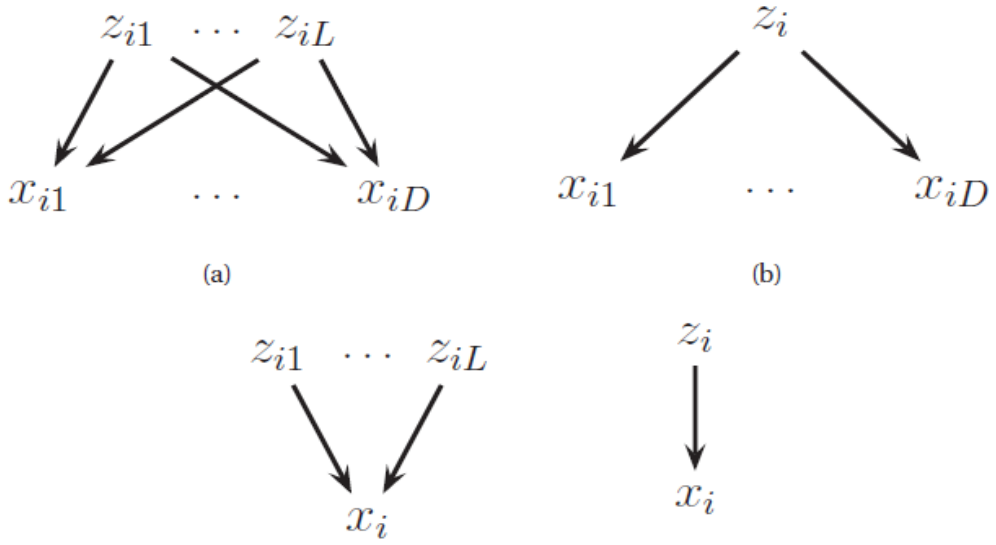
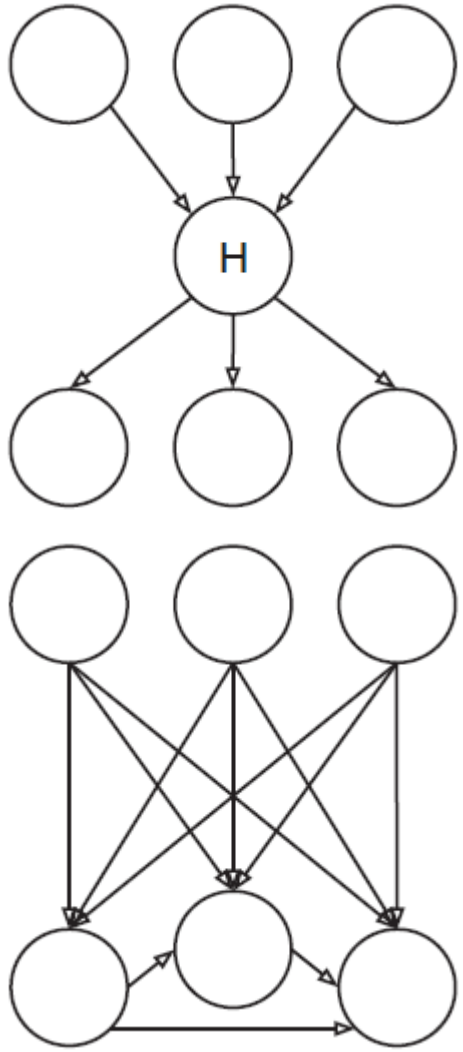
Mixture models and the EM algorithm

Latent variable models

- Latent / hidden: not observed in the data
- Assume that the observed variables are correlated because they arise from a hidden common “cause”
- Model with hidden variables are also known as **latent variable models** or **LVM**s.





The leaves represent medical symptoms. The roots represent primary causes, such as smoking, diet and exercise. The hidden variable can represent mediating factors, such as heart disease, which might not be directly visible.

Mixture models

- Most of the density functions we have considered so far are **unimodal**, that is, they have at most one peak.
- However, often we want to be able to represent densities with multiple modes. A common way to do this is to create a **mixture model**, which is a convex combination of pdf's:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p_k(\mathbf{x}_i|\boldsymbol{\theta})$$

- where *K* is the (fixed) number of mixture component, π is a vector of **mixing weights** ($0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$), and p_k is the *k'th* **base distribution** (can be of any type)

Mixture of Gaussians

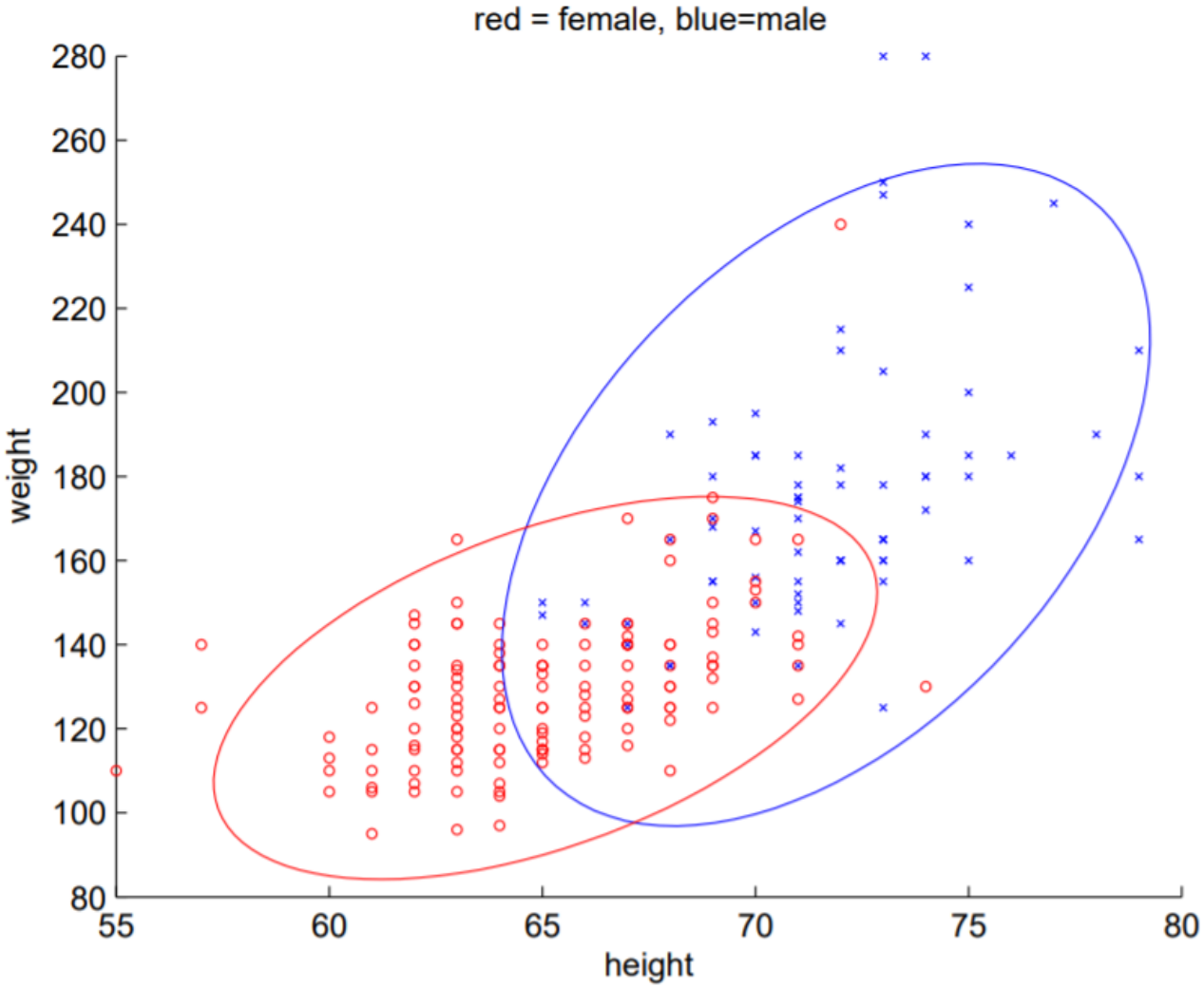
- The most widely used mixture model is the **mixture of Gaussians** (MOG) or **Gaussians mixture modes (GMM)**
- In this model, each base distribution in the mixture is a multivariate Gaussian with mean μ_k and covariance matrix Σ_k
- The model has the form

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where $\theta = \{\pi_k, \mu_k, \Sigma_k\}$ and *K* is the number of components

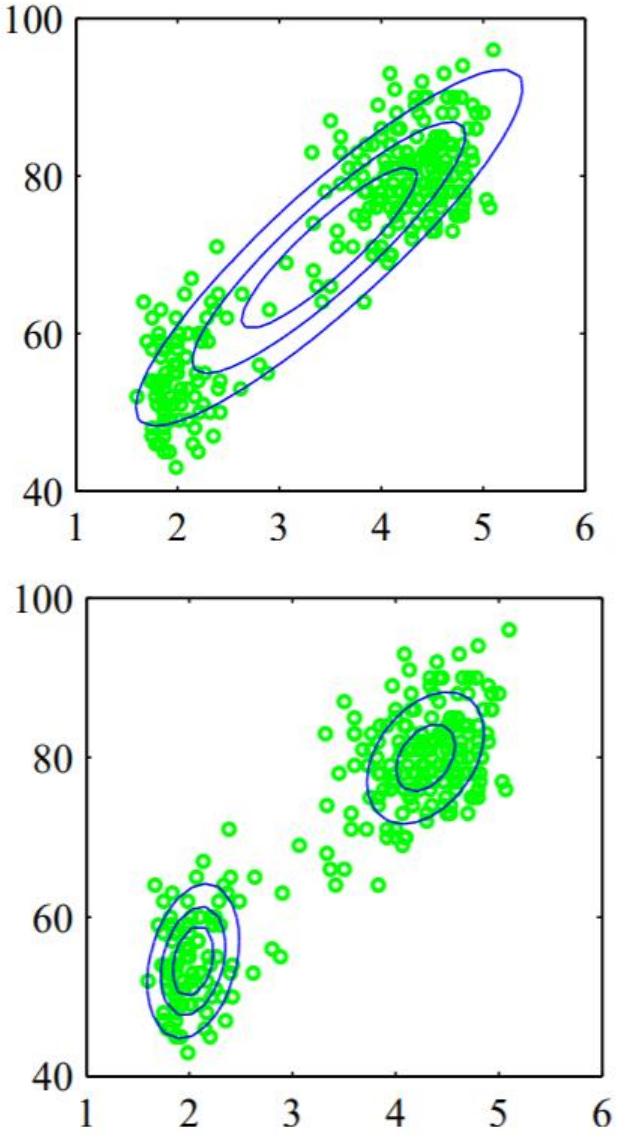
Mixture of Gaussians

- Dataset of height and weight
- It is clear that there are two subpopulations in this data set, and in this case they are easy to interpret: one represents males and the other females.
- Within each class or cluster, the data is fairly well represented by a 2D Gaussian (as can be seen from the fitted ellipses), **but to model the data as a whole, we need to use a Gaussian mixture model (GMM)**



Mixture of Gaussians

- Note that the clusters do not always have an obvious meaning.
- For example, in these figures we see a dataset which seems to have two clusters.
- Discovering the number and shape of clusters in data is an example of **unsupervised learning**.
- The Bayesian approach to fit a GMM (which we will explain later) has been used to discover clusters.



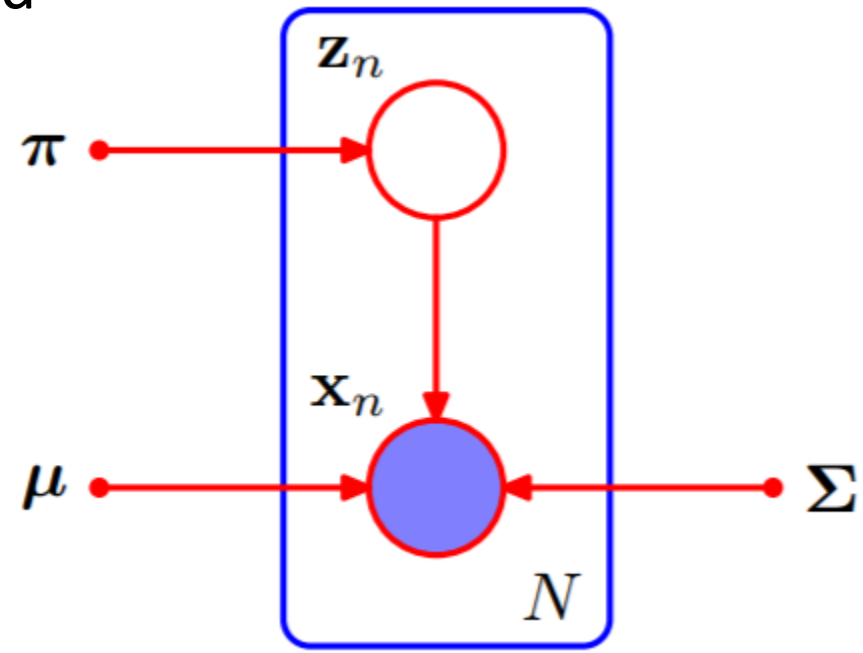
Horizontal axis is duration of the erruption in minutes. Vertical axis is time until the next erruption in minutes. (a) A single Gaussian. (b) A mixture of two Gaussians.

Mixture of Gaussians

- A helpful way to think about GMMs is to imagine that each data point $\boldsymbol{x_n}$ **has an associated latent indicator variable** $\boldsymbol{z_n} \in \{1, \ldots, K\}$ that specifies which mixture component that data point came from.
- These are **like the class labels** in a classifier, except we assume the labels are hidden (since this is unsupervised learning).
- We can write the complete data log likelihood as

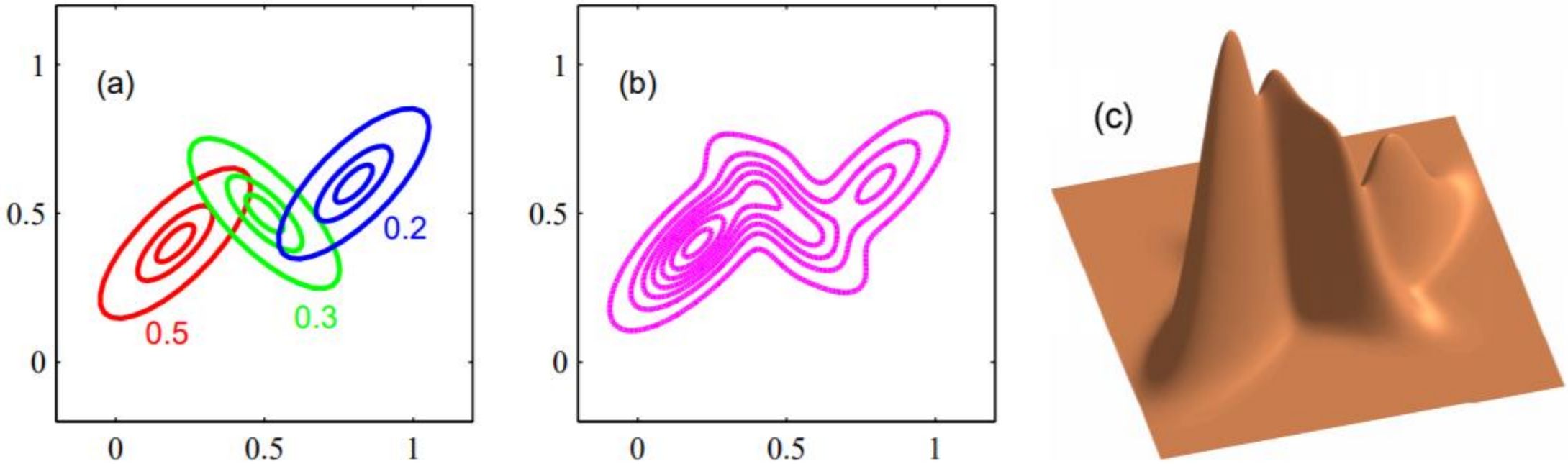
$$\begin{aligned} \ell_c(\theta) &= \log p(x_{1:N}, z_{1:N}|\theta) \\ &= \log \prod_n p(z_n|\pi) p(x_n|z_n, \theta) \end{aligned}$$

- Where $p(z_n = k|\pi) = \pi_k$ is a multinomial and $p(x_n|z_n = k, \theta) = \mathcal{N}(x_n|mu_k, \Sigma_k)$ is a Gaussian



Mixture of Gaussians

- A mixture of 3 Gaussians.
  - (a) We show the contours of constant probability and the mixing weights.
  - (b) A contour plot of the overall density.
  - (c) A 3D surface plot of the overall density



Mixture of multivariate Bernoullis

- A mixture of Gaussians is appropriate if the data is real-valued, but what about binary data?
- In this case, we can replace the Gaussian densities with a product of Bernoullis, and the class-conditional density is:

$$p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta}) = \prod_{j=1}^{D} \mathrm{Ber}(x_{ij}|\mu_{jk}) = \prod_{j=1}^{D} \mu_{jk}^{x_{ij}} (1 - \mu_{jk})^{1-x_{ij}}$$

- Where μ_{jk} is the probability that bit *j* turns on in cluster *k,* and the data consists of D-dimensional bit vectors

Using mixture models for clustering

- The most common application of mixture models is for clustering
- Clustering is the process of grouping similar objects together

Using mixture models for clustering

- We first fit the mixture model, and then compute $p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})$ which represents the posterior probability that point *i* belongs to cluster *k*.
- This is known as the **responsibility** of cluster *k* for point *i*, and can be computed using Bayes rule as follows:

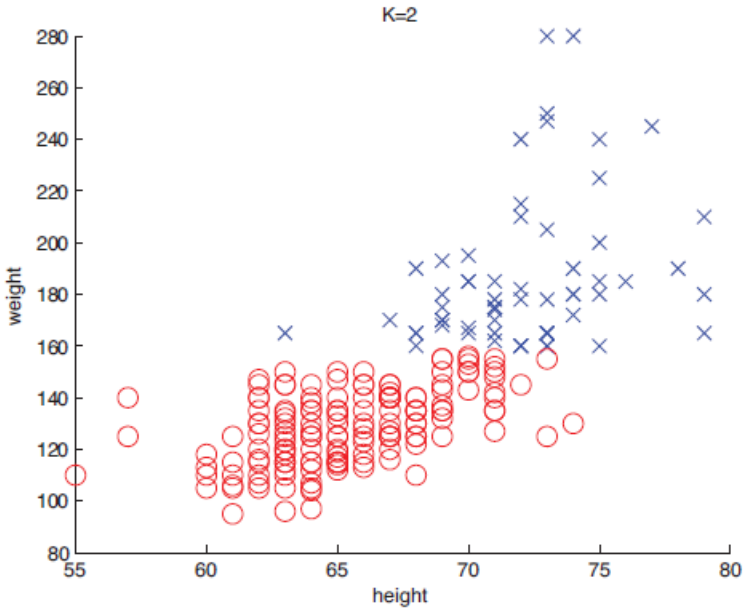
$$r_{ik} \triangleq p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(z_i = k|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta})}{\sum_{k'=1}^{K} p(z_i = k'|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k', \boldsymbol{\theta})}$$

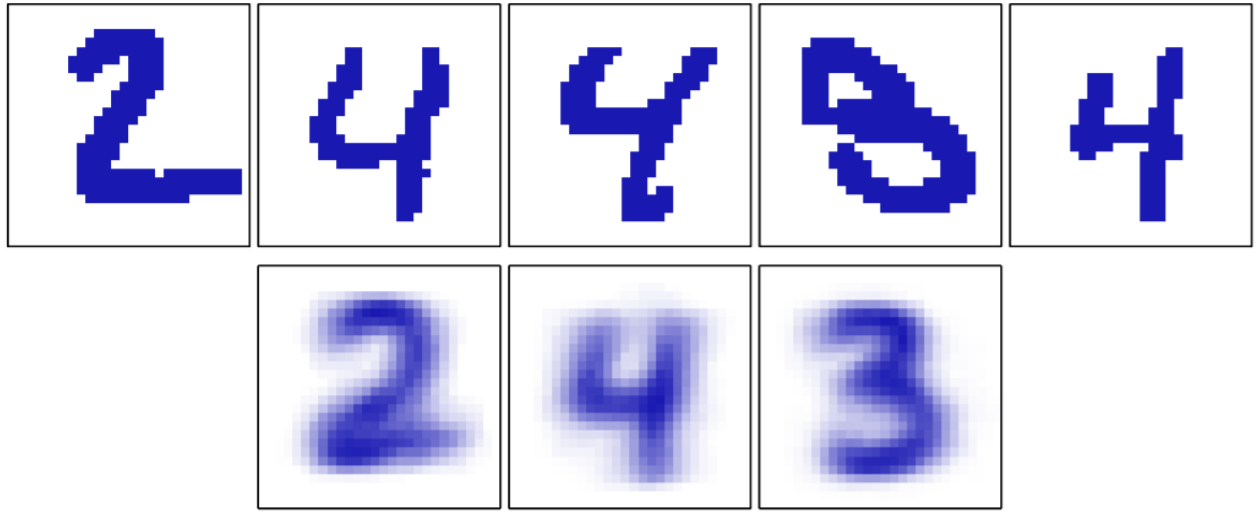
- This procedure is called **soft clustering**

Using mixture models for clustering

- **Hard clustering** can be computed using the MAP estimate, using the following equation:

$$z_i^* = \arg\max_k r_{ik} = \arg\max_k \log p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta}) + \log p(\mathbf{z}_i = k|\boldsymbol{\theta})$$





Example of clustering binary data, using mixture of Bernoullis.
Top: 3 kinds of binary digits. Each image is a 28 × 28 binary image.
Bottom: the 3 cluster centers that are learned.

