Unsupervised Learning - Clustering

- Assume training data set contains “unlabeled” samples
  - i.e. we are not told which class they belong to
- Why interested in such a *difficult* problem?
  - It arises in many real-life situations…
  - Class labels are unknown.
- In this context,
  - “discovering” different classes from unlabeled data…
  - leads us to change our “way of thinking” for learning
- Question:
  - Would it be possible to learn anything from *unlabeled data ?*
- Answer:
  - It depends on the conditions we are willing to accept
  - There is no theorem without any assumption!

Assumptions - Models

- Probability distribution of data:
  - *parametric* vs. *nonparametric*
- Parameters of distribution:
  - *known* vs. *unknown*
- Measures for similarity
- Quality assessment:
  - How good is a particular method?
  - Under what conditions?
  - How can quality be measured?
- all these discussed here…

Partitional methods:

- Maximum likelihood (expectation maximization)
- k-means
- Fuzzy k-means
- Support vector clustering
- Self-organizing maps (SOMs)
- Spectral/Graph clustering

Hierarchical models:

- Agglomerative (bottom-up)
  - Single-linkage (nearest-neighbor)
  - Complete-linkage (farthest-neighbor)
  - Centroid-linkage
  - Average-linkage
- Divisive (top-down)
  - Minimum spanning trees
  - Graph clustering approaches

Mixture of Densities

- Given a dataset of *unlabeled* samples: $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

Assumptions:

- Samples come from a *known* number classes, namely *c* classes
- Prior probabilities of classes: $P(\omega_i)$ *known*
- Forms of class-conditional probability densities: $p(\mathbf{x} \mid \omega_i, \boldsymbol{\theta}_i)$ *known*
- Values for the *c* parameter vectors: $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_c$ *unknown*
- Class labels (memberships) are *unknown*

We assume:

- The samples were obtained by selecting class ω_i, whose *a priori* prob. is $P(\omega_i)$
- Then, we select **x**

  based on prob. law $p(\mathbf{x} \mid \omega_i, \boldsymbol{\theta}_i)$
- Thus, the prob. density function for a sample **x** is:

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{i=1}^{c} p(\mathbf{x} \mid \omega_i, \boldsymbol{\theta}_i) P(\omega_i)$$

  where $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_c]^t$
- called the *mixture density*

Example: Univariate normal, $c = 2$

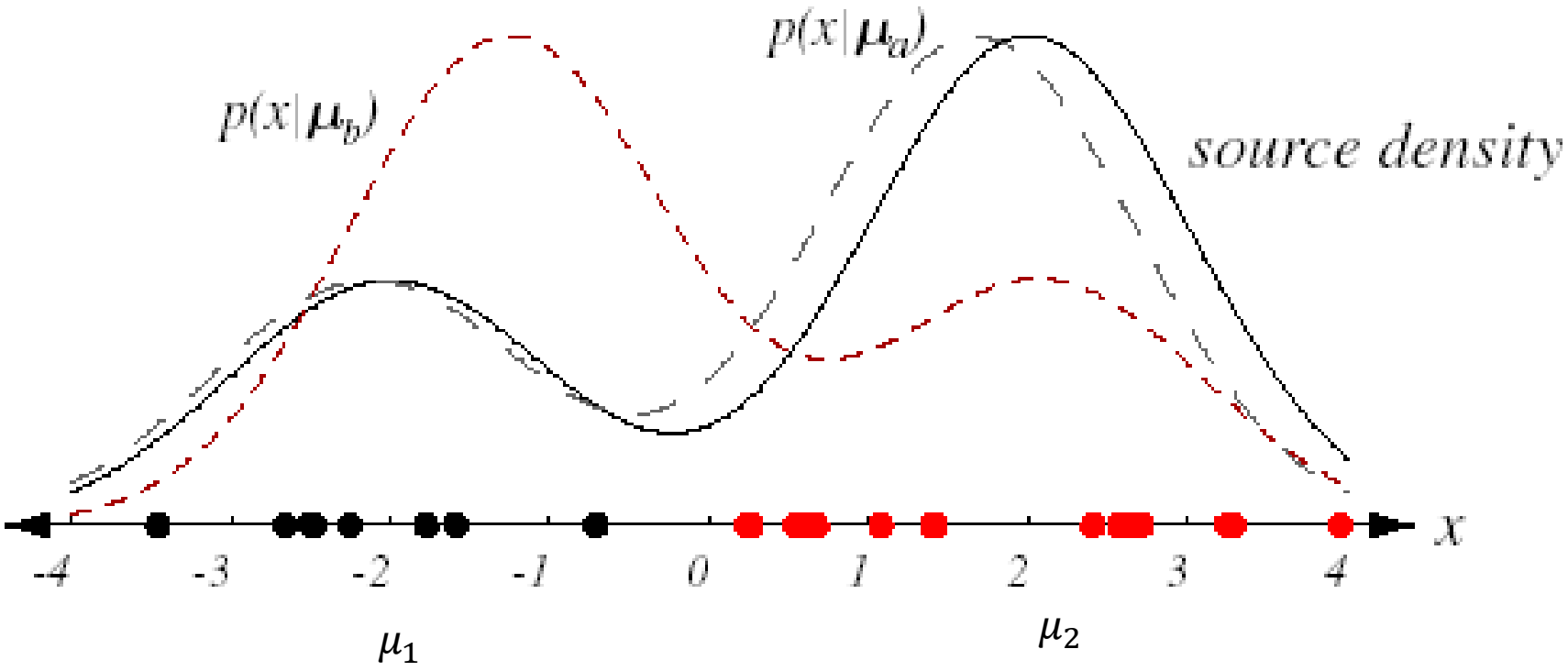


Figure from Duda et al.

