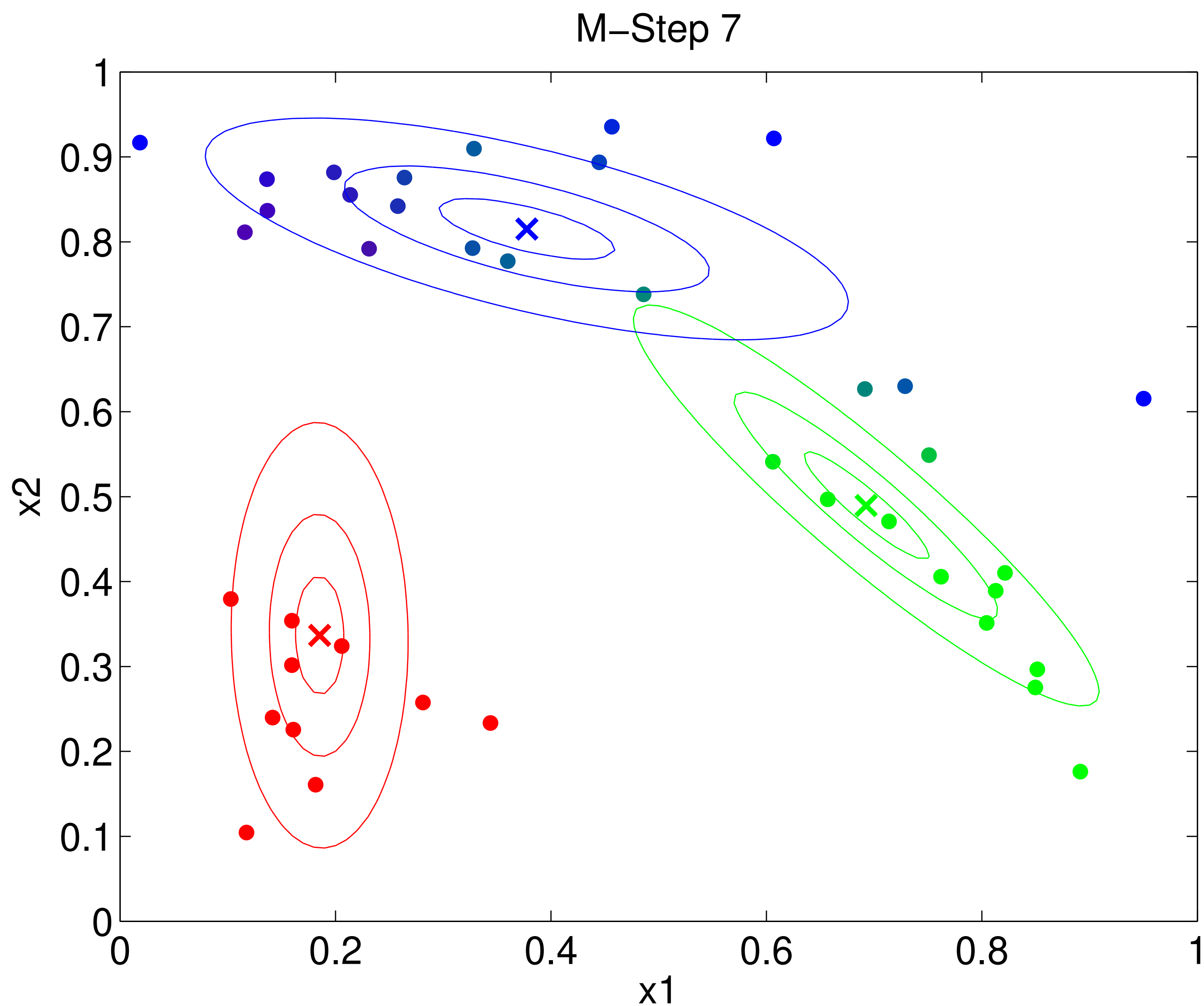
Soft EM for Mixture of Gaussians: Example



