CS 559: Machine Learning Fundamentals and Applications 13th Set of Notes

Instructor: Philippos Mordohai
Webpage: www.cs.stevens.edu/~mordohai
E-mail: Philippos.Mordohai@stevens.edu
Office: Lieb 215

Project Presentations

- Present project in class on:
  - December 7: regular time
  - December 9: 3:00-6:00pm
- Send me PPT/PDF file 2 hours before
  - 37 projects * 8 min = 296 minutes
  - 6 min presentation + 2 min Q&A
- Counts for 10% of total grade

Project Presentations

- Target audience: fellow classmates
- Content:
  - Define problem
  - Show connection to class material
    - What is being classified, what are the classes etc.
  - Describe data
    - Train/test splits etc.
  - Show results

Final Report

- Due December 12 (23:59)
- 6-10 pages including figures, tables and references
- Counts for 15% of total grade
- NO LATE SUBMISSIONS

Instructions for Final

- Emphasis on new material, not covered in Midterm
- Old material still in

- Will post reading list next week

- Open book, open notes, open homeworks and solutions
- No laptops, no cellphones
- Calculators OK
  - No graphical solutions. Show all computations

Overview

- Unsupervised Learning (slides by Olga Veksler)
  - Supervised vs. unsupervised learning
  - Unsupervised learning
  - Flat clustering (k-means)
  - Hierarchical clustering
- Expectation Maximization

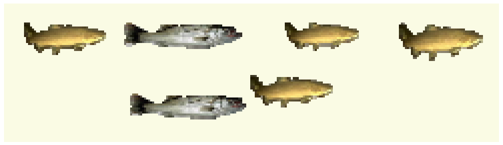
Supervised vs. Unsupervised Learning

- Up to now we considered supervised learning scenarios, where we are given:
  1. samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
  2. class labels for all samples
  - This is also called learning with teacher, since the correct answer (the true class) is provided

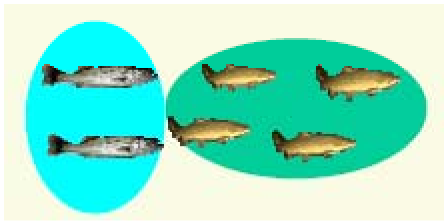
- Here, we consider unsupervised learning scenarios, where we are only given:
  1. samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
  - This is also called learning without teacher, since the correct answer is not provided
  - Do not split data into training and test sets

Unsupervised Learning

- Data are *not labeled*



- Parametric Approach
  - Assume parametric distribution of data
  - Estimate parameters of this distribution
    - Expectation Maximization
- Non-Parametric Approach
  - Group the data into ***clusters,*** each cluster (hopefully) says something about classes present in the data

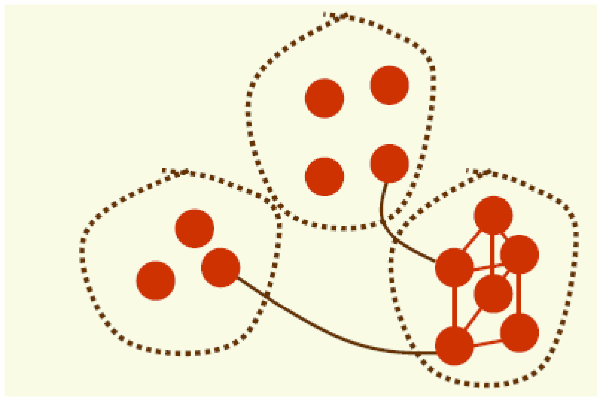


Why Unsupervised Learning?

- Unsupervised learning is harder
  - How do we know if results are meaningful? No answer (labels) is available
    - Let the experts look at the results (external evaluation)
    - Define an objective function on clustering (internal evaluation)
- We nevertheless need it because
  1. Labeling large datasets is very costly (speech recognition, object detection in images)
     - Sometimes can label only a few examples by hand
  2. May have no idea what/how many classes there are (data mining)
  3. May want to use clustering to gain some insight into the structure of the data before designing a classifier
     - Clustering as data description

Clustering

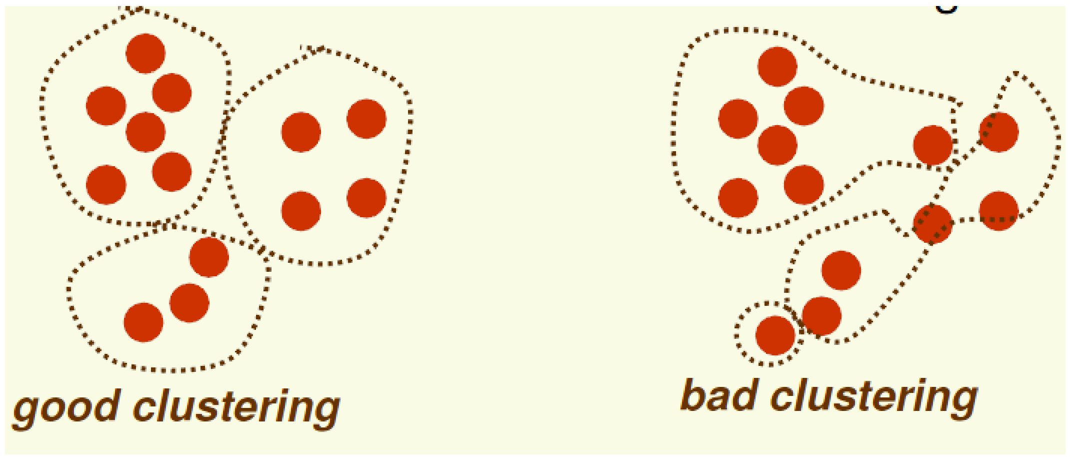
- Seek “natural” clusters in the data



- What is a good clustering?
  - internal (within the cluster) distances should be small
  - external (intra-cluster) should be large
- Clustering is a way to discover new categories (classes)

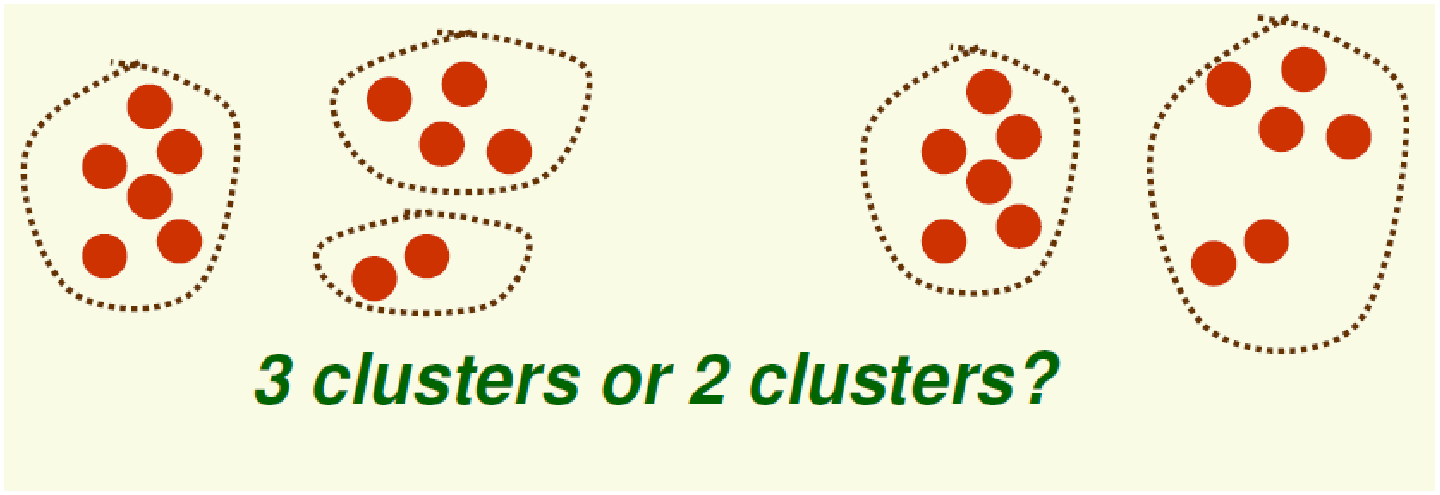
What we need for Clustering

1. Proximity measure, either
   - similarity measure **s(x_i,x_k)**: large if **x_i,x_k** are similar
   - dissimilarity(or distance) measure **d(x_i,x_k)**: small if **x_i,x_k** are similar
2. Criterion function to evaluate a clustering



3. Algorithm to compute clustering
   – For example, by optimizing the criterion function

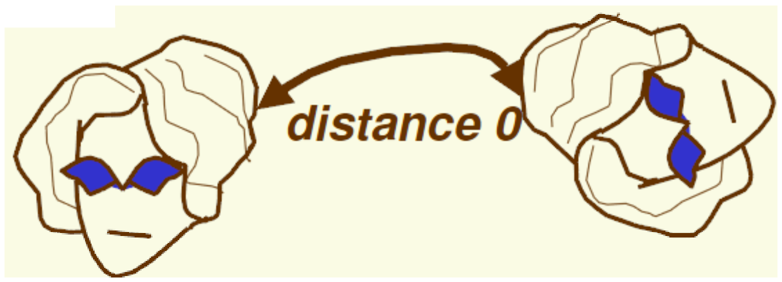
How Many Clusters?



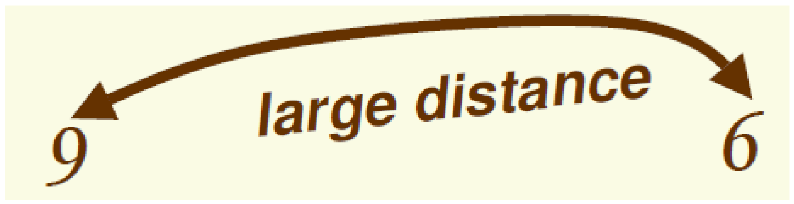
- Possible approaches
  1. Fix the number of clusters to ***k***
  2. Find the best clustering according to the criterion function (number of clusters may vary)

Proximity Measures

- A good proximity measure is VERY application dependent
  - Clusters should be invariant under the transformations "natural" to the problem
  - For example for object recognition, we should have invariance to rotation



  - For character recognition, invariance to rotation is bad



Distance Measures

- Euclidean distance

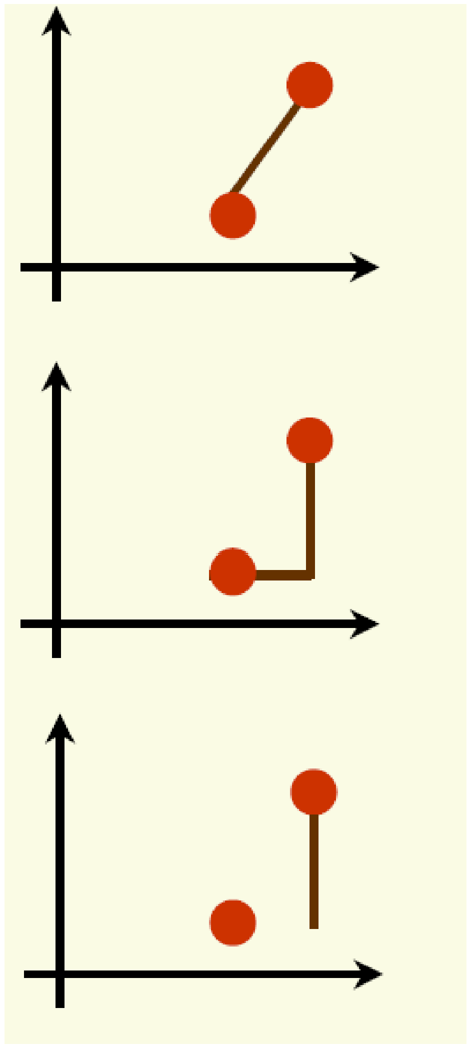
$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{d} \left(x_i^{(k)} - x_j^{(k)}\right)^2}$$

- Manhattan (city block) distance

$$d(x_i, x_j) = \sum_{k=1}^{d} \left| x_i^{(k)} - x_j^{(k)} \right|$$

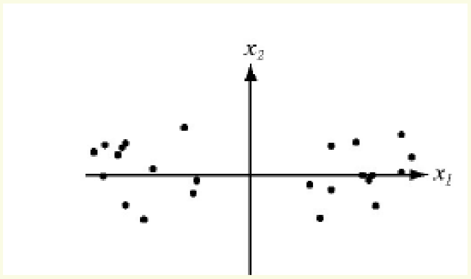
- Chebyshev distance

$$d(x_i, x_j) = \max_{1 \le k \le d} | x_i^{(k)} - x_j^{(k)} |$$

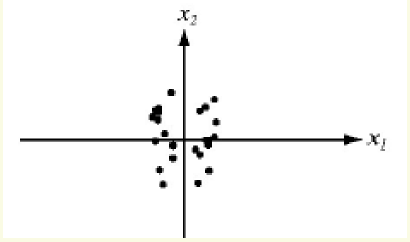


Feature Scaling

- Old problem: how to choose appropriate relative scale for features?
  - [length (in meters or cms?), weight (in grams or kgs?)]
- In supervised learning, we can normalize to zero mean unit variance with no problems
- In clustering this is more problematic
- *If variance in data is due to cluster presence, then normalizing features is not a good thing*



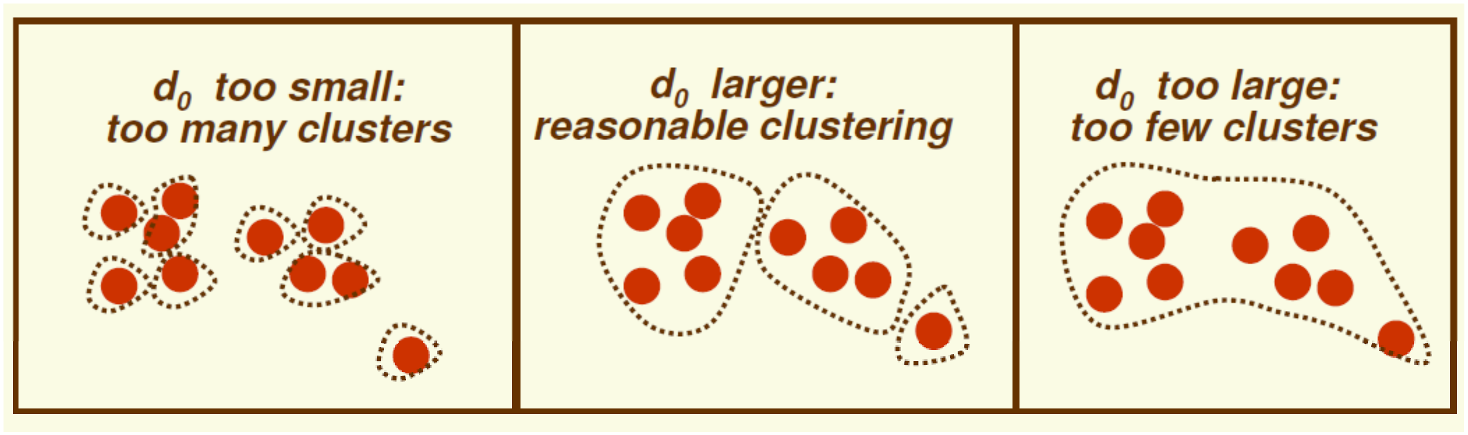
before normalization



after normalization

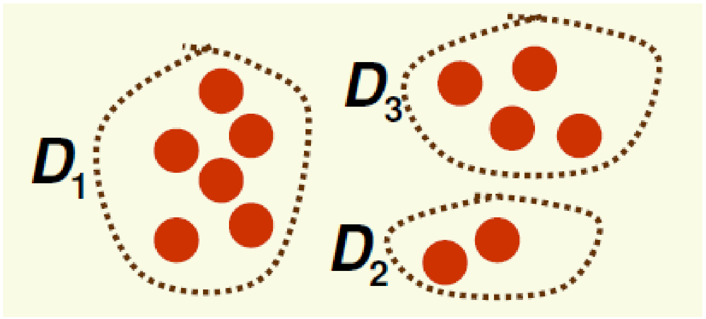
Simple Clustering Algorithm

- Having defined a proximity function, we can develop a simple clustering algorithm
  - go over all sample pairs, and put them in the same cluster if the distance between them is less then some threshold distance $\mathbf{d_0}$ (or if similarity is larger than $\mathbf{s_0}$)
- Pros: simple to understand and implement
- Cons: very dependent on $\mathbf{d_0}$ (or $\mathbf{s_0}$), automatic choice of $\mathbf{d_0}$ (or $\mathbf{s_0}$) is not easy



Criterion Functions for Clustering

- Given samples $x_1,\ldots,x_n$
- Partition them into **c** subsets $D_1,\ldots,D_c$



- There are approximately $c^n/c!$ distinct partitions
- Can define a criterion function $J(D_1,\ldots,D_c)$ which measures the quality of a partitioning $D_1,\ldots,D_c$
- Then clustering is a well defined problem
  - the optimal clustering is the partition which optimizes the criterion function