Maximum Likelihood Estimate (MLE)

- The unsupervised version of MLE
- Given a dataset of *unlabeled* samples:

$$D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$$

- drawn independently from the mixture density:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{i=1}^{c} p(\mathbf{x}|\omega_i, \boldsymbol{\theta}_i) P(\omega_i)$$

- where $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_c]^t$ is *fixed* but *unknown*
- **Aim:** Maximize the likelihood function

$$p(\mathcal{D}|\boldsymbol{\theta}) = \prod_{k=1}^{n} p(\mathbf{x}_k|\boldsymbol{\theta}) \qquad (1)$$

- How do we find that maximum?
- Again, like in the supervised MLE…
  - take the log-likelihood function,
  - apply the gradient operator wrt

$$\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_c]^t$$

  - and find the necessary (first order) conditions:

$$\nabla_{\boldsymbol{\theta}} l = \mathbf{0}$$

- Should also find the sufficient (second order) conditions

Normal Distributions

- In this context, called *mixture of normals*,
- The *component* densities are given by:

  where $\theta_i = [\mu_i , \Sigma_i]^t$

$$p(\mathbf{x}_k \mid \omega_i, \boldsymbol{\theta}_i) \sim N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

Case 1: Very restrictive, and *not common* in real life
Case 2: More realistic, but *more difficult*
Case 3: Completely *unknown* data… *extremely* difficult

Cases:

Case	μ_i	Σ_i	$P(\omega_i)$	c
1	???	known	known	known
2	???	???	???	known
3	???	???	???	???

Unknown Mean Vectors

Assume:

- Σ_i are *known*, but μ_i's are *unknown*
- Can be shown that the MLE must satisfy:

$$\sum_{k=1}^{n} P(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\mu}}) \boldsymbol{\Sigma}_i^{-1} (\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i) = 0$$

where $\hat{\boldsymbol{\mu}} = \left[\hat{\boldsymbol{\mu}}_1, \ldots, \hat{\boldsymbol{\mu}}_c\right]^t$, and $P(.|.)$ is the posterior prob.

- Pre multiplying by Σ, and rearranging:

$$\hat{\boldsymbol{\mu}}_i = \frac{\sum_{k=1}^{n} P(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\mu}}) \mathbf{x}_k}{\sum_{k=1}^{n} P(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\mu}})} \qquad (2)$$

- How to compute the posterior probs.?

$$P(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\mu}}) = \frac{p(\mathbf{x}_k \mid \omega_i, \hat{\boldsymbol{\mu}}_i) P(\omega_i)}{\sum_{j=1}^{c} p(\mathbf{x}_k \mid \omega_j, \hat{\boldsymbol{\mu}}_i) P(\omega_i)}$$

Note:

- The MLE for $\hat{\boldsymbol{\mu}}_i$ is the *weighted average* of the samples
- The *weight* for $\mathbf{x}_k$ is given by
  how "likely" is that $\mathbf{x}_k$ belongs to ω_i
- Eq. (2) doesn't give an explicit solution:
  $\hat{\boldsymbol{\mu}}$ contains $\hat{\boldsymbol{\mu}}_i$

An Iterative Algorithm

Find an *initial* estimate for $\hat{\boldsymbol{\mu}}(0)$, and repeat

for $i \leftarrow 1$ to c

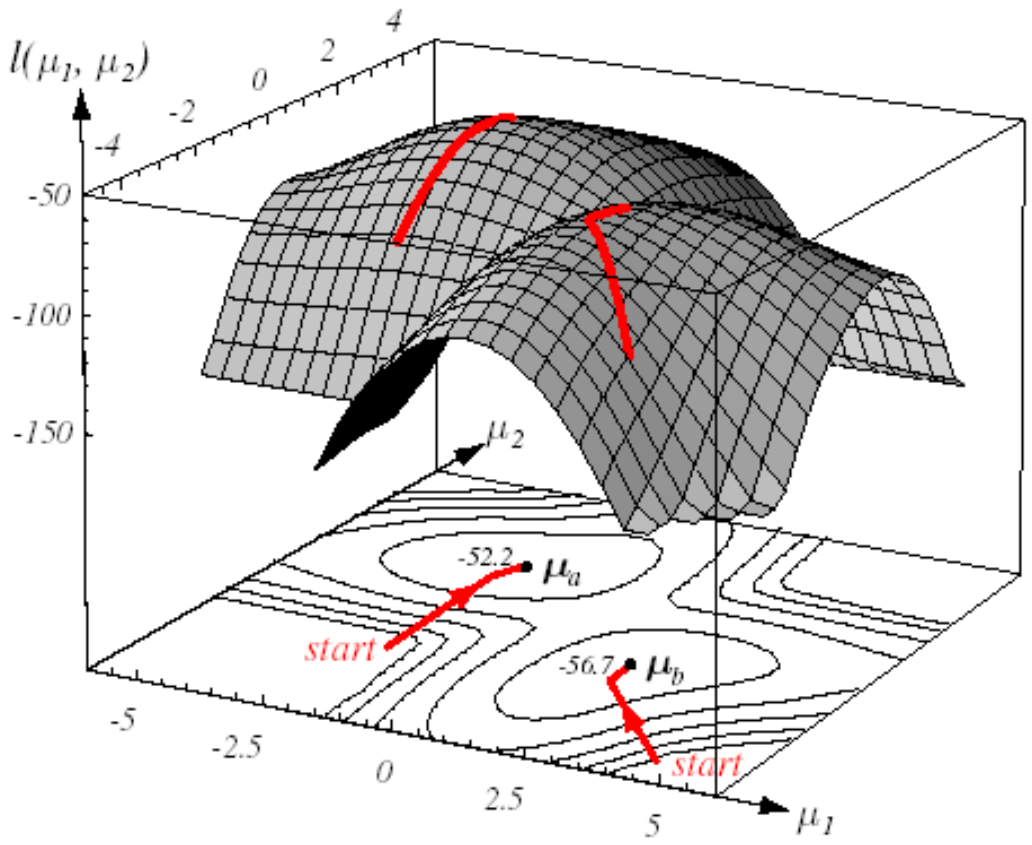
$$\hat{\boldsymbol{\mu}}_i(j+1) = \frac{\sum_{k=1}^{n} P(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\mu}}(j))\mathbf{x}_k}{\sum_{k=1}^{n} P(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\mu}}(j))} \qquad (3)$$

endfor

until *small* (or no) change in $\hat{\boldsymbol{\mu}}_i(j)$

It is a *gradient ascent* procedure for maximizing the likelihood

Example: Univariate normal, $c = 2$



All Parameters of Normal Dist. Unknown

Assume: $\hat{P}(\omega_i)$, $\boldsymbol{\mu}_i$'s and $\boldsymbol{\Sigma}_i$'s are *unknown*