Parameter estimation for mixture models

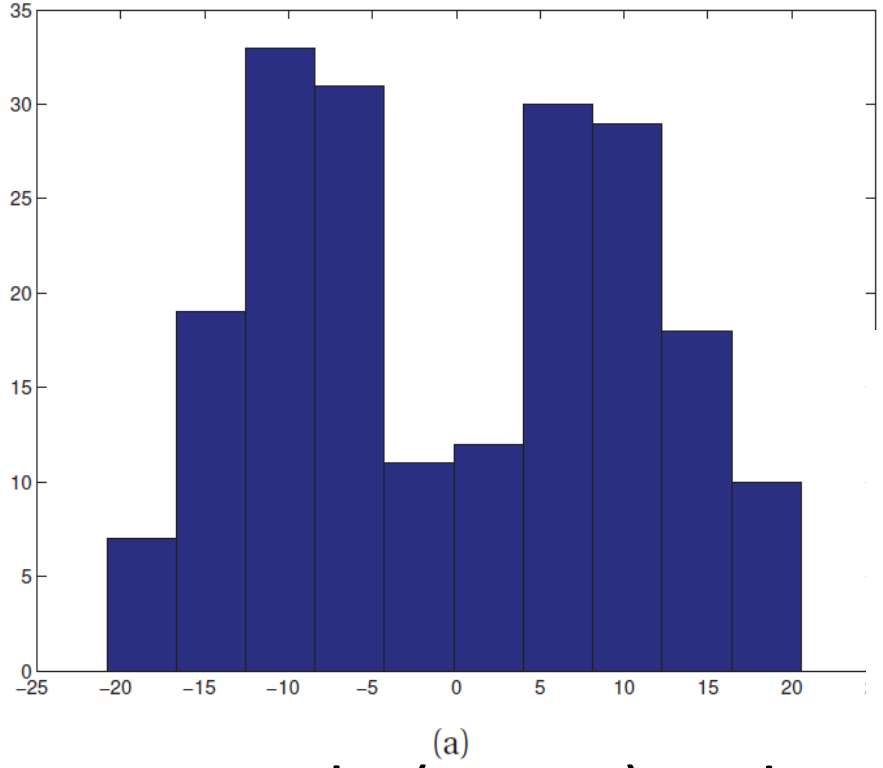
- We have shown how to compute the posterior over the hidden variables given the observed variables, assuming the parameters are known
- We need to **learn the parameters**

Parameter estimation for mixture models

- Not only do we have to estimate θ but also z_n.
- Since $\boldsymbol{z_n}$ **are hidden**, this **complicates the computation** of MAP and ML estimates.
- For each possible ways of "filling in" the z_i's, we get a different unimodal likelihood.
- Thus when we marginalize out over the z_i's, we get a multi-modal posterior for $p(\boldsymbol{\theta}|\mathcal{D})$
- This modes correspond to different labeling of the clusters

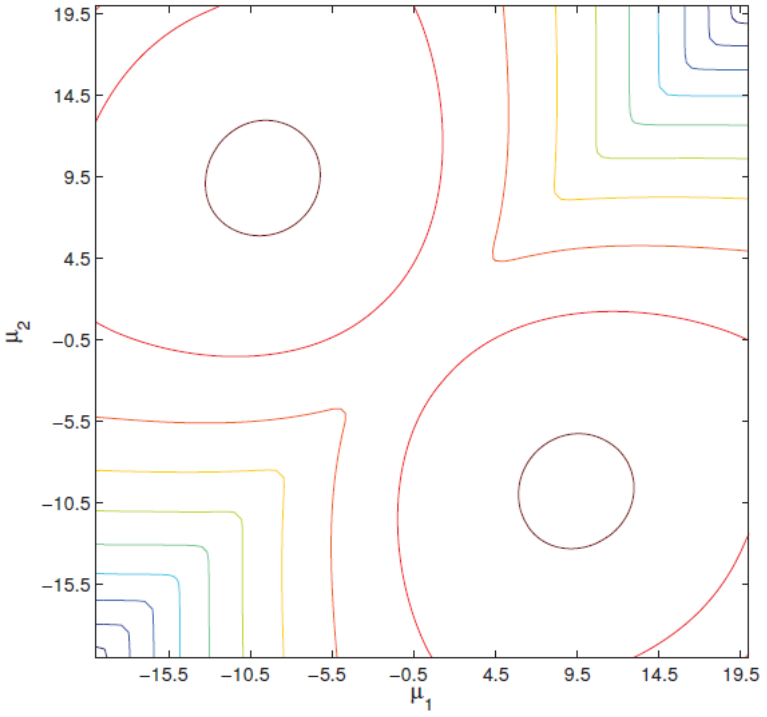
Parameter estimation for mixture models

- N = 200 data points sampled from a mixture of 2 Gaussians in 1d, with π_k = 0.5,
- σ_k = 5, μ_1 = −10 and μ_2 = 10. Right: Likelihood surface $p(D|\mu_1, \mu_2)$, with all other parameters set to their true values. We see the two symmetric modes, reflecting the unidentifiability of the parameters.



(a)

- Two peeks (-10,10) and (10,-10), we say parameters are **not identifiable**



(b)

Parameter estimation for mixture models

- There is not a unique MLE
- Therefore there cannot be a unique MAP estimate and hence the posterior must be multimodal
- **Computing the MAP estimate is non-convex**
- Hard to find global optimum, most algorithm will find only local optimum. Which one they find, depends on where they start
- In practice, we will run a local optimizer, perhaps using **multiple random restarts** to increase the chance of finding “good” local optimum
- Careful initialization can help a lot, too, which is case-by-case sensitive

The EM algorithm

- For many models in machine learning and statistics, computing the ML or MAP parameter estimate is easy provided we observe all the values of all the relevant random variables, i.e., if we have complete data.
- However, if we have missing data and/or latent variables, then computing the ML/MAP estimate becomes hard.
- Two possible solutions:
  - Use a gradient-based optimizer
  - Use Expectation Maximization
    - Often much simpler in these cases
    - Simple iterative algorithm, often with closed-form updates at each step

Expectation Maximization Algorithm

- Observation: Know values of z_i $\Rightarrow$ easy to maximize

- Key idea: iterative updates
  - Given parameter estimates, “infer” all z_i variables
  - Given inferred z_i variables, maximize with respect to parameters

Expectation Maximization - basic idea

- So let us define the **expected complete data log likelihood** as follows:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) = \mathbb{E}\left[\ell_c(\boldsymbol{\theta}) \middle| \mathcal{D}, \boldsymbol{\theta}^{t-1}\right]$$

  - *Q* is called the **auxiliary function**.
- The goal of the **E step** is to compute $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$
  - With this we cluster the data (calculate the probability $p(z_i=k|x_i)$ and responsibility r_{ik})
- In the **M step**, we optimize the Q function wrt θ:

$$\boldsymbol{\theta}^t = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$$

  - With this we find the new parameters based on the clustering in the E step
- To perform MAP estimation, we modify the M step as follows:

$$\boldsymbol{\theta}^t = \operatorname{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) + \log p(\boldsymbol{\theta})$$

- The EM algorithm monotonically increases the log likelihood of the observed data, or it stays the same

Expectation-Maximization

- The goal is to find MLE (or MAP) estimate. But for a model with latent variables, it is more difficult:

$$l(\theta) = \sum_{i=1}^{N} \log p(x_i|\theta) = \sum_{i=1}^{N} \log \sum_{z_i} p(x_i, z_i|\theta)$$

  - we need to marginalize out the unobserved latent variable z_i
- Therefore, EM defines **complete data log-likelihood**

$$l_c(\theta) = \sum_{i=1}^{N} \log p(x_i, z_i|\theta)$$
$$= \sum_{i=1}^{N} \log[p(z_i|\theta)p(x_i|z_i, \theta)]$$

  - again, we do not observe z_i

Expectation-Maximization

- However, we take the **expected value** with respect to the conditional distribution of z_i given the data and our previous value of the parameters, θ^{t-1}, called **auxiliary function**

$$Q(\theta, \theta^{t-1}) = E[l_c(\theta)|\mathcal{D}, \theta^{t-1}]$$
$$= \sum_{i=1}^{N} E[\log[p(z_i|\theta)p(x_i|z_i, \theta)]]$$

  - Instead of using exact values of zi, we use the expected value
  - In this way we can fill in the zi values, so we have function only of the values we want to optimize – θ

Expectation-Maximization

- If we assume that z_i is discrete, we can write the previous equation as:

$$
\begin{aligned}
Q(\theta, \theta^{t-1}) &= \sum_{i=1}^{N} E[\log[p(z_i|\theta)p(x_i|z_i,\theta)]] \\
&= \sum_{i=1}^{N} E\big[\log \Pi_{k=1}^{K} [p(z_i = k|\theta)p(x_i|z_i = k,\theta)]^{I(z_i=k)}\big] \\
&= \sum_{i=1}^{N} E\big[\sum_{k=1}^{K} \log[p(z_i = k|\theta)p(x_i|z_i = k,\theta)]^{I(z_i=k)}\big]
\end{aligned}
$$

  - the indicator function acts like a "filter" for the products over k, taking out the exact value of zi.

Expectation-Maximization

- Further simplification to the auxiliary function:

$$
\begin{aligned}
Q(\theta, \theta^{t-1}) &= \sum_{i=1}^{N} E\Big[\sum_{k=1}^{K} \log[p(z_i = k|\theta) p(x_i|z_i = k, \theta)]^{I(z_i=k)} \Big] \\
&= \sum_{i=1}^{N} E\Big[\sum_{k=1}^{K} I(z_i = k) \log[p(z_i = k|\theta) p(x_i|z_i = k, \theta)] \Big] \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} E[I(z_i = k)] \log[p(z_i = k|\theta) p(x_i|z_i = k, \theta)] \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} p(z_i = k|\mathcal{D}, \theta^{t-1}) \log[p(z_i = k|\theta) p(x_i|z_i = k, \theta)]
\end{aligned}
$$

  - expectation is only performed over the indicator function, while the probability statements in the log are readily evaluated to functions of only the parameters

EM summary

- To summarize, the EM loop aims to maximize the expected complete data log-likelihood, or auxiliary function $Q(\theta,\theta t-1)$ in two steps:
  1. Given the parameters $\theta t-1$ from the previous iteration, evaluate the Q function so that it's only in terms of θ.
  2. Maximize this simplified Q function in terms of θ. These parameters becomes the starting point for the next iteration.

EM for GMM

- Fit a mixture of Gaussians using EM (assuming the number of mixture components is K)
- We usually initialize by assigning each μ_k to one of the data points chosen at random, and setting Σ_k to be a small fraction of the global covariance.
- The **E step** is to compute

$$p(z_n = k|x_n, \theta^{old}) = r_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}$$

where r_{nk} is called the **responsibility** of cluster *k* for data point *n*. (the probability that it belongs to cluster *k*)

normalized to sum to one (over clusters)

EM for GMM

- The **M step**
  - Maximizing the expected complete data log likelihood (Q) with respect to π and μ_k, Σ_k separately.
  - For each cluster (Gaussian) update the parameters
  - For π we have

$$\pi_k = \frac{1}{N}\sum_i r_{ik} = \frac{r_k}{N}$$

Total responsibility allocated to cluster *k*

Fraction of total assigned to cluster *k*

where $r_k \triangleq \sum_i r_{ik}$ is the weighted number of points assigned to cluster *k*

- The estimates of the parameters μ_k and Σ_k are given by

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik}\mathbf{x}_i}{r_k}$$

Weighted mean of assigned data

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik}\mathbf{x}_i\mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k\boldsymbol{\mu}_k^T$$

Weighted covariance of assigned data (use new weighted means here)

EM for GMMs

- **Auxiliary function**

$$Q(\theta, \theta^{t-1}) = \sum_{i=1}^{N} \sum_{k=1}^{K} p(z_i = k|\mathcal{D}, \theta^{t-1}) \log[p(z_i = k|\theta) p(x_i|z_i = k, \theta)]$$
$$= \sum_{i=1}^{N} \sum_{k=1}^{K} \left[r_{ik} \log p(z_i = k|\theta) + r_{ik} \log p(x_i|z_i = k, \theta) \right]$$

- **E step**

$$r_{ik} = \frac{\pi_k p(\mathbf{x}_i|\boldsymbol{\theta}_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(\mathbf{x}_i|\boldsymbol{\theta}_{k'}^{(t-1)})}$$

- **M step**

$$\pi_k = \frac{1}{N} \sum_i r_{ik} = \frac{r_k}{N}$$

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik} \mathbf{x}_i}{r_k}$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} = \frac{\sum_i r_{ik} \mathbf{x}_i \mathbf{x}_i^T}{r_k} - \boldsymbol{\mu}_k \boldsymbol{\mu}_k^T$$

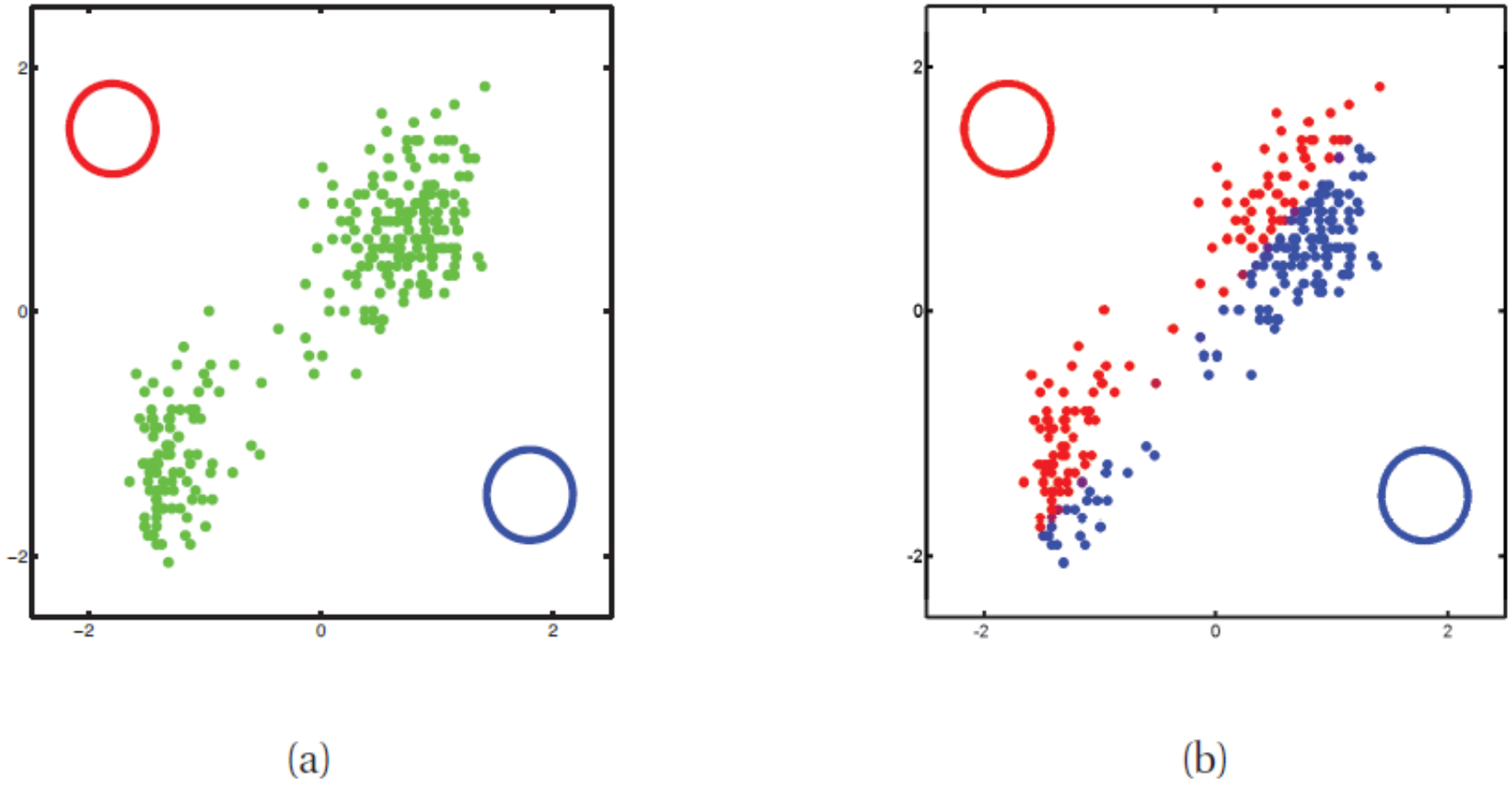
EM for GMM

- After computing the new estimates, we set $\boldsymbol{\theta}^t = (\pi_k, \mu_k, \boldsymbol{\Sigma}_k)$ for $k = 1 : K$, and go to the next E step.
- Example
- We start with $\boldsymbol{\mu}_1 = (-1, 1)$, $\boldsymbol{\Sigma}_1 = \mathbf{I}$, $\boldsymbol{\mu}_2 = (1, -1)$, $\boldsymbol{\Sigma}_2 = \mathbf{I}$. We color code points such that blue points come from cluster 1 and red points from cluster 2. More precisely, we set the color to

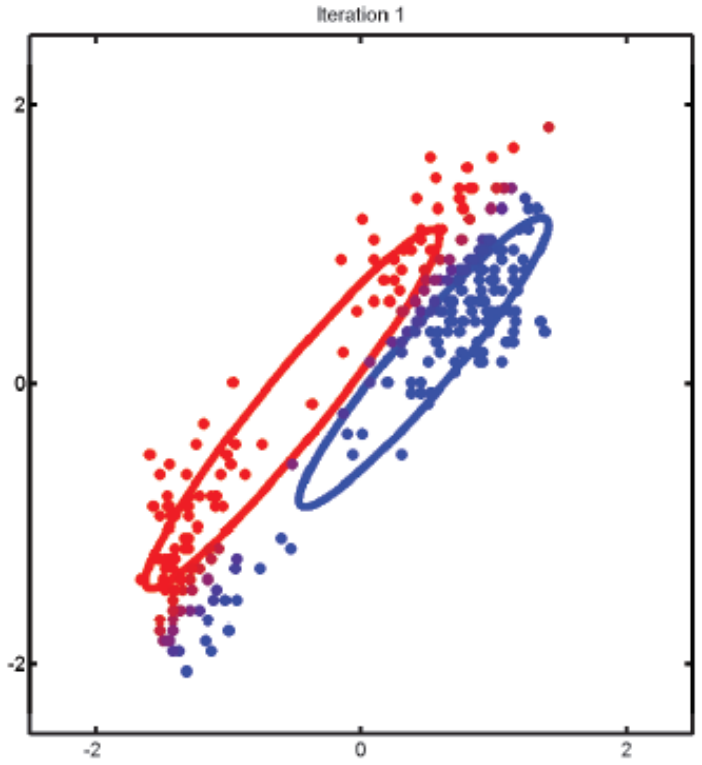
$$\text{color}(i) = r_{i1}\text{blue} + r_{i2}\text{red}$$

  so ambiguous points appear purple.
- After 20 iterations, the algorithm has converged on a good clustering. (The data was standardized, by removing the mean and dividing by the standard deviation, before processing. This often helps convergence.)