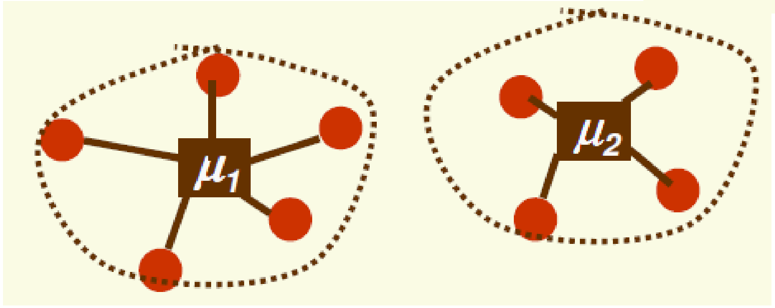
SSE Criterion Function

- Let $\mathbf{n_i}$ be the number of samples in D_i, and define the mean of samples in D_i

$$\mu_i = \frac{1}{n_i} \sum_{x \in D_i} x$$

- Then the sum-of-squared errors criterion function (to minimize) is:

$$J_{SSE} = \sum_{i=1}^{c} \sum_{x \in D_i} \| x - \mu_i \|^2$$

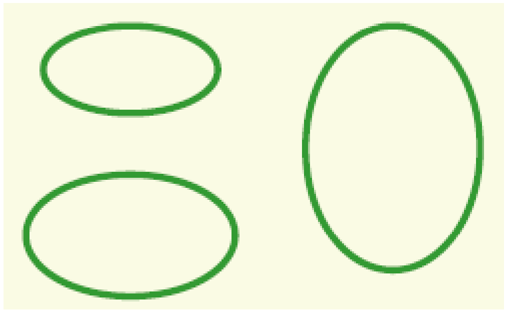


- Note that the number of clusters, $\mathbf{c}$, is fixed

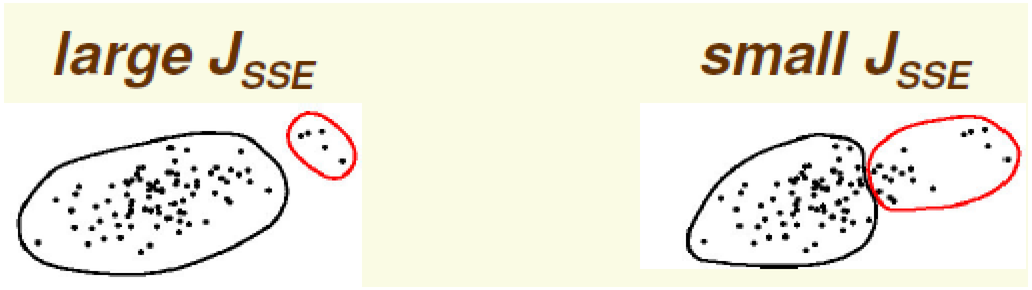
SSE Criterion Function

$$J_{SSE} = \sum_{i=1}^{c} \sum_{x \in D_i} || x - \mu_i ||^2$$

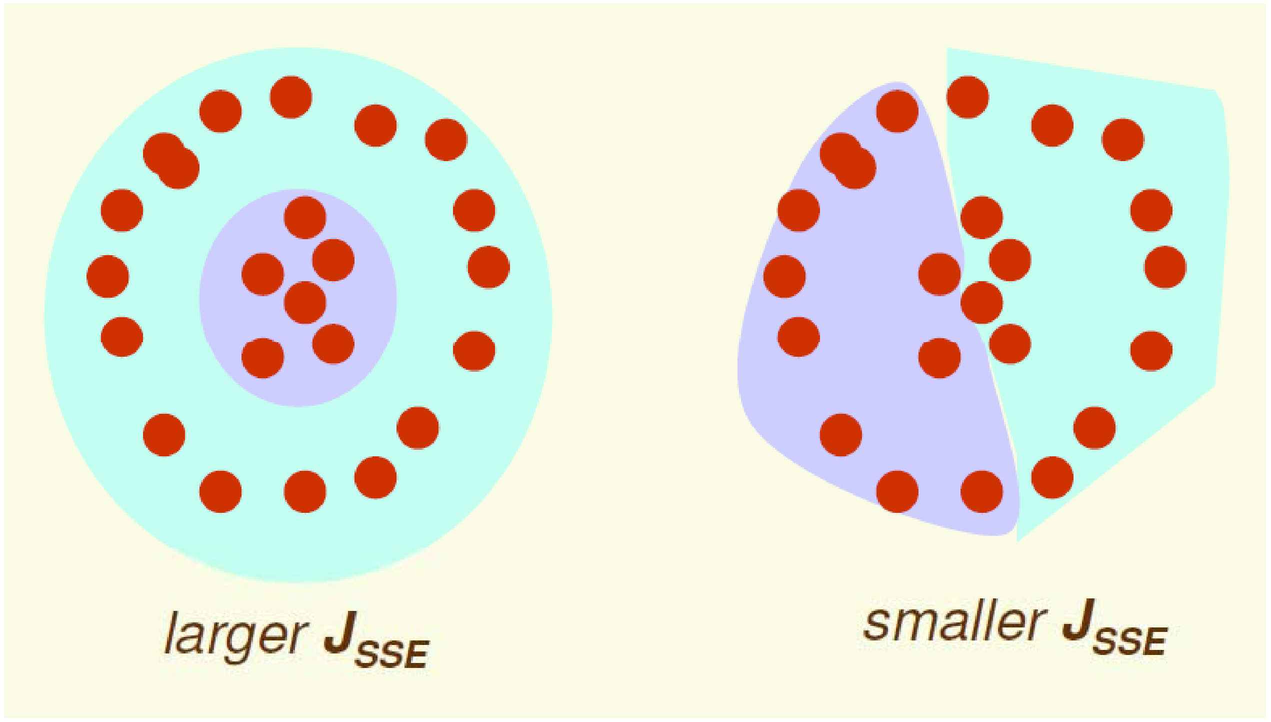
- The SSE criterion is appropriate when data forms compact clouds that are relatively well separated



- SSE criterion favors equally sized clusters, and may not be appropriate when “natural” groupings have very different sizes



Example of SSE Failure



- The problem is that one of the “natural” clusters is not compact (the outer ring)

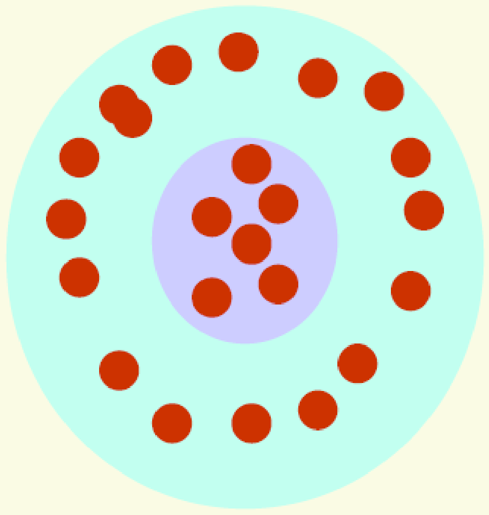
Other Clustering Criteria

- Can obtain other criterion functions by replacing $||x - y||^2$ by any other measure of distance between points in D_i
- We can also use the median, maximum, etc. instead of the average distance

Maximum Distance Criterion

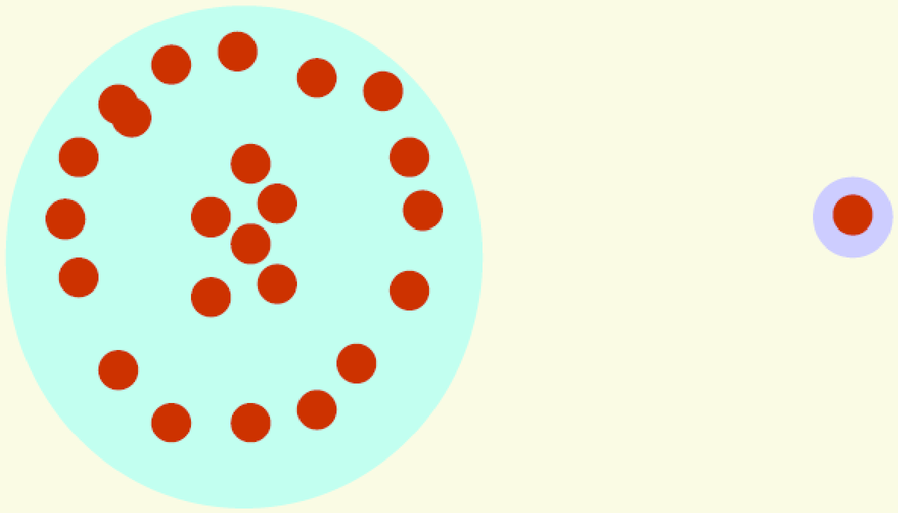
- Consider $J_{\max} = \sum_{i=1}^{c} n_i \left[\max_{y \in D_i, x \in D_i} \| x - y \|^2 \right]$

- Solves previous case



*smallest **J***J_{max}

- However J_{max} is not robust to outliers

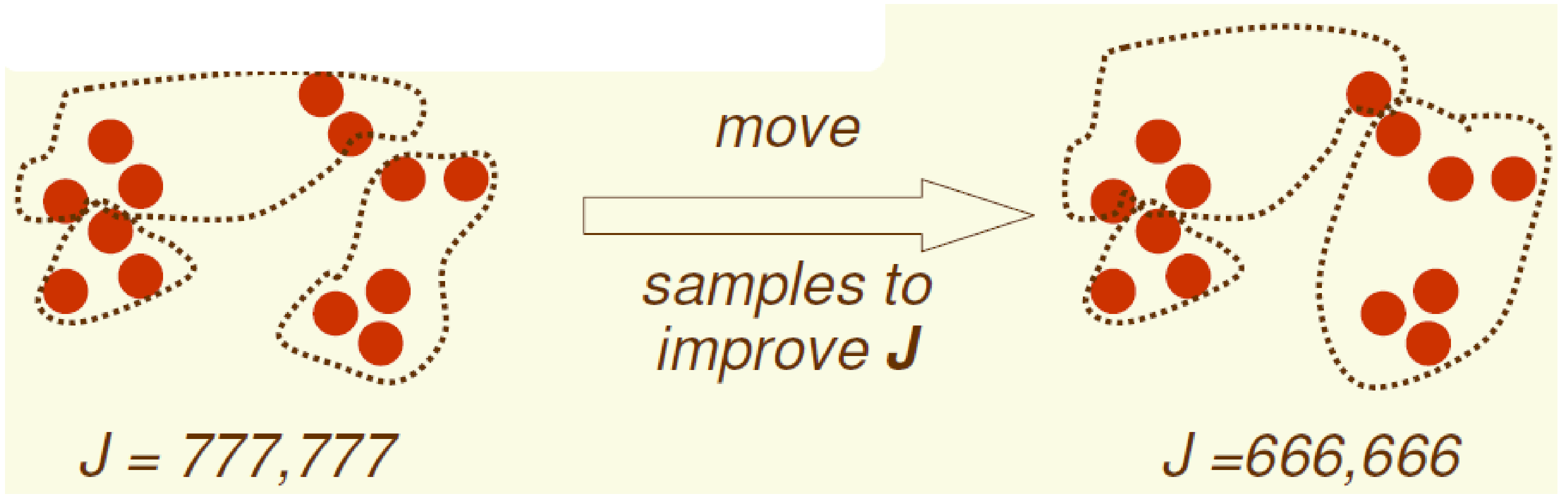


smallest J_{max}

K-means Clustering

Iterative Optimization Algorithms

- Having proximity measure and criterion function, we need an algorithm to find the optimal clustering
- Exhaustive search is impossible, since there are approximately **$c^n/c!$** possible partitions
- Usually, iterative algorithms are used
  1. Find a reasonable initial partition
  2. Repeat: move samples from one group to another s.t. the objective function J is improved



Iterative Optimization Algorithms

- Iterative optimization algorithms are similar to gradient descent
  - Move in a direction of descent, but not in the steepest descent direction since they have no derivative of the objective function
  - Solution depends on the initial point
  - Cannot find global minimum
- Main Issue
  - How to move from current partitioning to the one which improves the objective function

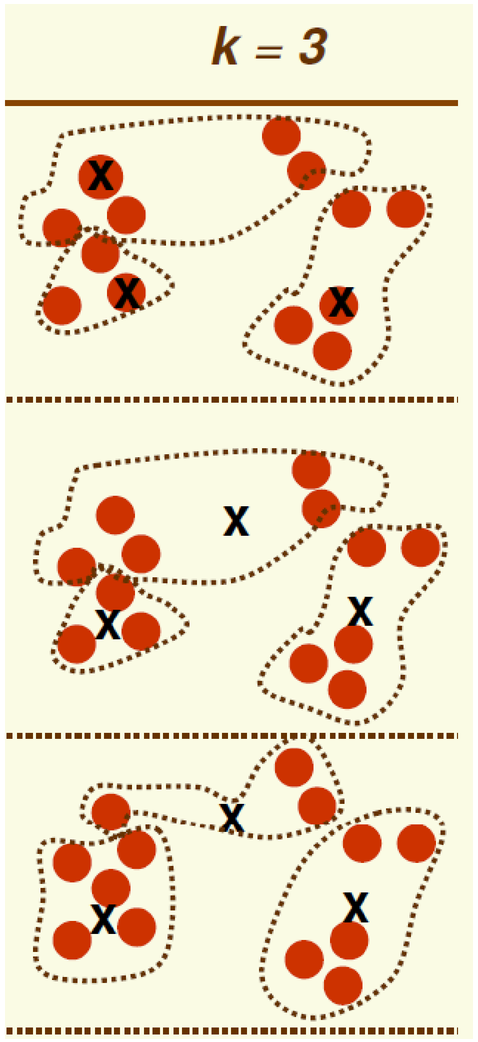
K-means Clustering

- We now consider an example of iterative optimization algorithm for the special case of J_{SSE} objective function

$$J_{SSE} = \sum_{i=1}^{k} \sum_{x \in D_i} || \mathbf{x} - \mu_i ||^2$$

- K-means is probably the most famous clustering algorithm
  - It has a smart way of moving from current partitioning to the next one
- Fix number of clusters to **k** (**c** = **k**)

K-means Clustering

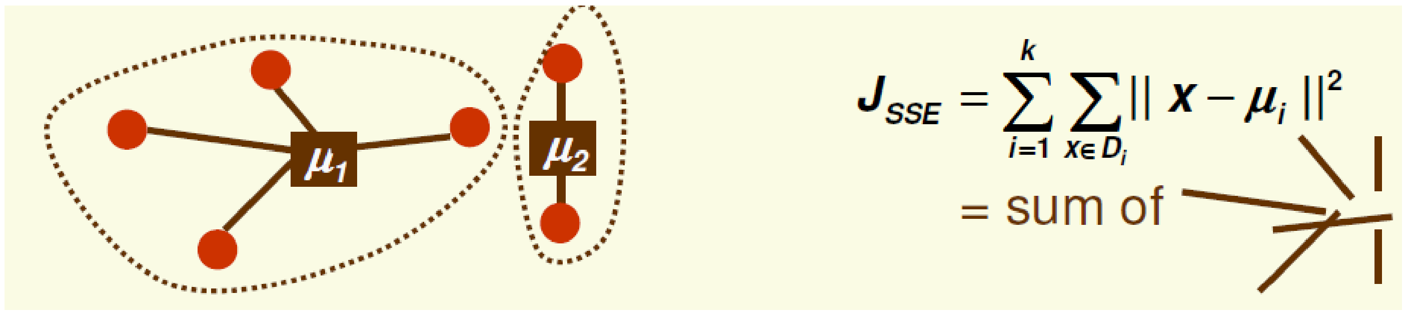


1. Initialize
   - Pick *k* cluster centers arbitrarily
   - Assign each example to closest center
2. Compute sample means for each cluster
3. Reassign all samples to the closest mean
4. If clusters changed at step 3, go to step 2

K-means Clustering

Consider steps 2 and 3 of the algorithm

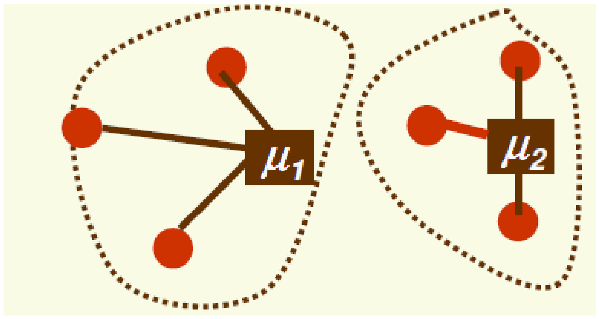
2. compute sample means for each cluster



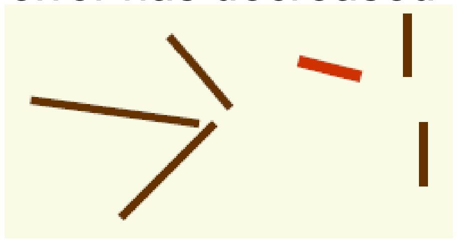
$$J_{SSE} = \sum_{i=1}^{k} \sum_{x \in D_i} || \mathbf{x} - \mu_i ||^2$$

= sum of

3. reassign all samples to the closest mean

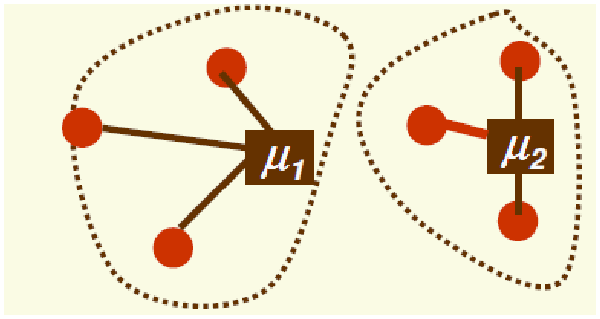


If we represent clusters by their old means, the error has decreased

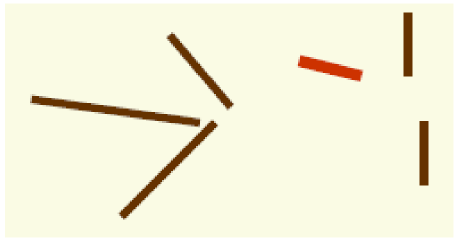


K-means Clustering

3. reassign all samples to the closest mean



If we represent clusters by their old means, the error has decreased



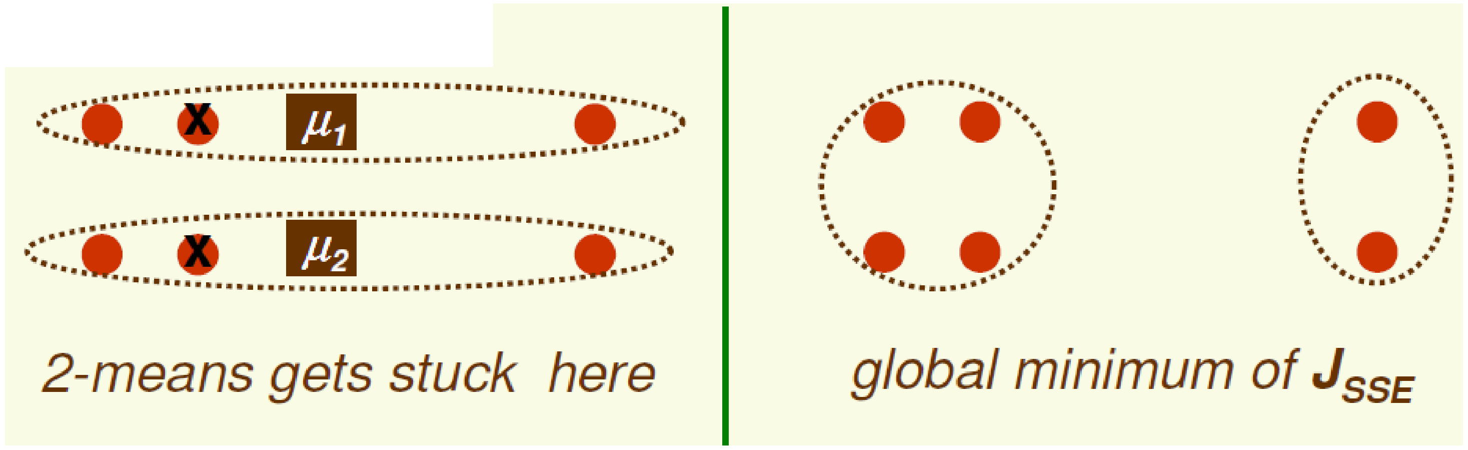
- However we represent clusters by their new means, and the mean always results in the smallest sum of squared distances

$$\frac{\partial}{\partial z}\sum_{x\in D_i}\frac{1}{2}\|x-z\|^2=\frac{\partial}{\partial z}\sum_{x\in D_i}\frac{1}{2}\left(\|x\|^2-2x^t z+\|z\|^2\right)=\sum_{x\in D_i}(-x+z)=0$$

$$\Rightarrow z=\frac{1}{n_i}\sum_{x\in D_i}x$$

K-means Clustering

- Proved that by repeating steps 2 and 3, the objective function is reduced
  - Found a "smart " move which decreases the objective function
- Thus k-means converges after a finite number of iterations of steps 2 and 3
- However k-means is not guaranteed to find a global minimum