Solution is given by:

$$\hat{\boldsymbol{\mu}}_i = \frac{\sum_{k=1}^{n} \hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}})\mathbf{x}_k}{\sum_{k=1}^{n} \hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}})} \quad (4)$$

$$\hat{\boldsymbol{\Sigma}}_i = \frac{\sum_{k=1}^{n} \hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}})(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)^t}{\sum_{k=1}^{n} \hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}})} \quad (5)$$

$$\hat{P}(\omega_i) = \frac{1}{n}\sum_{k=1}^{n} \hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}}) \quad (6)$$

where:

$$\hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}}) = \frac{p(\mathbf{x}_k \mid \omega_i, \hat{\boldsymbol{\theta}}_i)\hat{P}(\omega_i)}{\sum_{i=1}^{c} p(\mathbf{x}_k \mid \omega_i, \hat{\boldsymbol{\theta}}_i)\hat{P}(\omega_i)}$$

$$= \frac{|\hat{\boldsymbol{\Sigma}}_i|^{-1/2}\, e^{-\frac{1}{2}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)^t \hat{\boldsymbol{\Sigma}}_i^{-1}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)} \hat{P}(\omega_i)}{\sum_{j=1}^{c} |\hat{\boldsymbol{\Sigma}}_j|^{-1/2}\, e^{-\frac{1}{2}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_j)^t \hat{\boldsymbol{\Sigma}}_j^{-1}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_j)} \hat{P}(\omega_j)} \quad (7)$$

Algorithm (*Expectation Maximization*):

- Use initial estimates for (4), (5) and (6)
- Iteratively re-compute (7), (4), (5) and (6)
- Repeat until small change in $\hat{\boldsymbol{\mu}}_i(j)$ and $\hat{\Sigma}_i(j)$

k-Means Clustering

Notes:

- It is a *gradient ascent* procedure
- Results depend upon the starting parameters
- Can be stuck in a local optima
- Computationally expensive:
  - Compute normal distribution function for each $\mathbf{x}_k$
  - Compute inverse of all Σ_i's at each step

Simplification:

- Assume diagonal covariance matrices
- Simpler: Assume the *c* covariances are *equal*
- Even simpler!... use the identity matrix…

- One of the *important* algorithms in machine learning
- Introduced by J. MacQueen in 1967
- It's very simple…
  - Simplifies computations and
  - accelerates convergence
- Why *k*?
  - Recall our aim is to find *c* classes
  - So, should call it *c*-Means algorithm
  - Historically, been called *k*-Means

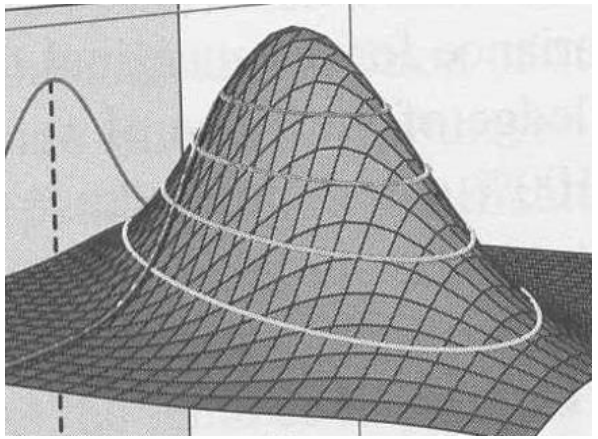
Mahalanobis Distance

$$\hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}}) = \frac{p(\mathbf{x}_k \mid \omega_i, \hat{\boldsymbol{\theta}}_i)\hat{P}(\omega_i)}{\sum_{i=1}^{c} p(\mathbf{x}_k \mid \omega_i, \hat{\boldsymbol{\theta}}_i)\hat{P}(\omega_i)}$$

- How does it work?
- Lets analyze Eq. (7)

$$= \frac{|\hat{\boldsymbol{\Sigma}}_i|^{-1/2} \, e^{-\frac{1}{2}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)^t \hat{\boldsymbol{\Sigma}}_i^{-1}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)} \hat{P}(\omega_i)}{\sum_{j=1}^{c} |\hat{\boldsymbol{\Sigma}}_j|^{-1/2} \, e^{-\frac{1}{2}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_j)^t \hat{\boldsymbol{\Sigma}}_j^{-1}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_j)} \hat{P}(\omega_j)}$$

- $\hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}})$ is *large* when the squared Mahalanobis distance $(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)^t \hat{\boldsymbol{\Sigma}}_i^{-1}(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)$ is *small*



Recall:

- When using normal distributions,
  - points with the same Mahalanobis distance are equally likely
- smaller the distance $\Rightarrow$ higher probability

- Suppose we assume $\hat{\boldsymbol{\Sigma}}_i = \mathbf{I}$
- Then, we just compute the Euclidean distance:

$$(\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i)^t \mathbf{I} (\mathbf{x}_k - \hat{\boldsymbol{\mu}}_i) = \left\| \mathbf{x}_k - \hat{\boldsymbol{\mu}}_i \right\|^2$$

- Equivalent to finding the mean $\boldsymbol{\mu}_j$ closest to $\mathbf{x}_k$ using the Euclidean distance, and change $\hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}})$ to:

$$\hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}}) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- These assumptions and simplifications
  - lead to the *k*-Means algorithm

k-Means Algorithm

Algorithm ***k*-Means**

Input: Number of clusters, k

A training dataset, $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

begin

Initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

repeat

Assign $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ to the class (cluster) of the nearest $\boldsymbol{\mu}_i$