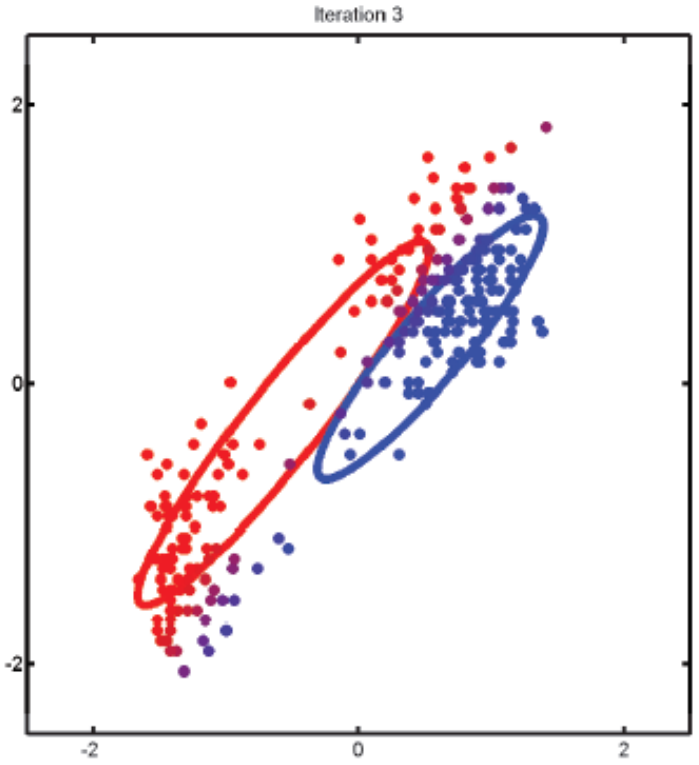
Example of EM for GMM



Example of EM for GMM

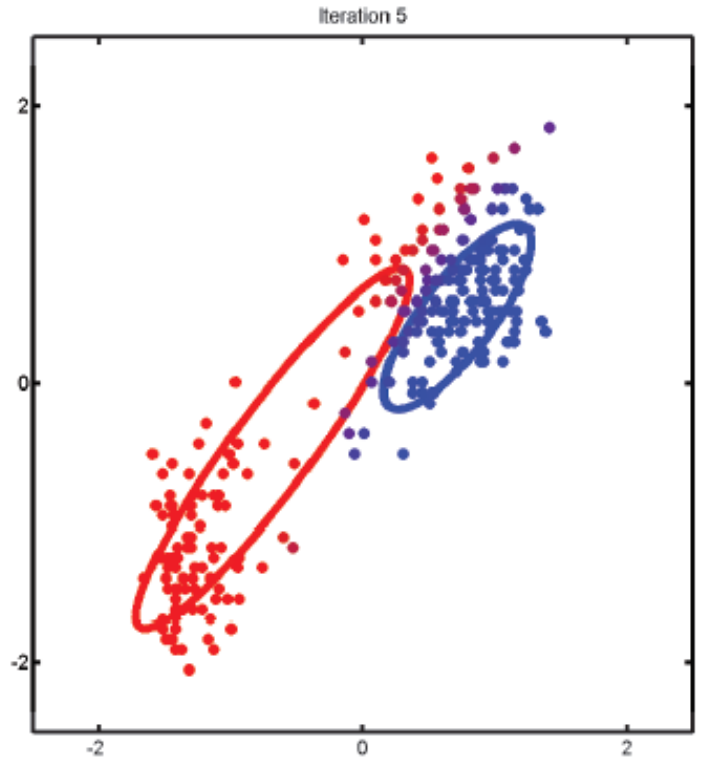


(c)

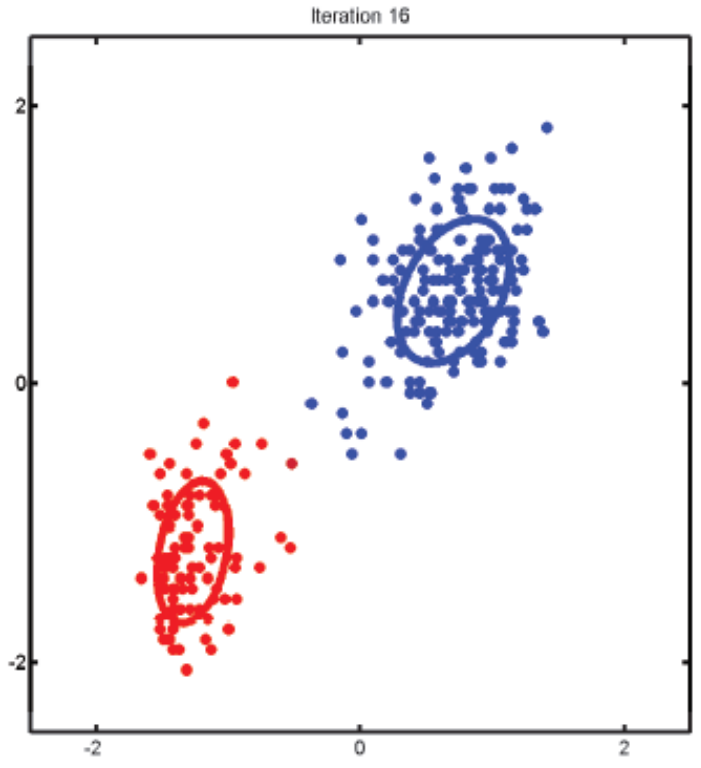


(d)

Example of EM for GMM



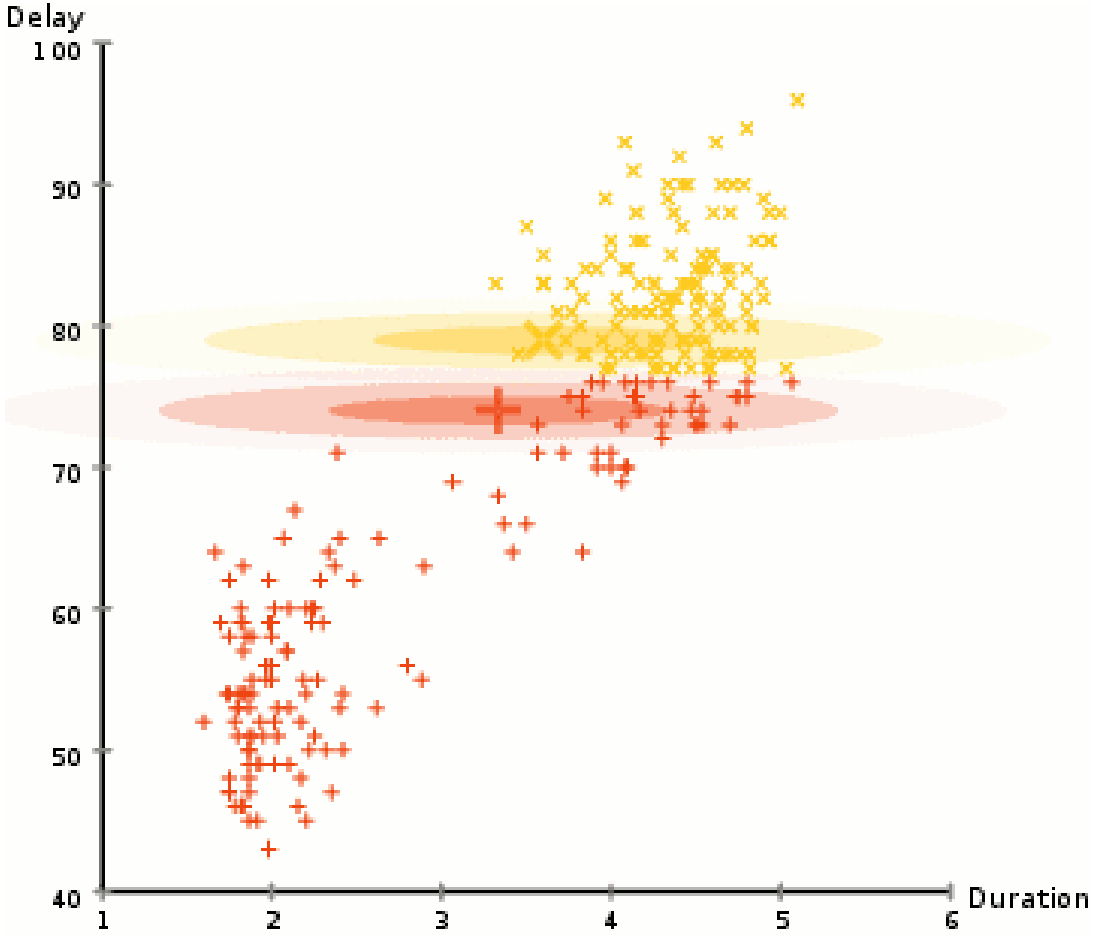
(e)

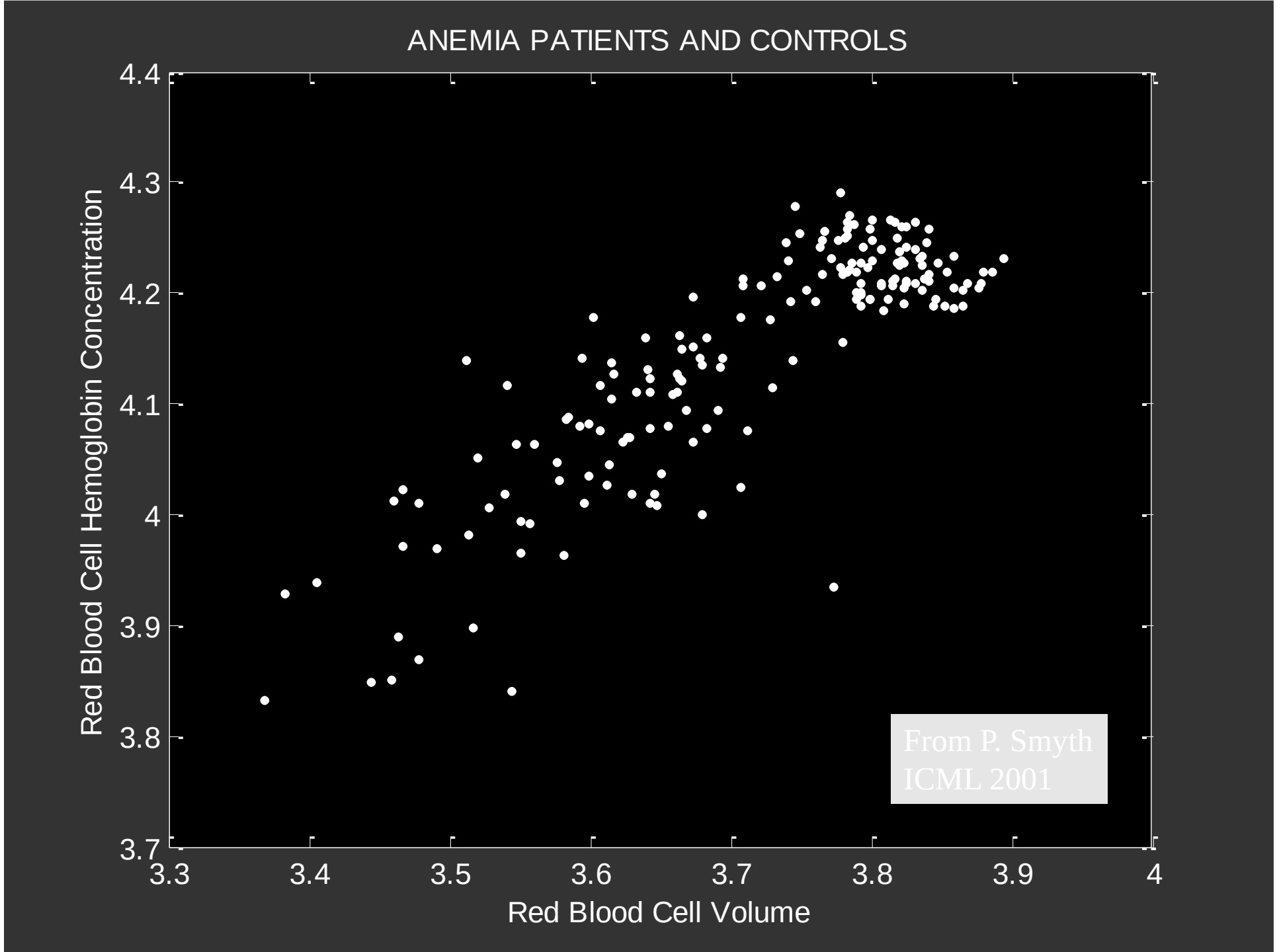


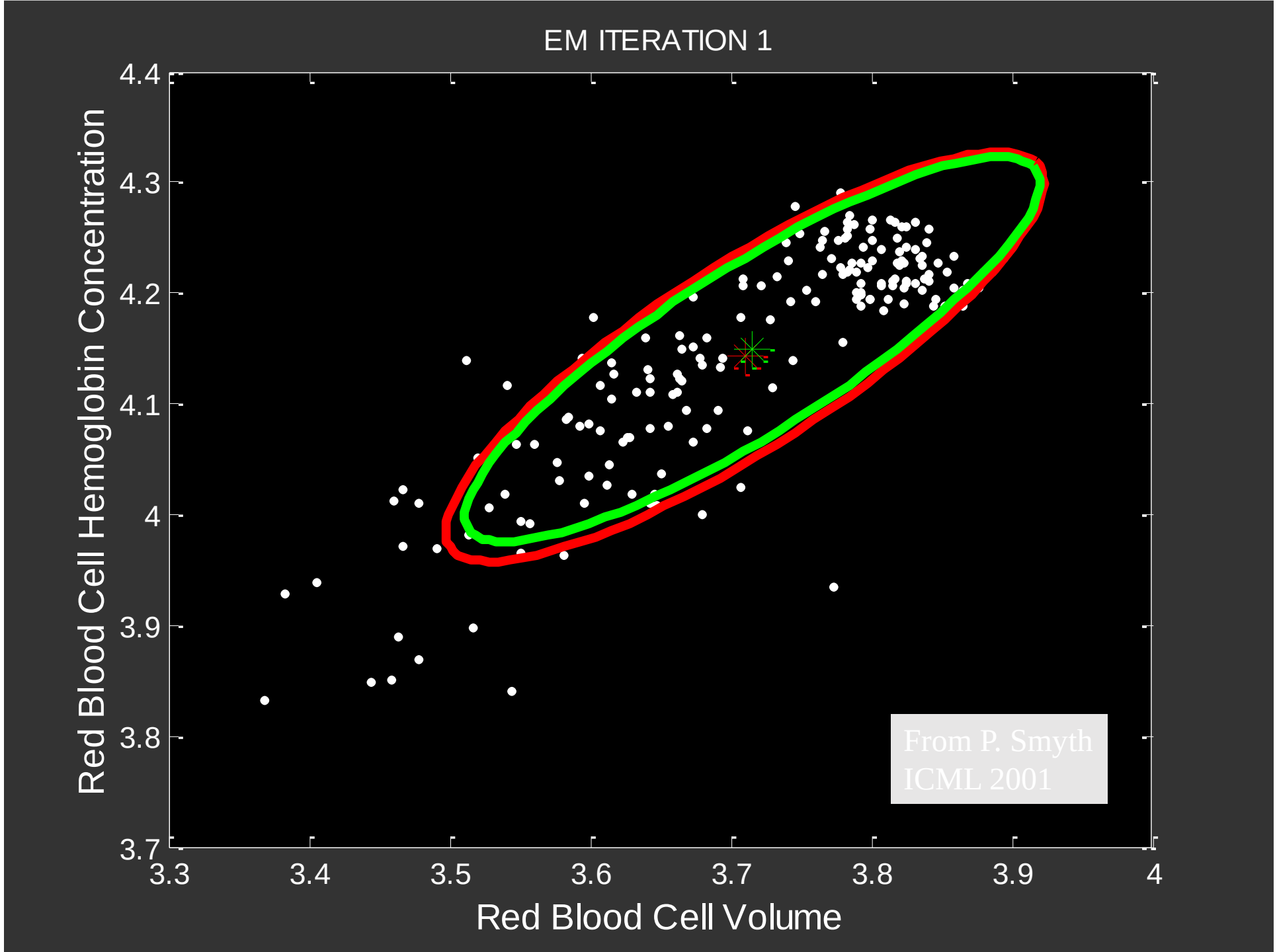
(f)

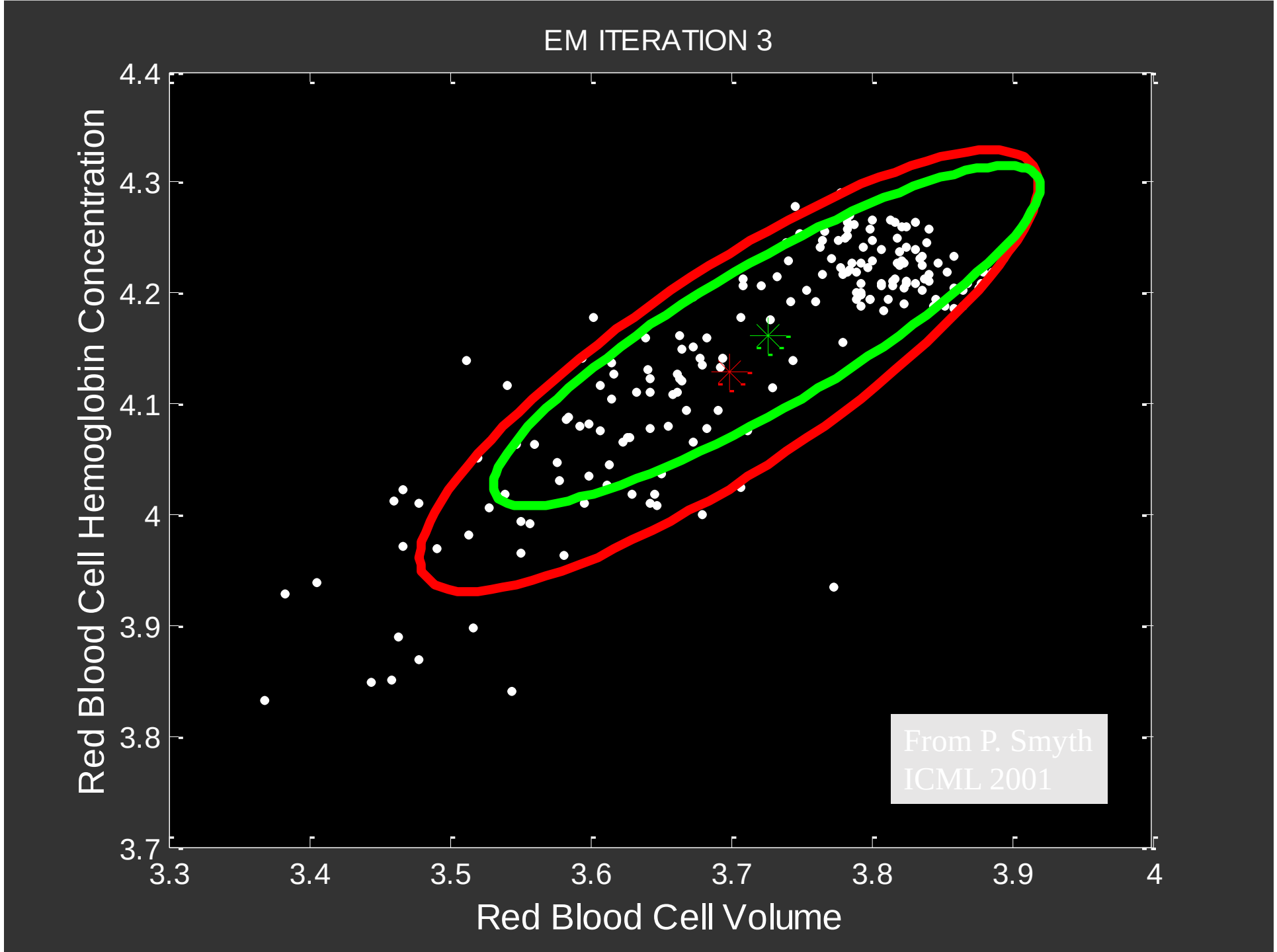
EM for GMM example

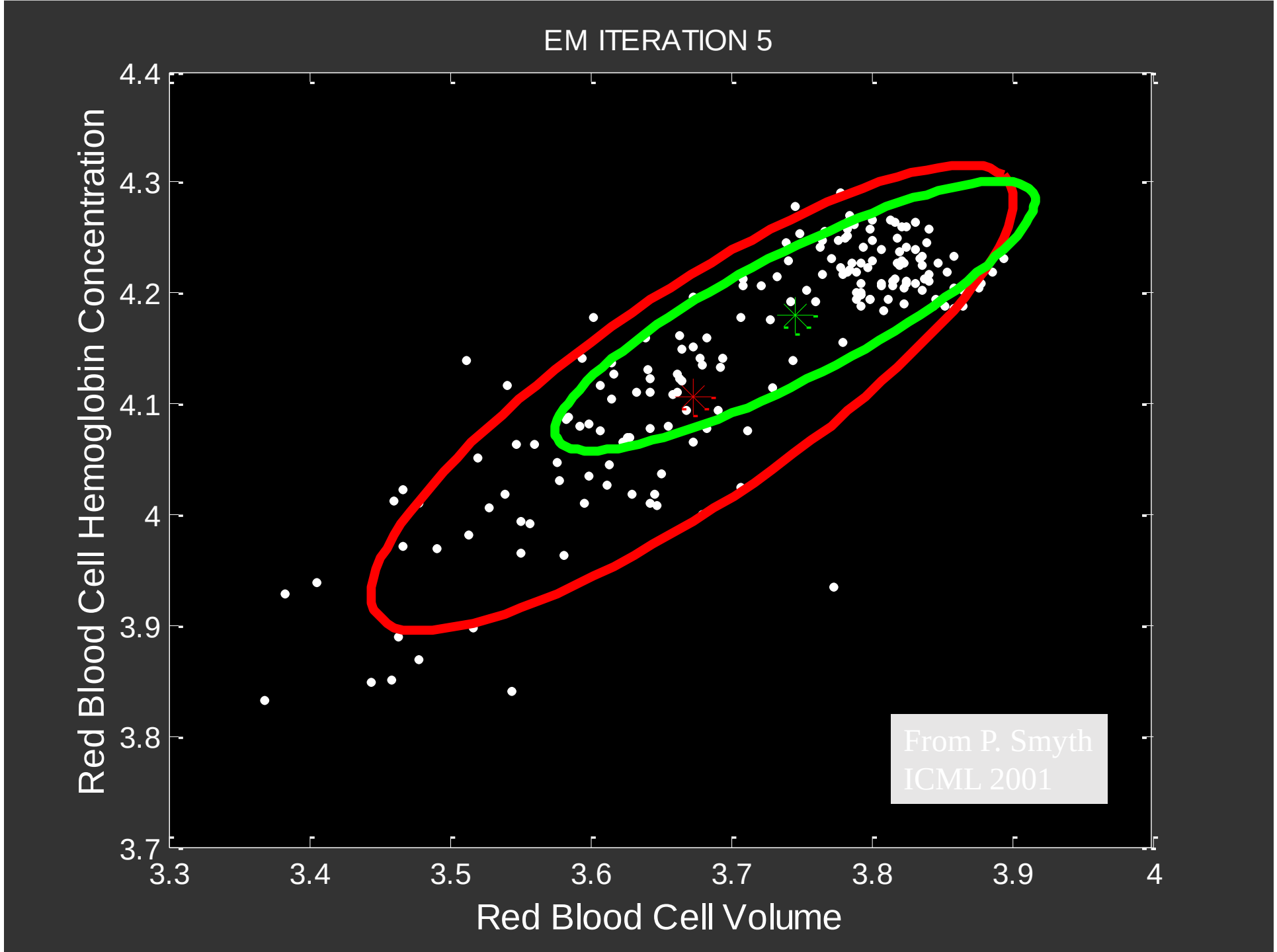
- EM clustering of Old Faithful eruption data.
- The random initial model (which, due to the different scales of the axes, appears to be two very flat and wide spheres) is fit to the observed data. In the first iterations, the model changes substantially, but then converges to the two modes of the geyser.