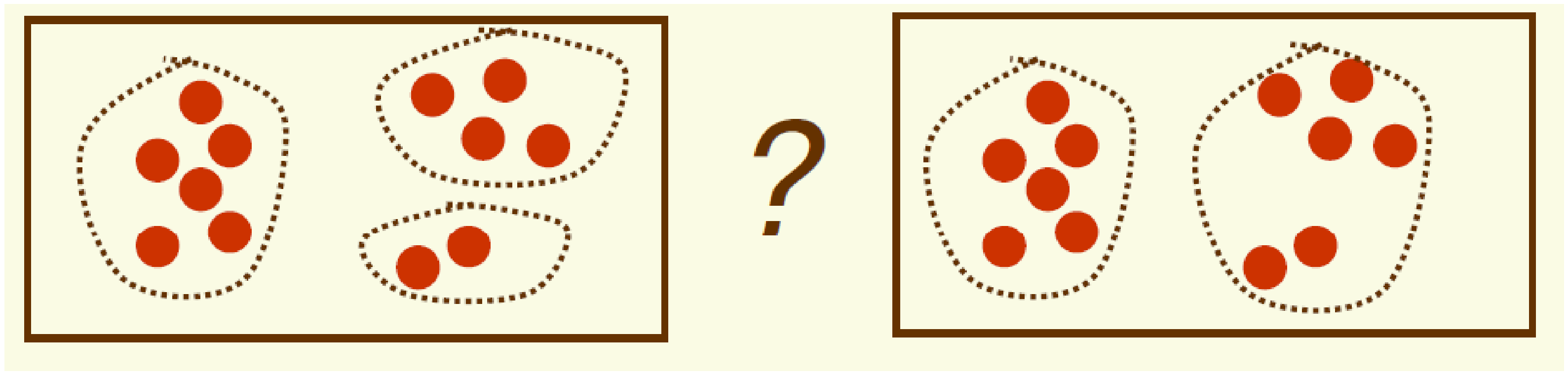
K-means Clustering

- Finding the optimum of J_{SSE} is NP-hard
- In practice, k-means clustering usually performs well
- It can be very efficient
- Its solution can be used as a starting point for other clustering algorithms
- Hundreds of papers on variants and improvements of k-means clustering are published every year

Hierarchical Clustering

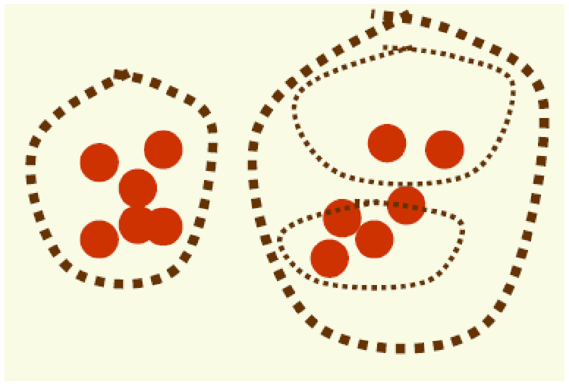
Hierarchical Clustering

- Up to now considered flat clustering

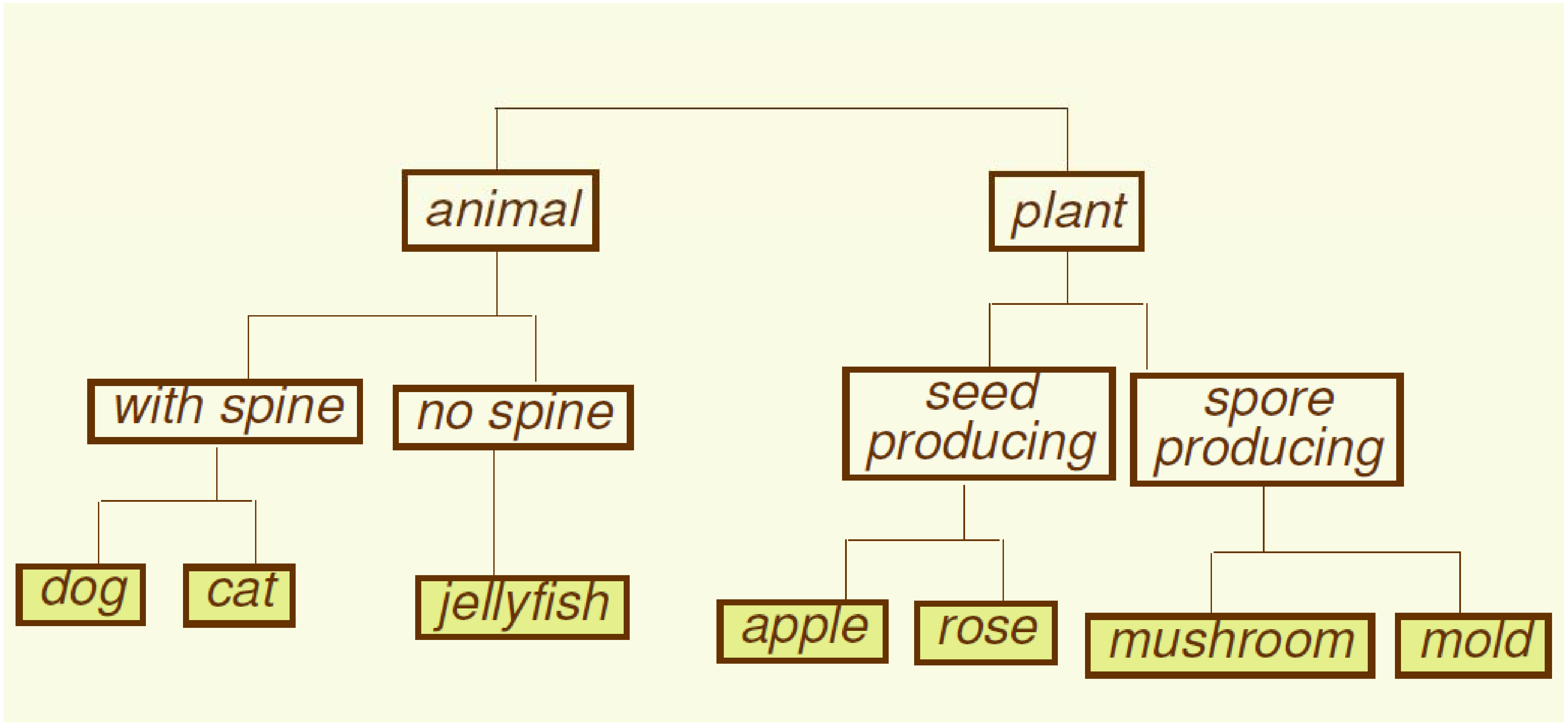


- For some data, hierarchical clustering is more appropriate than “flat” clustering

- Hierarchical clustering

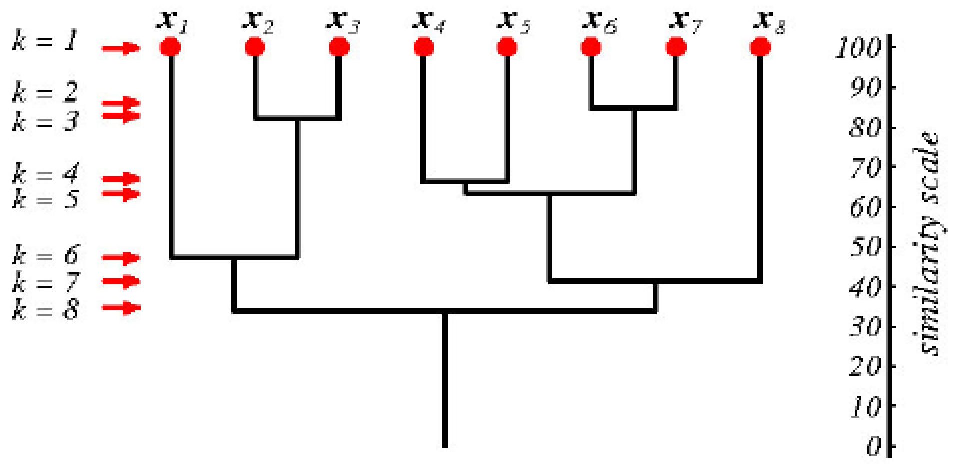


Example of Hierarchical Clustering



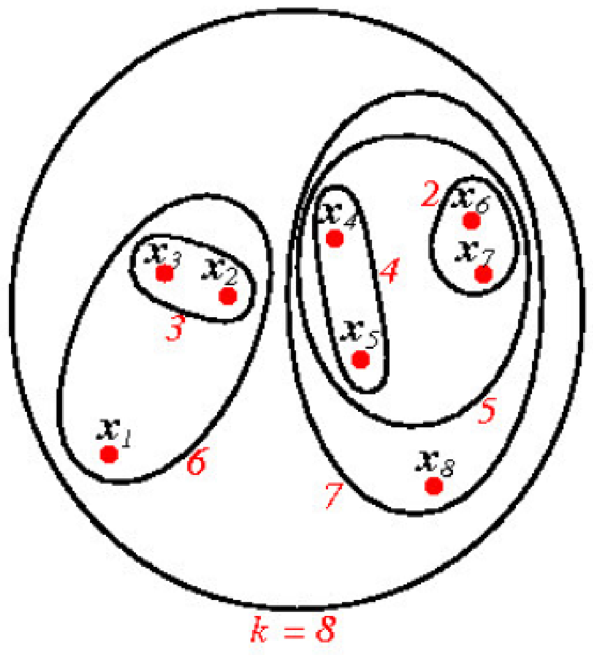
Hierarchical Clustering: Dendrogram

- The preferred way to represent a hierarchical clustering is a dendrogram
  - Binary tree
  - Level ***k*** corresponds to partitioning with ***n-k+1*** clusters
  - If ***k*** clusters required, use clustering from level ***n-k+1***
  - If samples are in the same cluster at level ***k***, they stay in the same cluster at higher levels
  - The dendrogram typically shows the similarity of grouped clusters



Hierarchical Clustering: Venn Diagram

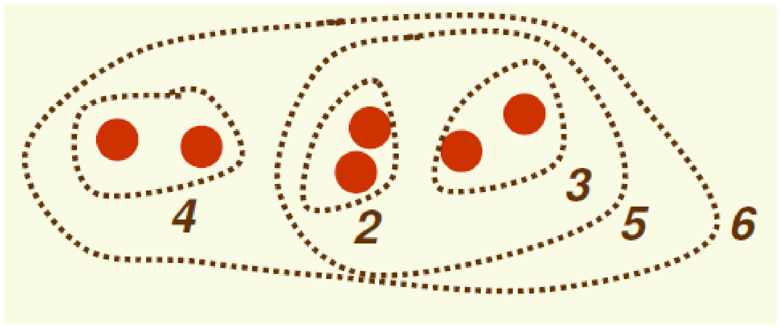
- Can also use Venn diagram to show hierarchical clustering, but similarity is not represented quantitatively



Hierarchical Clustering

- Algorithms for hierarchical clustering can be
- divided into two types:

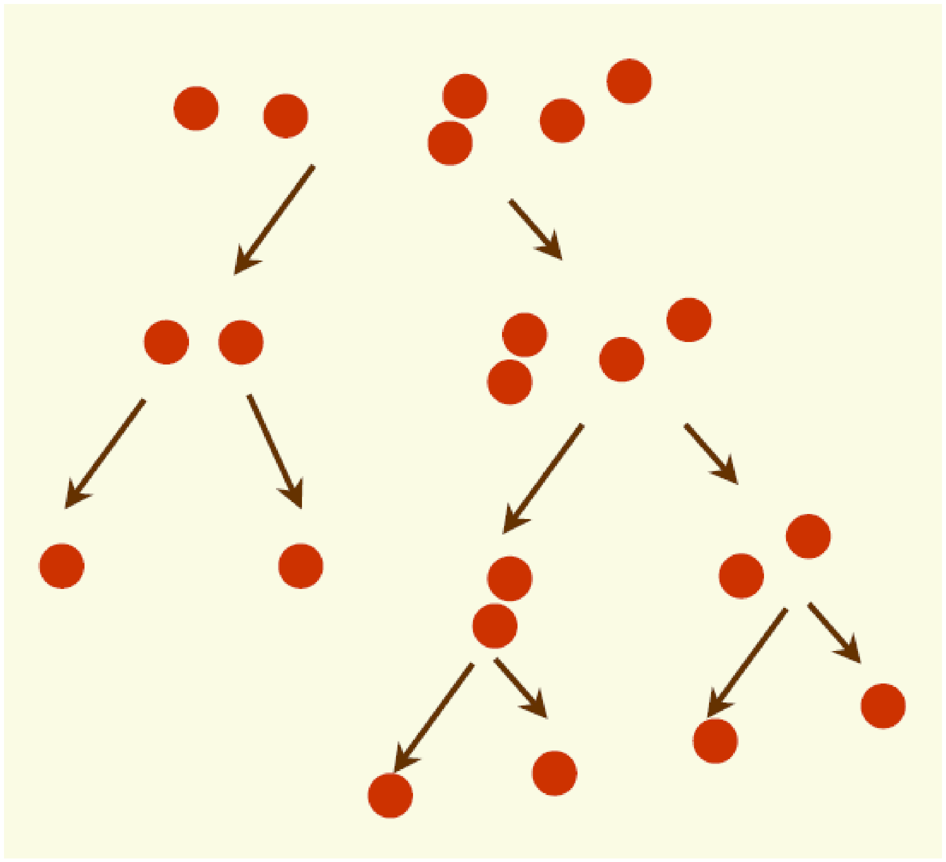
1. Agglomerative (bottom up) procedures
   - Start with *n* singleton clusters
   - Form hierarchy by merging most similar clusters



2. Divisive (top down) procedures
   - Start with all samples in one cluster
   - Form hierarchy by splitting the “worst” clusters

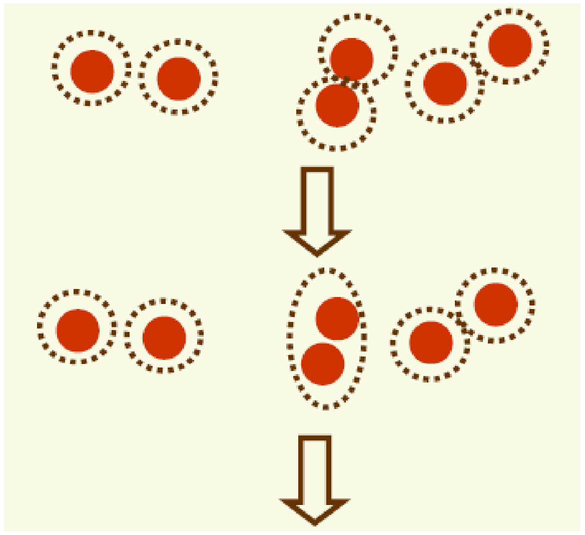
Divisive Hierarchical Clustering

- Any "flat" algorithm which produces a fixed number of clusters can be used
  - Set $c = 2$



Agglomerative Hierarchical Clustering

```
initialize with each example in
singleton cluster
while there is more than 1 cluster
        1. find 2 nearest clusters
        2. merge them
```



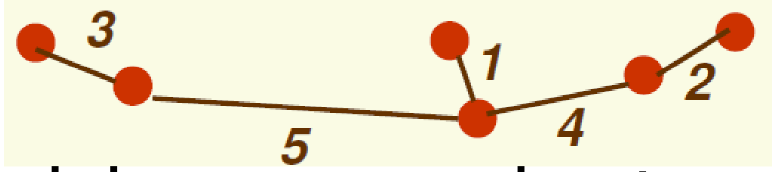
- Four common ways to measure cluster distance

1. minimum distance $d_{\min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \| x - y \|$
2. maximum distance $d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \| x - y \|$
3. average distance $d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \| x - y \|$
4. mean distance $d_{mean}(D_i, D_j) = \| \mu_i - \mu_j \|$

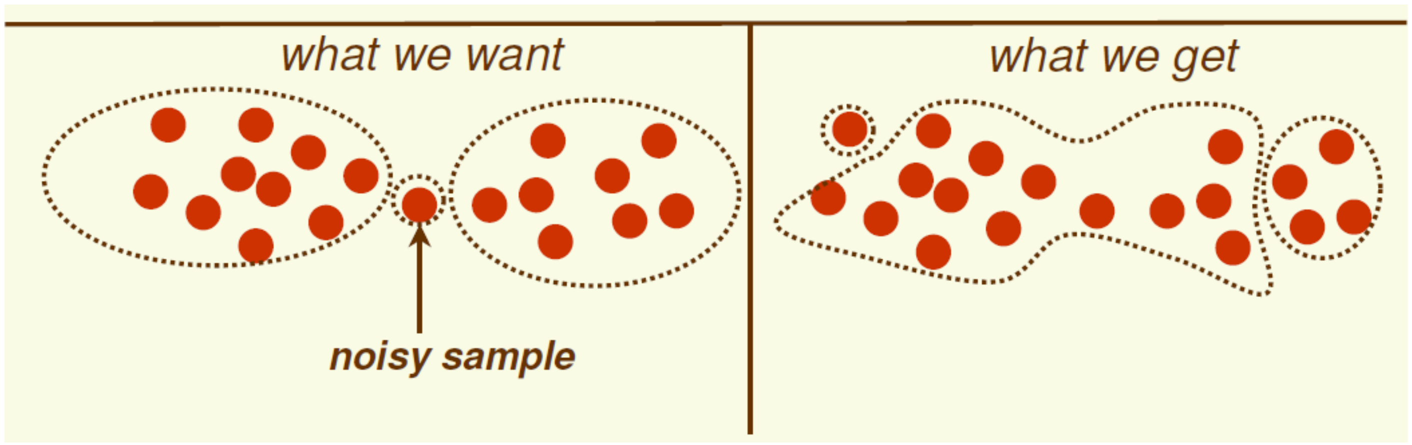
Single Linkage or Nearest Neighbor

- Agglomerative clustering with minimum distance

$$d_{\min}(D_i, D_j) = \min_{x \in D_i, y \in D_j} \| x - y \|$$



- Generates minimum spanning tree
- Encourages growth of elongated clusters
- Disadvantage: very sensitive to noise