Re-compute $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

until no change in $\boldsymbol{\mu}_i$

return $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

Notes on *k*-Means

- Need to provide the number of clusters, *k*
  - Need some knowledge about the problem
  - When not known, use indices of validity (later)
- The *nearest* mean is found by using

  an arbitrary similarity measure
- Can use Euclidean distance, for example
- Other distances can also be used
  - Convergence may be affected though

Complexity

- Initial values of $\boldsymbol{\mu}_i$ affect the convergence
- We may guess initial values based on knowledge of the problem
- or may choose k random samples from $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$

  See figure on next page.
- Complexity of k-Means is $O(ndkT)$
  - where n is the number of *training samples*
  - d is the *dimension* of the feature space
  - k is the number of *clusters* (or *classes*)
  - T is the number of *iterations*
- Average-case complexity is $O(n^2dk)$
  - Though under certain assumptions
- In practice, $T < n$, but worst-case complexity?

Example: 2 clusters, 1D

Different values of $\boldsymbol{\mu}_i$ lead to *different* results

Note: *k*-Means can be seen as a *gradient ascent* method to maximize the log-likelihood function

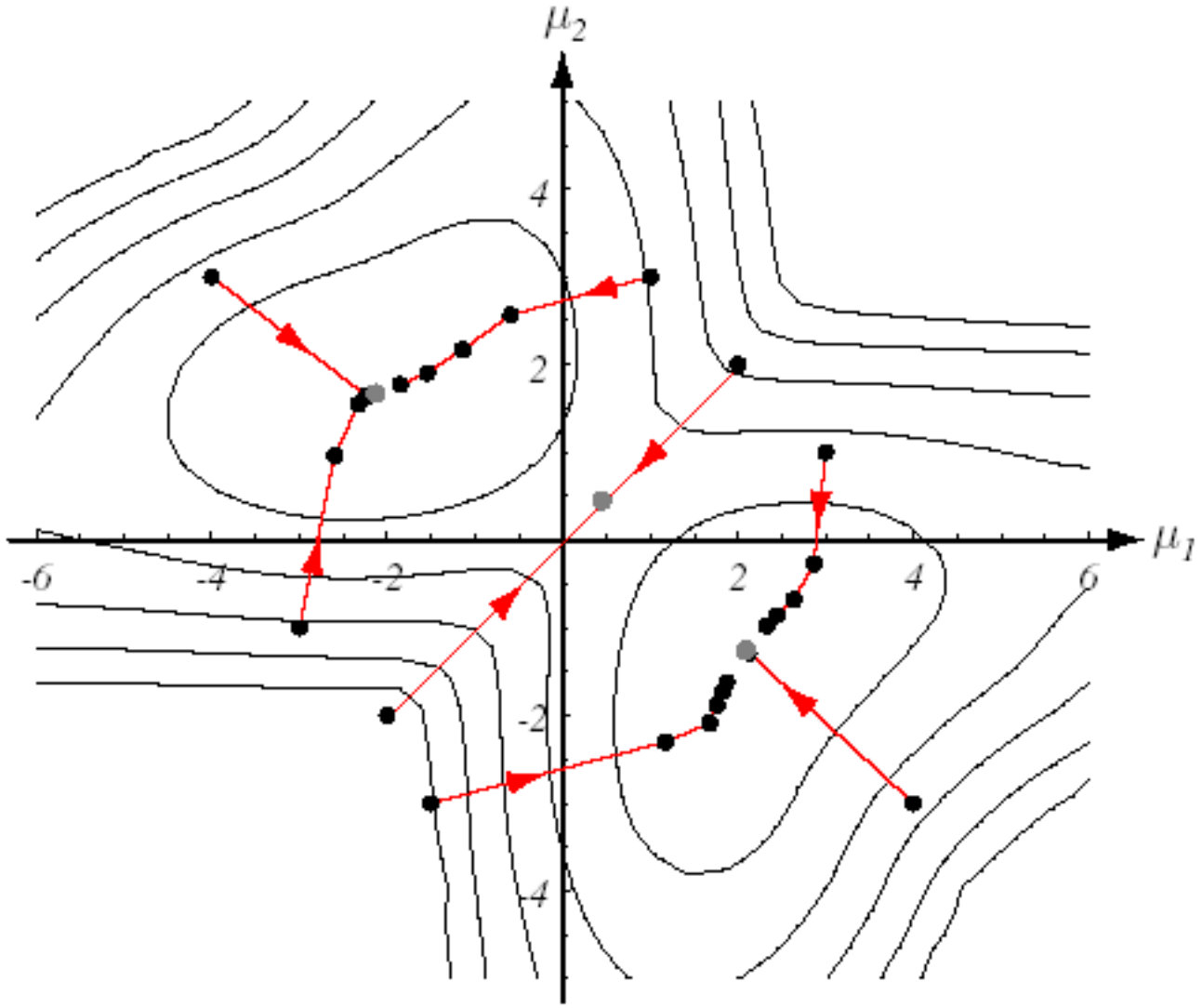


Figure from Duda et al.