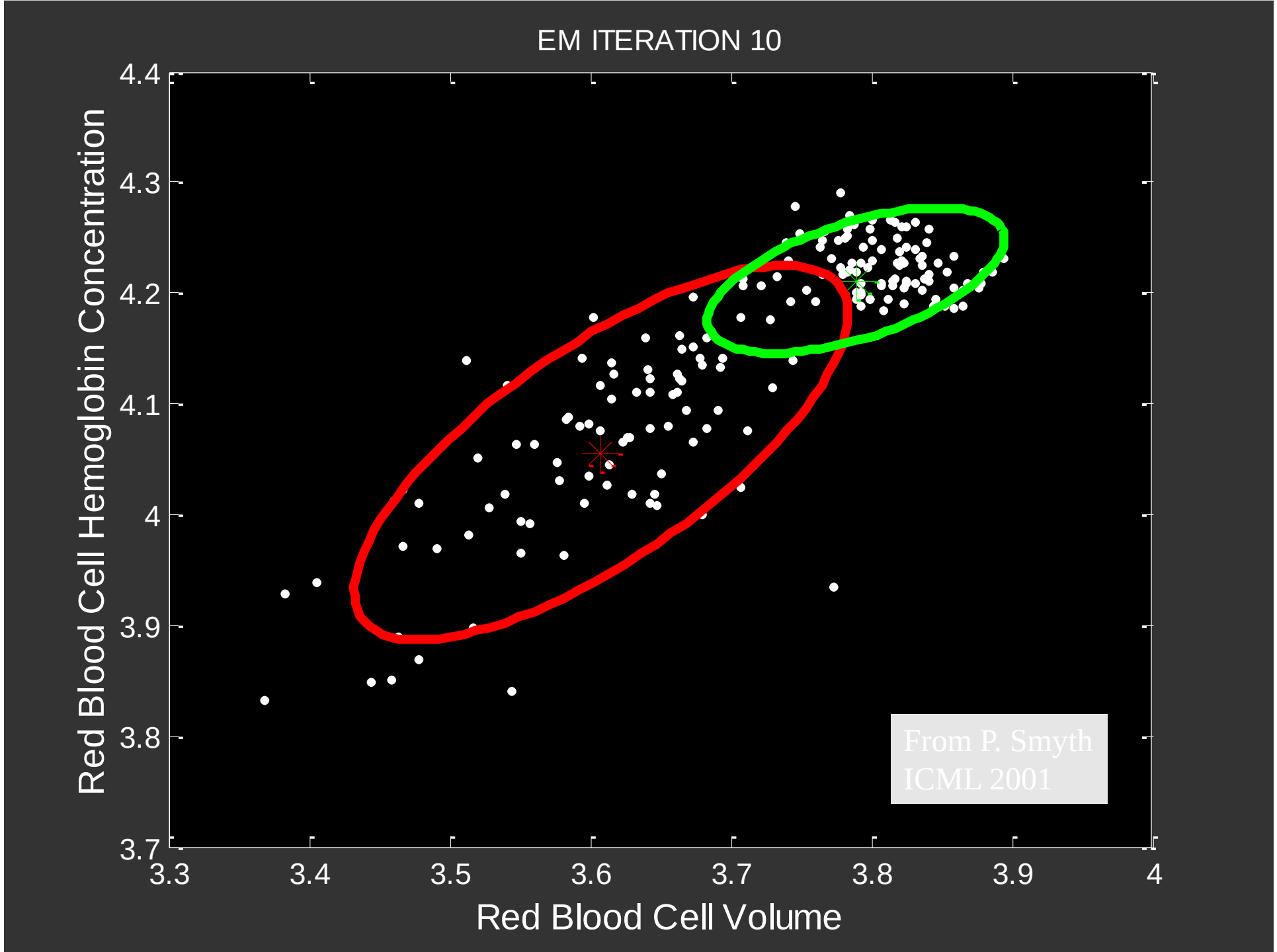


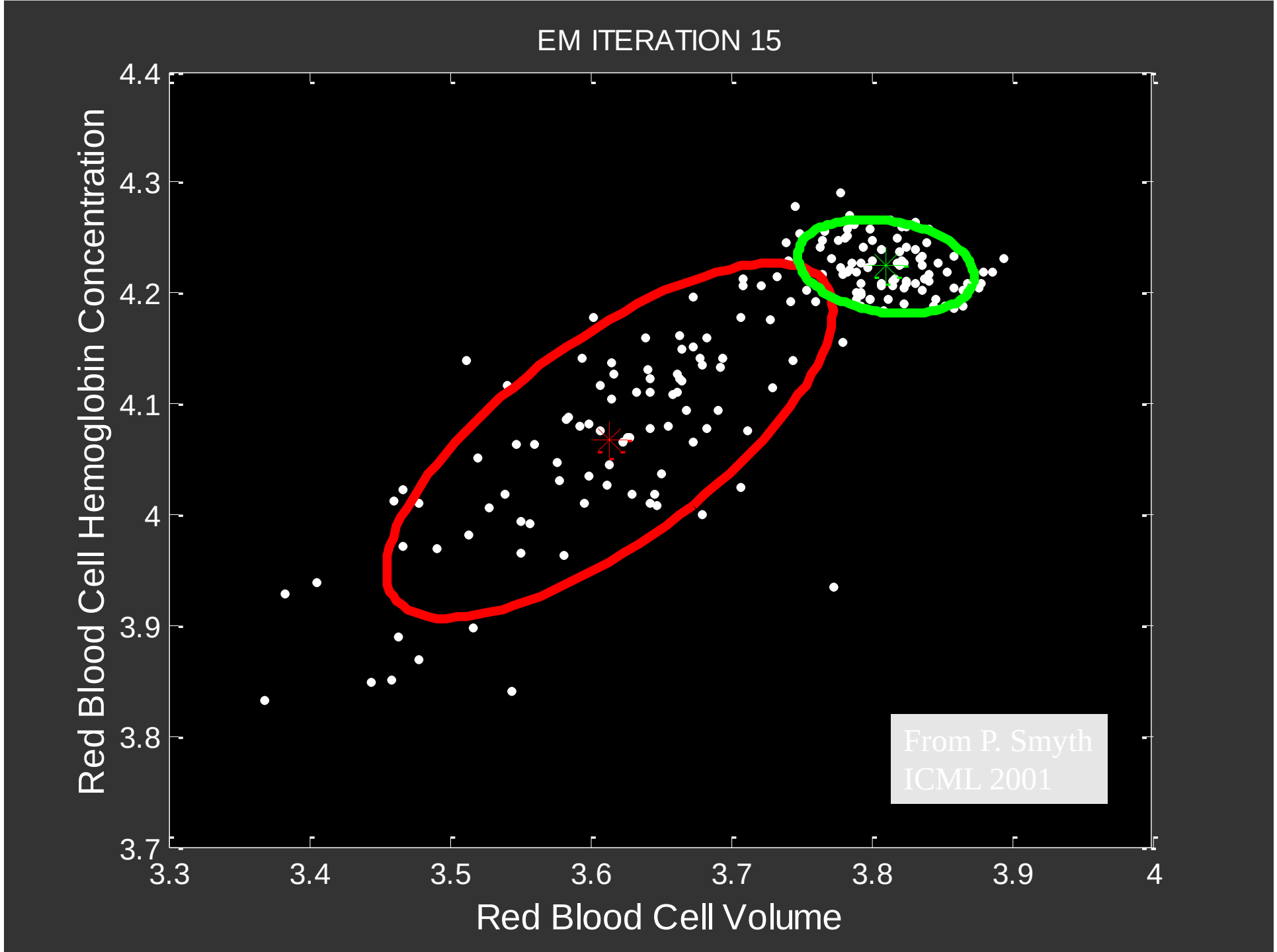

ANEMIA PATIENTS AND CONTROLS
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001

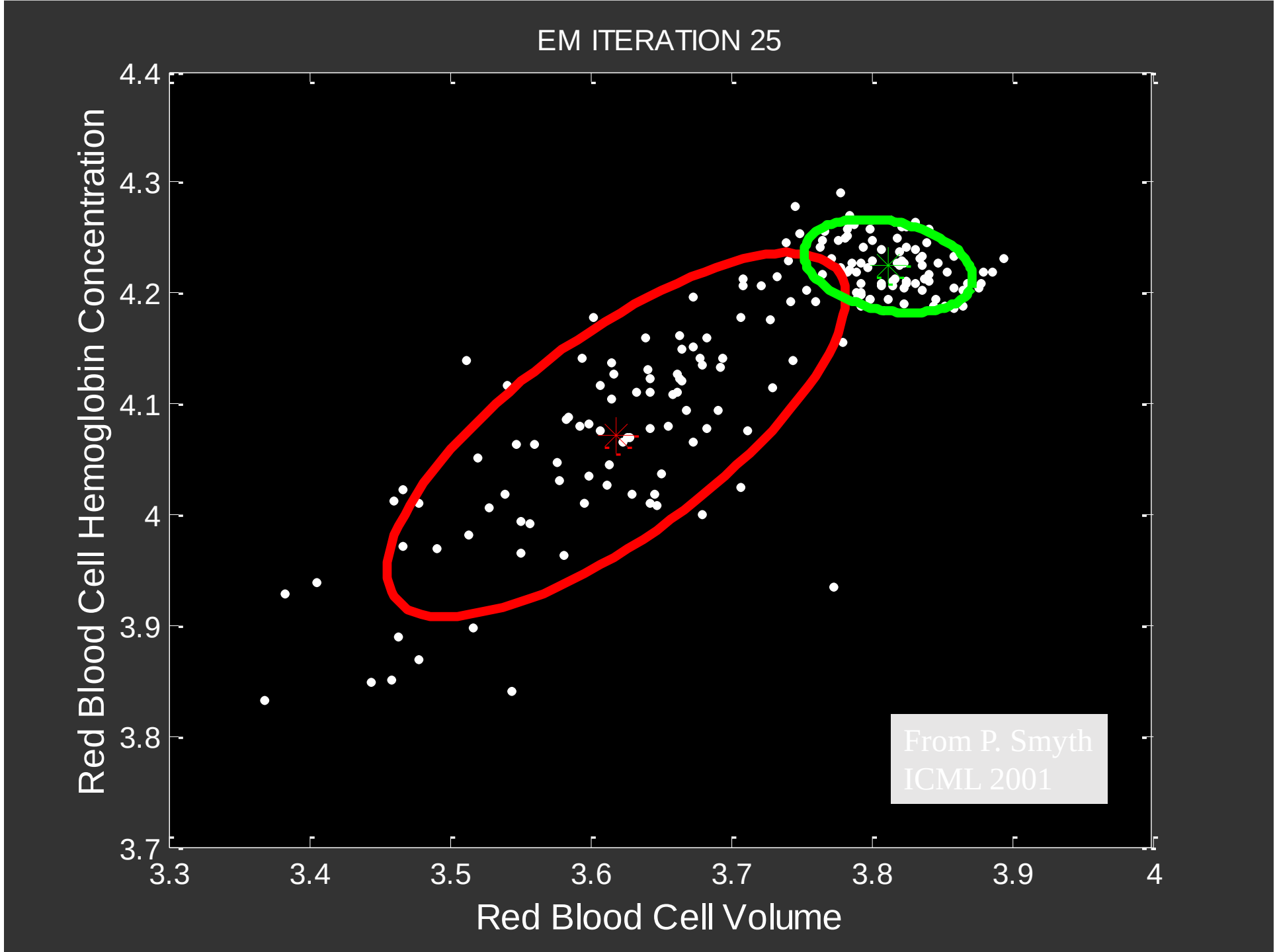

EM ITERATION 1
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001

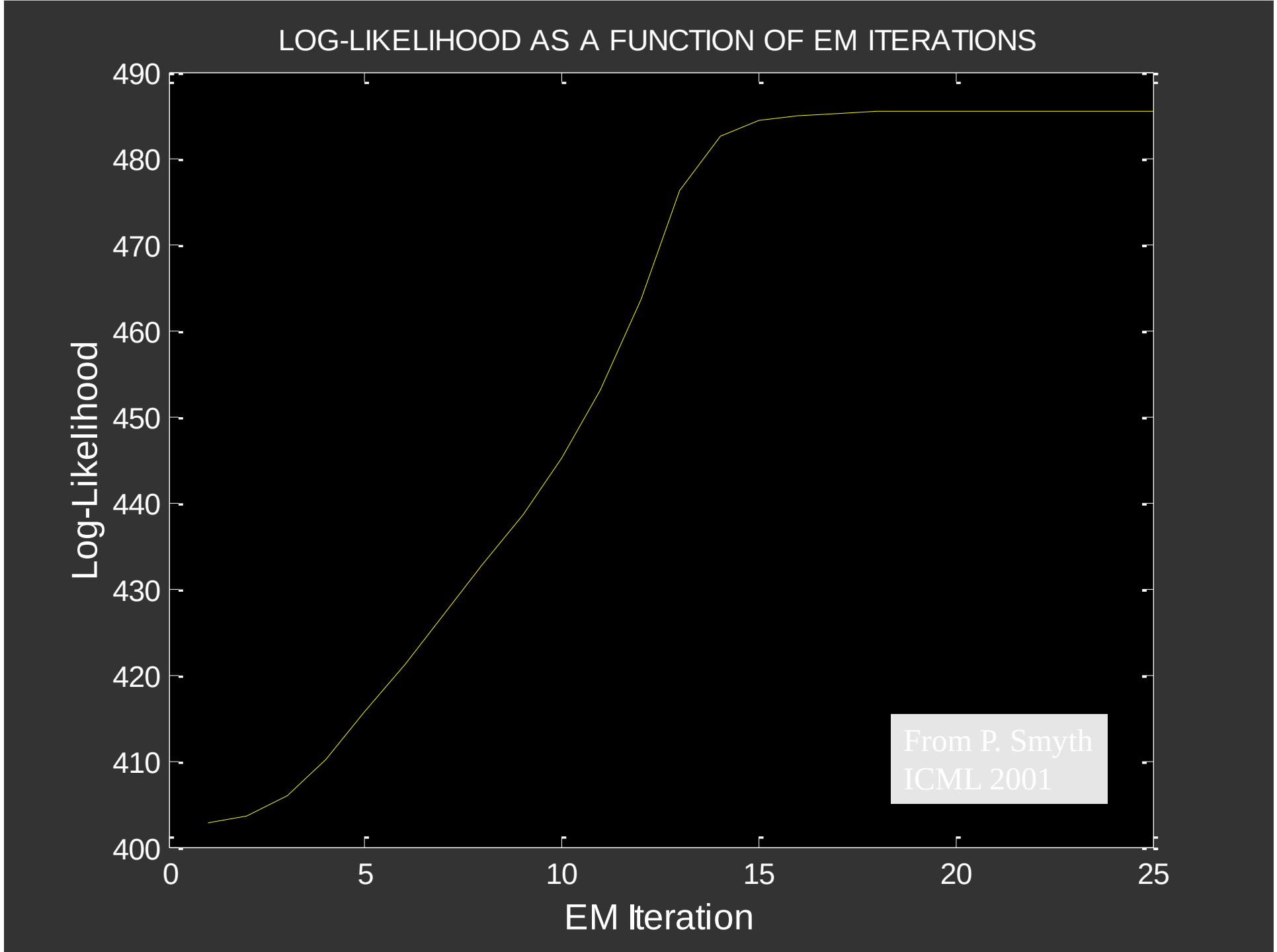

EM ITERATION 3
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 5
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 10
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 15
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


EM ITERATION 25
Red Blood Cell Hemoglobin Concentration
Red Blood Cell Volume
4.4
4.3
4.2
4.1
4
3.9
3.8
3.7
3.3
3.4
3.5
3.6
3.7
3.8
3.9
4
From P. Smyth
ICML 2001


LOG-LIKELIHOOD AS A FUNCTION OF EM ITERATIONS
Log-Likelihood
EM Iteration
490
480
470
460
450
440
430
420
410
400
0
5
10
15
20
25
From P. Smyth
ICML 2001

The K means algorithm

- The K-means algorithm is a variant of the EM algorithm for GMMs, with the following assumptions:
  - $\boldsymbol{\Sigma}_k = \sigma^2 \boldsymbol{I}_D$ is fixed,
  - $\pi_k = 1/K$ is fixed
  - Therefore only the cluster centers, $\mu_k \in R^D$, have to be estimated
- It assumes that all the clusters have the same fixed spherical covariance matrix, which is not updated.
- Also, in the E step, K-means uses a **hard assignment**, which means it assigns each data point z_n to the nearest cluster center μ_k, rather than computing a weighted combination of points.
- Consider the following delta-function approximation to the posterior computed during the **E step**:

$$p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}) \approx \mathbb{I}(k = z_i^*)$$

where

$$z_i* = \text{argmax}_k \, p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})$$

The K means algorithm

- Since we assumed an equal spherical covariance matrix for each cluster, the most probable cluster for x_i can be computed by finding the nearest prototype:

$$z_i^* = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$$

- Hence in each E step, we must find the **Euclidean distance** between *N* data points and *K* cluster centers
- Given the hard cluster assignments, the **M step** updates each cluster center by computing the mean of all points assigned to it:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$$

K-means vs EM for GMM

- The K-means algorithm is **faster** than EM for GMMs with full covariance matrices,
- But its **results** tend to be **not as good**, since
  - it cannot model the **shape of the clusters** (hence it does not learn a distance metric).
  - it is not robust to **ambiguous points** that are equidistant between clusters, because it uses hard assignment. (Probabilistic inference in the E step would softly assign such ambiguous points to multiple clusters.)

The K means algorithm

Algorithm 11.1: K-means algorithm

1 *initialize* $\mathbf{m}_k$;
2 **repeat**
3 | Assign each data point to its closest cluster center
4 | Update each cluster center by computing the mea
| $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$;
5 **until** *converged*;

```
function mu = kmeans(data, K, maxIter, thresh)

[N D] = size(data);
% initialization by picking random pixels
% mu(k,:) = k'th center
perm = randperm(N);
mu = data(perm(1:K),:);

converged = 0;
iter = 1;
while ~converged & (iter < maxIter)
  newmu = zeros(K,D);
  % dist(i,k) = squared distance from pixel i to center k
  dist = sqdist(data', mu');
  [junk, assign] = min(dist,[],2);
  for k=1:K
    newmu(k,:) = mean(data(assign==k,:), 1);
  end
  delta = abs(newmu(:) - mu(:));
  if max(delta./abs(mu(:))) < thresh
    converged = 1;
  end
  mu = newmu;
  iter = iter + 1
end

if ~converged
  error(sprintf('did not converge within %d iterations', maxIter))
end
```

K-means algorithm

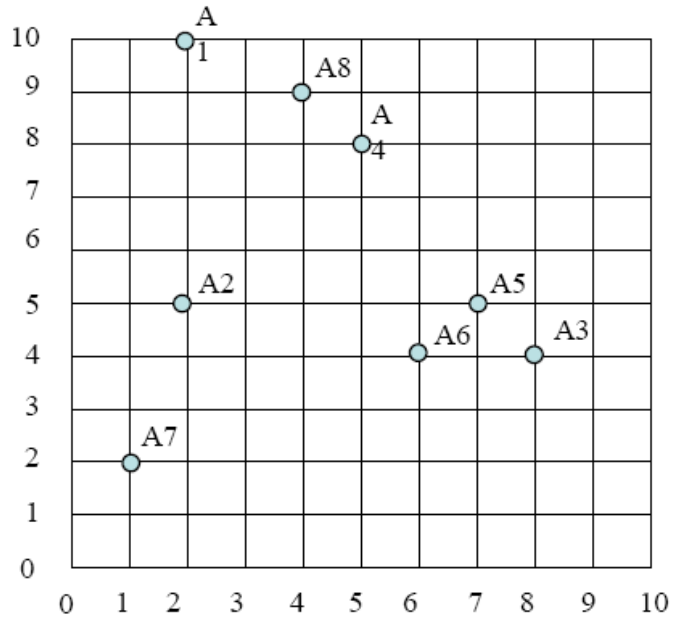
- In each E step:
  - we must find the Euclidean distance between *N* data points and *K* cluster centers and assign a cluster to each data point (analogous to finding the responsibility in the standard EM algorithm)
    - Although Euclidean distance is the standard metric (as defined by the GMM and other assumptions, we can make adjustments to use other distance metric we think would fit the data better)
- In each M step:
  - update each cluster center by computing the mean of all points assigned to it (analogous to finding the parameters in the standard EM algorithm)

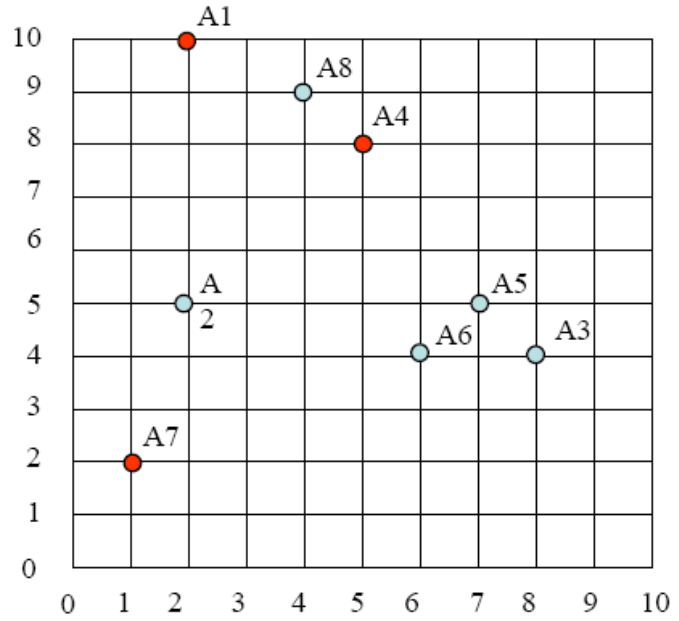
$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$$

- Repeat until convergence
  - Until the parameters (and the clusters) don't change in the next iteration
- Note: K-means is a non-probabilistic method because of the hard-assignment

K-means Example

- Using k-means clustering and Euclidean distance group the following 8 data points into 3 clusters:
  A_1=(2,10), A_2=(2,5), A_3=(8,4), A_4=(5,8), A_5=(7,5), A_6=(6,4), A_7=(1,2), A_8=(4,9).
- For the initial centroids, we take the points A_1, A_4 и A_7.
  C_1=A_1=(2,10), C_2=A_4=(5,8), C_3=A_7=(1,2)





Epoch 1-Step E

- In step E of the K-means algorithm, analogously to the EM algorithm, we calculate the responsibilities (or in this case hard assignments) of each point.