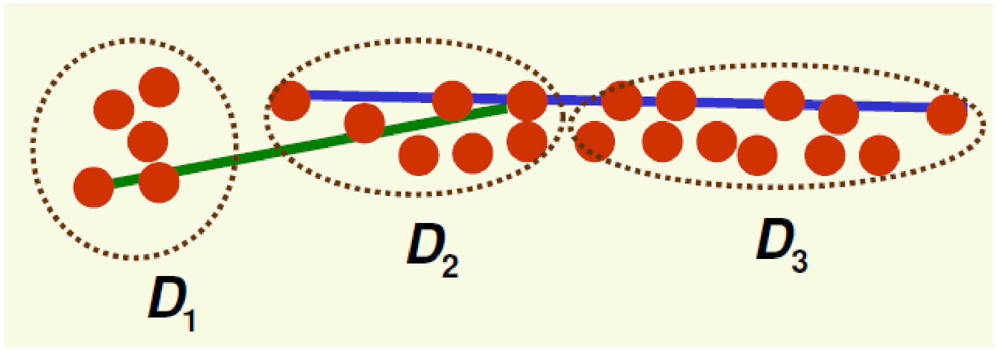


Complete Linkage or Farthest Neighbor

- Agglomerative clustering with maximum distance

$$d_{\max}(D_i, D_j) = \max_{x \in D_i, y \in D_j} \| x - y \|$$

- Encourages compact clusters
- Does not work well if elongated clusters are present



- $d_{\max}(D_1, D_2) < d_{\max}(D_2, D_3)$
- thus D_1 and D_2 are merged instead of D_2 and D_3

Average and Mean Agglomerative Clustering

- Agglomerative clustering is more robust under the average or the mean cluster distance

$$d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{x \in D_i} \sum_{y \in D_j} \| x - y \|$$

$$d_{mean}(D_i, D_j) = \| \mu_i - \mu_j \|$$

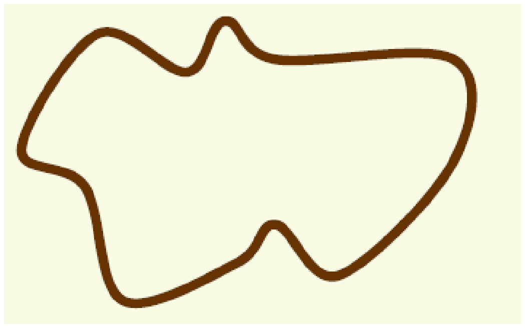
- Mean distance is cheaper to compute than the average distance
- Unfortunately, there is not much to say about agglomerative clustering theoretically, but it does work reasonably well in practice

Agglomerative vs. Divisive

- Agglomerative is faster to compute, in general
- Divisive may be less “blind” to the global structure of the data

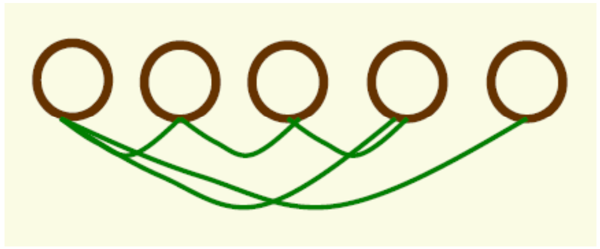
Divisive

- when taking the first step (split), it has access to all the data; can find the best possible split in 2 parts



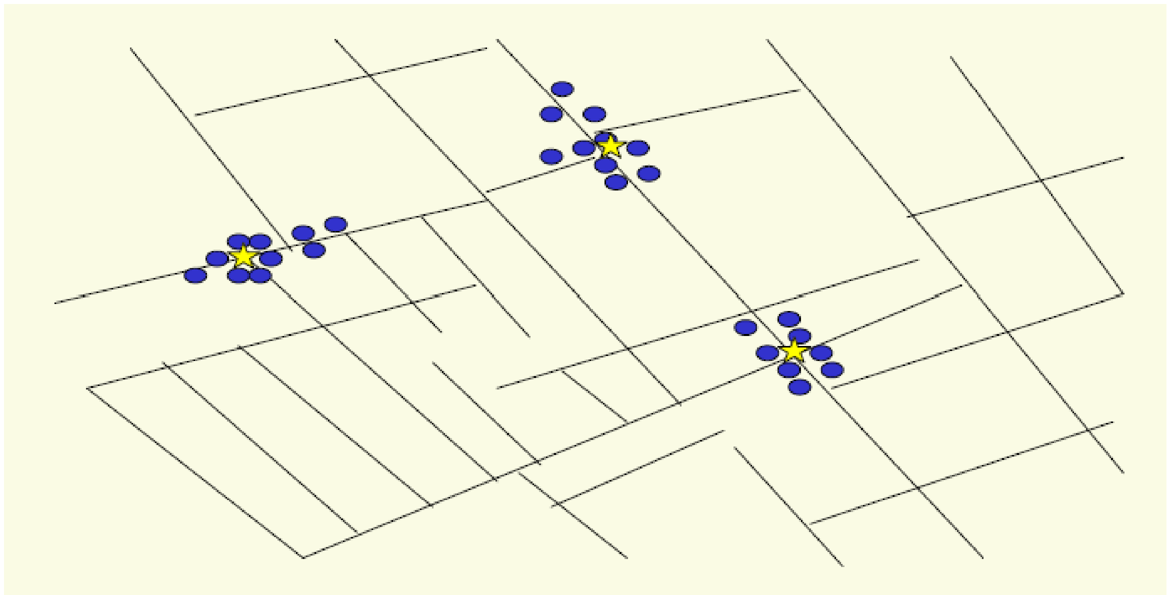
Agglomerative

- when taking the first step (merge), it does not consider the global structure of the data, only looks at pairwise structure



First (?) Application of Clustering

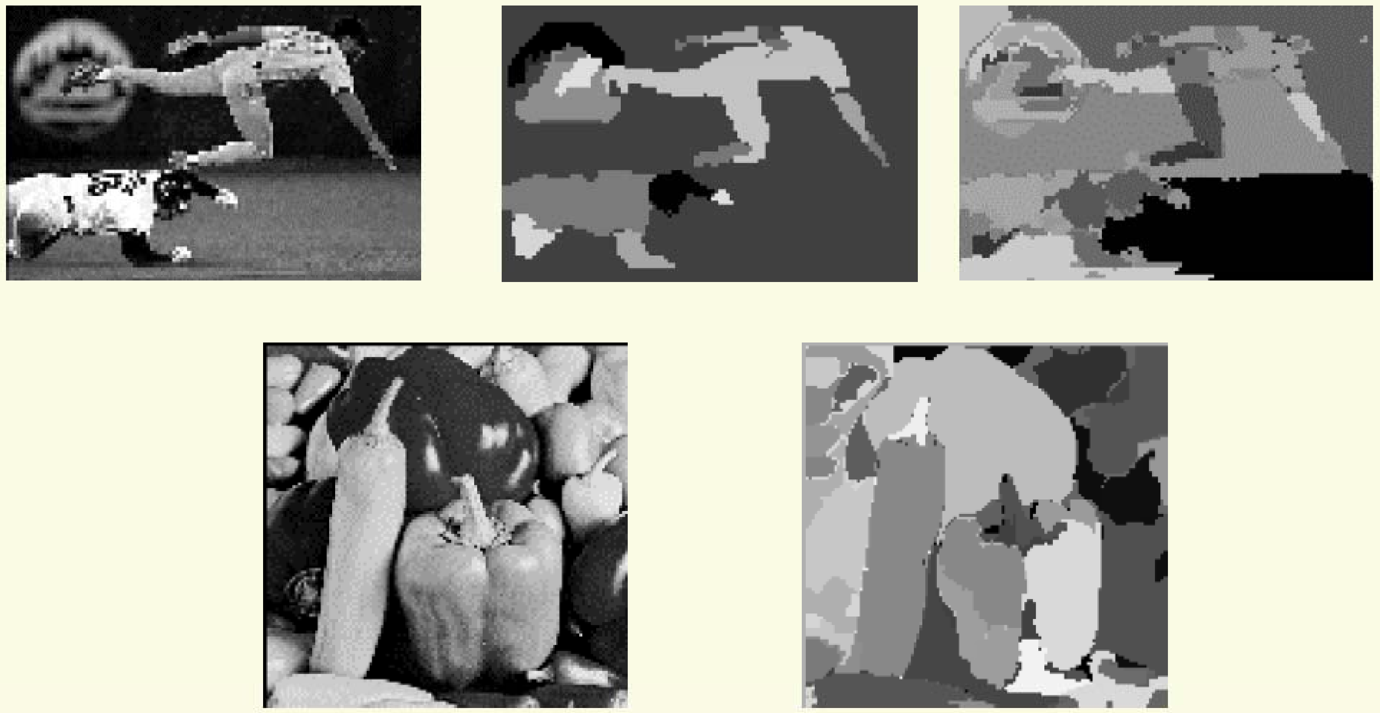
- John Snow, a London physician plotted the location of cholera deaths on a map during an outbreak in the 1850s.
- The locations indicated that cases were clustered around certain intersections where there were polluted wells -- thus exposing both the problem and the solution.



From: Nina Mishra HP Labs

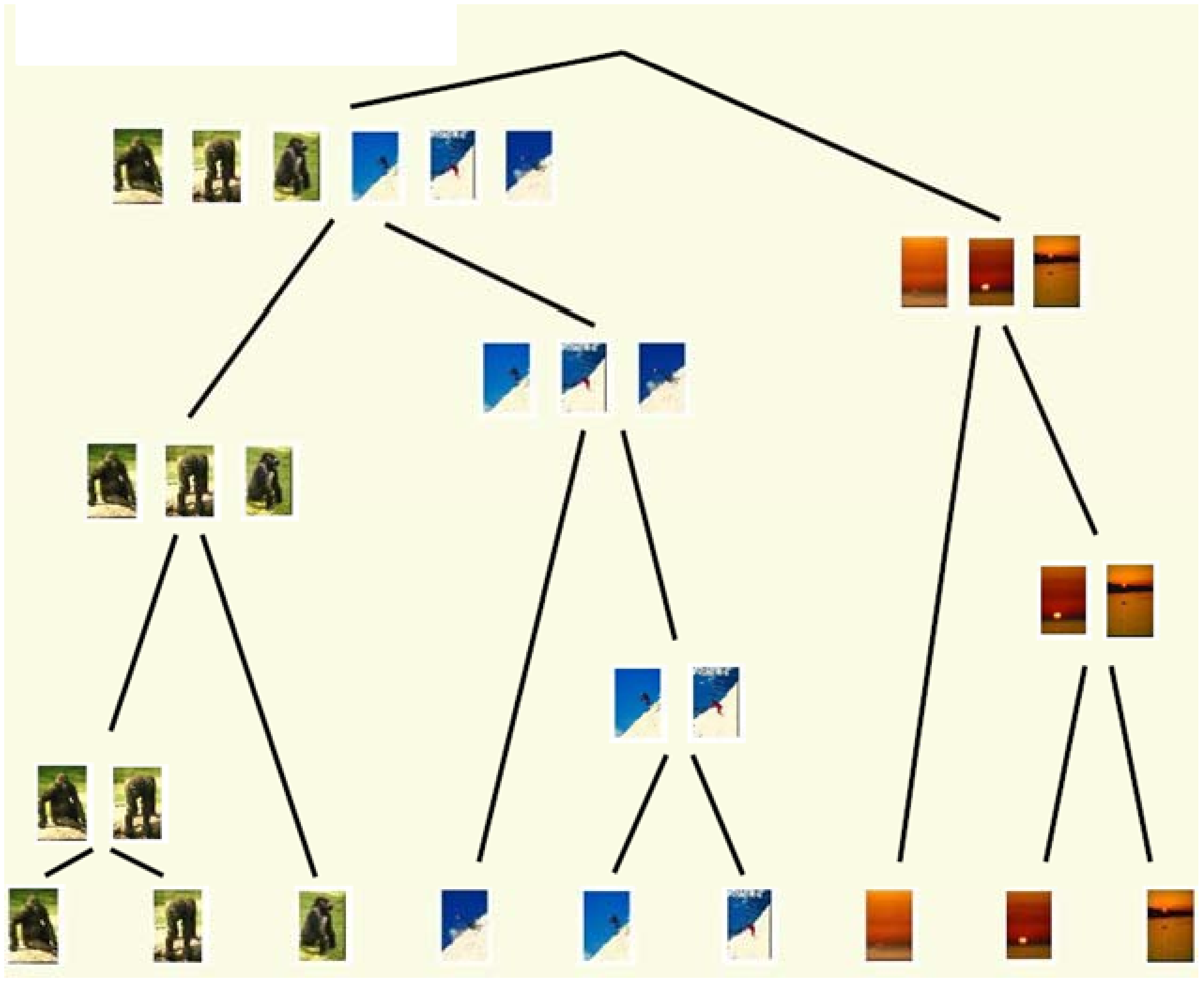
Applications of Clustering

- Image segmentation
  - Find coherent "objects" in images



From: Image Segmentation by Nested Cuts, O. Veksler, CVPR2000

Image Database Organization



Clustering Summary

- Clustering (nonparametric learning) is useful for discovering inherent structure in data
- Clustering is immensely useful in different fields
- Clustering comes naturally to humans (in up to 3 dimensions), but not so to computers
- It is very easy to design a clustering algorithm, but it is very hard to make theoretical claims on performance
- General purpose clustering is unlikely to exist; for best results, clustering should be tuned to application at hand

Expectation Maximization

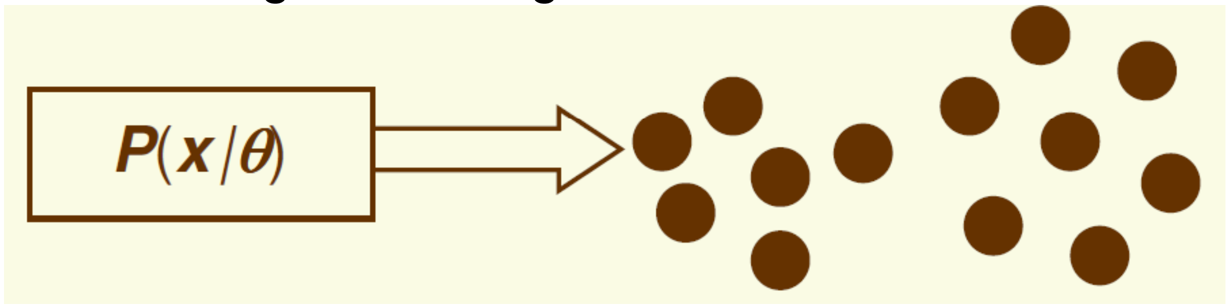
Slides based on Olga Veksler’s

Unsupervised Learning

- In unsupervised learning, where we are only given samples $x_1, \ldots, x_n$ without class labels
- Nonparametric approach: clustering
- Parametric approach:
  - assume parametric distribution of data
  - estimate parameters of this distribution
  - much “harder” than the supervised learning case

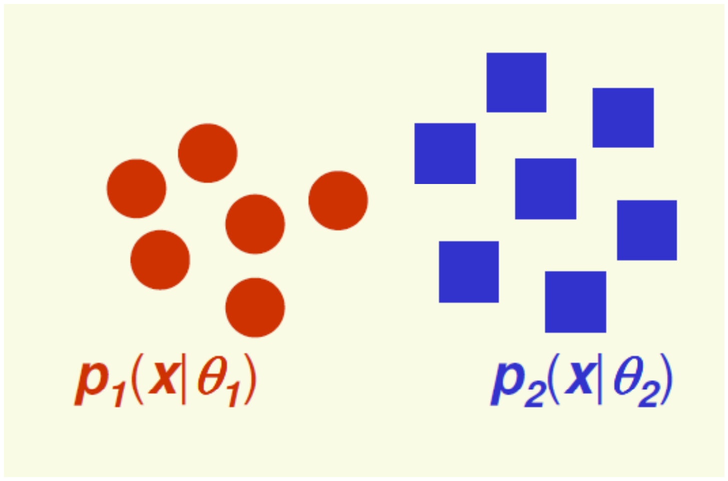
Parametric Unsupervised Learning

- Assume the data was generated by a model with known shape but unknown parameters
- Advantages of having a model
  - Gives a meaningful way to cluster data
  - adjust the parameters of the model to maximize the probability that the model produced the observed data
  - Can sensibly measure if a clustering is good
    - compute the likelihood of data induced by clustering
  - Can compare 2 clustering algorithms
    - which one gives the higher likelihood of the observed data?