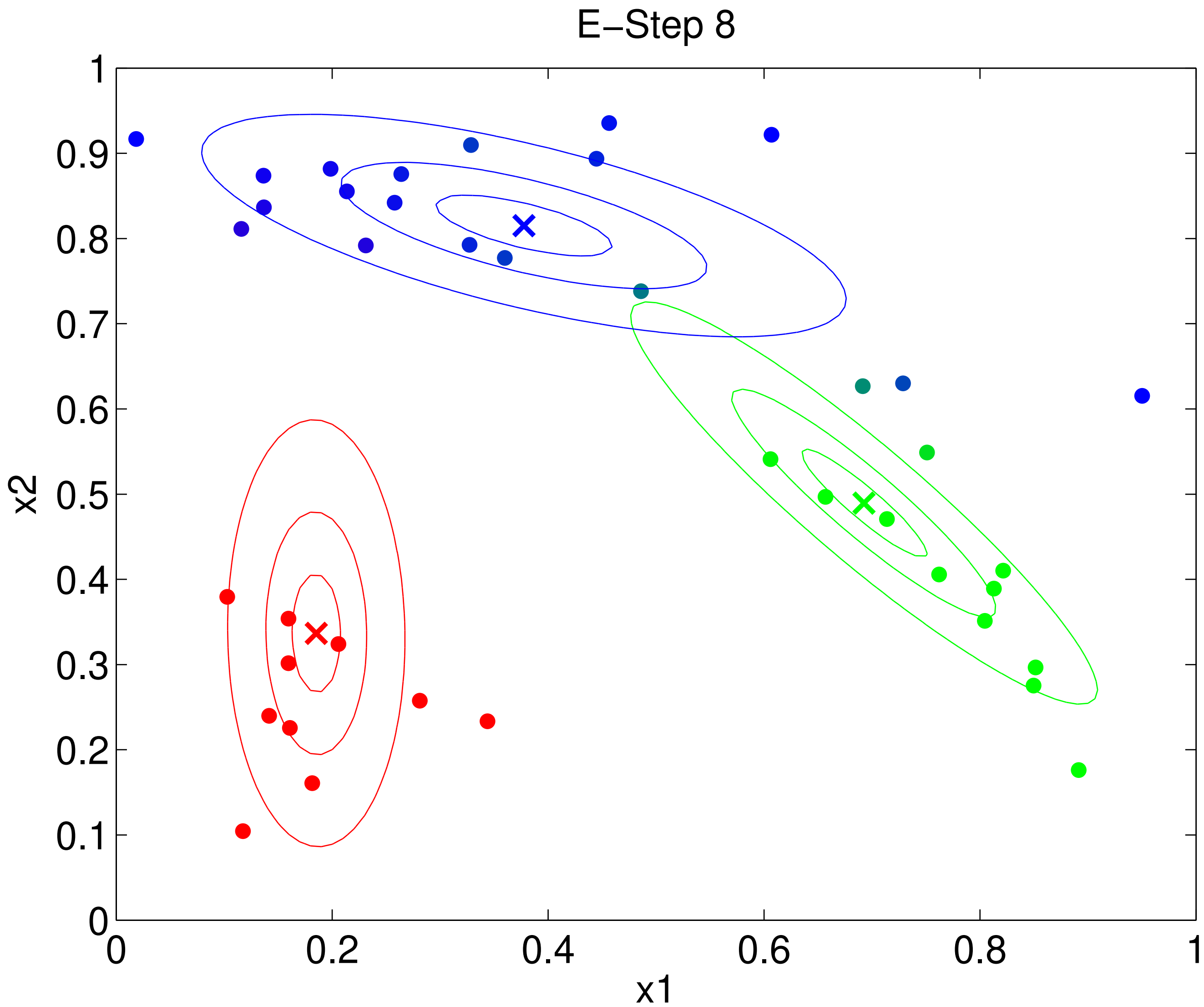
Soft EM for Mixture of Gaussians: Example



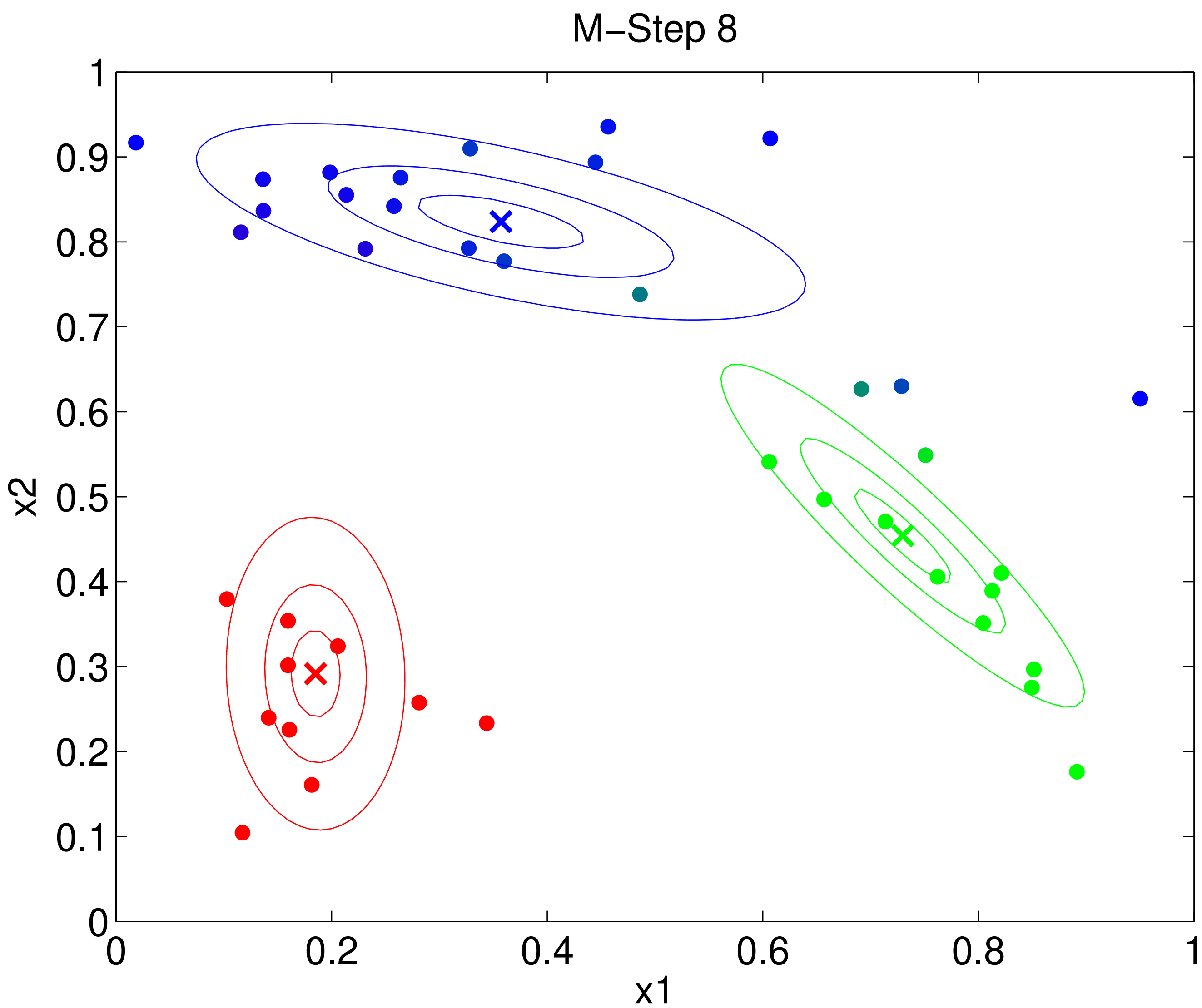
Soft EM for Mixture of Gaussians: Example