- A_1:
  1. **$d(A_1, C_1) = 0$ as A_1 is C_1**
  2. $d(A_1, C_2) > 0$
  3. $d(A_1, C_3) > 0$
  - $\Rightarrow A_1 \in$ cluster 1
- A_2:
  1. $d(A_2, C_1) = 5$
  2. $d(A_2, C_2) = 4.24$
  3. **$d(A_2, C_3) = 3.16$**
  - $\Rightarrow A_2 \in$ cluster 3

- A_3:
  1. $d(A_3, C_1) = 8.5$
  2. **$d(A_3, C_2) = 5$**
  3. $d(A_3, C_3) = 7.28$
  - $\Rightarrow A_3 \in$ cluster 2
- A_4:
  1. $d(A_4, C_1) > 0$
  2. **$d(A_4, C_2) = 0$ as A_4 is C_2**
  3. $d(A_4, C_3) > 0$
  - $\Rightarrow A_4 \in$ cluster 2

Epoch 1-Step E

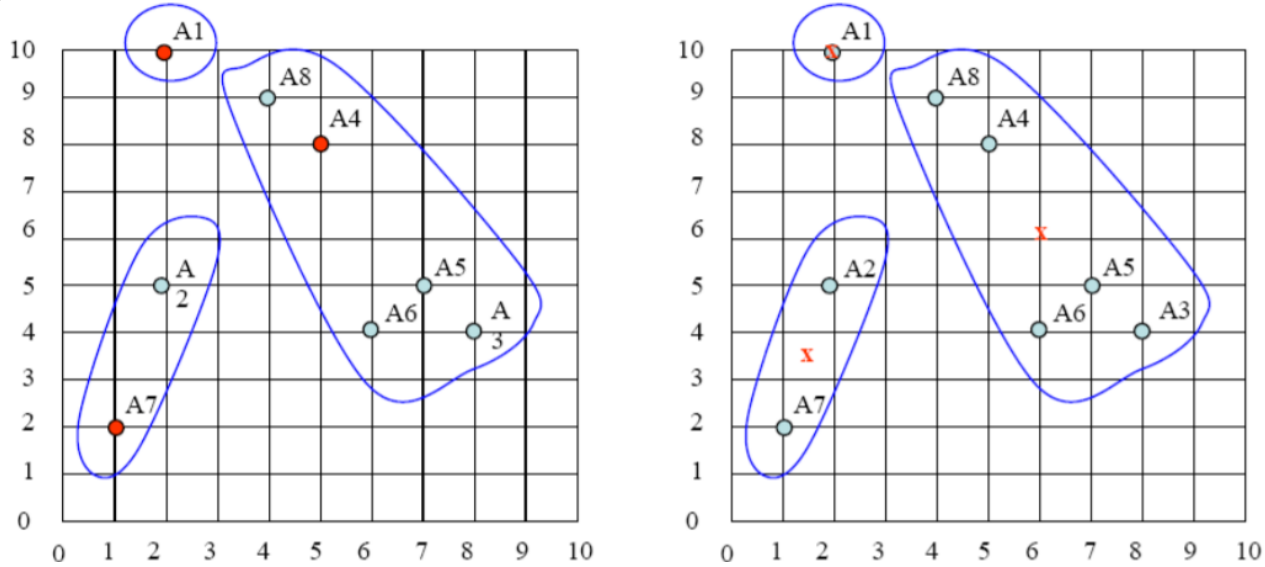
- A_5:
  1. $d(A_5, C_1) = 7.07$
  2. **$d(A_5, C_2) = 3.60$**
  3. $d(A_5, C_3) = 6.70$
  - $\Rightarrow A_5 \in$ cluster 2
- A_6:
  1. $d(A_6, C_1) = 7.21$
  2. **$d(A_6, C_2) = 4.12$**
  3. $d(A_6, C_3) = 5.38$
  - $\Rightarrow A_6 \in$ cluster 2

- A_7:
  1. $d(A_7, C_1) > 0$
  2. $d(A_7, C_2) > 0$
  3. **$d(A_7, C_3) = 0$ as A_7 is C_3**
  - $\Rightarrow A_7 \in$ cluster 3
- A_8:
  1. $d(A_8, C_1) = 2.23$
  2. **$d(A_8, C_2) = 1.41$**
  3. $d(A_8, C_3) = 7.61$
  - $\Rightarrow A_8 \in$ cluster 2

Epoch 1-Step M

- In step M of the K-means algorithm, analogously to the EM algorithm, we calculate the parameters (or in this case the centroids which represent the mean of the cluster)

- After the first epoch the clusters are:
  - 1: $\{A_1\}$
  - 2: $\{A_3, A_4, A_5, A_6, A_8\}$
  - 3: $\{A_2, A_7\}$
- The centroids are:
  - The new centroids are calculated as the means of the points belonging in the cluster
    For instance:
    $C_2\_x = (A_3\_x + A_4\_x + A_5\_x + A_6\_x + A_7\_x)/5$
    $C_2\_y = (A_3\_y + A_4\_y + A_5\_y + A_6\_y + A_7\_y)/5$
  - $C_1 = (2, 10)$
  - $C_2 = ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6, 6)$
  - $C_3 = ((2+1)/2, (5+2)/2) = (1.5, 3.5)$



The clusters and centroids after the first iteration

Epoch 2-Step E

- A_1:
  1. **$d(A_1, C_1) = 0$ as A_1 is C_1**
  2. $d(A_1, C_2) > 0$
  3. $d(A_1, C_3) > 0$
  - $\Rightarrow A_1 \in$ cluster1
- A_2:
  1. $d(A_2, C_1) = 5$
  2. $d(A_2, C_2) = 4.1$
  3. **$d(A_2, C_3) = 1.58$**
  - $\Rightarrow A_2 \in$ cluster3

- A_3:
  1. $d(A_3, C_1) = 8.48$
  2. **$d(A_3, C_2) = 2.82$**
  3. $d(A_3, C_3) = 6.51$
  - $\Rightarrow A_3 \in$ cluster2
- A_4:
  1. $d(A_4, C_1) = 3.6$
  2. **$d(A_4, C_2) = 2.23$**
  3. $d(A_4, C_3) > 5.7$
  - $\Rightarrow A_4 \in$ cluster2

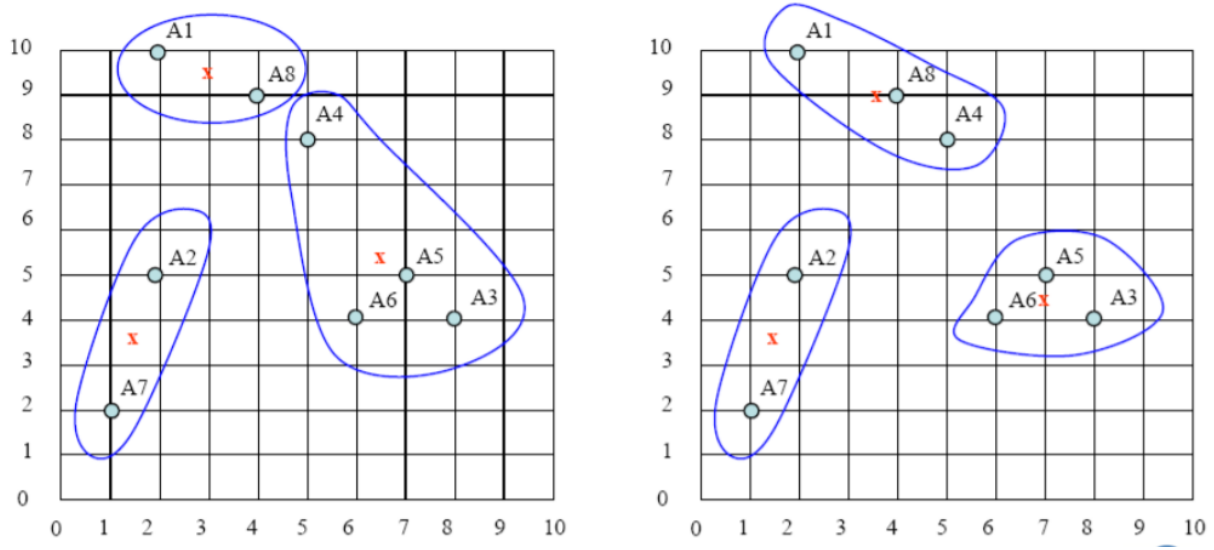
Epoch 2-Step E

- A_5:
  1. $d(A_5, C_1) = 7.07$
  2. **$d(A_5, C_2) = 1.41$**
  3. $d(A_5, C_3) = 5.70$
  - $\Rightarrow A_5 \in$ cluster2
- A_6:
  1. $d(A_6, C_1) = 7.21$
  2. **$d(A_6, C_2) = 2$**
  3. $d(A_6, C_3) = 4.52$
  - $\Rightarrow A_6 \in$ cluster2

- A_7:
  1. $d(A_7, C_1) = 8.06$
  2. $d(A_7, C_2) = 6.4$
  3. **$d(A_7, C_3) = 1.58$**
  - $\Rightarrow A_7 \in$ cluster3
- A_8:
  1. **$d(A_8, C_1) = 2.23$**
  2. $d(A_8, C_2) = 3.60$
  3. $d(A_8, C_3) = 6.04$
  - $\Rightarrow A_8 \in$ cluster1

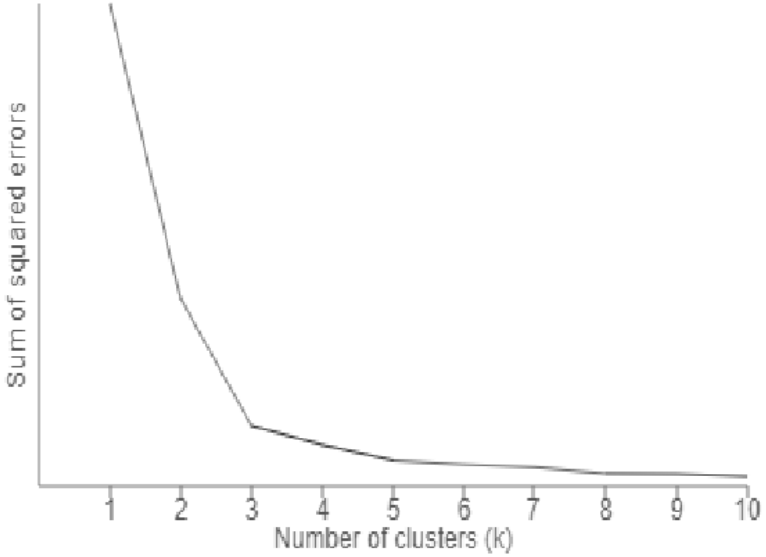
Epoch 2 and 3-Step M

The clusters and centroids after the second and third iterations



- After the second epoch the clusters are :
  - 1: {A_1, A_8}
  - 2: {A_3, A_4, A_5, A_6}
  - 3: {A_2,A_7}
- The centroids are:
  - C_1=(3, 9.5)
  - C_2=(6.5, 5.25)
  - C_3=(1.5, 3.5)
- We should continue with the iterative approach until convergence
- After the third epoch, the clusters are: 1: {A_1, A_4, A_8}, 2: {A_3, A_5, A_6}, 3: {A_2, A_7} with centroids C_1=(3.66, 9), C_2=(7, 4.33) и C_3=(1.5, 3.5), and this is the final result.
- In every calculation after this, the points will be assigned the same clusters, and therefore the centroids won't move.

Estimating the number of mixture components

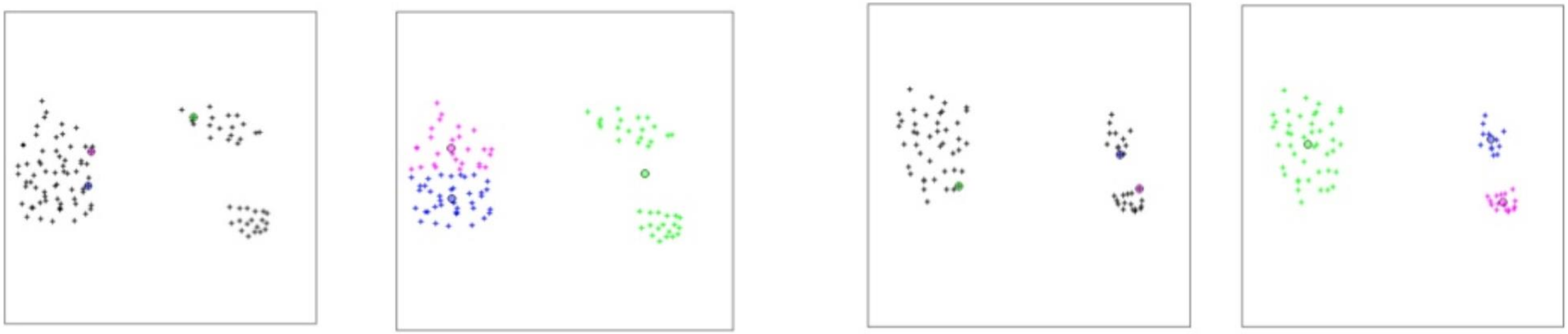


- Defined by the application
- Plot data and check for clusters
- Maximum likelihood will always favor the largest possible value of K, since that gives the greatest number of parameters and maximizes the ability to fit the data. But this may result in overfitting. A simple alternative is to use **cross validation**, i.e., to find the value of K that maximizes the log likleihood of a **validation set.**
- Incremental (leader-cluster) algorithm: Add one at a time until “elbow”
- Manual check for meaning

Initialization and avoiding local minima

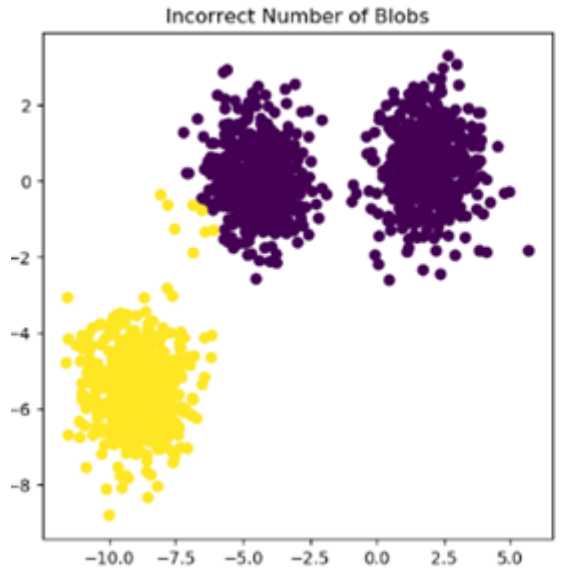
- Both K-means and EM need to be initialized. It is common to pick *K* data points at random, and to make these be the initial cluster centers.
- Or we can pick the centers sequentially so as to try to "cover" the data.
  1. Pick the initial point uniformly at random.
  2. Pick each subsequent point from the remaining points with probability proportional to its squared distance to the points' closest cluster center
  - This is known as **farthest point clustering** or **k-means++**

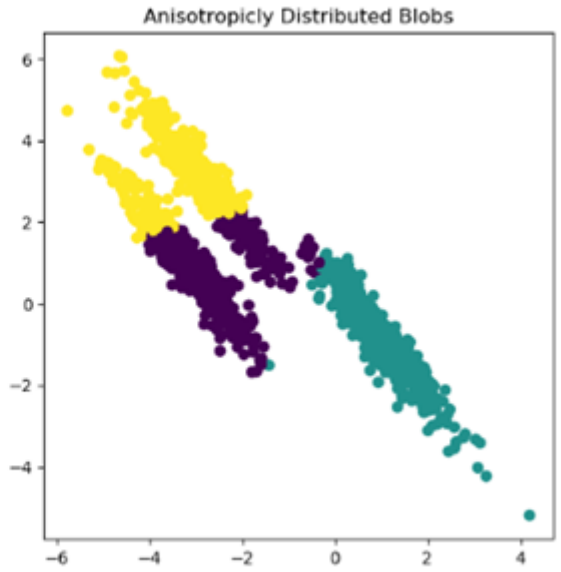
Bad vs Good initialization of centroids



Cons of K-means

- Cons:
  - The parameter K representing the number of clusters must be known beforehand
  - Choice of a distance metric
  - Takes into account the distance, but not the size of the clusters (due to the equal, fixed, diagonal covariance matrix assumption)
  - Bad results for clusters that aren't convex and isotropic (again due to the covariance matrix assumption)
  - Non probabilistic (Hard assignments)





Vector quantization

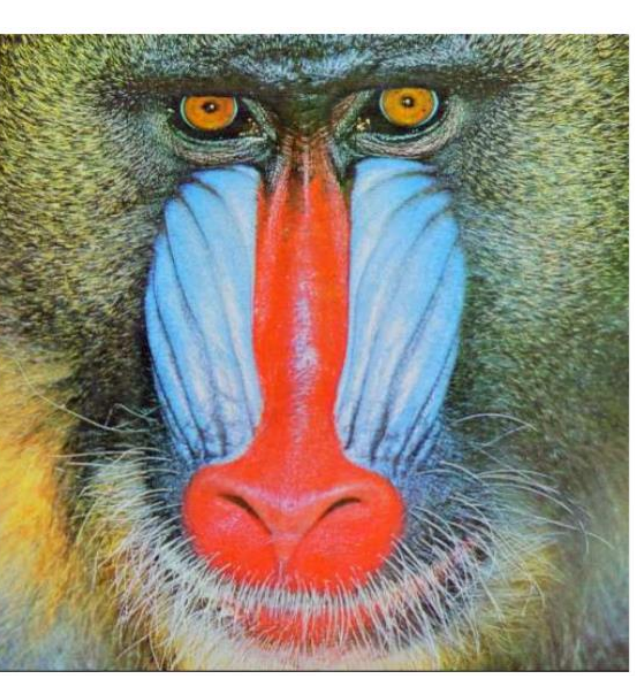
- The K centers μ_k learned by K-means are often called a **codebook**.
- This can then be used for discretizing data - this is called **vector quantization (VQ)**.
- The idea is that each new data point x is replaced by its nearest prototype μ_k
- The result is a set of N discrete symbols
- This can take much less space to store than the original data and is therefore an example of **lossy data compression**.