Parametric Supervised Learning

- We have m classes
- with samples $x_1, \dots, x_n$ from each class 1, 2,…, m
- D_i holds samples from class i
- the probability distribution for class i is $p_i(x|\theta_i)$



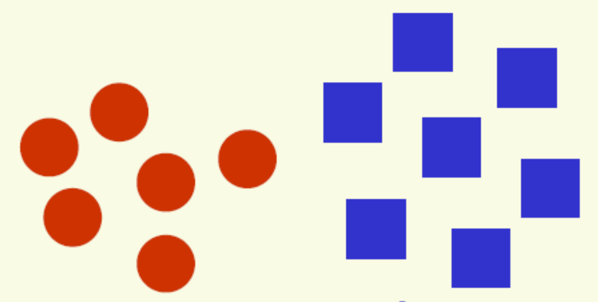
Parametric Supervised Learning

- Use the ML method to estimate parameters θ_i
  - Find θ_i which maximizes the likelihood function $F(\theta_i)$

$$p(D_i \mid \theta_i) = \prod_{x \in D_i} p(x \mid \theta_i) = F(\theta_i)$$

- or, equivalently, find θ_i which maximizes the log likelihood $l(\theta_i)$

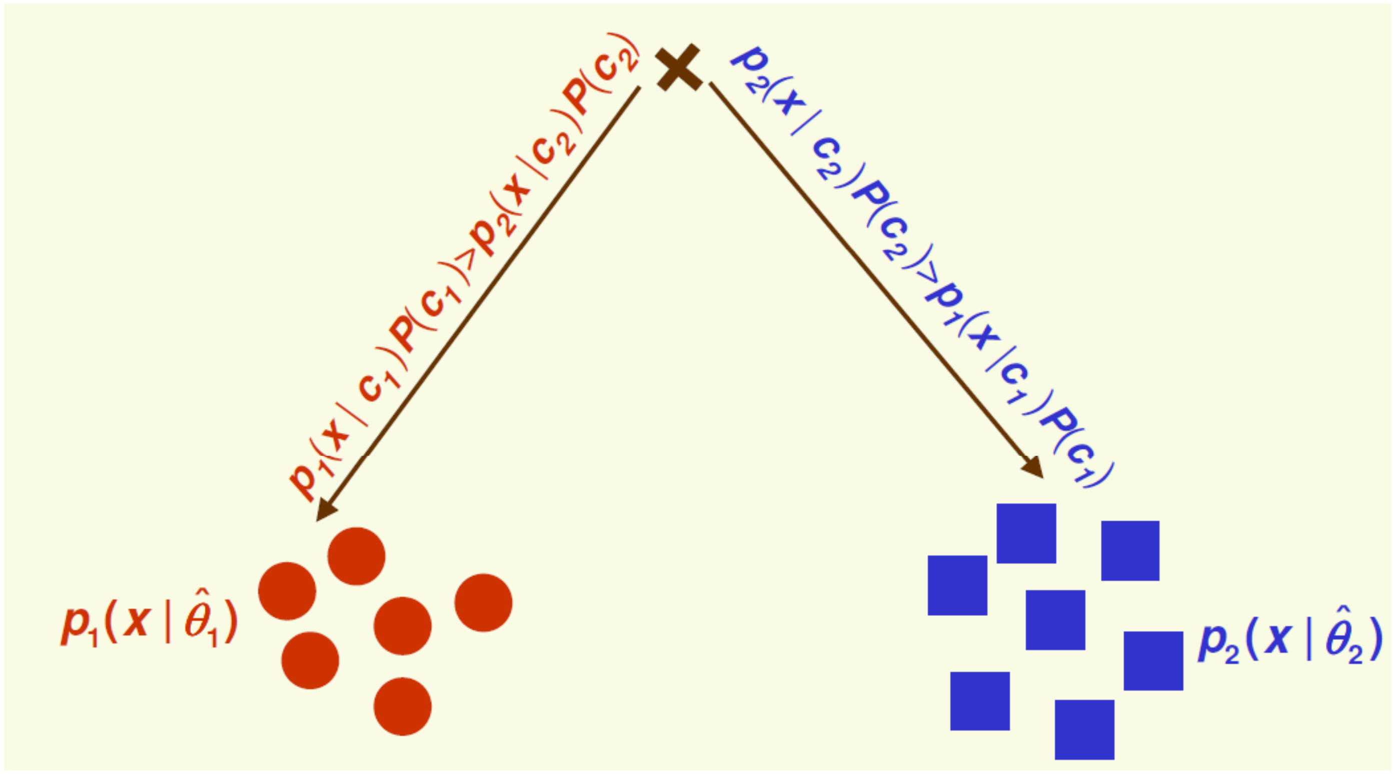
$$l(\theta_i) = \ln p(D_i \mid \theta_i) = \sum_{x \in D_i} \ln p(x \mid \theta_i)$$



$$\hat{\theta}_1 = \arg\max_{\theta_1} [\ln p(D_1 \mid \theta_1)] \qquad \hat{\theta}_2 = \arg\max_{\theta_2} [\ln p(D_2 \mid \theta_2)]$$

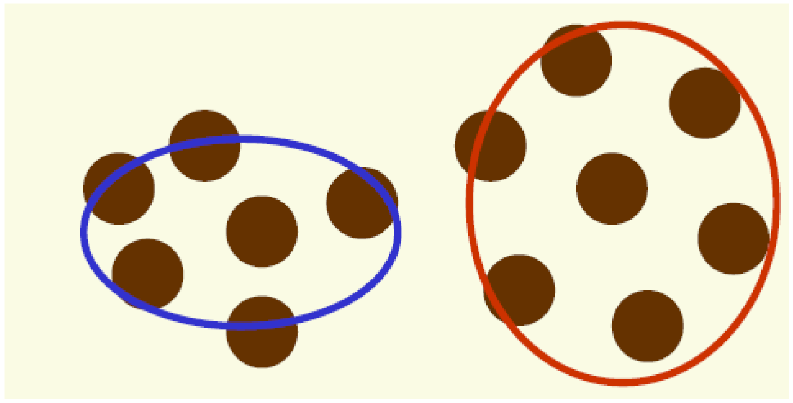
Parametric Supervised Learning

- Now the distributions are fully specified
- We can classify unknown sample using MAP classifier


$p_1(x \mid c_1)P(c_1) > p_2(x \mid c_2)P(c_2)$
$p_2(x \mid c_2)P(c_2) > p_1(x \mid c_1)P(c_1)$
$p_1(x \mid \hat{\theta}_1)$
$p_2(x \mid \hat{\theta}_2)$

Parametric Unsupervised Learning

- In unsupervised learning, no one tells us the true classes for samples. We still know that:
  - we have ***m classes***
  - we have samples $x_1,\ldots, x_n$ from unknown class
  - the probability distribution for class i is $p_i(x|\theta_i)$
- Can we determine the classes and parameters simultaneously?

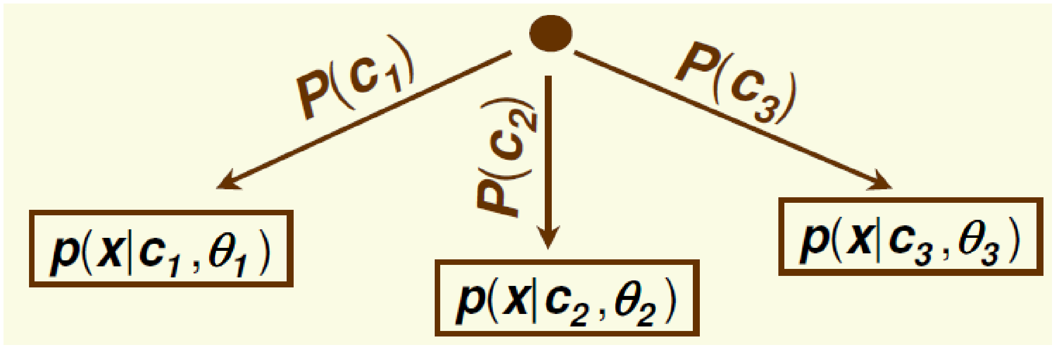


Mixture Density Model

- Model data with mixture density

$$p(x \mid \theta) = \sum_{j=1}^{m} \overbrace{p(x \mid c_j, \theta_j)}^{\text{component densities}} \underbrace{P(c_j)}_{\text{mixing parameters}}$$

  - where $\theta=\{\theta_1,\dots, \theta_m\}$
  - $P(c_1)+P(c_2)\dots+P(c_m)=1$
- To generate a sample from distribution $p(x|\theta)$:
  - first select class j with probability $P(c_j)$
  - then generate x according to probability law $p(x|c_j, \theta_j)$



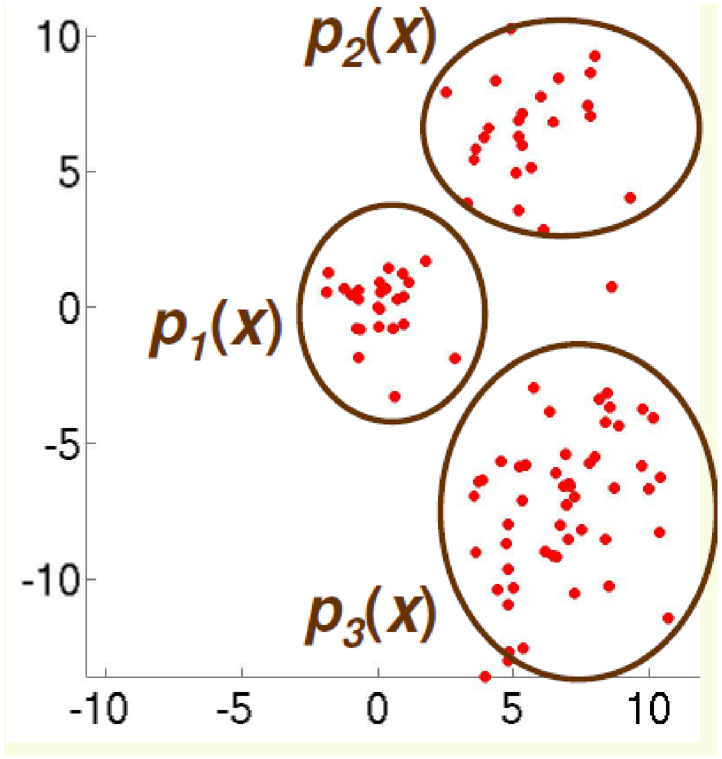
Example: Gaussian Mixture Density

- Mixture of 3 Gaussians

$$p_1(x) \cong N\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

$$p_2(x) \cong N\left([6,6], \begin{bmatrix}4 & 0\\0 & 4\end{bmatrix}\right)$$

$$p_3(x) \cong N\left([7,-7], \begin{bmatrix}6 & 0\\0 & 6\end{bmatrix}\right)$$



$$p(x) = 0.2p_1(x) + 0.3p_2(x) + 0.5p_3(x)$$

Mixture Density

$$p(x \mid \theta) = \sum_{j=1}^{m} p(x \mid c_j, \theta_j) P(c_j)$$

- $P(c_1), \ldots, P(c_m)$ can be known or unknown
  - Suppose we know how to estimate $\theta_1, \ldots, \theta_m$ and $P(c_1), \ldots, P(c_m)$
- Can "break apart" mixture $p(x|\theta)$ for classification
- To classify sample x, use MAP estimation, that is choose class i which maximizes

$$P(c_i \mid x, \theta_i) \propto \underbrace{p(x \mid c_i, \theta_i)}_{\text{probability of component } \boldsymbol{i} \text{ to generate } \boldsymbol{x}} \underbrace{P(c_i)}_{\text{probability of component } \boldsymbol{i}}$$

ML Estimation for Mixture Density

$$p(x \mid \theta, \rho) = \sum_{j=1}^{m} p\left(x \mid c_j, \theta_j\right) P\left(c_j\right) = \sum_{j=1}^{m} p\left(x \mid c_j, \theta_j\right) \rho_i$$

- Use Maximum Likelihood estimation for a mixture density; need to estimate
  - $\theta = \{\theta_1, \ldots, \theta_m\}$
  - $\rho_1 = P(c_1), \ldots, \rho_m = P(c_m)$, and $\rho = \{\rho_1, \ldots, \rho_m\}$
- As in the supervised case, form the log likelihood function

$$l(\theta, \rho) = \ln p(D \mid \theta, \rho) = \sum_{k=1}^{n} \ln p(x_k \mid \theta, \rho) = \sum_{k=1}^{n} \ln \left[\sum_{j=1}^{m} p\left(x \mid c_j, \theta_j\right) \rho_i \right]$$

ML Estimation for Mixture Density

  - Need to maximize $l(\theta, \rho)$ with respect to θ and ρ
- $l(\theta, \rho)$ is not the easiest function to maximize
  - If we take partial derivatives with respect to θ, ρ and set them to 0, typically we have a "coupled" nonlinear system of equations
  - usually closed form solution cannot be found
- We could use the gradient ascent method
  - in general, it is not the best method to use, should only be used as last resort
- There is a better algorithm, called EM

Mixture Density

- Before EM, let's look at the mixture density again

$$p(x \mid \theta, \rho) = \sum_{j=1}^{m} p\left(x \mid c_j, \theta_j\right)\rho_j$$

- Suppose we know how to estimate $\theta_1,\ldots,\theta_m$ and $\rho_1,\ldots,\rho_m$
- Estimating the class of x is easy with MAP, maximize:

$$p(x \mid c_i, \theta_i)P(c_i) = p(x \mid c_i, \theta_i)\rho_i$$

- Suppose we know the class of samples $x_1,\ldots,x_n$
  - This is just the supervised learning case, so estimating $\theta_1,\ldots,\theta_m$ and $\rho_1,\ldots,\rho_m$ is easy

$$\hat{\theta}_i = \underset{\theta_i}{\arg\max}\left[\ln p(D_i \mid \theta_i)\right] \qquad \hat{\rho}_i = \frac{|D_i|}{n}$$

- This is an example of chicken-and-egg problem
  - The EM algorithm approaches this problem by adding "hidden" variables

Expectation Maximization

EM is an algorithm for ML parameter estimation when the data have missing values. It is used when:

1. Data are incomplete
   - Some features are missing for some samples due to data corruption, partial survey responses, etc.
   - This scenario is not covered here
2. The data X are complete, but $p(X|\theta)$ is hard to optimize. We introduce hidden variables Z, whose values are missing, hoping to make the optimization of the "complete" likelihood function $p(X,Z|\theta)$ easier.
   - This scenario is useful for the mixture density estimation, and is the way we will look at EM

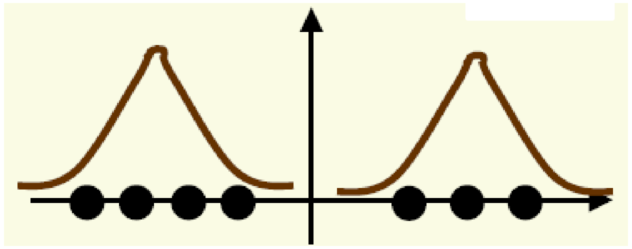
EM: Hidden Variables for Mixture Density

$$p(x \mid \theta) = \sum_{j=1}^{m} p(x \mid c_j, \theta_j) \rho_j$$

- For simplicity, assume component densities are

$$p(x \mid c_j, \theta_j) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2} \right)$$

- assume for now that the variance is known
  - need to estimate $\theta = \{\mu_1, \dots, \mu_m\}$



- If we knew which sample came from which component (that is the class label), the ML parameter estimation is easy
- Thus to get an easier problem, introduce hidden variables which indicate which component each sample belongs to

EM: Hidden Variables for Mixture Density

- For i ϵ [1, n], k ϵ [1, m], define hidden variables $z_i^{(k)}$

$$z_i^{(k)} = \begin{cases} 1 & \text{if sample } \boldsymbol{i} \text{ was generated by component } \boldsymbol{k} \\ 0 & \text{otherwise} \end{cases}$$

$$\boxed{\boldsymbol{x}_i \rightarrow \left\{\boldsymbol{x}_i, \boldsymbol{z}_i^{(1)}, \ldots, \boldsymbol{z}_i^{(m)}\right\}}$$

- $z_i^{(k)}$ are indicator random variables, they indicate which Gaussian component generated sample x_i

EM: Hidden Variables for Mixture Density

- Let $z_i = \{z_i^{(1)},\ldots, z_i^{(m)}\}$, be indicator r.v. corresponding to sample x_i
- Conditioned on z_i, the distribution of x_i is Gaussian

$$p(x_i \mid z_i, \theta) \sim N(\mu_k, \sigma^2)$$

- where k is s.t. $z_i^{(k)} = 1$

EM: Joint Likelihood

- Let $z_i = \{z_i^{(1)},\dots, z_i^{(m)}\}$ and $Z = \{z_1,\dots, z_n\}$
- The complete likelihood is

$$p(X,Z \mid \theta) = p(x_1,\dots,x_n,z_1,\dots,z_n \mid \theta) = \prod_{i=1}^{n} p(x_i, z_i \mid \theta)$$

$$= \prod_{i=1}^{n} \underbrace{p(x_i \mid z_i,\theta)}_{\text{gaussian}} \underbrace{p(z_i)}_{\text{part of } \rho_c}$$

- If we actually observed Z, the log likelihood $\ln[p(X,Z|\theta)]$ would be trivial to maximize with respect to θ and ρ_i
- The problem, is, of course, that the values of Z are missing, since we made it up (that is, Z is hidden)

EM Derivation

- Instead of maximizing $\ln[p(X,Z|\,\theta)]$ the idea behind EM is to maximize some function of $\ln[p(X,Z|\,\theta)]$, usually its expected value

$$E_Z\,[\ln p(X,Z \mid \theta\,)]$$

  - If θ makes $\ln[p(X,Z|\,\theta)]$ large, then θ tends to make $E[\ln\, p(X,Z|\,\theta)]$ large
  - the expectation is with respect to the missing data Z
  - that is with respect to density $p(Z\,|X,\,\theta)$
- however θ is our ultimate goal

The EM Algorithm

- The EM solution is:
  - Start with initial parameters $\theta^{(0)}$
  - Iterate the following 2 steps until convergence

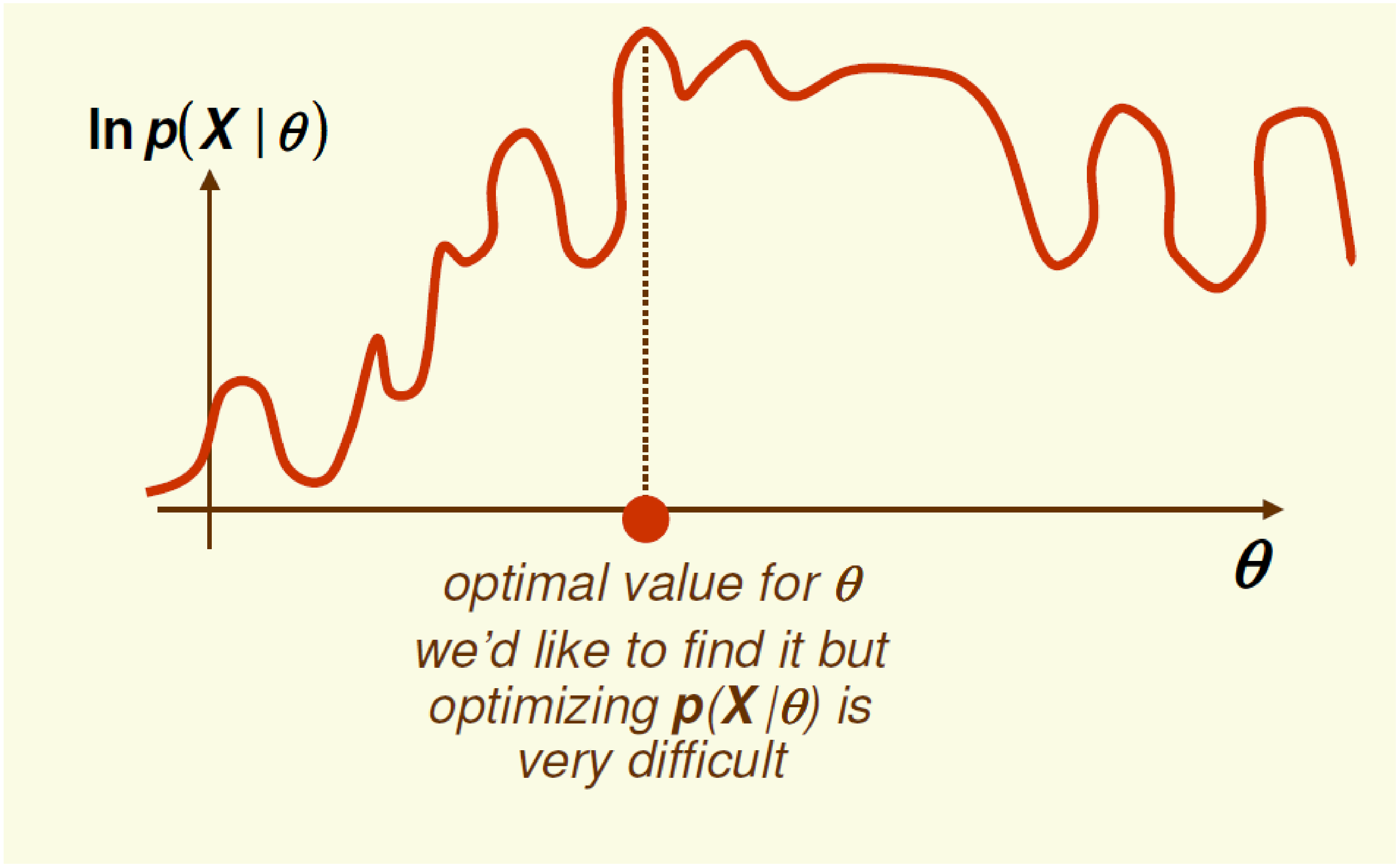
    E. compute the expectation $Q(\theta \mid \theta^{(t)})$ of the log likelihood with respect to current estimate $\theta^{(t)}$ and X

$$Q\left(\theta \mid \theta^{(t)}\right) = E_Z\left[\ln p(X, Z \mid \theta) \mid X, \theta^{(t)}\right]$$