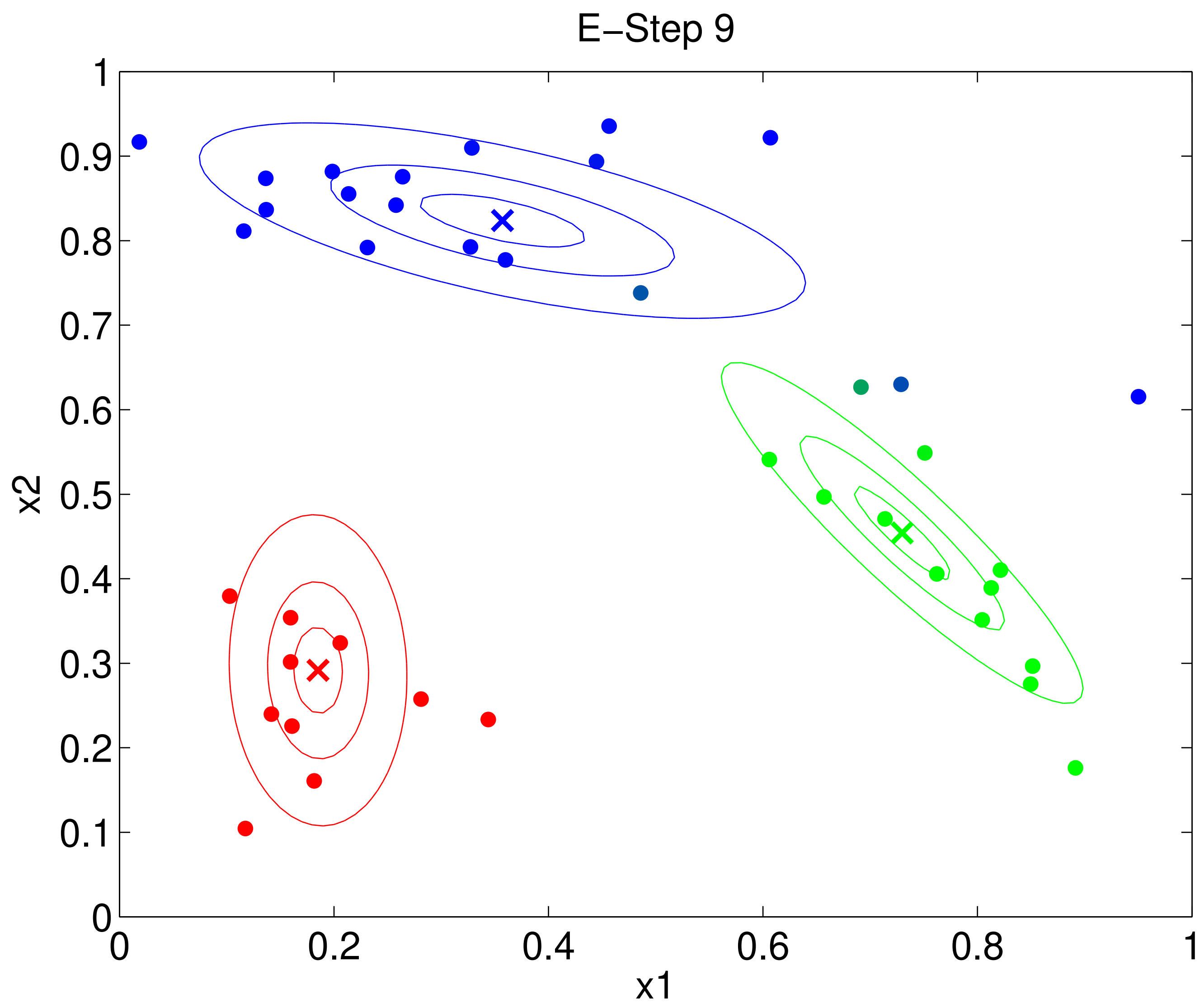
Soft EM for Mixture of Gaussians: Example