Another example: 3 clusters, 2D

- Three initial cluster centers $\boldsymbol{\mu}_i$ chosen randomly
- Metric: Euclidean distance
- Each step of *k*-Means gives a Voronoi diagram (tessellation), where the prototypes are $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$

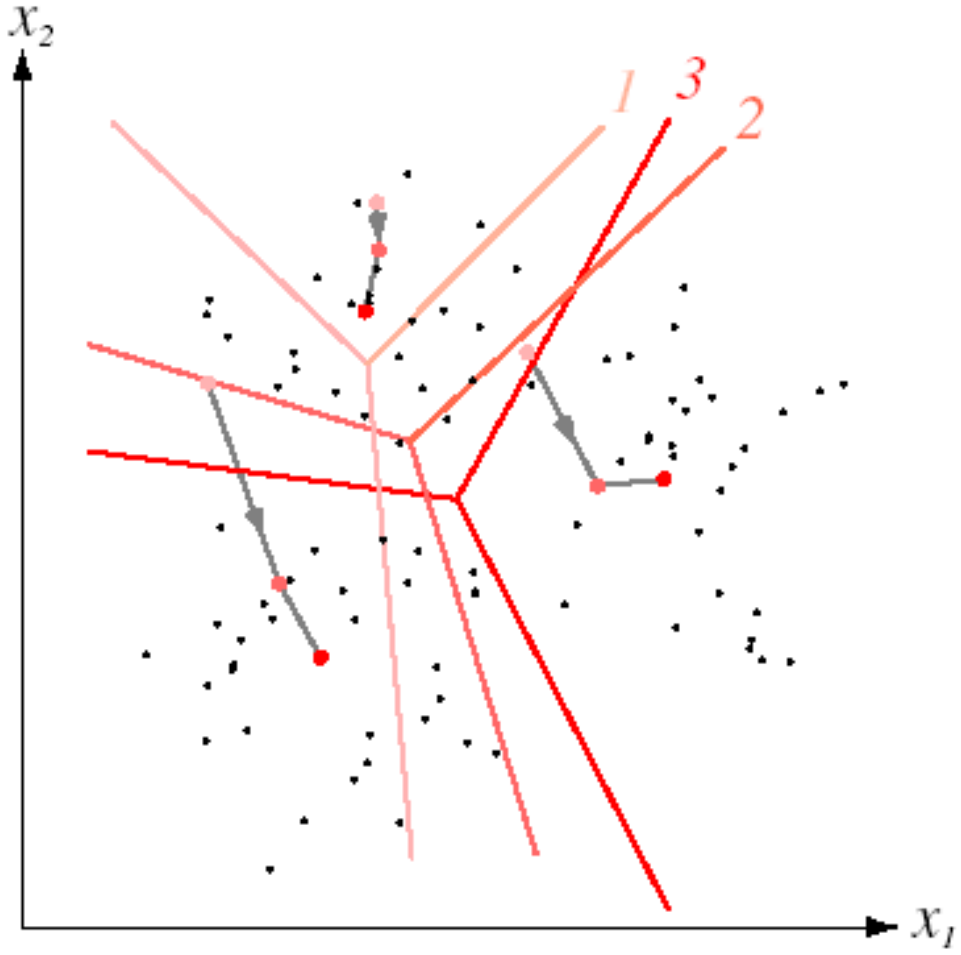


Figure from Duda et al.

Example

Unlabeled samples:

D={[7,7],[5,9],[6,9],[7,9],[5,11],[5,3],[6,1],[6,2],[7,1],[7,2],[7,3],[8,4],[8,6],[9,3],[9,4],[9,5],[10,4],[10,5],[10,6],[9,7]}^{t}

where $\mathbf{x} \in D, n = |D|$ and $k = 3$

Initialize $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$ randomly:

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 7 \\ 7 \end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix} 8 \\ 6 \end{bmatrix}, \boldsymbol{\mu}_3 = \begin{bmatrix} 8 \\ 4 \end{bmatrix}$$

Iteration 1:

- Assign $\mathbf{x}_1$, $\mathbf{x}_2$. …. $\mathbf{x}_n$ to the class of the nearest $\boldsymbol{\mu}_i$, using the Euclidean distance from $\mathbf{x}_j$ to $\boldsymbol{\mu}_i$ with j = 1,..,n and i=1,...,k

	Distance			Membership		
	μ_1	μ_2	μ_3	ω_1	ω_2	ω_3
$\mathbf{x}_1$	0	1.4	3.2	1	0	0
$\mathbf{x}_2$	2.8	4.2	5.8	1	0	0
$\mathbf{x}_3$	2.2	3.6	5.4	1	0	0
$\mathbf{x}_4$	2	3.2	5.1	1	0	0
$\mathbf{x}_5$	4.5	5.8	7.6	1	0	0
$\mathbf{x}_6$	4.5	4.2	3.2	0	0	1
$\mathbf{x}_7$	6.1	5.4	3.6	0	0	1
$\mathbf{x}_8$	5.1	4.5	2.8	0	0	1
$\mathbf{x}_9$	6	5.1	3.2	0	0	1
$\mathbf{x}_{10}$	5	4.1	2.2	0	0	1
$\mathbf{x}_{11}$	4	3.2	1.4	0	0	1
$\mathbf{x}_{12}$	3.2	2	0	0	0	1
$\mathbf{x}_{13}$	1.4	0	2	0	1	0
$\mathbf{x}_{14}$	4.5	3.2	1.4	0	0	1
$\mathbf{x}_{15}$	3.6	2.2	1	0	0	1
$\mathbf{x}_{16}$	2.8	1.4	1.4	0	1	0
$\mathbf{x}_{17}$	4.2	2.8	2	0	0	1
$\mathbf{x}_{18}$	3.6	2.2	2.2	0	1	0
$\mathbf{x}_{19}$	3.2	2	2.8	0	1	0
$\mathbf{x}_{20}$	2	1.4	3.2	0	1	0

Recompute $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix} 9.2 \\ 5.8 \end{bmatrix}, \boldsymbol{\mu}_3 = \begin{bmatrix} 7.4 \\ 2.7 \end{bmatrix}$$

Iteration 2:

- Assign $\mathbf{x}_1$, $\mathbf{x}_2$. …. $\mathbf{x}_n$ to the class of the nearest $\boldsymbol{\mu}_i$, using the Euclidean distance from $\mathbf{x}_j$ to $\boldsymbol{\mu}_i$ with j = 1,..,n and i=1,...,k