    M. maximize $Q(\theta \mid \theta^{(t)})$

$$\theta^{(t+1)} = \arg\max_{\theta} Q\left(\theta \mid \theta^{(t)}\right)$$

EM in Pictures



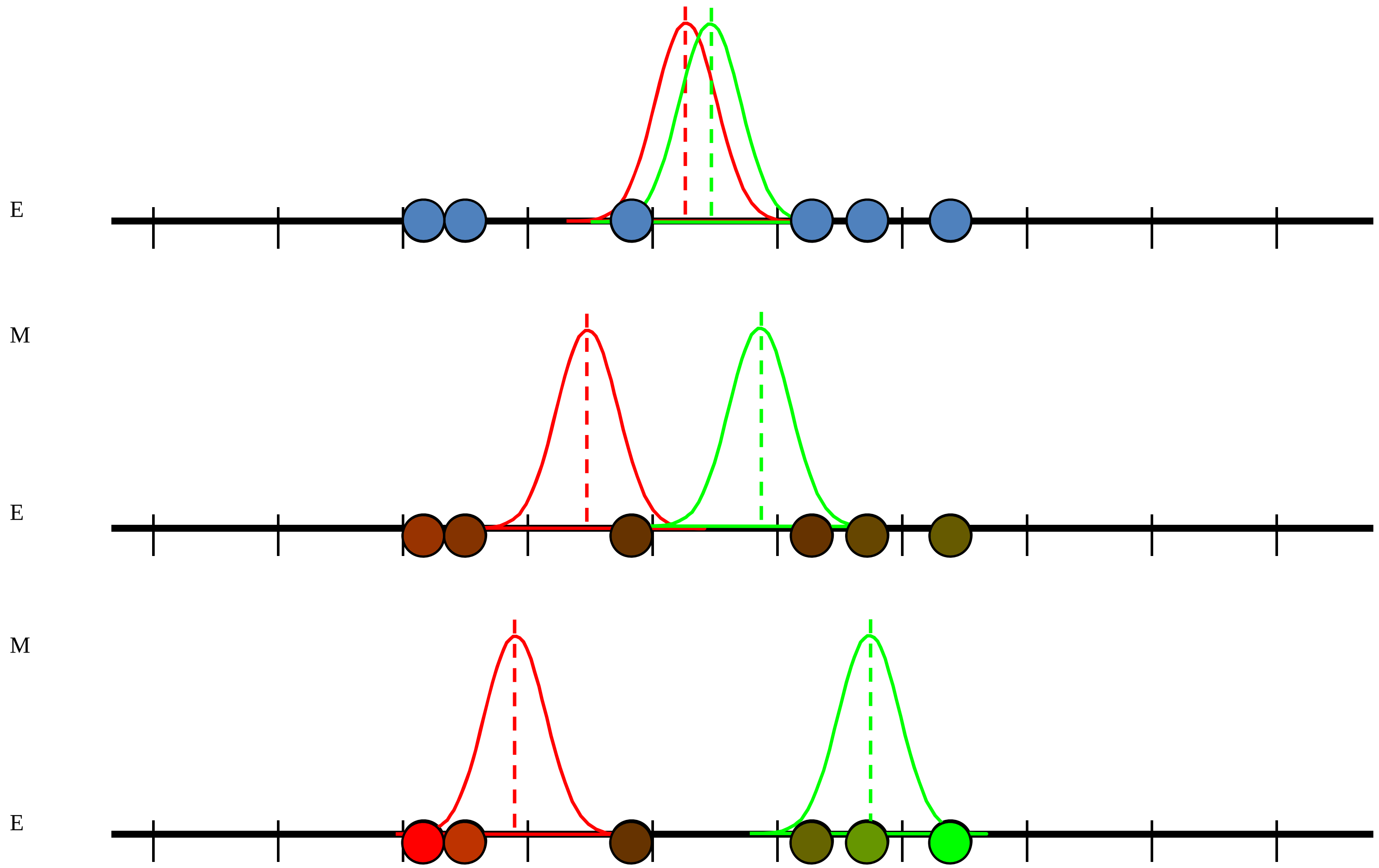
optimal value for θ

we'd like to find it but optimizing $\mathbf{p}(\boldsymbol{X} \mid \theta)$ is very difficult

EM in Pictures

E-step: Compute a *distribution* on the labels of the points, using current parameters

M-step: Update parameters using current guess of label distribution.



Convergence

- It can be proven that the EM algorithm converges to a local maximum of the log-likelihood

$$\ln p(X \mid \theta)$$

EM for Mixture of Gaussians: E step

- Let's revisit the example: $p(x \mid \theta, \rho) = \sum_{j=1}^{m} p(x \mid c_j, \theta_j)\rho_j$
  - with: $p(x \mid c_j, \theta_j) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)$
  - Need to estimate $\theta_1, \ldots, \theta_m$ and $\rho_1, \ldots, \rho_m$
- Define: $z_i^{(k)} = \begin{cases} 1 & \text{if sample } \boldsymbol{i} \text{ was generated by component } \boldsymbol{k} \\ 0 & \text{otherwise} \end{cases}$
  - and $z_i = \{z_i^{(1)}, \ldots, z_i^{(m)}\}$ and $Z = \{z_1, \ldots, z_n\}$
- We need the log-likelihood of observed X and hidden Z

$$\ln p(X, Z \mid \theta) = \ln \prod_{i=1}^{n} p(x_i, z_i \mid \theta) = \sum_{i=1}^{n} \ln p(x_i \mid z_i, \theta) P(z_i)$$

EM for Mixture of Gaussians: E step

- Omitting several steps
- …
- We need to compute $E_Z[z_i^{(k)}]$ (the expected value of the latent variables

$$E_Z\left[z_i^{(k)}\right] = 0 * P\left(z_i^{(k)} = 0 \mid \theta^{(t)}, x_i\right) + 1 * P\left(z_i^{(k)} = 1 \mid \theta^{(t)}, x_i\right)$$

$$= P\left(z_i^{(k)} = 1 \mid \theta^{(t)}, x_i\right) = \frac{p\left(x_i \mid \theta^{(t)}, z_i^{(k)} = 1\right) P\left(z_i^{(k)} = 1 \mid \theta^{(t)}\right)}{p\left(x_i \mid \theta^{(t)}\right)}$$

$$= \frac{\rho_k^{(t)} \exp\left(-\frac{1}{2\sigma^2}\left(x_i - \mu_k^{(t)}\right)^2\right)}{\sum_{j=1}^{m} P\left(x_i \mid \theta^{(t)}, z_i^{(j)} = 1\right) P\left(z_i^{(j)} = 1 \mid \theta^{(t)}\right)} = \frac{\rho_k^{(t)} \exp\left(-\frac{1}{2\sigma^2}\left(x_i - \mu_k^{(t)}\right)^2\right)}{\sum_{j=1}^{m} \rho_j^{(t)} \exp\left(-\frac{1}{2\sigma^2}\left(x_i - \mu_j^{(t)}\right)^2\right)}$$

EM for Mixture of Gaussians: M step

$$Q(\theta \mid \theta^{(t)}) = \sum_{i=1}^{n}\sum_{k=1}^{m} E_Z[z_i^{(k)}]\left(\ln\frac{1}{\sigma\sqrt{2\pi}} - \frac{(x_i-\mu_k)^2}{2\sigma^2} + \ln\rho_k\right)$$

- Need to maximize Q with respect to all parameters
- First differentiate with respect to μ_k

$$\frac{\partial}{\partial\mu_k} Q(\theta \mid \theta^{(t)}) = \sum_{i=1}^{n} E_Z[z_i^{(k)}]\frac{(x_i-\mu_k)}{\sigma^2} = 0$$

$$\Rightarrow new\ \mu_k = \mu_k^{(t+1)} = \frac{\sum_{i=1}^{n} E_Z[z_i^{(k)}]x_i}{\sum_{i=1}^{n} E_Z[z_i^{(k)}]}$$

the mean for class k is the weighted average of all samples, and this weight is proportional to the current estimate of probability that the sample belongs to class k

EM for Mixture of Gaussians: M step

$$Q(\theta \mid \theta^{(t)}) = \sum_{i=1}^{n} \sum_{k=1}^{m} E_Z[z_i^{(k)}] \left(\ln \frac{1}{\sigma\sqrt{2\pi}} - \frac{(x_i - \mu_k)^2}{2\sigma^2} + \ln \rho_k \right)$$

- For ρ_k we have to use Lagrange multipliers to preserve the constraint: $\sum_{j=1}^{m} \rho_j = 1$
- Thus we need to differentiate $F(\lambda, \rho) = Q(\theta \mid \theta^{(t)}) - \lambda \left(\sum_{j=1}^{m} \rho_j - 1 \right)$

$$\frac{\partial}{\partial \rho_k} F(\lambda, \rho) = \sum_{i=1}^{n} \frac{1}{\rho_k} E_Z[z_i^{(k)}] - \lambda = 0 \quad \Rightarrow \sum_{i=1}^{n} E_Z[z_i^{(k)}] - \lambda \rho_k = 0$$

- Summing up over all components: $\sum_{k=1}^{m} \sum_{i=1}^{n} E_Z[z_i^{(k)}] = \sum_{k=1}^{m} \lambda \rho_k$

Since $\sum_{k=1}^{m} \sum_{i=1}^{n} E_Z[z_i^{(k)}] = n$ and $\sum_{k=1}^{m} \rho_k = 1$ we get $\lambda = n$

$$\boxed{\rho_k^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} E_Z[z_i^{(k)}]}$$

The EM Algorithm: Univariate Gaussian Case

The algorithm on this slide applies ONLY to the univariate Gaussian case with known variances

- randomly initialize $\mu_1,.., \mu_m$ and $\rho_1,.., \rho_m$ (subject to $\Sigma\rho_i=1$)
- iterate the following 2 steps until there is no change in $\mu_1,.., \mu_m$ and $\rho_1,.., \rho_m$

E. For all i, k compute

$$E_z\left[z_i^{(k)}\right] = \frac{\rho_k \exp\left(-\frac{1}{2\sigma^2}(x_i-\mu_k)^2\right)}{\sum_{j=1}^{m}\rho_j \exp\left(-\frac{1}{2\sigma^2}(x_i-\mu_j)^2\right)}$$

M. for all k, do parameter update

$$\mu_k = \frac{\sum_{i=1}^{n} E_z\left[z_i^{(k)}\right] x_i}{\sum_{i=1}^{n} E_z\left[z_i^{(k)}\right]}$$

$$\rho_k = \frac{1}{n}\sum_{i=1}^{n} E_z\left[z_i^{(k)}\right]$$

The EM Algorithm

- For the more general case of multivariate Gaussians with unknown means and variances
- E step

$$E_Z\left[z_i^{(k)}\right] = \frac{\rho_k\, p(x / \mu_k, \Sigma_k)}{\sum_{j=1}^{m} \rho_j\, p(x / \mu_j, \Sigma_j)}$$

$$p(x / \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{d/2} \left|\Sigma_k^{-1}\right|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_k)^t \Sigma_k^{-1}(x-\mu_k)\right]$$

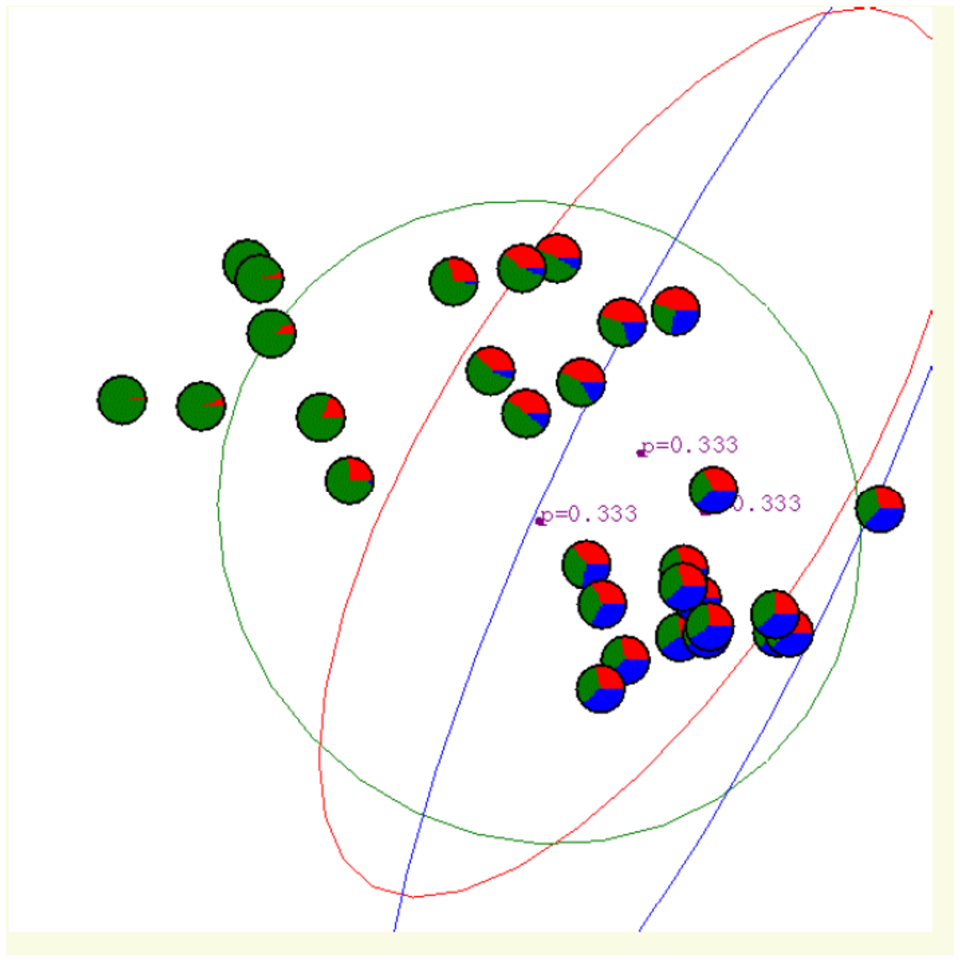
- M step

$$\rho_k = \frac{1}{n}\sum_{i=1}^{n} E_Z\left[z_i^{(k)}\right]$$

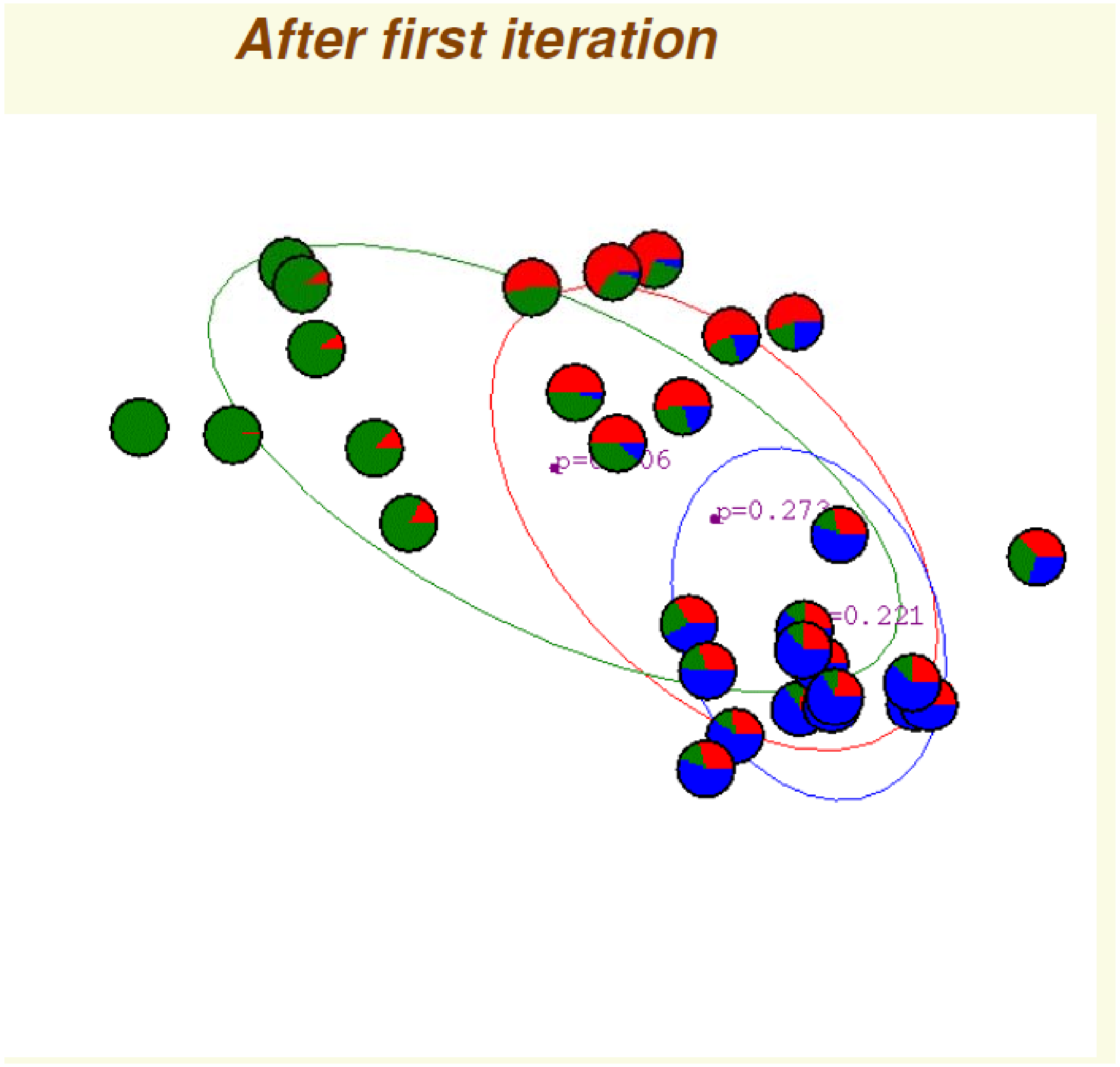
$$\mu_k = \frac{\sum_{i=1}^{n} E_Z\left[z_i^{(k)}\right] x_i}{\sum_{i=1}^{n} E_Z\left[z_i^{(k)}\right]}$$

$$\Sigma_k = \frac{\sum_{i=1}^{n} E_Z\left[z_i^{(k)}\right](x_i-\mu_k)(x_i-\mu_k)^T}{\sum_{i=1}^{n} E_Z\left[z_i^{(k)}\right]}$$