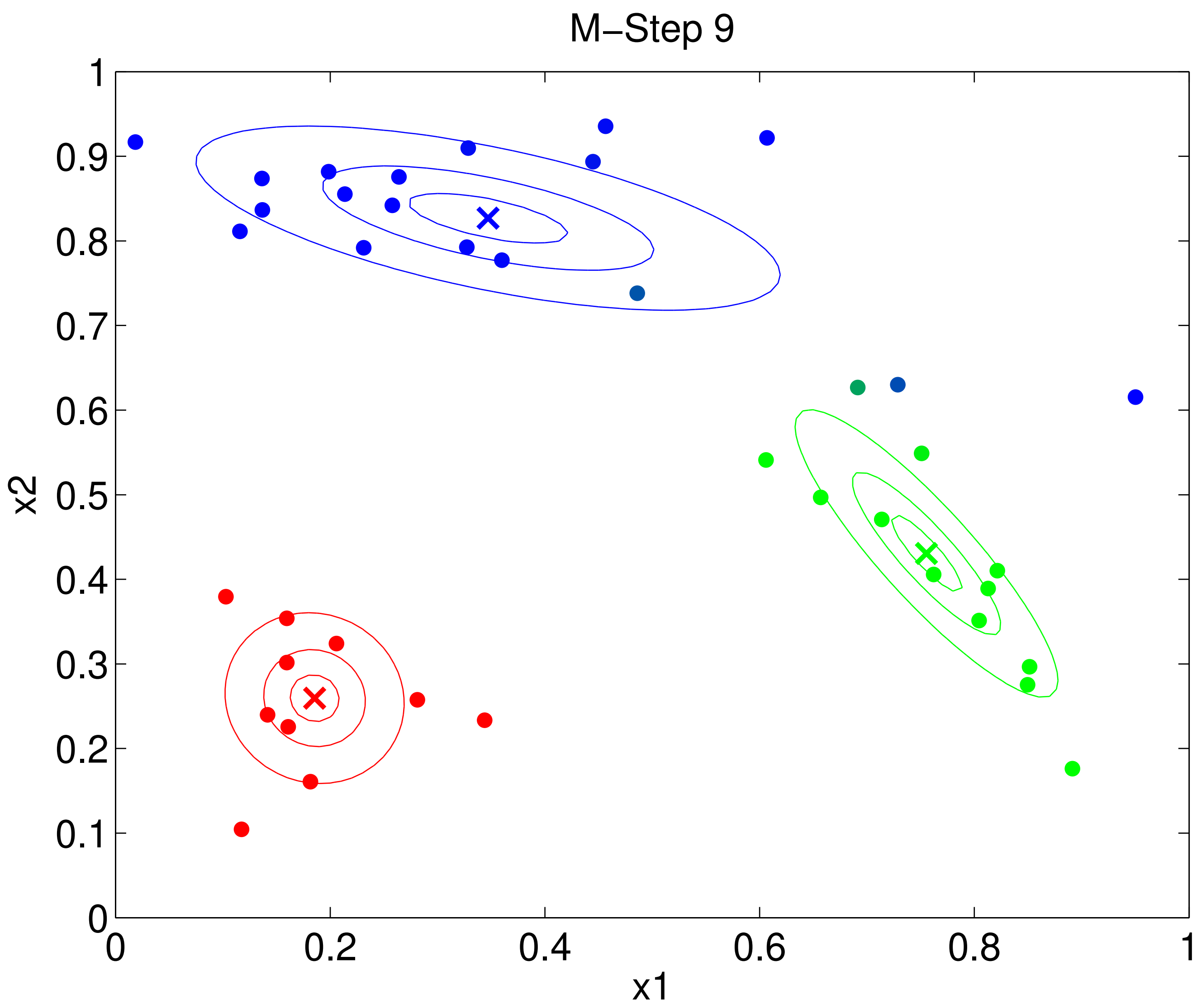
Soft EM for Mixture of Gaussians: Example



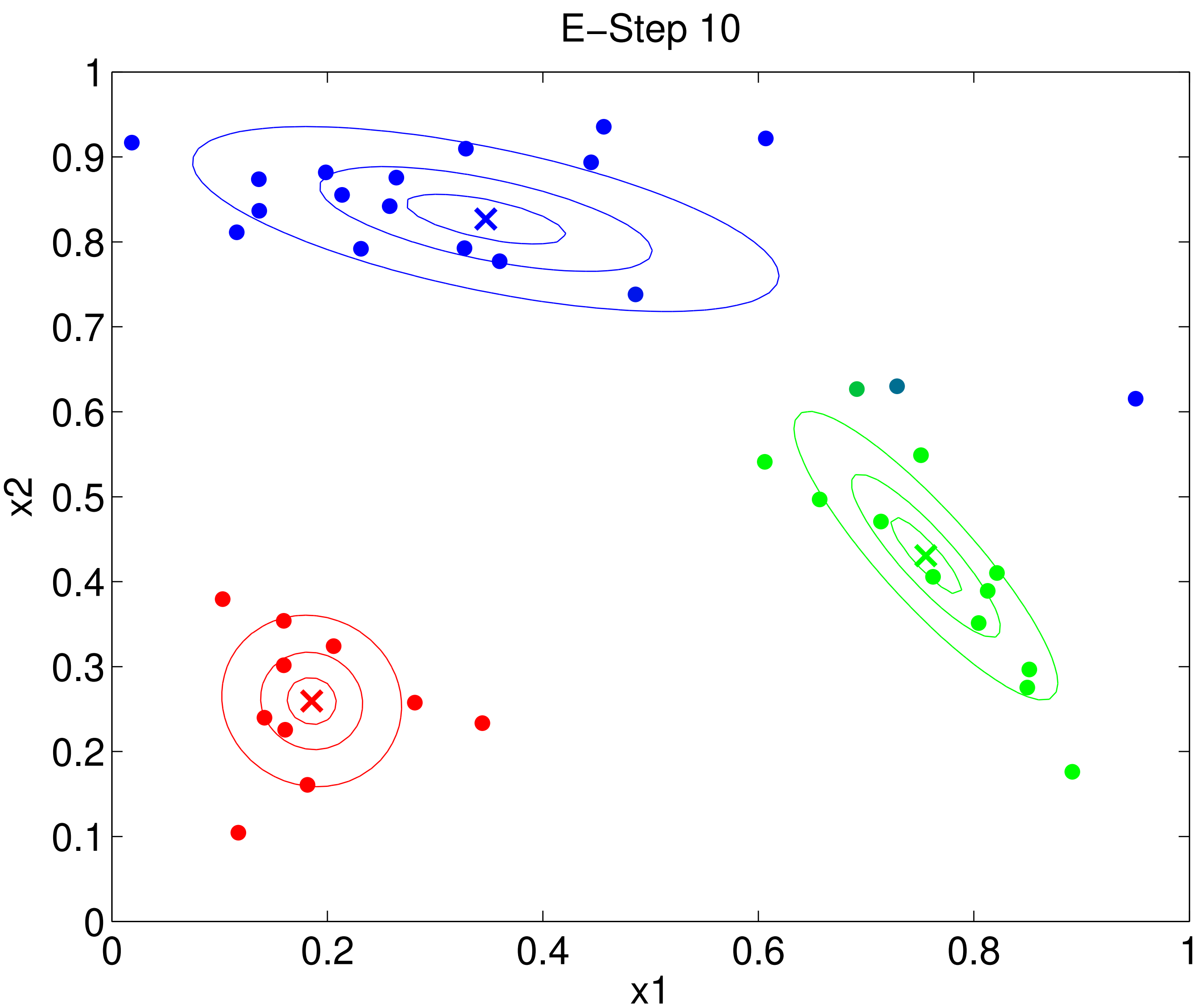
Soft EM for Mixture of Gaussians: Example