Vector quantization

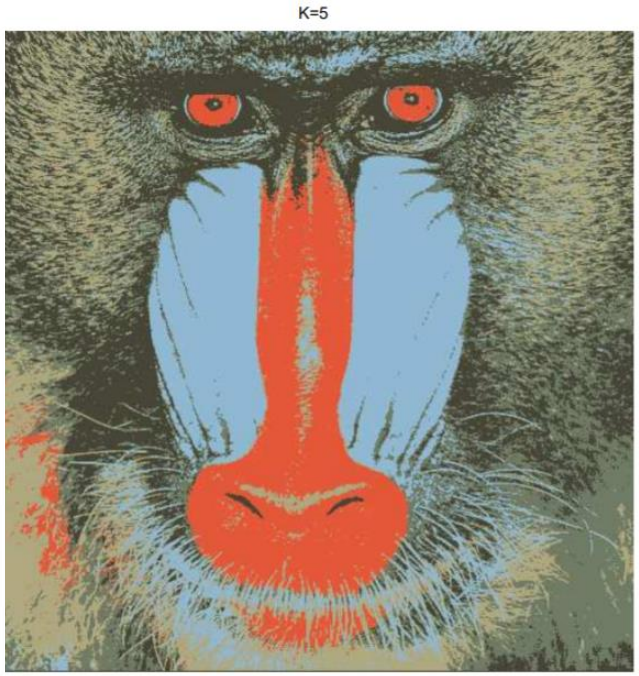
- For example, consider N = 512 × 512 = 262,144 pixel image, and each pixel is represented by three 8-bit numbers (each ranging from 0 to 255) that represent the green, red and blue intensity values for that pixel. The straightforward representation of this image therefore takes about 24N = 6, 291, 456 bits. We applied vector quantization to this using K = 5, 10, 15 codewords using the code below.

```
% vqDemo

% Learn code book on small image (for speed)
A = double(imread('mandrill-small.tiff'));
figure; imshow(uint8(round(A)));
[nrows ncols ncolors] = size(A);
% data(i,:) = rgb value for pixel i
data = reshape(A, [nrows*ncols ncolors]);
maxIter = 100; % usually converges long before 100 iterations!
thresh = 1e-2;
K = 15;
mu = kmeansKPM(data, K, maxIter, thresh);

% Apply codebook to quantize  large image
B = double(imread('mandrill-large.tiff'));
imshow(uint8(round(B)));
[nrows ncols ncolors] = size(B);
data = reshape(B, [nrows*ncols ncolors]);
[N D] = size(data);
dist = sqdist(data', mu');
[junk, assign] = min(dist,[],2);
qdata = zeros(size(data));
for k=1:K
  ndx = find(assign==k);
  Nassign = length(ndx);
  qdata(ndx, :) = repmat(mu(k,:), Nassign, 1);
end
Qimg = reshape(qdata, [nrows ncols ncolors]);
figure; imshow(uint8(round(Qimg)));
title(sprintf('K=%d',K))
```

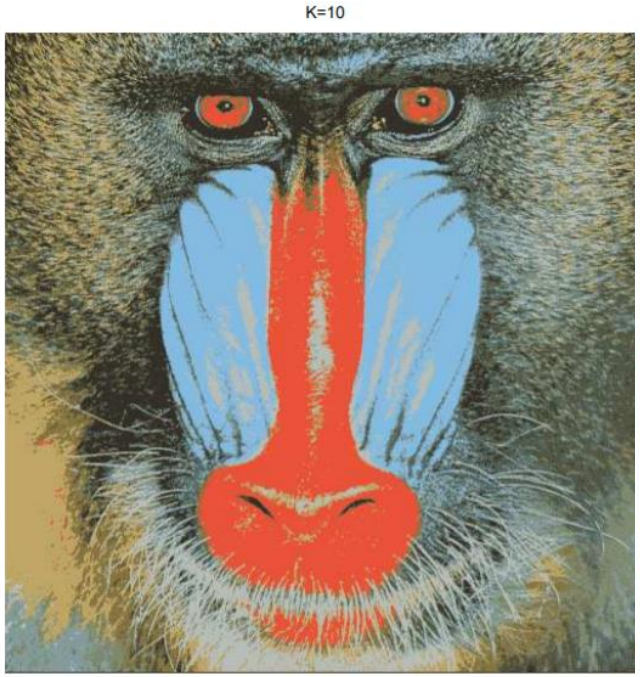
Vector quantization



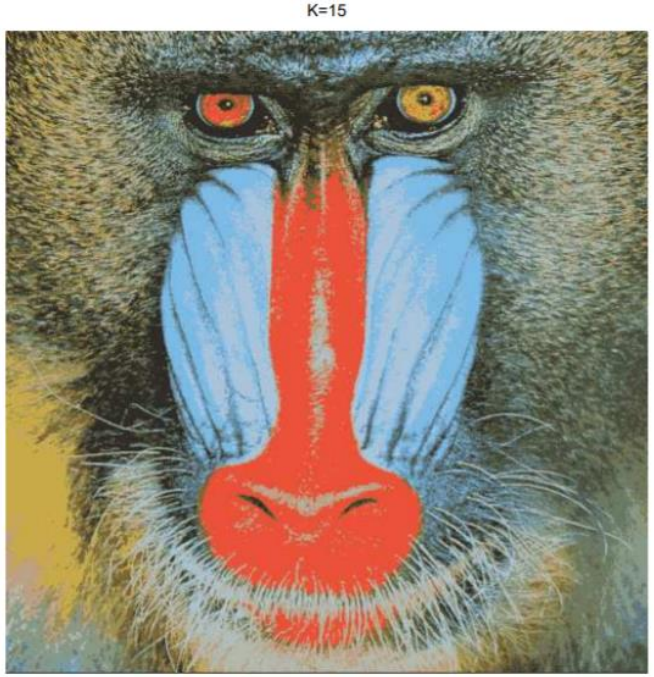
K=5



K=10



K=15



K-medoids

- When using Euclidean distance, as above, these centers are averages of the data points.
- For spaces in which such averaging does not make sense, it is possible to replace the Euclidean distance measure by some other measure of similarity/ distance, $V(x_n, \mu_k)$, so the **E step** becomes:

$$z_n^* = \arg\min_k V(x_n, \mu_k)$$

- In the **M step**, we search over all N_k points assigned to cluster k to find the one with minimum distance to all the others, and that is chosen as the prototype μ_k.

K-medoids кластерирање

- Алгоритамот k-medoids може да се гледа како дискретизирана верзија на k-means алгоритамот
- Тој гласи:

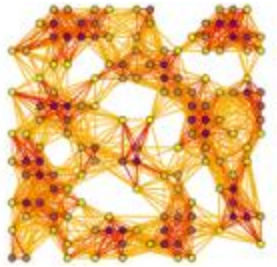
  (1) Случајно избери k примероци за да служат како “seeds” за k кластери

  (2) Додели ги останатите податоци на кластерот од најблискиот “seed” користејќи некоја метрика за растојание D

  (3) Пресметај го новиот medoid (еден од податоците) за секој кластер како репрезент на тој кластер; и

  (4) Повтори ги чекорите 2 и 3 користејќи ги “medoid”-ите како “seeds” се додека не се стабилизираат кластерите.

Medoids are representative objects of a data set or a cluster with a data set whose average dissimilarity to all the objects in the cluster is minimal. **Medoids** are similar in concept to means or centroids, but **medoids** are always members of the data set.



K-medoids кластерирање на граф

- Интуитивна мерка D е геодетското растојание или бројот на скокови (hops) помеѓу јазлите
  1. Ги иницијализираме нашите кластери со случајно избирање на k јазли во графот како *seeds* и ги доделуваме сите јазли на кластерот од најблискиот *seed* јазел.
  2. Ги избираме *medoids* со пресметување на локалната closeness centrality помеѓу јазлите во секој кластер и се избира јазелот со најголема оцена за closeness centrality.

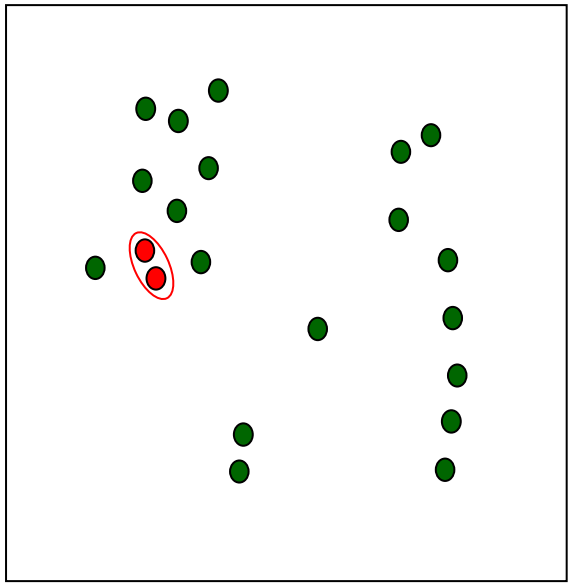
$$C(x) = \frac{1}{\sum_y d(y,x)}$$

  3. Процесот завршува кога се стабилизираат *medoids* на кластерите.

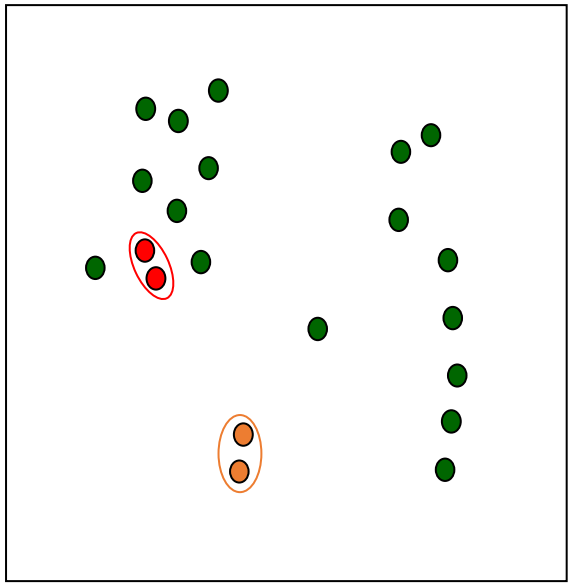
Hierarchical clustering

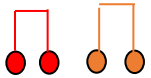
- GMMs can be used to perform clustering, but the clustering is “flat”.
- Often, we want to learn hierarchical clusters (nested clusters).
- The most popular approach is to use **agglomerative clustering**, which we will describe below.
- Agglomerative clustering starts with N groups, **each initially containing one training instance**, merging similar groups to form larger groups, until there is a single group.
- At each iteration of an agglomerative algorithm, we choose the two closest groups to merge.

Iteration 1



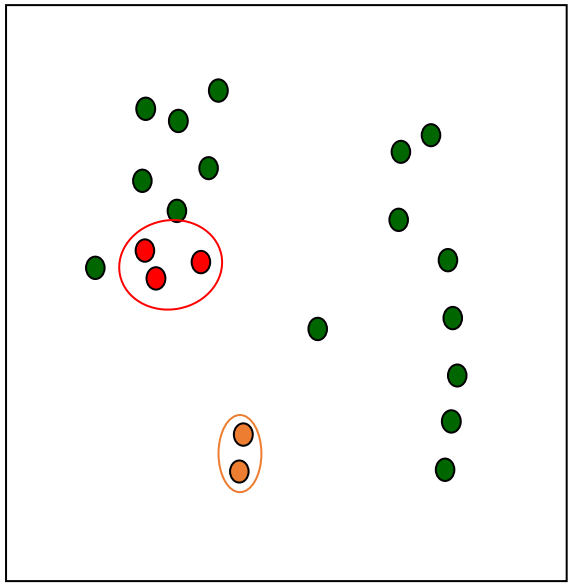
Iteration 2

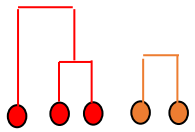




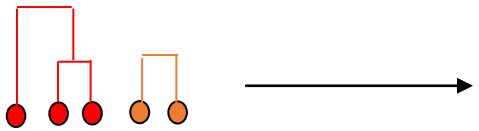
Iteration 3

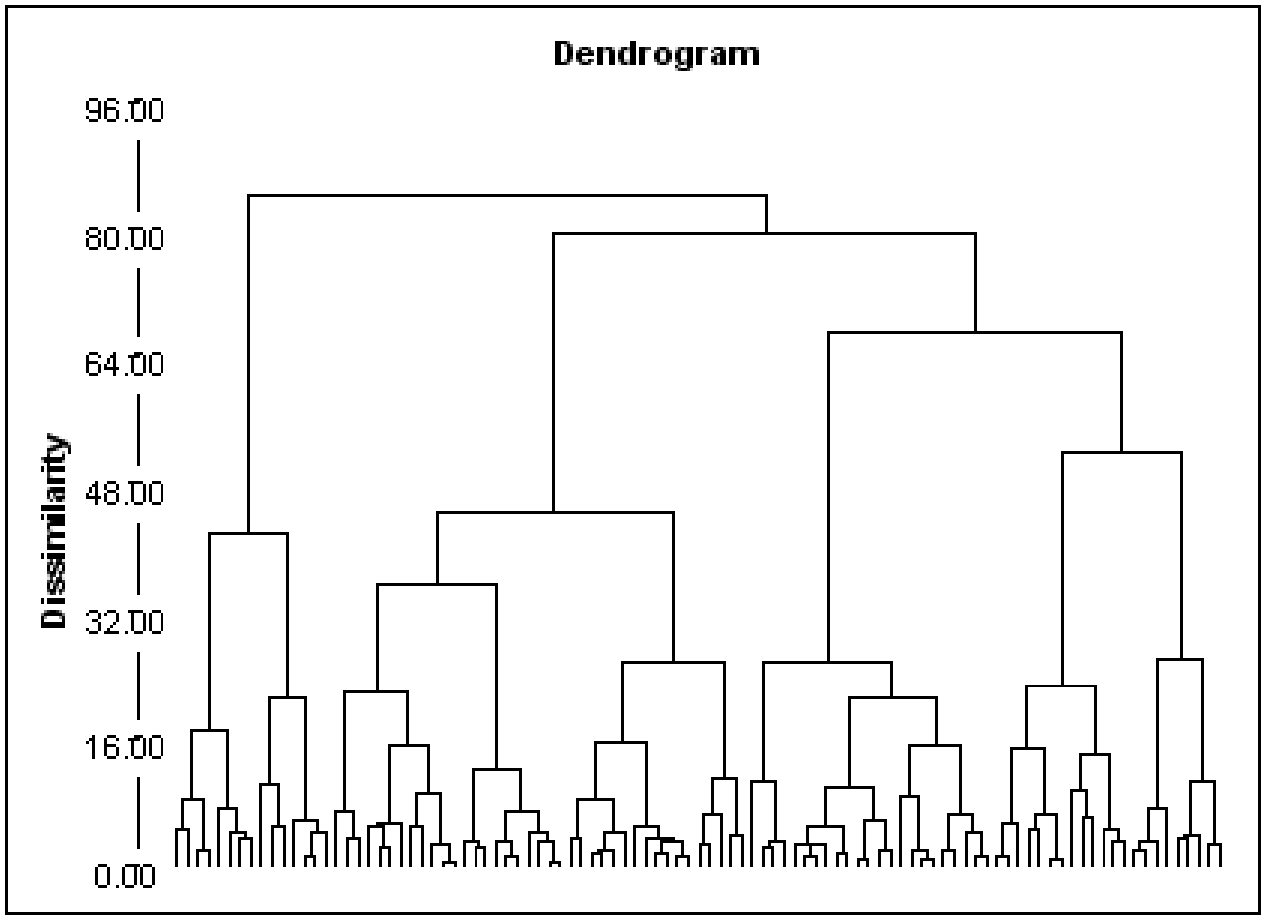
- Builds up a sequence of clusters (“hierarchical”)





Dendrogram

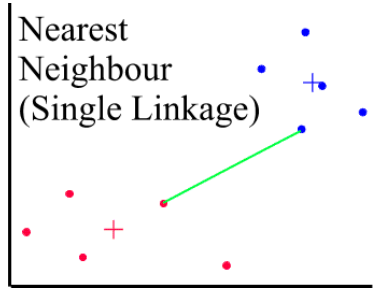


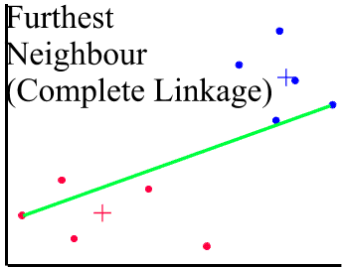

Dendrogram
96.00
80.00
64.00
48.00
32.00
16.00
0.00
Dissimilarity

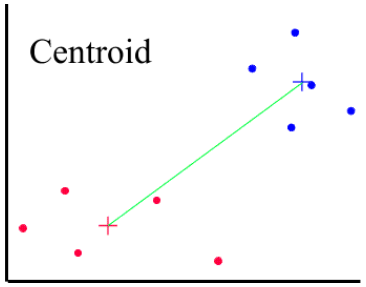
Hierarchical clustering

- Distance between two groups
  - **single link clustering -** the distance between two groups is defined as the distance between the two closest members of each group: $$D_{SL}(G_i, G_j) = \min_{x^r \in G_i, x^s \in G_j} D(x^r, x^s)$$
  - **complete link clustering -** the distance between two groups is defined as the distance between the two most distant pairs: $$D_{CL}(G_i, G_j) = \max_{x^r \in G_i, x^s \in G_j} D(x^r, x^s)$$
  - **average link clustering**, which measures the average distance between all pairs
  - **centroid** clustering
  - **ward** clustering – how much the sum of squares will increase if we merge the two clusters A and B

$$\Delta(A, B) = \sum_{i \in A \bigcup B} ||\vec{x_i} - \vec{m}_{A \bigcup B}||^2 - \sum_{i \in A} ||\vec{x_i} - \vec{m}_A||^2 - \sum_{i \in B} ||\vec{x_i} - \vec{m}_B||^2$$



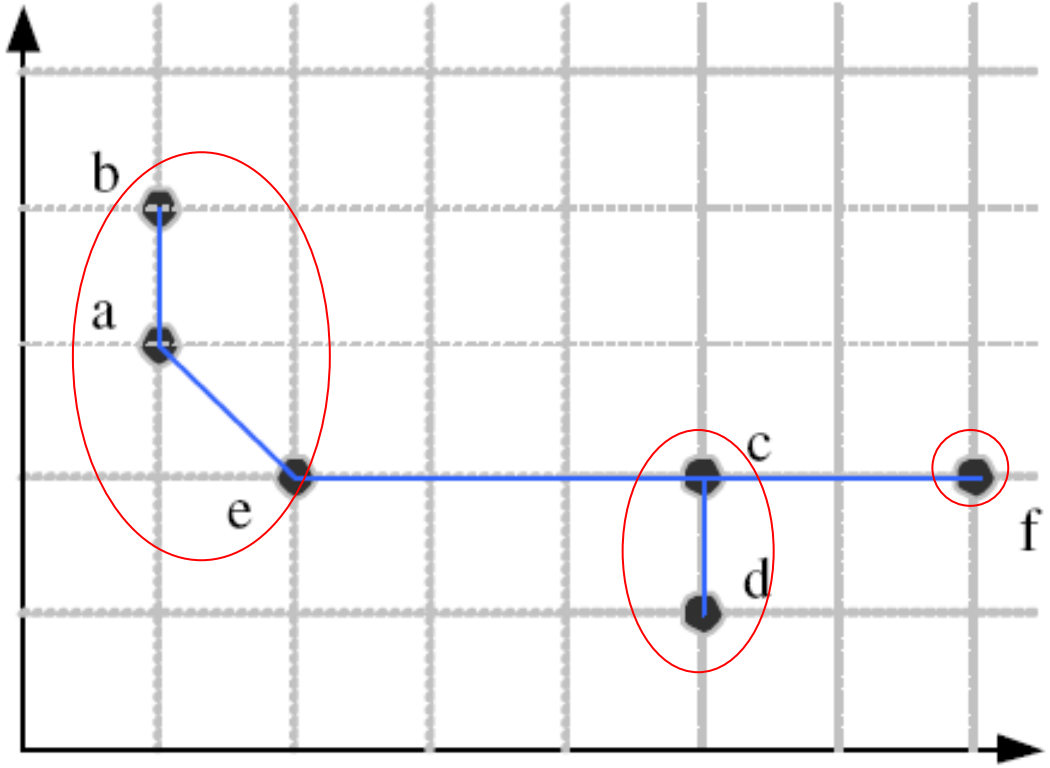


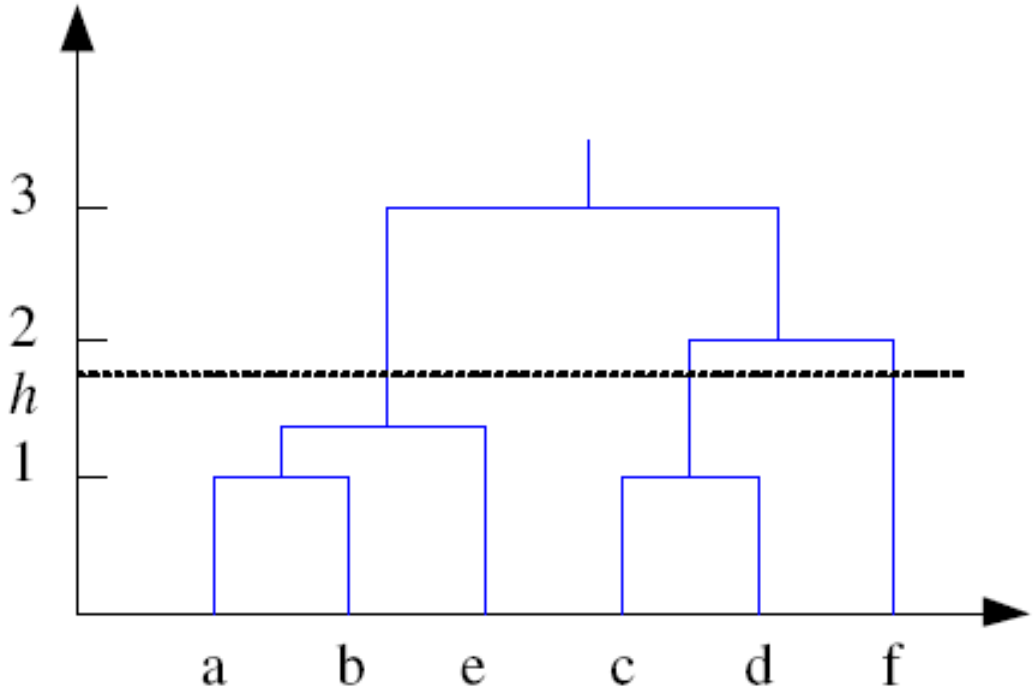


Hierarchical clustering

```
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='complete')
cluster.fit_predict(data)
```

Example: Single-Link Clustering





Dendrogram

Hierarchical clustering

- Distance measure between instances $\boldsymbol{x}^r$ and $\boldsymbol{x}^s$

  Minkowski (L_p) (Euclidean for $p = 2$)

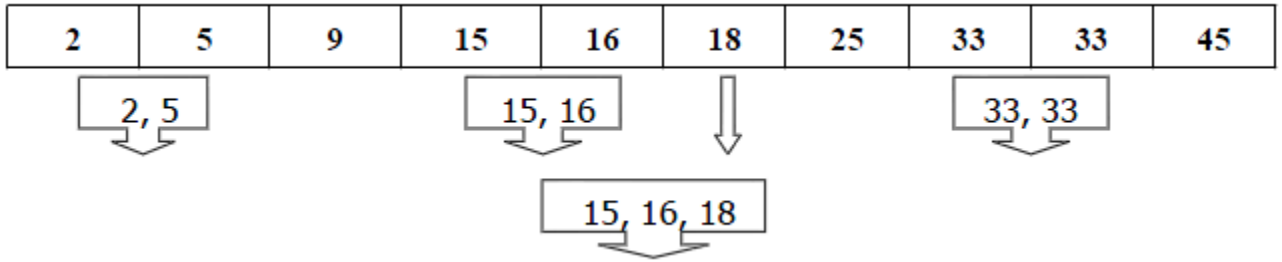
$$d_m\left(\boldsymbol{x}^r, \boldsymbol{x}^s\right) = \left[\sum_{j=1}^{d}\left(x_j^r - x_j^s\right)^p\right]^{1/p}$$