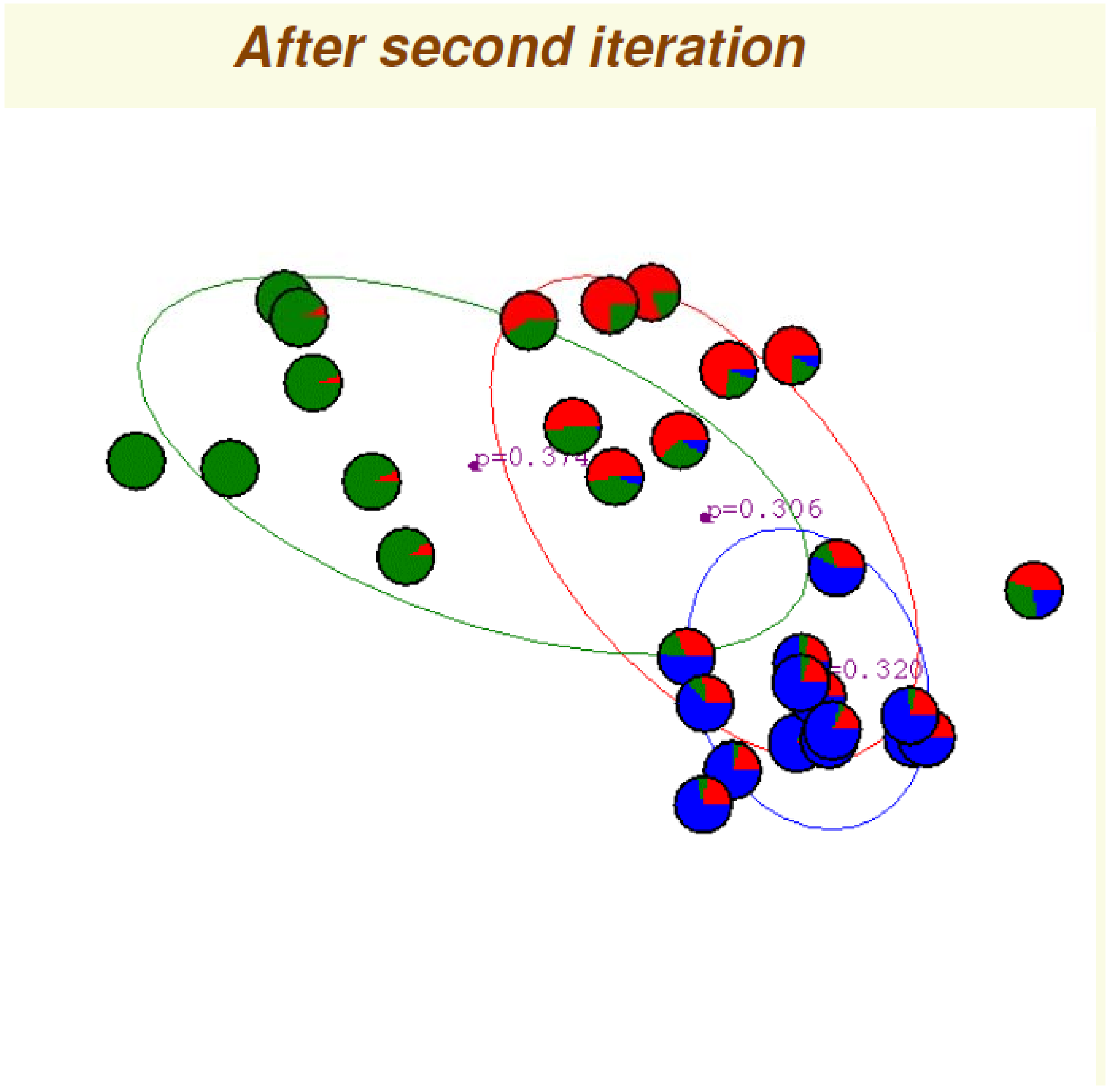
EM Gaussian Mixture Example



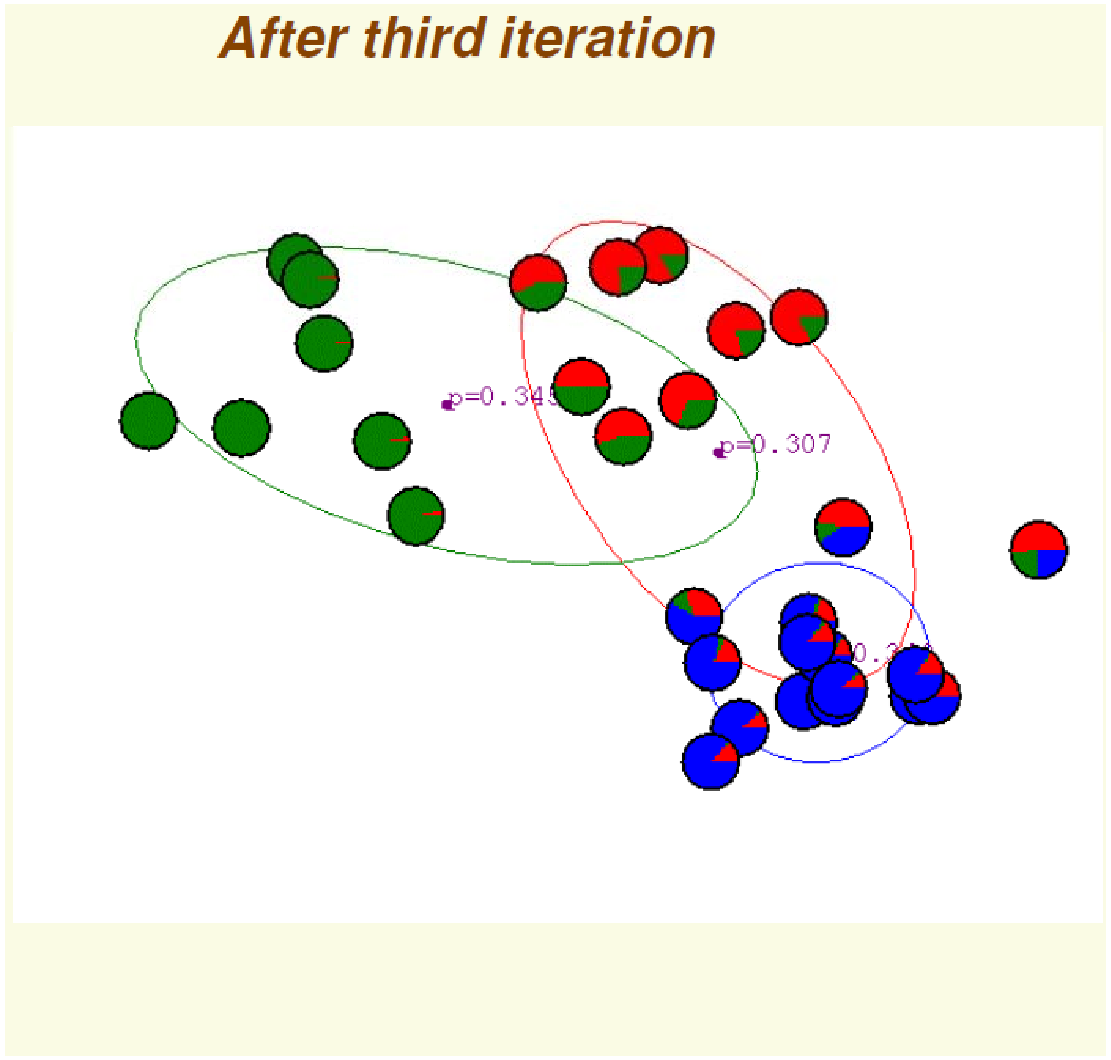
EM Gaussian Mixture Example



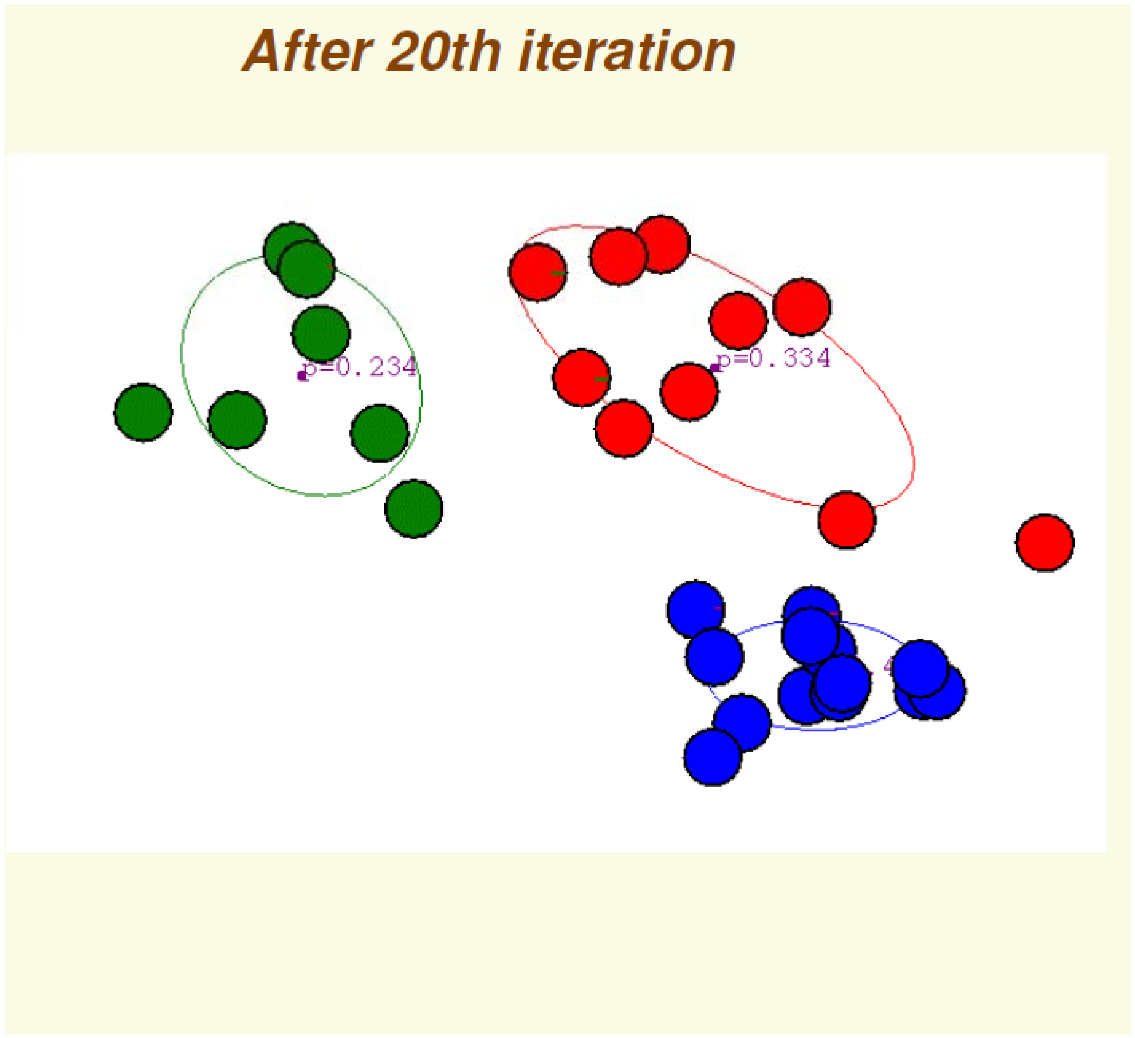
EM Gaussian Mixture Example



EM Gaussian Mixture Example

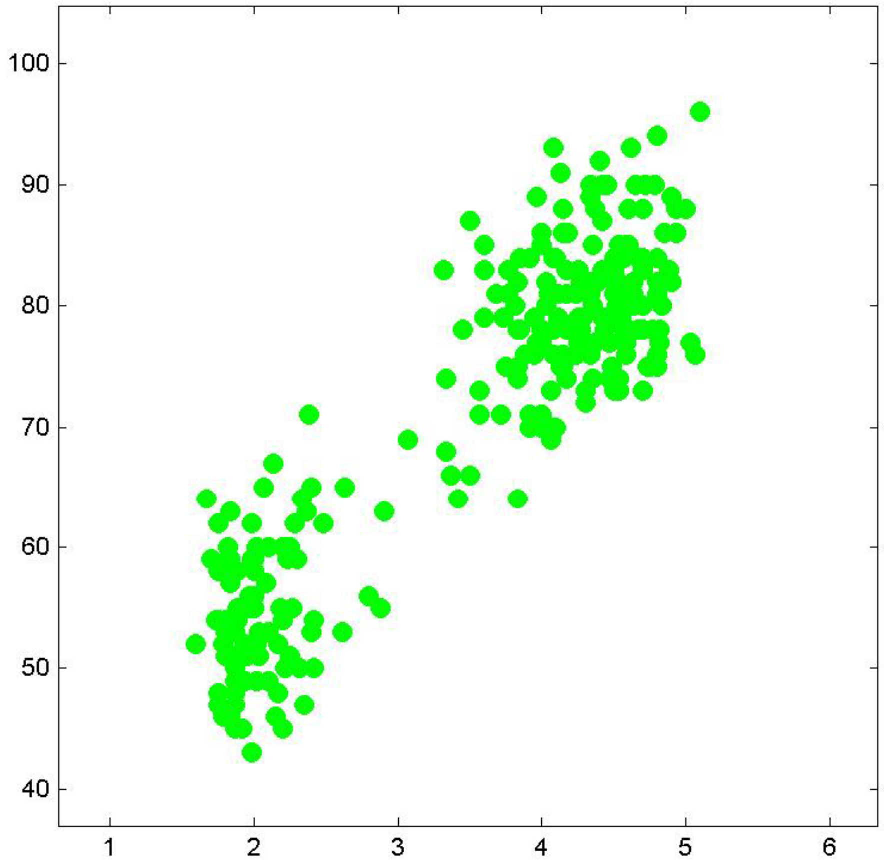


EM Gaussian Mixture Example

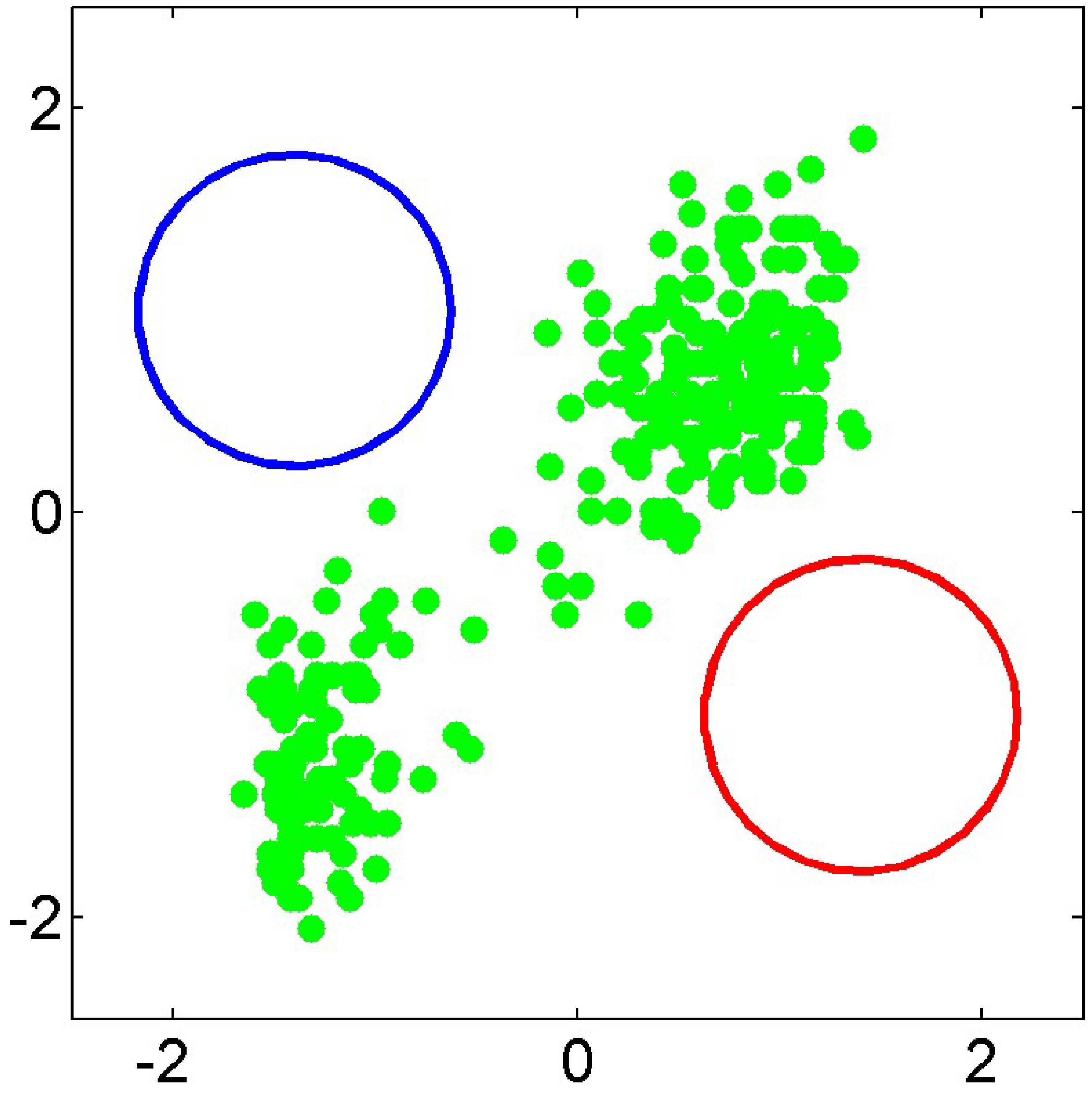


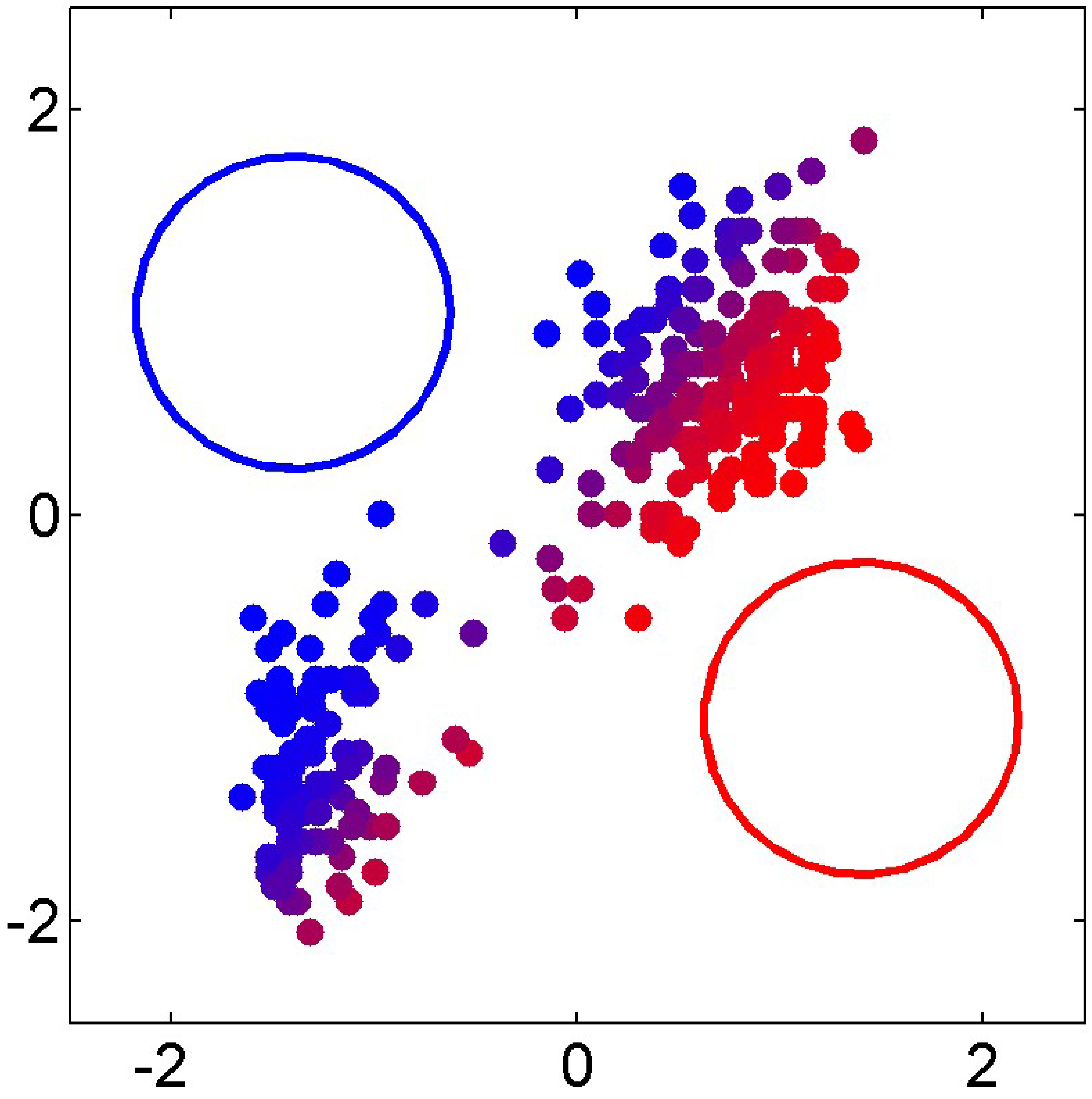
Volcano Eruption Data Set

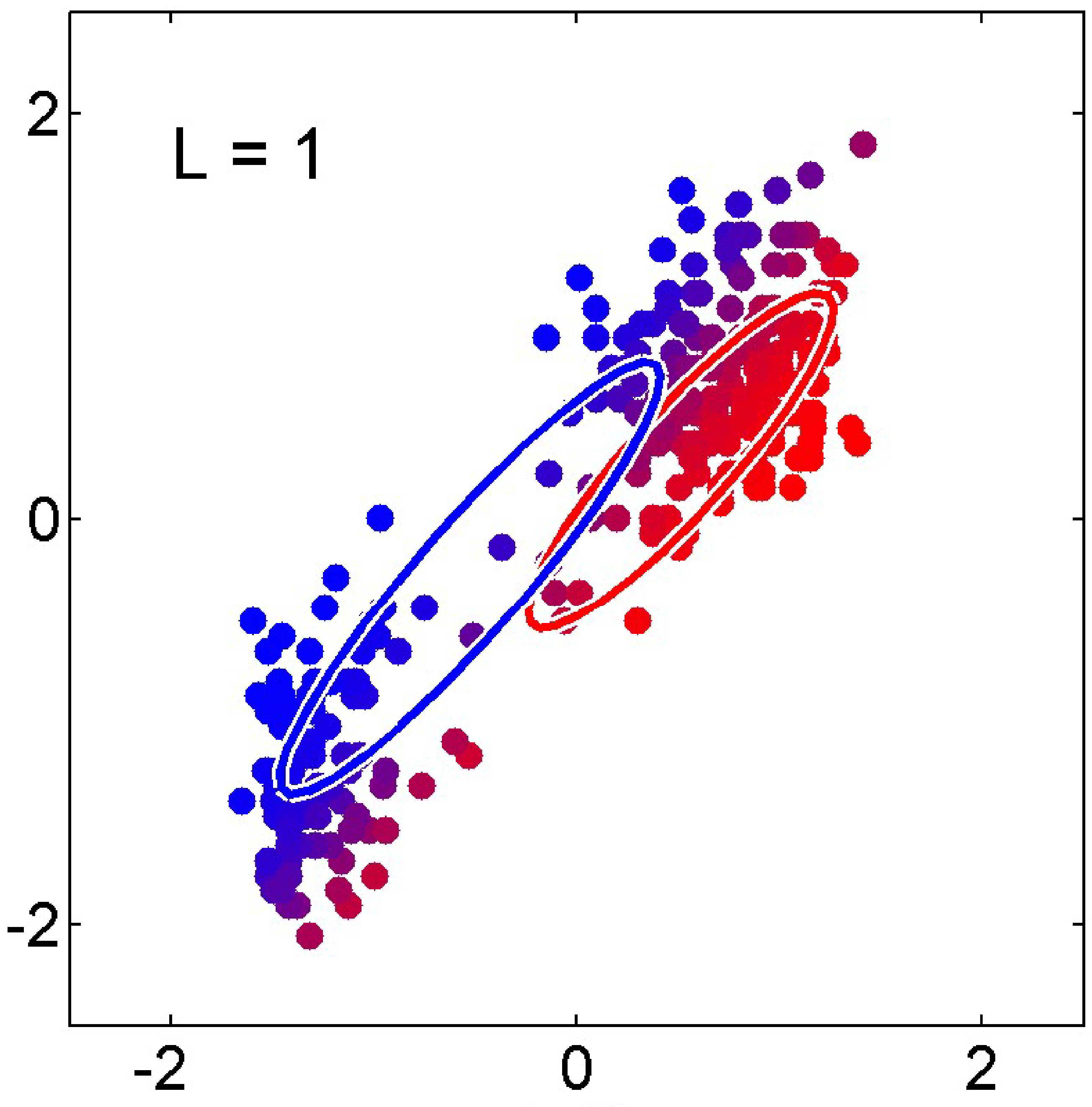
Time between eruptions (minutes)