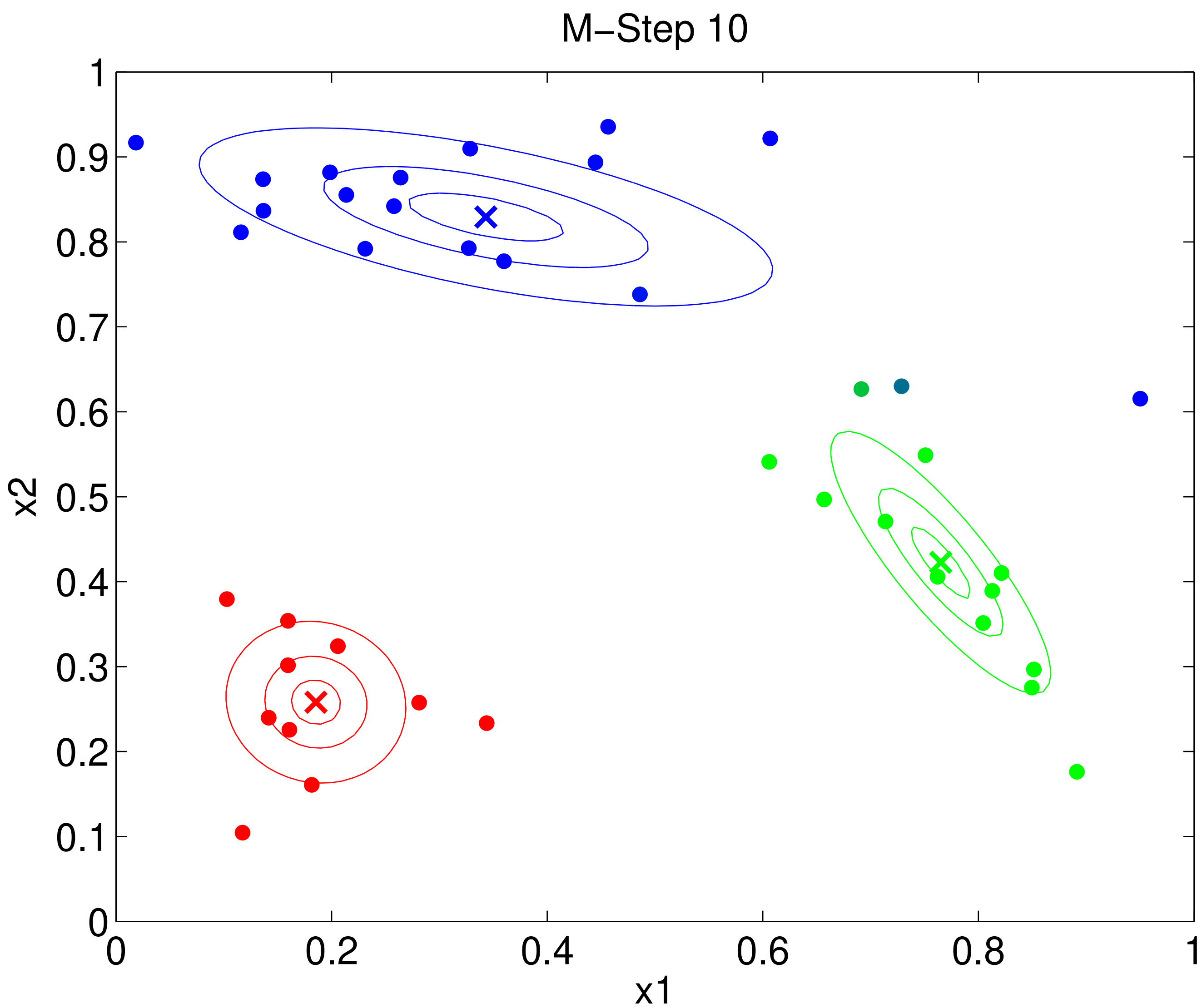
Soft EM for Mixture of Gaussians: Example



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?