	Distance			Membership		
	μ_1	μ_2	μ_3	ω_1	ω_2	ω_3
x_1	2.2	2.5	4.3	1	0	0
x_2	1	5.3	6.7	1	0	0
x_3	0	4.5	6.5	1	0	0
x_4	1	3.9	6.3	1	0	0
x_5	2.2	6.7	8.6	1	0	0
x_6	6.1	5	2.4	0	0	1
x_7	8	5.8	2.2	0	0	1
x_8	7	5	1.6	0	0	1
x_9	8.1	5.3	1.7	0	0	1
x_{10}	7.1	4.4	0.8	0	0	1
x_{11}	6.1	3.6	0.5	0	0	1
x_{12}	5.4	2.2	1.4	0	0	1
x_{13}	3.6	1.2	3.4	0	1	0
x_{14}	6.7	2.8	1.6	0	0	1
x_{15}	5.8	1.8	2.1	0	1	0
x_{16}	5	0.8	2.8	0	1	0
x_{17}	6.4	2	2.9	0	1	0
x_{18}	5.7	1.1	3.5	0	1	0
x_{19}	5	0.8	4.2	0	1	0
x_{20}	3.6	1.2	4.6	0	1	0

Recompute: $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix} 9.28 \\ 5.29 \end{bmatrix}, \boldsymbol{\mu}_3 = \begin{bmatrix} 6.88 \\ 2.38 \end{bmatrix}$$

Iteration 3:

- Assign $\mathbf{x}_1$, $\mathbf{x}_2$. …. $\mathbf{x}_n$ to the class of the nearest $\boldsymbol{\mu}_i$, using the Euclidean distance from $\mathbf{x}_j$ to $\boldsymbol{\mu}_i$ with j = 1,..,n and i=1,...,k

	Distance			Membership		
	μ_1	μ_2	μ_3	ω_1	ω_2	ω_3
x_1	2.2	2.9	4.6	1	0	0
x_2	1	5.7	6.9	1	0	0
x_3	0	5	6.7	1	0	0
x_4	1	4.4	6.6	1	0	0
x_5	2.2	7.1	8.8	1	0	0
x_6	6.1	4.9	2	0	0	1
x_7	8	5.4	1.6	0	0	1
x_8	7	4.6	1	0	0	1
x_9	8.1	4.9	1.4	0	0	1
x_{10}	7.1	4	0.4	0	0	1
x_{11}	6.1	3.2	0.6	0	0	1
x_{12}	5.4	1.8	2	0	1	0
x_{13}	3.6	1.5	3.8	0	1	0
x_{14}	6.7	2.3	2.2	0	0	1
x_{15}	5.8	1.3	2.7	0	1	0
x_{16}	5	0.4	3.4	0	1	0
x_{17}	6.4	1.5	3.5	0	1	0
x_{18}	5.7	0.8	4.1	0	1	0
x_{19}	5	1	4.8	0	1	0
x_{20}	3.6	1.7	5.1	0	1	0

Recompute: $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix} 9.13 \\ 5.13 \end{bmatrix}, \boldsymbol{\mu}_3 = \begin{bmatrix} 6.71 \\ 2.14 \end{bmatrix}$$

Iteration 4:

- Assign $\mathbf{x}_1$, $\mathbf{x}_2$. …. $\mathbf{x}_n$ to the class of the nearest $\boldsymbol{\mu}_i$, using the Euclidean distance from $\mathbf{x}_j$ to $\boldsymbol{\mu}_i$ with $j = 1,..,n$ and $i=1,...,k$

	Distance			Membership		
	μ_1	μ_2	μ_3	ω_1	ω_2	ω_3
$\mathbf{x}_1$	2.24	2.83	4.87	1	0	0
$\mathbf{x}_2$	1	5.66	7.07	1	0	0
$\mathbf{x}_3$	0	4.98	6.89	1	0	0
$\mathbf{x}_4$	1	4.42	6.86	1	0	0
$\mathbf{x}_5$	2.24	7.18	9.02	1	0	0
$\mathbf{x}_6$	6.08	4.64	1.92	0	0	1
$\mathbf{x}_7$	8.00	5.18	1.35	0	0	1
$\mathbf{x}_8$	7.00	4.42	0.73	0	0	1
$\mathbf{x}_9$	8.06	4.64	1.18	0	0	1
$\mathbf{x}_{10}$	7.07	3.78	0.32	0	0	1
$\mathbf{x}_{11}$	6.08	3.01	0.90	0	0	1
$\mathbf{x}_{12}$	5.39	1.59	2.26	0	1	0
$\mathbf{x}_{13}$	3.61	1.43	4.07	0	1	0
$\mathbf{x}_{14}$	6.71	2.13	2.44	0	1	0
$\mathbf{x}_{15}$	5.83	1.13	2.95	0	1	0
$\mathbf{x}_{16}$	5	0.18	3.66	0	1	0
$\mathbf{x}_{17}$	6.40	1.43	3.77	0	1	0
$\mathbf{x}_{18}$	5.66	0.88	4.35	0	1	0
$\mathbf{x}_{19}$	5	1.24	5.07	0	1	0
x_{20}	3.61	1.88	5.37	0	1	0

Recompute: $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix} 9.11 \\ 4.89 \end{bmatrix}, \boldsymbol{\mu}_3 = \begin{bmatrix} 6.33 \\ 2 \end{bmatrix}$$

Iteration 5:

- Assign $\mathbf{x}_1$, $\mathbf{x}_2$. …. $\mathbf{x}_n$ to the class of the nearest $\boldsymbol{\mu}_i$, using the Euclidean distance from $\mathbf{x}_j$ to $\boldsymbol{\mu}_i$ with j = 1,..,n and i=1,...,k