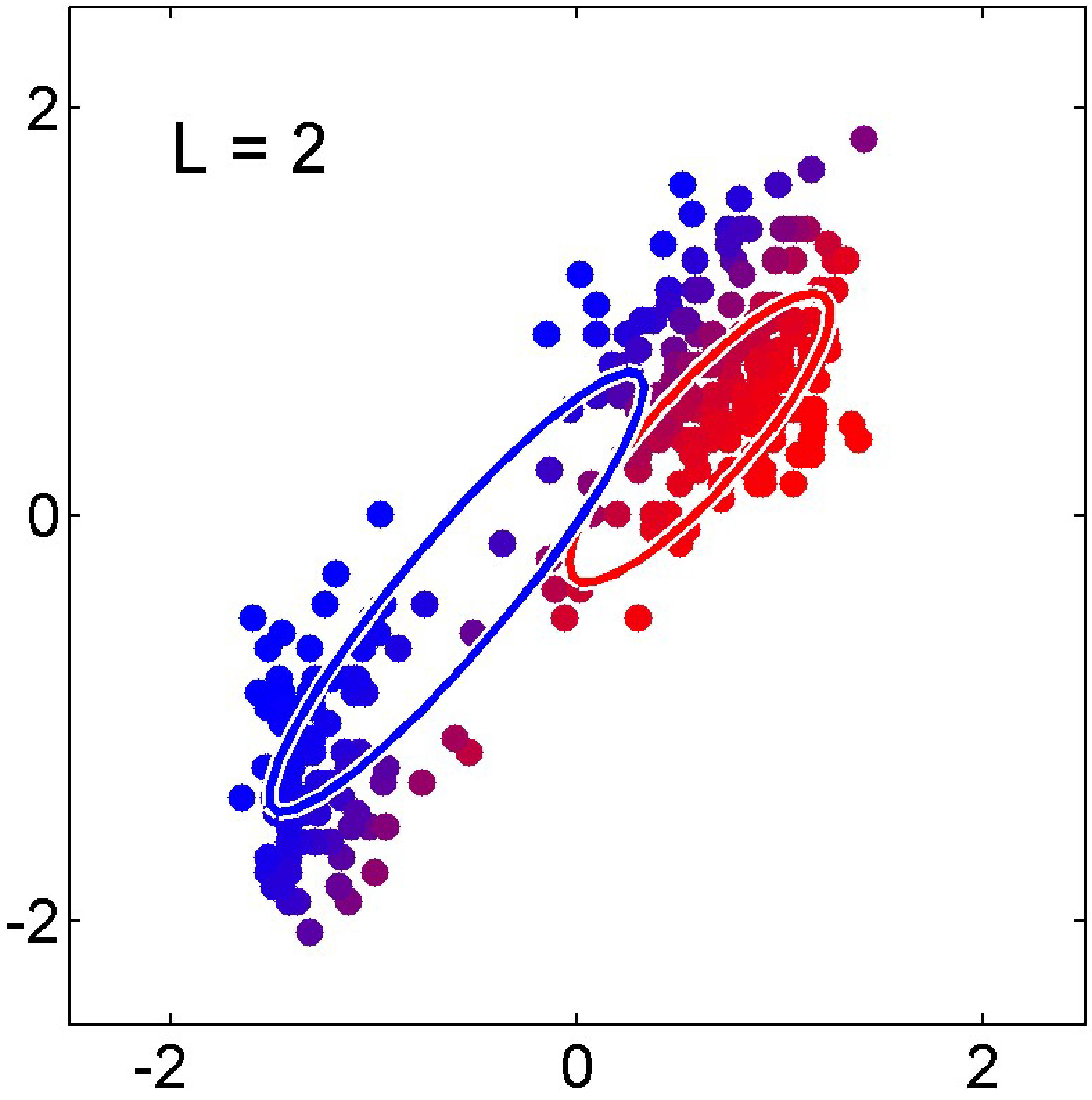


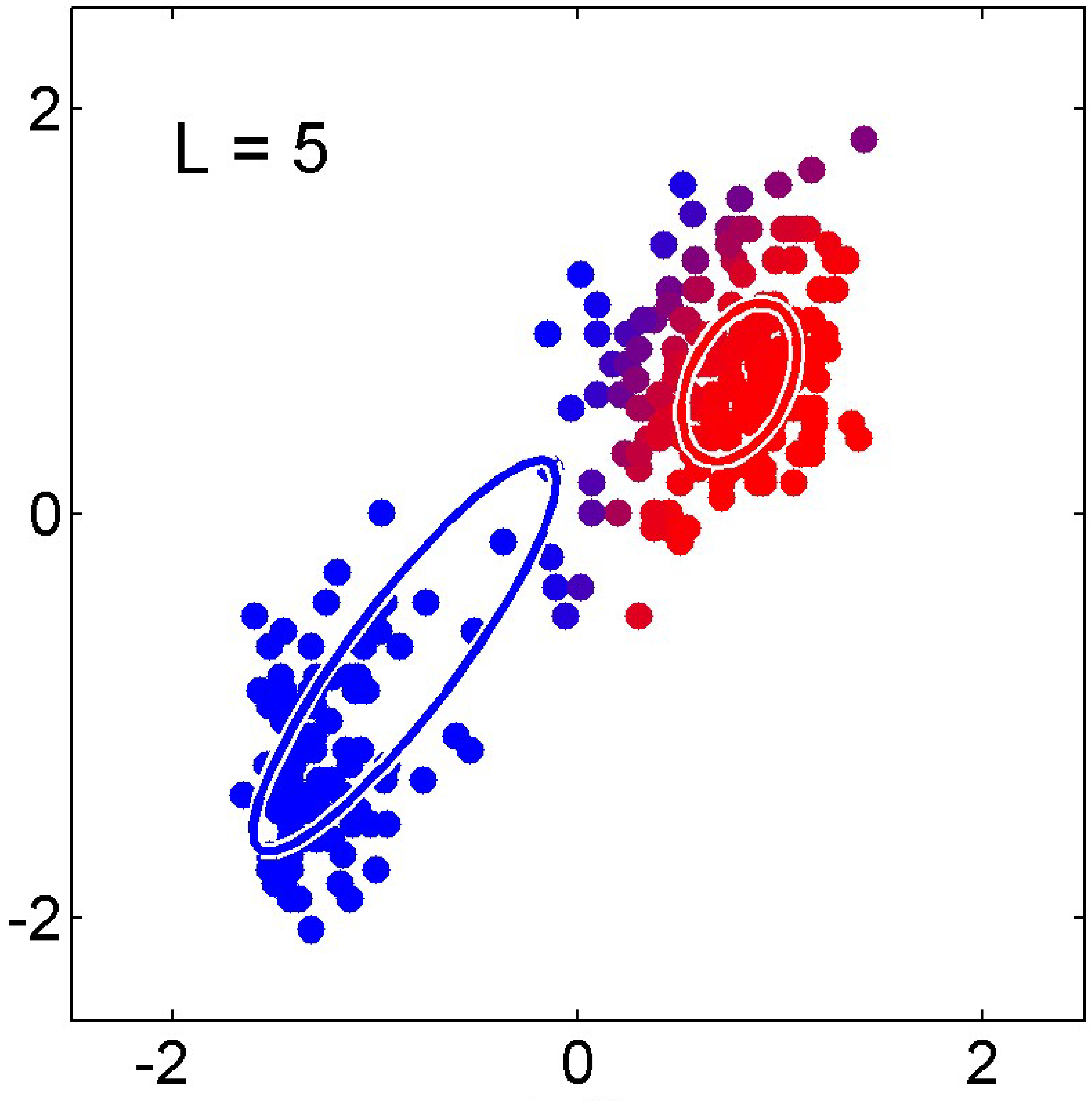
Duration of eruption (minutes)

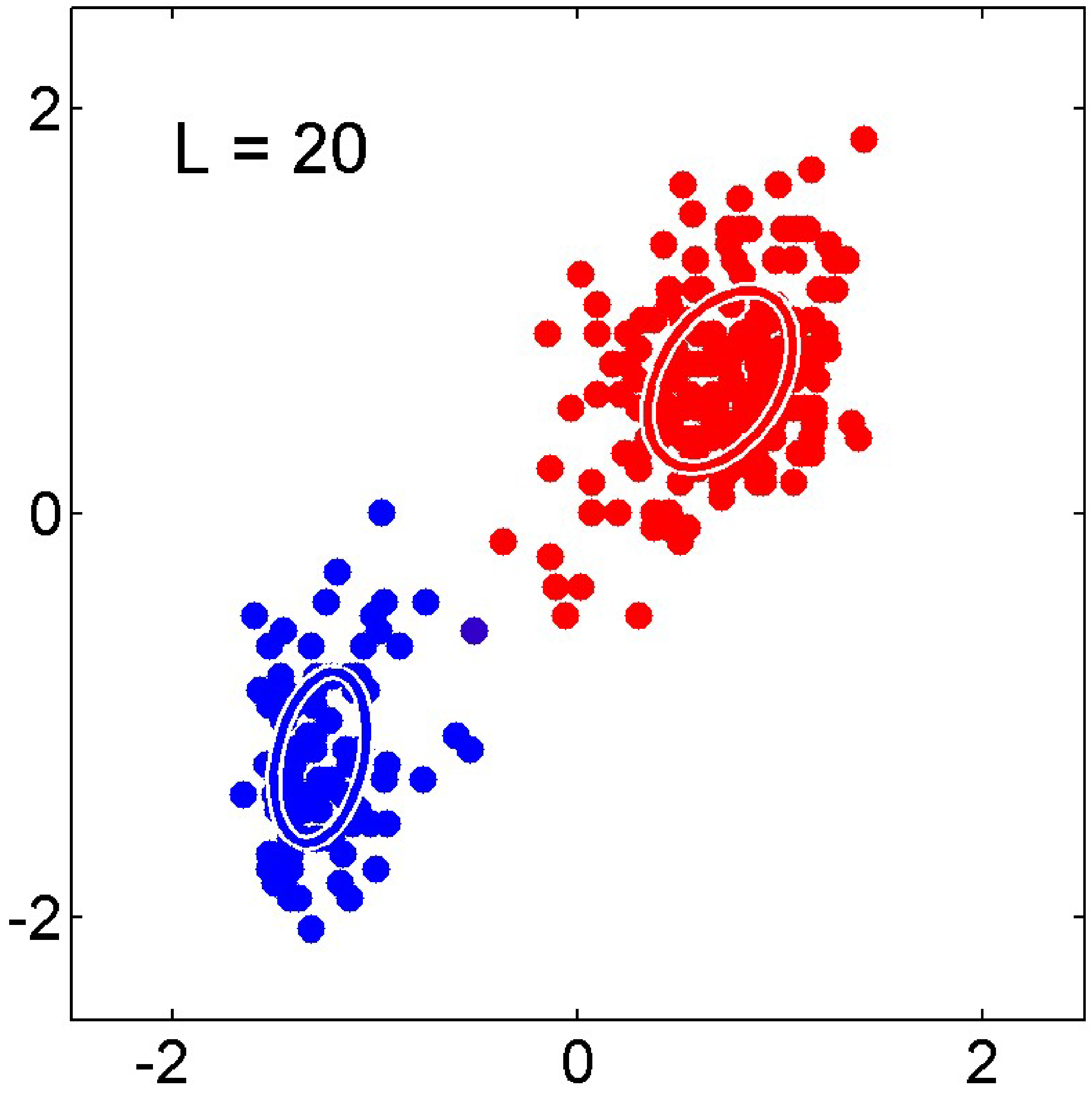

2
0
-2
-2
0
2


2
0
-2
-2
0
2


L = 1
2
0
-2
-2
0
2


L = 2
2
0
-2
-2
0
2


L = 5
2
0
-2
-2
0
2


L = 20
2
0
-2
-2
0
2

EM Summary

- Advantages
  - If the assumed data distribution is correct, the algorithm works well
- Disadvantages
  - If the assumed data distribution is wrong, results can be quite bad
  - In particular, bad results if incorrect number of mixture components is used