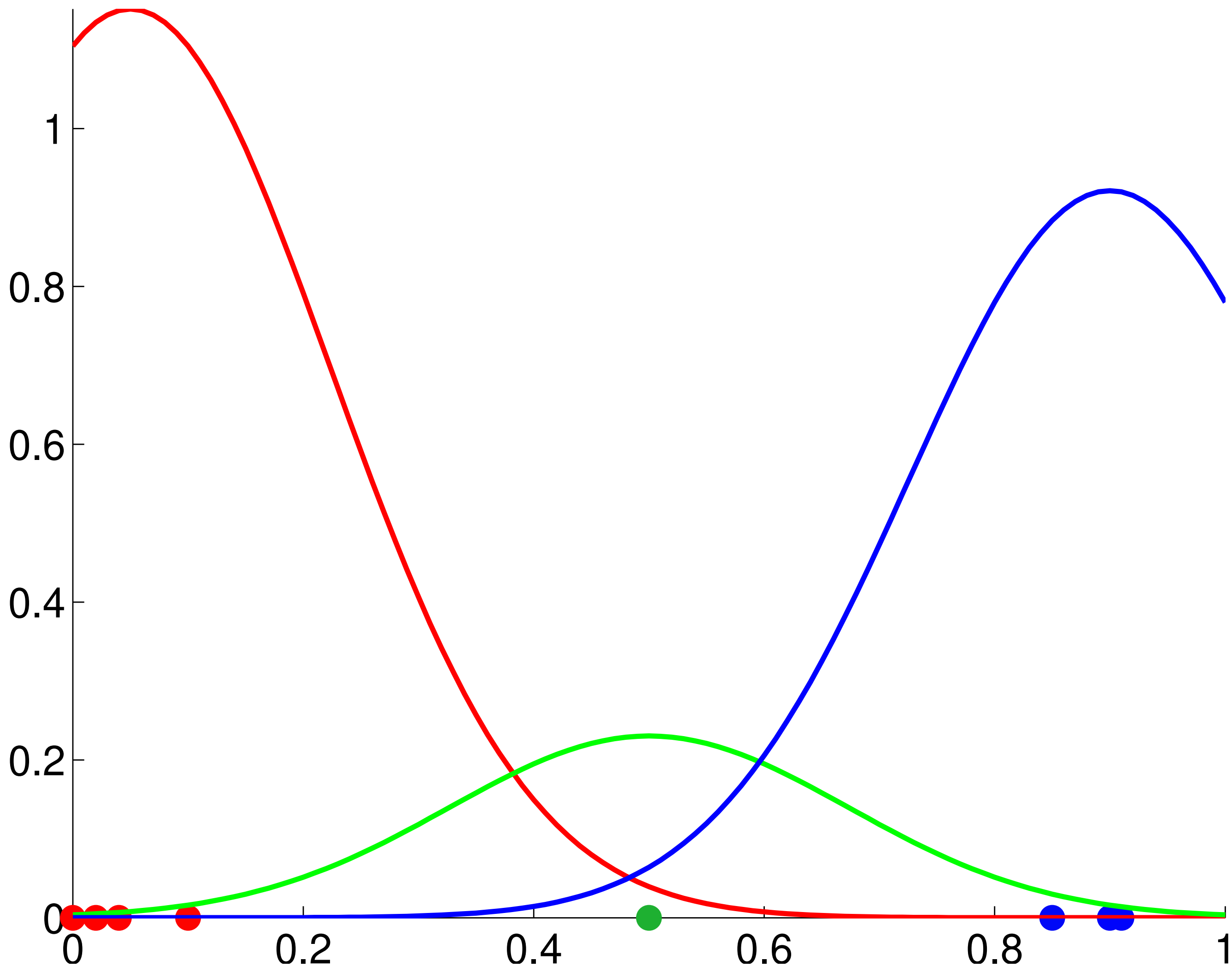
Theoretical properties of EM

- Each iteration improves the likelihood of the data given the class assignments, p_j, μ_j, and Σ_j.
  - Straightforward for Hard EM. Why?
  - Less obvious for Soft EM. Why?
- If the parameters do not change in one iteration, then the gradient of the log-likelihood function is zero.
  This means that we're at a min, max or saddle point.
- In practice, convergence only occurs at local maxima
- Does it necessarily converge?