	Distance			Membership		
	μ_1	μ_2	μ_3	ω_1	ω_2	ω_3
x_1	2.2	3	5	1	0	0
x_2	1	5.8	7.1	1	0	0
x_3	0	5.2	7	1	0	0
x_4	1	4.6	7	1	0	0
x_5	2.2	7.4	9.1	1	0	0
x_6	6.1	4.5	1.7	0	0	1
x_7	8	5	1.1	0	0	1
x_8	7	4.2	0.3	0	0	1
x_9	8.1	4.4	1.2	0	0	1
x_{10}	7.1	3.6	0.7	0	0	1
x_{11}	6.1	2.8	1.2	0	0	1
x_{12}	5.4	1.4	2.6	0	1	0
x_{13}	3.6	1.6	4.3	0	1	0
x_{14}	6.7	1.9	2.8	0	1	0
x_{15}	5.8	0.9	3.3	0	1	0
x_{16}	5	0.2	4	0	1	0
x_{17}	6.4	1.3	4.2	0	1	0
x_{18}	5.7	0.9	4.7	0	1	0
x_{19}	5	1.4	5.4	0	1	0
x_{20}	3.6	2.1	5.7	0	1	0

- Recompute: $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3$

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \boldsymbol{\mu}_2 = \begin{bmatrix} 9.11 \\ 4.89 \end{bmatrix}, \boldsymbol{\mu}_3 = \begin{bmatrix} 6.33 \\ 2 \end{bmatrix}$$

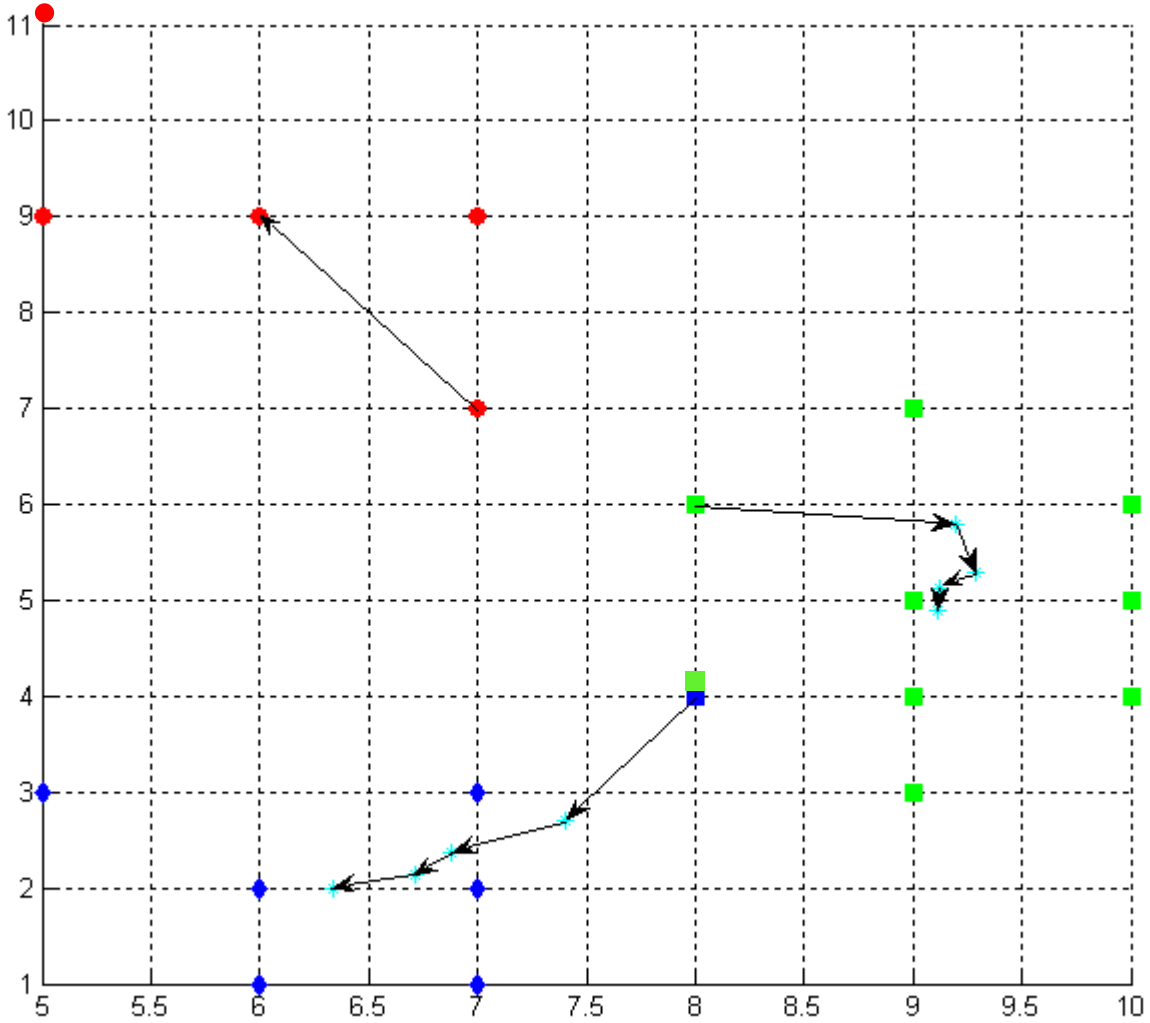
Iteration 5 does not change the values of $\boldsymbol{\mu}_i$

- Then the membership matrix does not change either

The resulting membership matrix:

	ω_1	ω_2	ω_3
$\mathbf{x}_1$	1	0	0
$\mathbf{x}_2$	1	0	0
$\mathbf{x}_3$	1	0	0
$\mathbf{x}_4$	1	0	0
$\mathbf{x}_5$	1	0	0
$\mathbf{x}_6$	0	0	1
$\mathbf{x}_7$	0	0	1
$\mathbf{x}_8$	0	0	1
$\mathbf{x}_9$	0	0	1
$\mathbf{x}_{10}$	0	0	1
$\mathbf{x}_{11}$	0	0	1
$\mathbf{x}_{12}$	0	1	0
$\mathbf{x}_{13}$	0	1	0
$\mathbf{x}_{14}$	0	1	0
$\mathbf{x}_{15}$	0	1	0
$\mathbf{x}_{16}$	0	1	0
$\mathbf{x}_{17}$	0	1	0
$\mathbf{x}_{18}$	0	1	0
$\mathbf{x}_{19}$	0	1	0
$\mathbf{x}_{20}$	0	1	0