  City-block distance

$$d_{cb}\left(\boldsymbol{x}^r, \boldsymbol{x}^s\right) = \sum_{j=1}^{d}\left|x_j^r - x_j^s\right|$$

Example: Single-Linkage Clustering

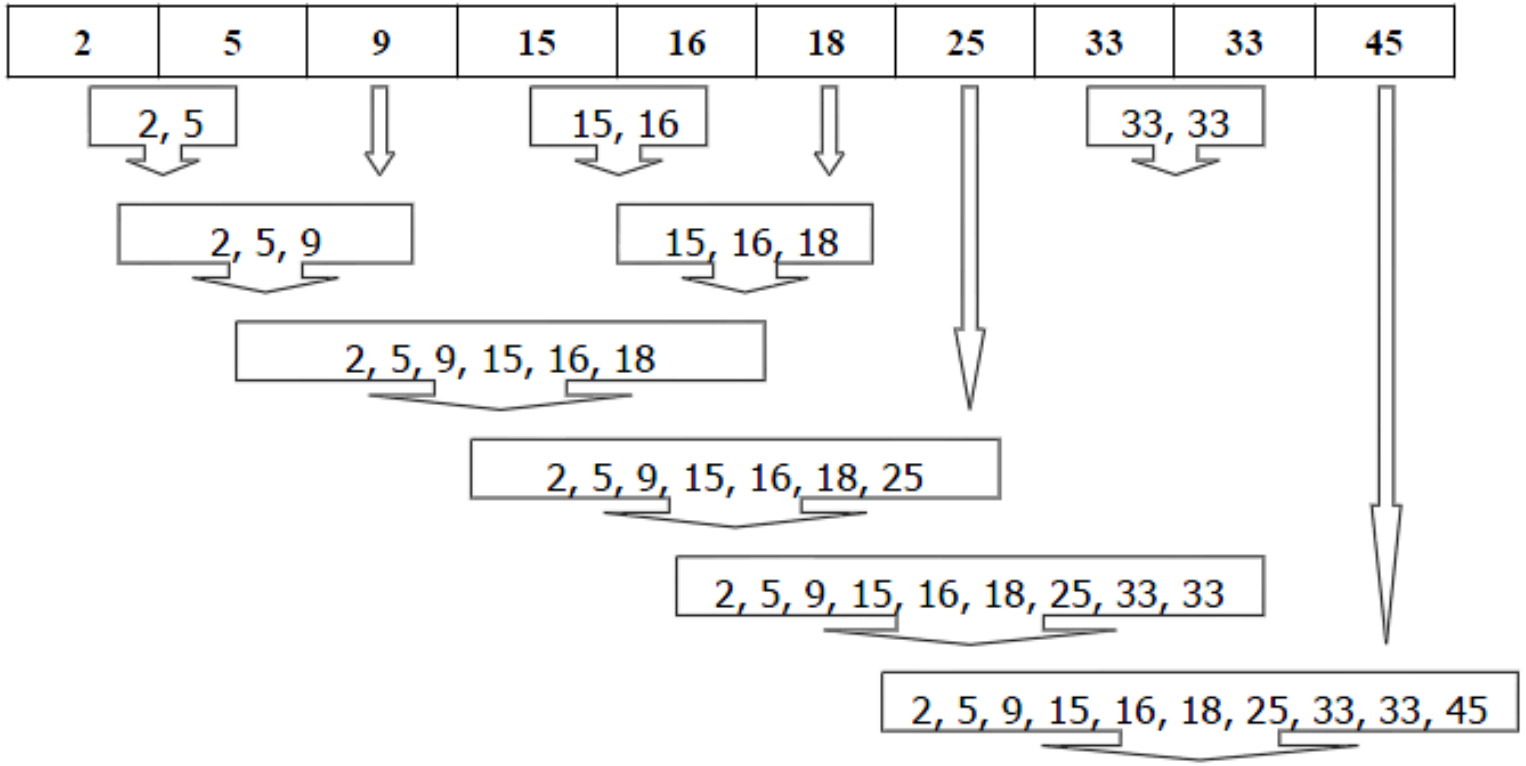
- На почеток, секој податок е во посебен кластер
- Single-linkage бара минимално растојание меѓу било кои два записи во различни кластери
  - Чекор 1:Минималното растојание е меѓу кластерите {33} и {33}. Раст.= 0, кластерите се спојуваат
  - Чекор 2:Кластерите {15} се {16} се спојуваат, каде раст. = 1
  - Чекор 3:Кластер {15, 16} се спојува со кластер {18}
  - Чекор 4:Кластерите {2} се {5} се спојуваат



Example: Single-Linkage Clustering

2	5	9	15	16	18	25	33	33	45

Example: Single-Linkage Clustering



- Агломерацијата продолжува со чекори од 4 –9
- Последниот кластер {2, 5, 9, 15, 16, 18, 25, 33, 33, 45} ги содржи сите податоци од множеството

Example: Complete-Linkage Clustering

2	5	9	15	16	18	25	33	33	45

Пример: Complete-Linkage Clustering

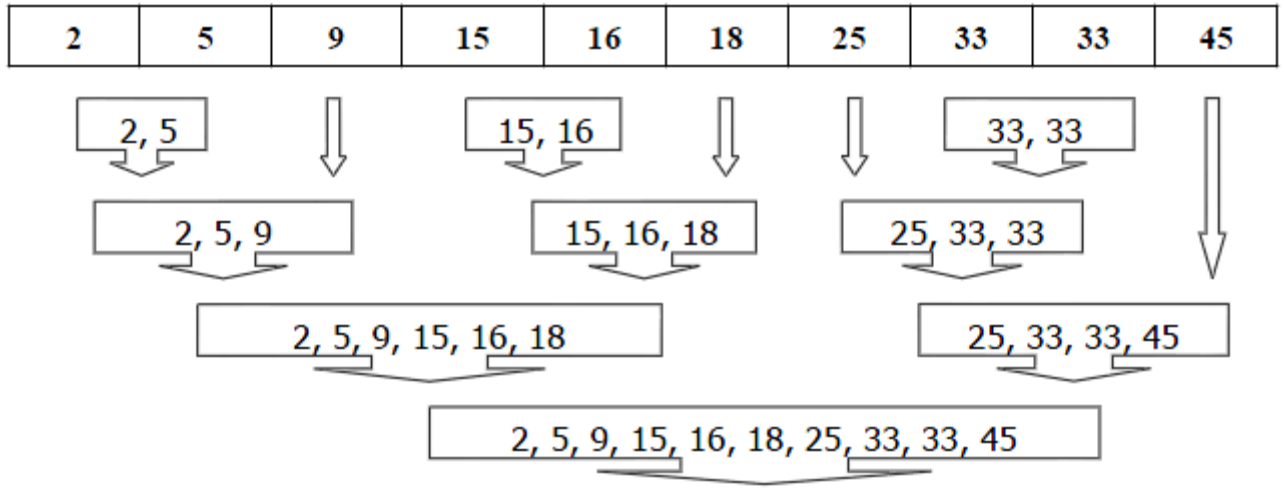
- *Complete-linkage*со примена на истото податочно множество

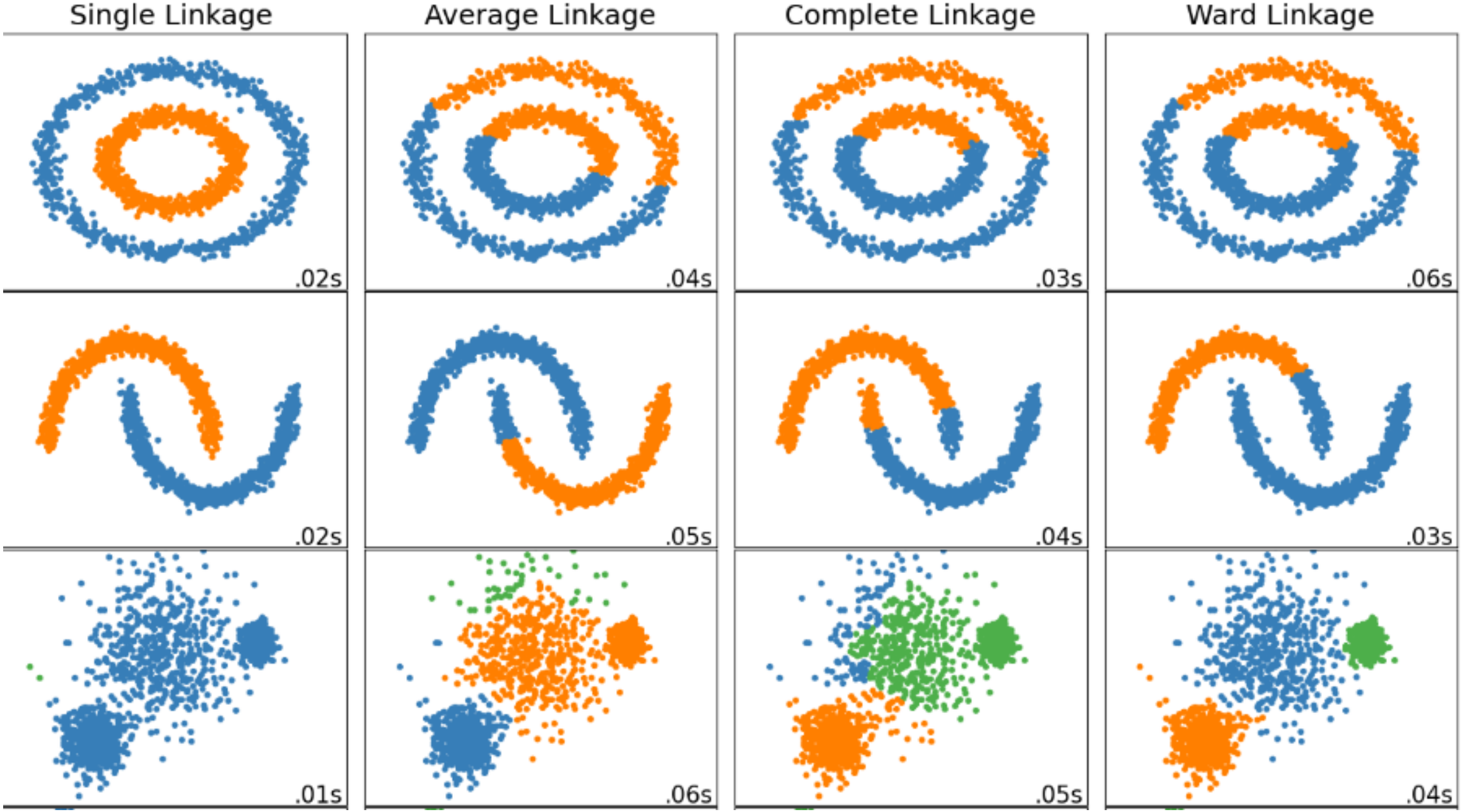
  2 5 9 15 16 18 25 33 33 45

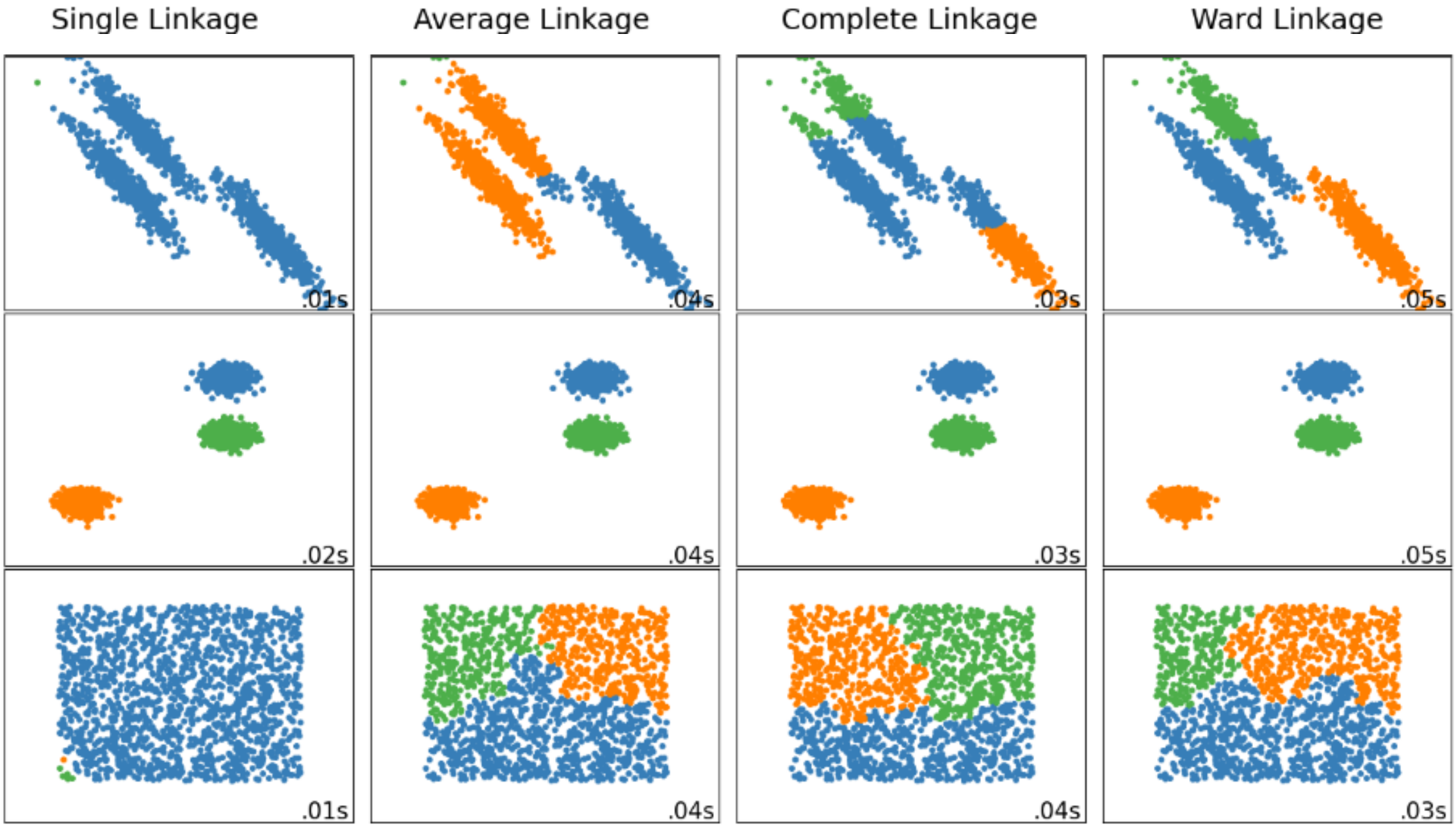
- Минимално растојание меѓу податоците во два кластери најоддалечени еден од друг
- Чекор 1: Секој податок е во посебен кластер
  - Овде нема разлика меѓу *Single* и *Complete-linkage*
  - Кластерите {33} и {33} се спојуваат
- Чекор 2: Кластерите {15} и {16} се спојуваат
  - Овде нема разлика меѓу *Single* и *Complete-linkage*

Пример: Complete-Linkage Clustering

- Чекор 3:*Complete-linkage*сега се разликува од *Single-linkage*
  - Најраздалечени соседи меѓу {15, 16} и {18} се 15 и 18, раст.= 3
  - Кластерите {2} и {5} исто имаат раст.= 3
  - Нема правило ако има исти растојанија
  - Резултат {2, 5} избран случајно
- *Complete-linkage*процедурата продолжува со чекорите 4 –9, се додека сите податоци не се содржат во еден кластер




Single Linkage
Average Linkage
Complete Linkage
Ward Linkage
.02s
.04s
.03s
.06s
.02s
.05s
.04s
.03s
.01s
.06s
.05s
.04s


Single Linkage
Average Linkage
Complete Linkage
Ward Linkage
.01s
.04s
.03s
.05s
.02s
.04s
.03s
.05s
.01s
.04s
.04s
.03s

Comparing different hierarchical linkage methods

- The main observations to make are:
  - **single linkage** is fast, and can perform well on non-globular data, but it performs poorly in the presence of noise.
  - **average and complete linkage** perform well on cleanly separated globular clusters but have mixed results otherwise.
  - **Ward** is the most effective method for noisy data.

Calinski-Harabasz (CH) Index

- used to evaluate the model of how well the clustering has been done
- also known as **Variance ratio criterion**
- measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation)
  - cohesion is estimated based on the distances from the data points in a cluster to its cluster centroid
  - separation is based on the distance of the cluster centroids from the global centroid
  - has a form of *(Separation)/(Cohesion)*

Calinski-Harabasz (CH) Index

- The CH index for K number of clusters on a dataset D =[d_1 , d_2 , d_3 , … d_N] is defined as

$$CH = \left[\frac{\sum_{k=1}^{K} n_k \left\| c_k - c \right\|^2}{K-1} \right] / \left[\frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} \left\| d_i - c_k \right\|^2}{N-K} \right]$$

- where, n_k and c_k are the no. of points and centroid of the k^{th} cluster respectively, c is the global centroid, N is the total no. of data points.
- Higher value of CH index means the clusters are dense and well separated, although there is no “acceptable” cut-off value.
- We need to choose that solution which gives a peak or at least an abrupt elbow on the line plot of CH indices.

Silhouette Coefficient

- The Silhouette Coefficient is defined for each sample and is composed of two scores:
  - **a**: The mean distance between a sample and all other points in the same class.
  - **b**: The mean distance between a sample and all other points in the *next nearest cluster*.
- The Silhouette Coefficient *s* for a single sample is then given as:

$$s = \frac{b - a}{max(a, b)}$$

- The Silhouette Coefficient for a set of samples is given as the mean of the Silhouette Coefficient for each sample.

Davies-Bouldin Index

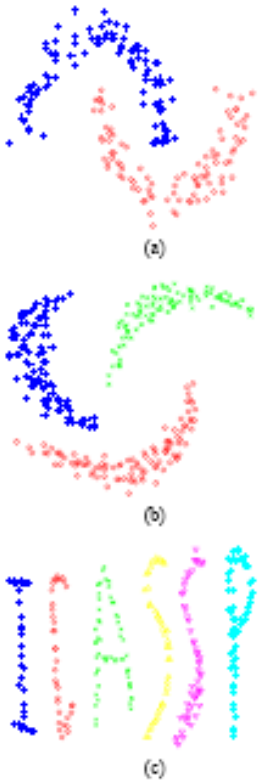
- In the context of this index, similarity is defined as a measure R_{ij} that trades off:
  - s_i, the average distance between each point of cluster i and the centroid of that cluster – also know as cluster diameter.
  - d_{ij}, the distance between cluster centroids i and j.
- A simple choice to construct R_{ij} so that it is nonnegative and symmetric is:

$$R_{ij} = \frac{s_i + s_j}{d_{ij}}$$

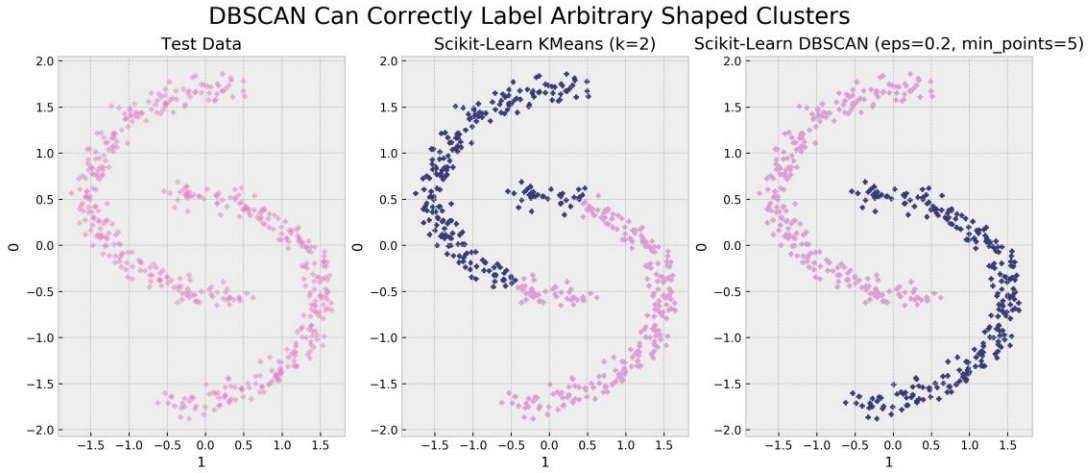
- Then the Davies-Bouldin index is defined as: $DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij}$

DBSCAN

Density-Based Spatial Clustering of Applications with Noise



- DBSCAN is a density-based algorithm
  - Density = number of points within a specified radius (Eps)
  - A point is a core point if it has more than a specified number of points (MinPts) within Eps
    - These are points that are at the interior of a cluster
  - A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point
  - A noise point is any point that is not a core point or a border point.




```
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
```

DBSCAN: Core, Border, and Noise Points

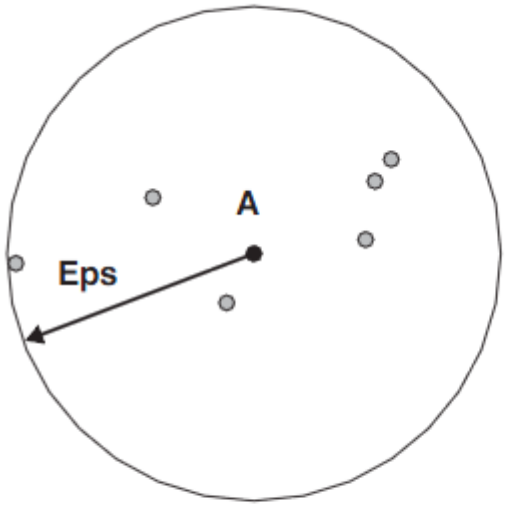


Figure 8.20. Center-based density.