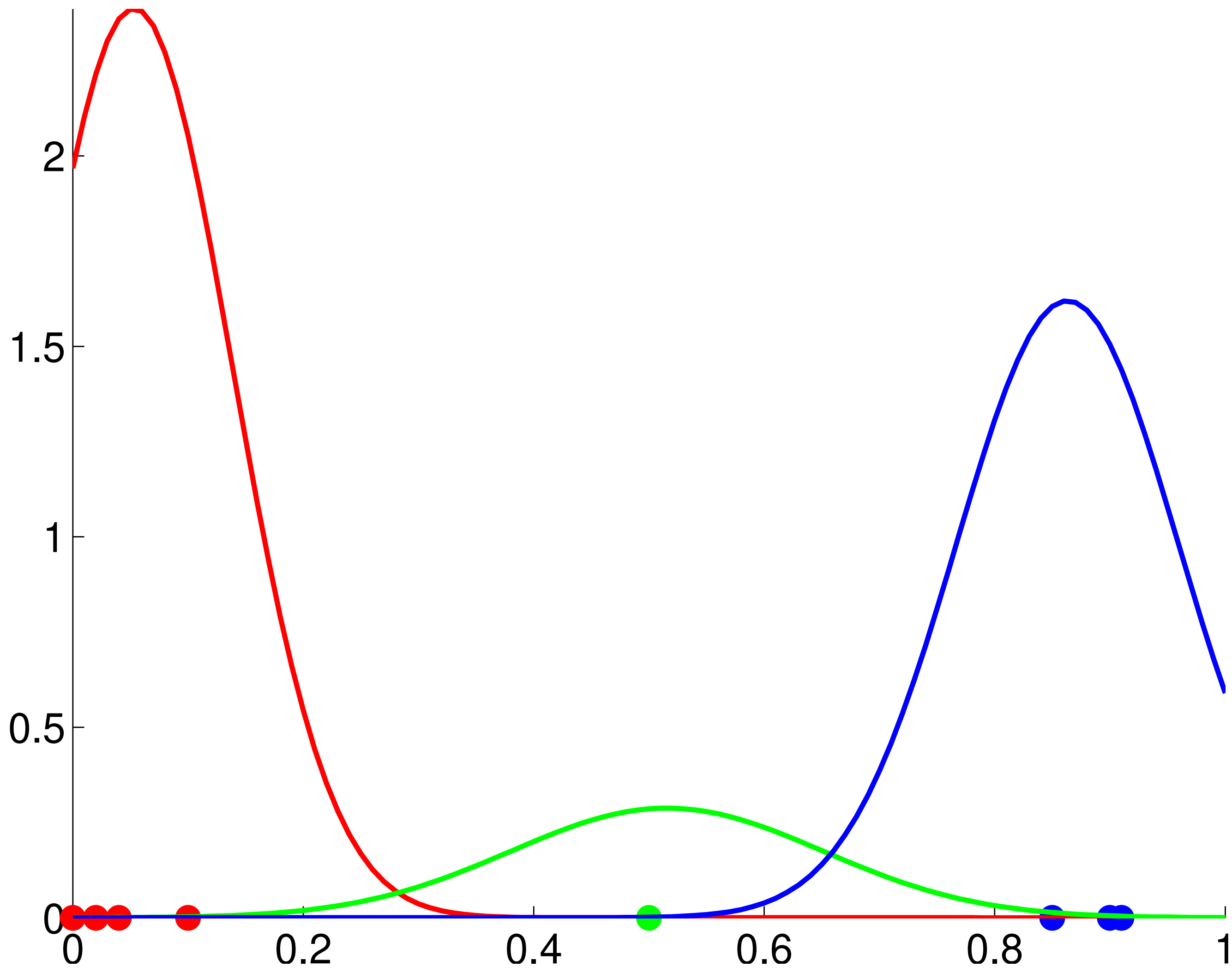
Mixture components converging to a point

- □ 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.)

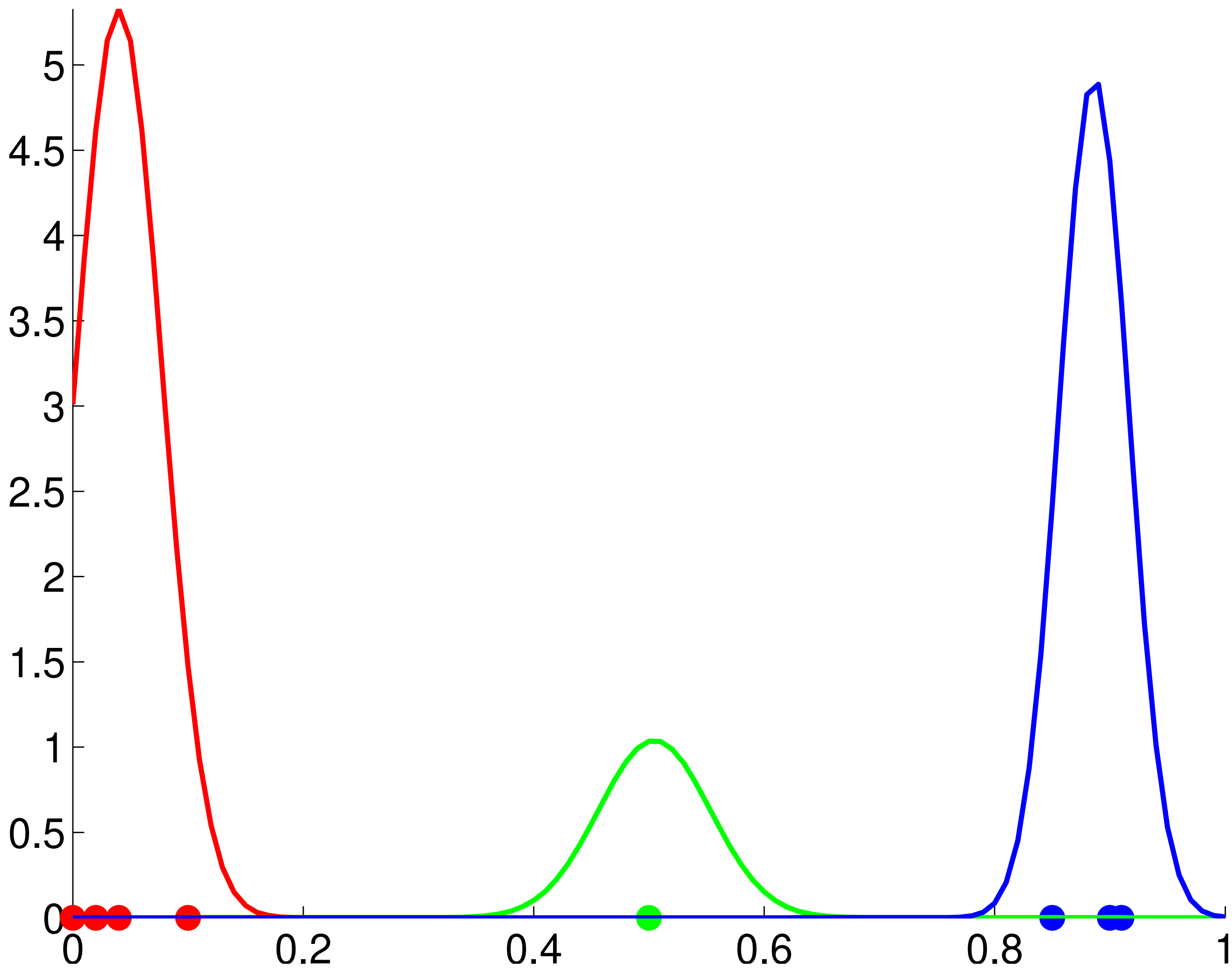
Mixture converging to a point: Example



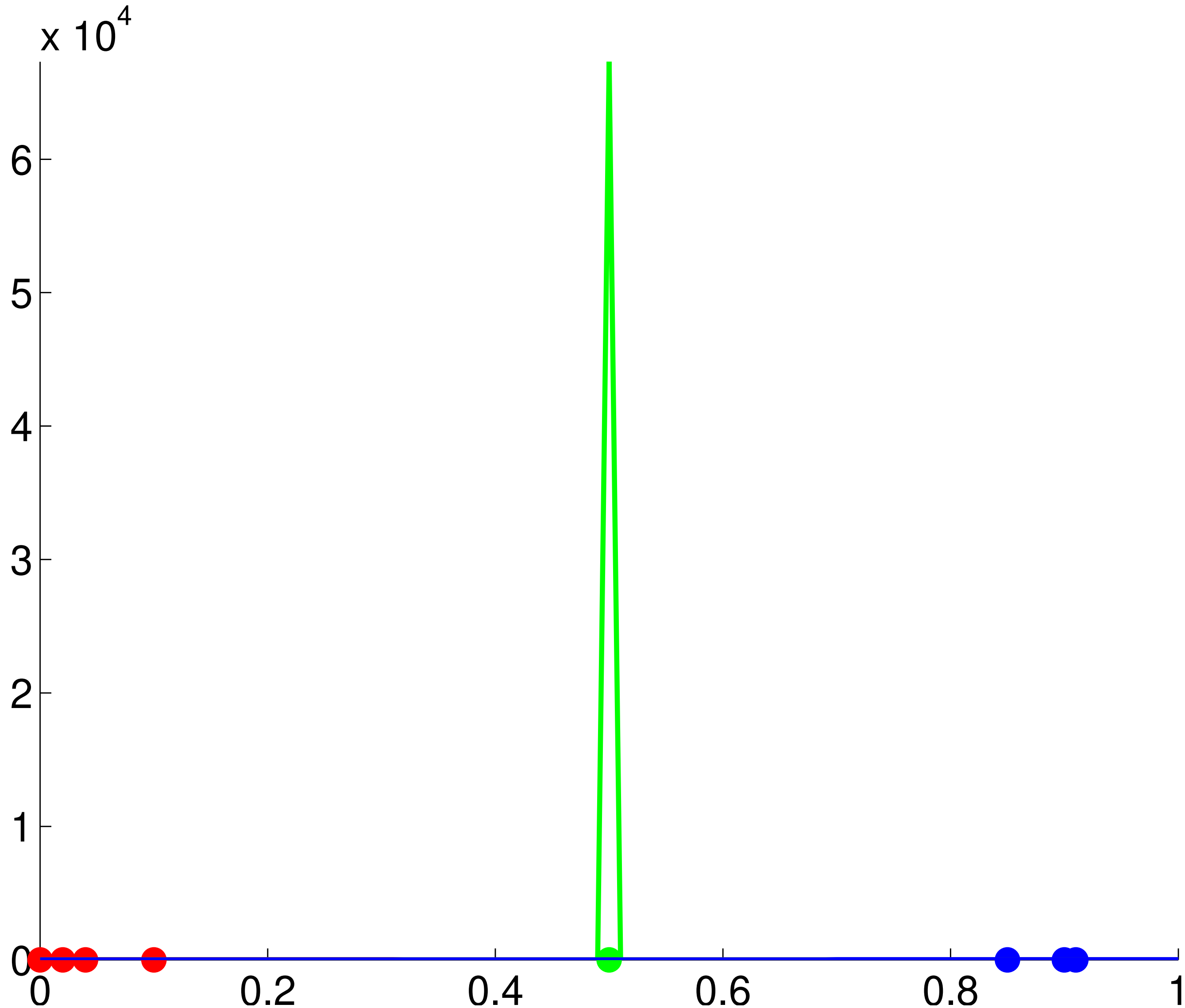
Mixture converging to a point: Example



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.

Flat clustering summary

- K-means clustering is a 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 ($\Sigma_j = \sigma^2 I$, $p_j = 1/K$)

- EM (hard or soft) explicitly fits a mixture distribution to the data
- Allows for different priors, p_j, and arbitrary multivariate Gaussian distributions for each class.
- In Hard EM, class assignments are definite; in Soft EM, probabilistic.
- Both converge–except for the issue of point-Gaussians.