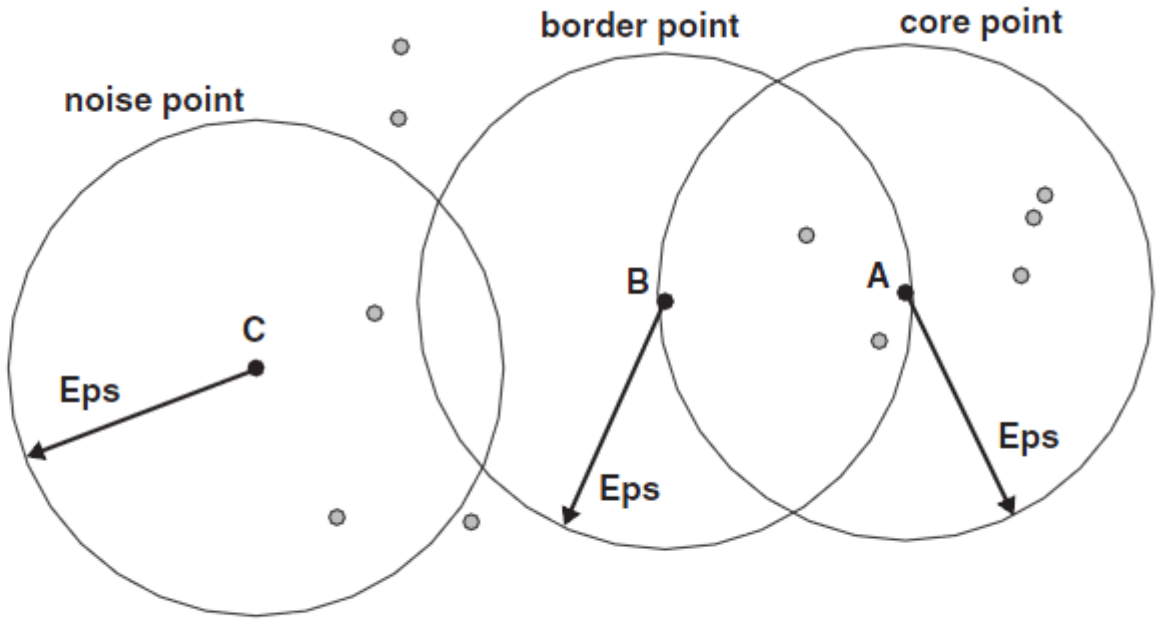


Figure 8.21. Core, border, and noise points.

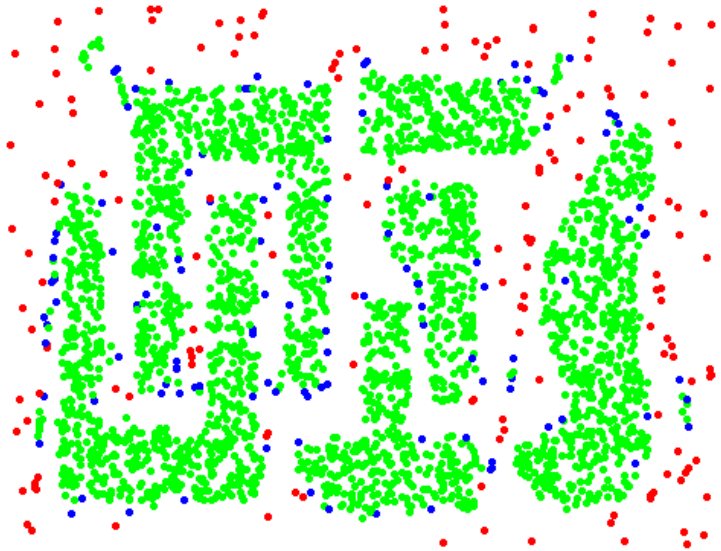
MinPts=7

The DBSCAN algorithm

1. Label all points as core, border or noise points
2. Eliminate noise points
3. Put an edge between all core points that are within *Eps* of each other
4. Make each group of connected core points into a separate cluster
5. Assign each border point to one of the clusters of its associated core points

DBSCAN: Core, Border and Noise Points

Original Points

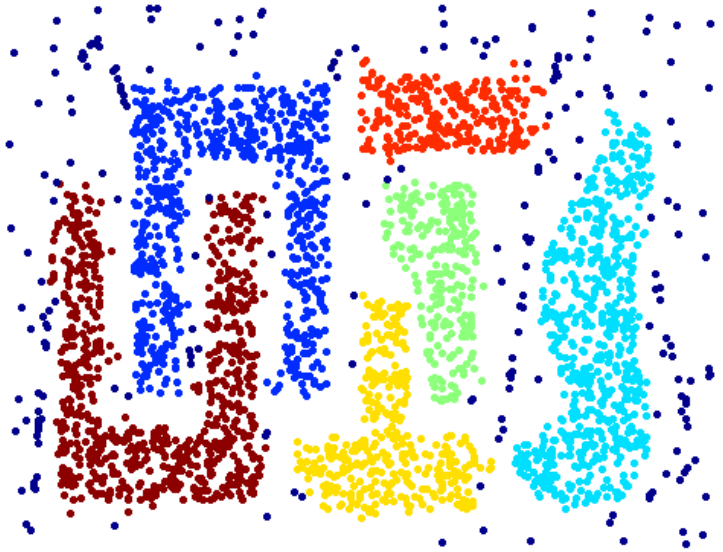


Point types: core, border and noise

Eps = 10, MinPts = 4

When DBSCAN Works Well

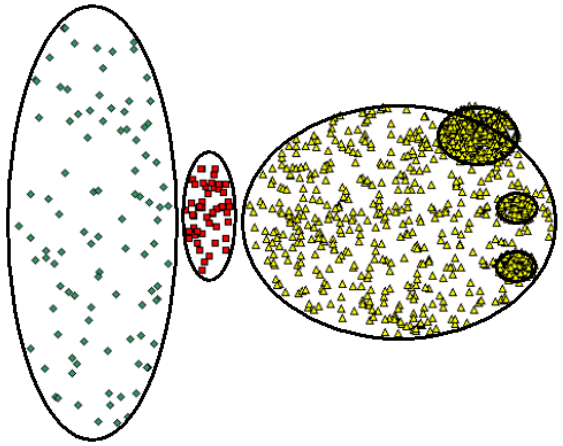
Original Points



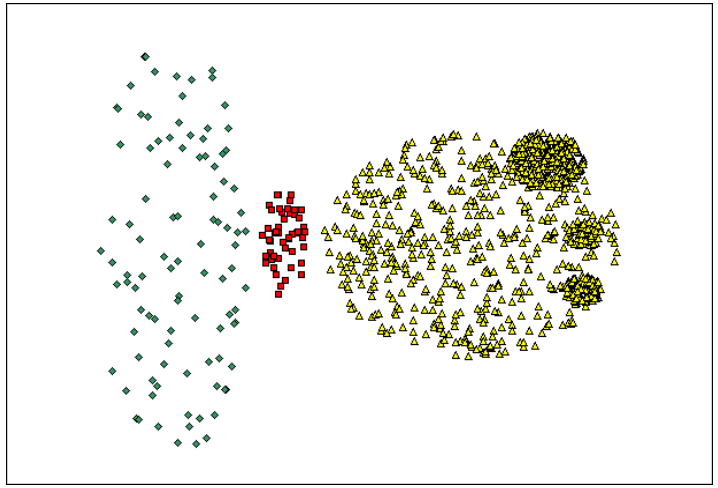
Clusters

- Resistant to Noise
- Can handle clusters of different shapes and sizes

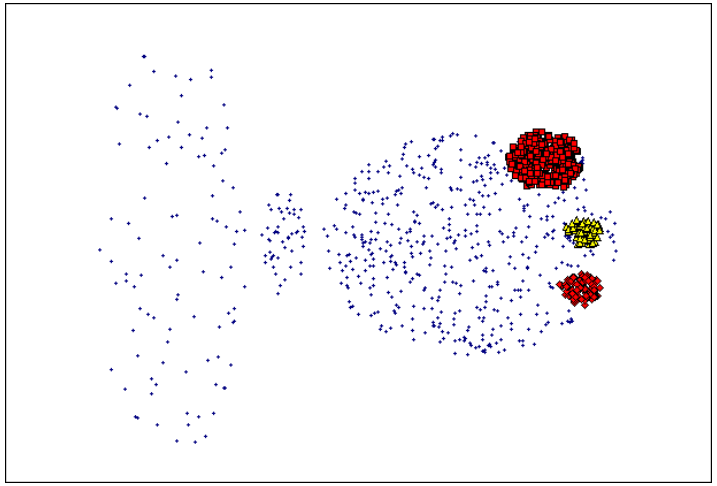
When DBSCAN Does NOT Work Well



Original Points



(MinPts=4, Eps=9.92)



(MinPts=4, Eps=9.75).

- Varying densities
- High-dimensional data

Choose the optimal parameters

- MinPts - rules of thumb
  - The larger the data set, the larger the value of MinPts should be
  - If the data set is noisier, choose a larger value of MinPts
  - Generally, MinPts should be greater than or equal to the dimensionality of the data set
  - For 2-dimensional data, use DBSCAN's default value of MinPts = 4 or 5
  - If your data has more than 2 dimensions, choose MinPts = 2*dim, where dim= the dimensions of your data set

Choose the optimal parameters

- After you select your MinPts value, you can move on to determining ε.
  - One technique to automatically determine the optimal ε value is to calculates the average distance between each point and its k nearest neighbors, where k = the MinPts value you selected.
  - The average k-distances are then plotted in ascending order on a k-distance graph.
  - The optimal value for ε is at the point of maximum curvature (i.e. where the graph has the greatest slope-knee).

3. If *Epsilon* is 2 and *minpoint* is 3, what are the clusters that DBScan would discover with the following 8 examples: A_1=(1,1), A_2=(1,3), A_3=(1,5), A_4=(1,8), A_5=(3,1), A_6=(4,4), A_7=(4,6), A_8=(5,1), A_9=(6,4), A_{10}=(6,6), A_{11}=(7,1).

Draw the 10 by 10 space and illustrate the discovered clusters. What if Epsilon is increased to 3 or 4 ?

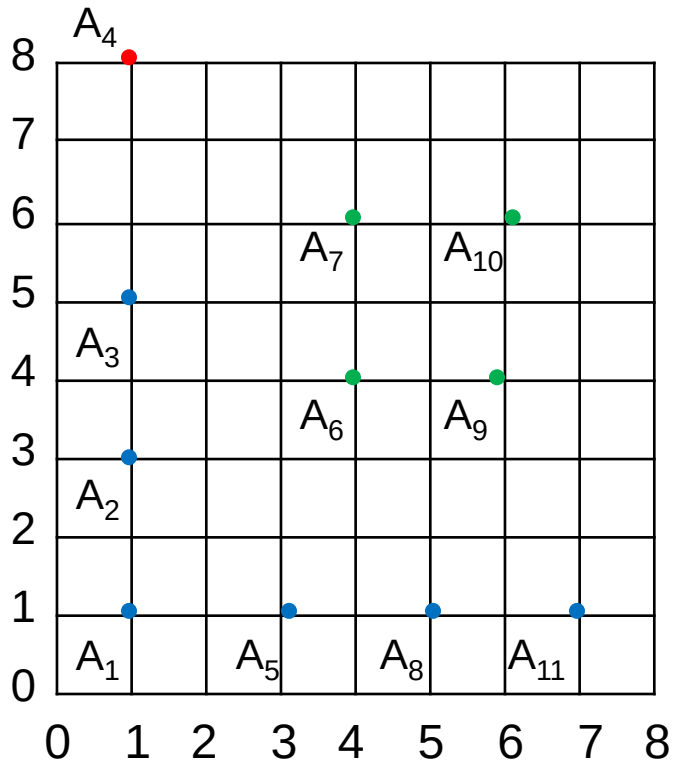
Solution:

What is the Epsilon neighborhood of each point?

$N_2(A_1)=\{A_1, A_2, A_5\}$; $N_2(A_2)=\{A_1, A_2, A_3\}$; $N_2(A_3)=\{A_2, A_3\}$; $N_2(A_4)=\{A_4\}$; $N_2(A_5)=\{A_1, A_5, A_8\}$; $N_2(A_6)=\{A_6, A_7, A_9\}$; $N_2(A_7)=\{A_6, A_7, A_{10}\}$; $N_2(A_8)=\{A_5, A_8, A_{11}\}$; $N_2(A_9)=\{A_6, A_9, A_{10}\}$; $N_2(A_{10})=\{A_7, A_9, A_{10}\}$; $N_2(A_{11})=\{A_8, A_{11}\}$;

So A_4 is an outlier, A_3 and A_{11} are border points, and we have two clusters $C_1=\{A_1, A_2, A_3, A_5, A_8, A_{11}\}$ and $C_2=\{A_6, A_7, A_9, A_{10},\}$

Прашање

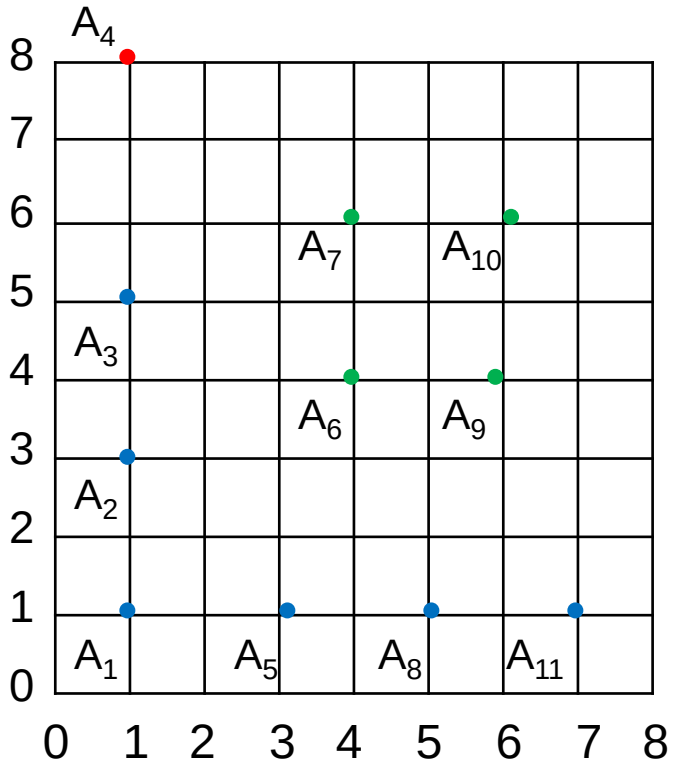


- Што ќе се случи доколку Epsilon стане 3?

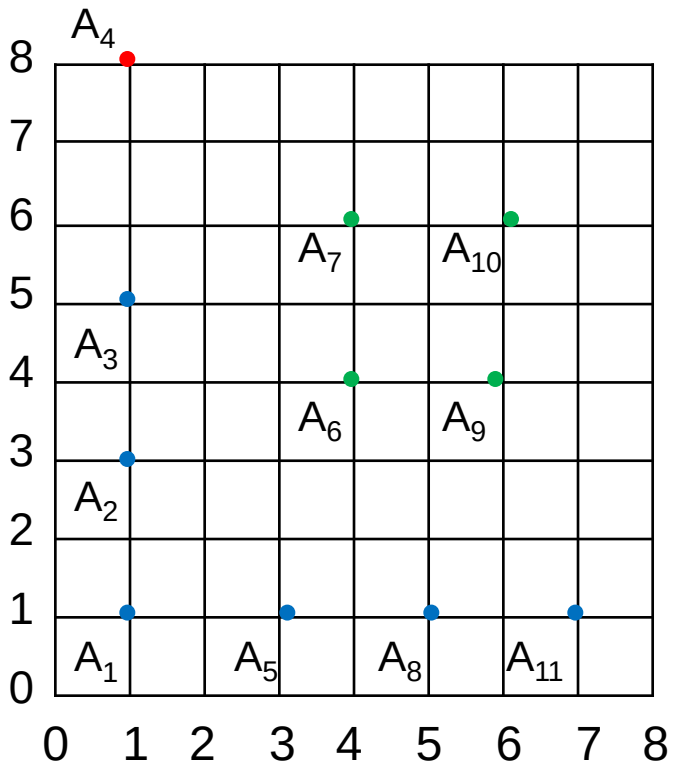
a) Ќе се спојат сите во еден кластер
b) Нема да има шум во кластерите
c) Нема да има промени во однос на случајот кога Epsilon е 2
d) Ќе се добијат три кластери

Прашање

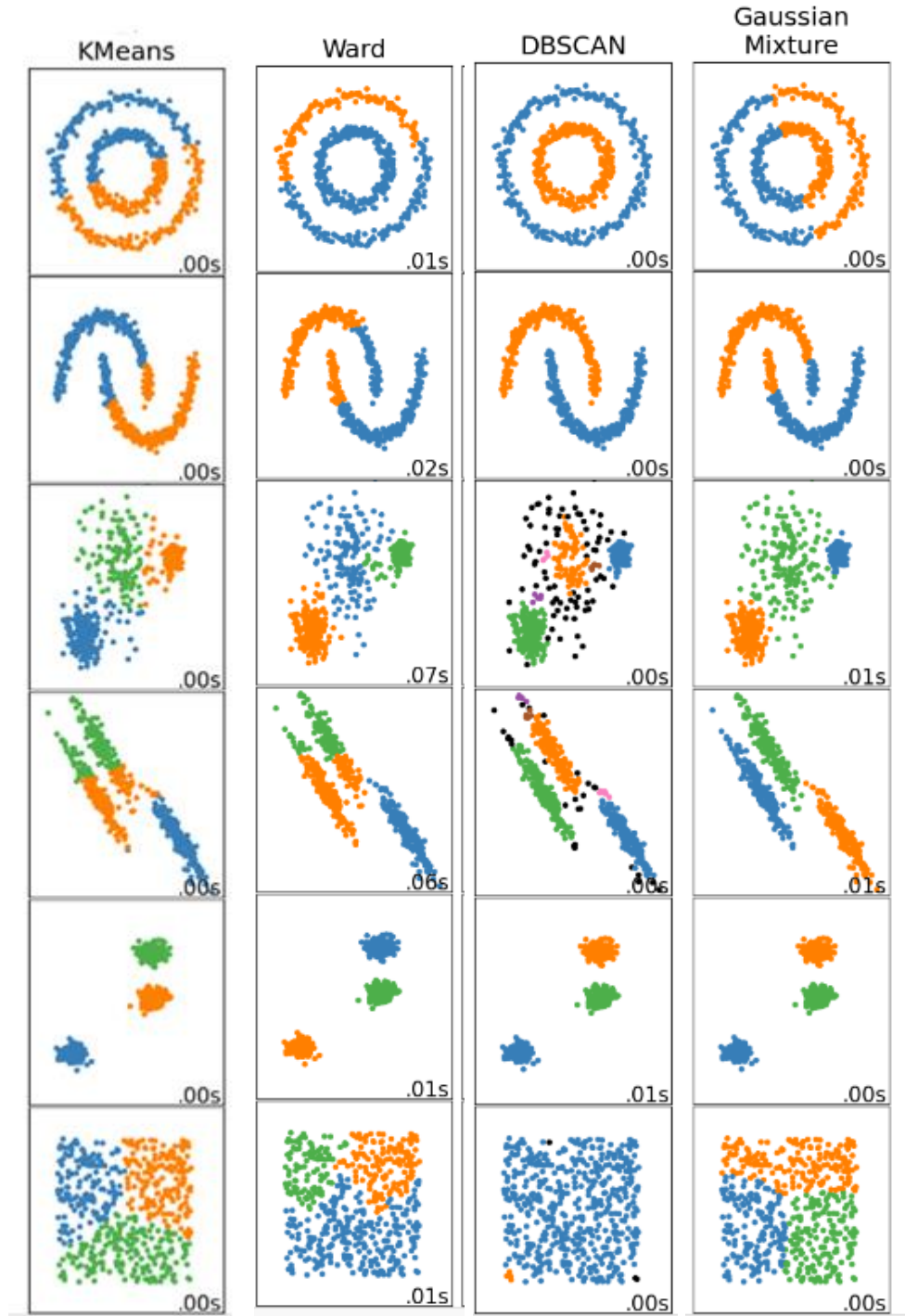
- Што ќе се случи доколку Epsilon стане 4?



a) Ќе се спојат сите во еден кластер
b) Нема да има шум во кластерите
c) Нема да има промени во однос на случајот кога Epsilon е 2
d) Ќе се добијат три кластери



- If Epsilon is 3, then the neighborhood of A_3, A_4, A_6, A_7, A_9 and A_{10} will increase and A_4 would join the cluster C_1.
- If Epsilon is 4, then the neighborhood of several points will increase and the two clusters will merge into one.


KMeans
Ward
DBSCAN
Gaussian Mixture
.00s
.01s
.00s
.00s
.00s
.02s
.00s
.00s
.00s
.07s
.00s
.01s
.00s
.06s
.00s
.01s
.00s
.01s
.01s
.00s
.00s
.01s
.00s
.00s