Graphically



Fuzzy *k*-Means Clustering

Consider this as an *unsupervised* learning scenario:

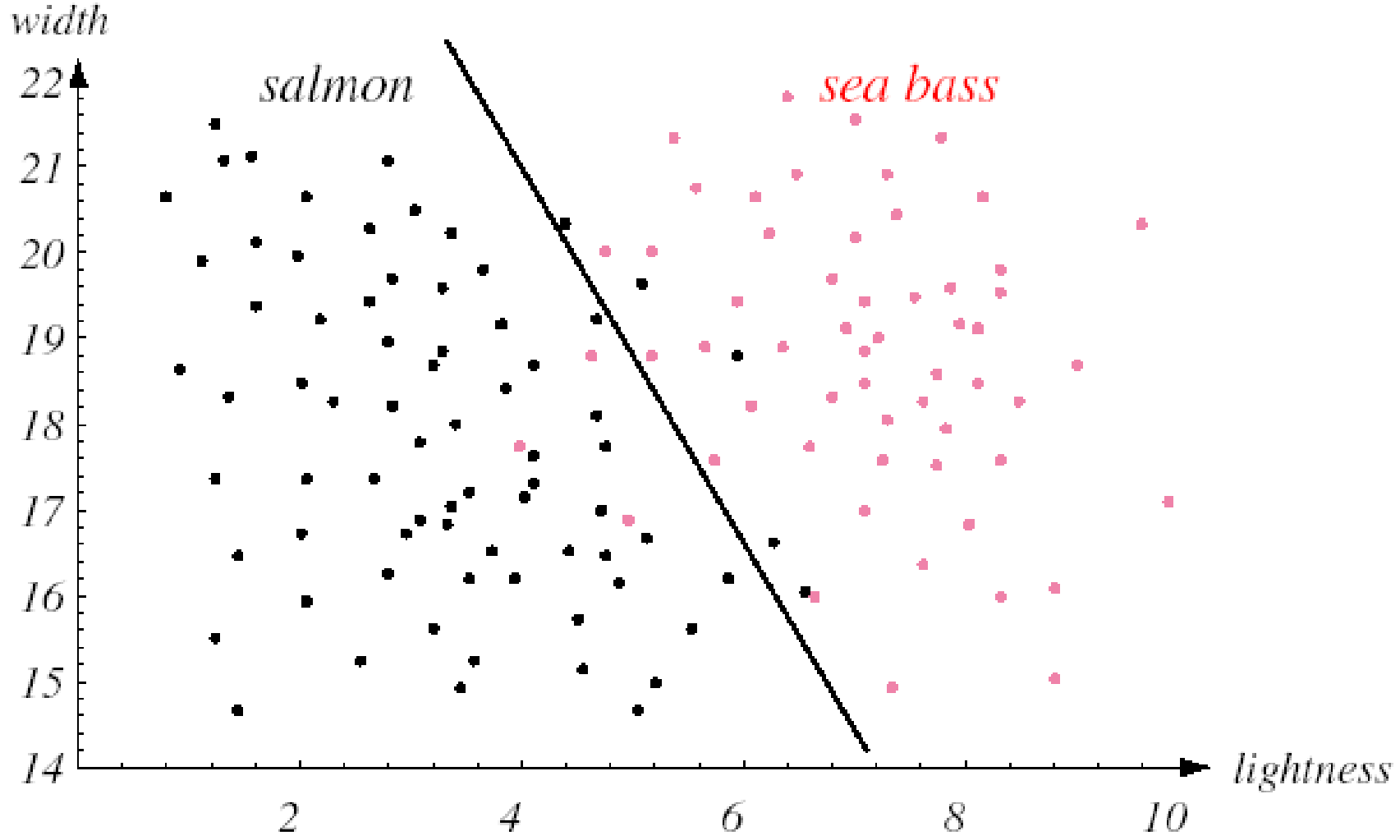


Figure from Duda et al.

- 2 classes: *sea bass* and *salmon*
- It would then be *natural* to consider *two* clusters
- Assume *k*-Means is used
- Two clusters are obtained
- but…
  - some **red** points are assigned to *salmon*, while
  - some **black** points are assigned to *sea bass*.
- Then…
  - Would it be right to have this "strict" definition of membership?
- What if our membership function were *fuzzy* ?

- The “new” definition of membership is equivalent to

$$\hat{P}(\omega_i \mid \mathbf{x}_k, \hat{\boldsymbol{\theta}})$$

as in (7), where $\hat{\boldsymbol{\theta}}$ is the parameter vector for the membership functions

Fuzzy *k*-Means seeks a *minimum* of the following heuristic global cost function:

$$J_{fuz} = \sum_{i=1}^{k} \sum_{j=1}^{n} \left[\hat{P}(\omega_i \mid \mathbf{x}_j, \hat{\boldsymbol{\theta}})\right]^b \left\|\mathbf{x}_j - \boldsymbol{\mu}_i\right\|^2 \qquad (11)$$

where *b* is a parameter chosen to adjust the “blending” or “overlapping” of the clusters

Notes on *b*

- *b* set to 0 means that
  - J_{fuz} is the sum-of-squared errors criterion,
  - where each $\mathbf{x}_j$ assigned to a different ω_i
- Typical values: $b > 1$, which means that
  - each $\mathbf{x}_j$ belongs to multiple clusters

- How can we then minimize J_{fuz} ?

- Suppose the memberships for each $\mathbf{x}_j$ are normalized:

$$\sum_{i=1}^{k} \hat{P}(\omega_i \mid \mathbf{x}_j) = 1, \quad j = 1, \ldots, n \quad (12)$$

  where there is an *implicit* dependence on $\hat{\boldsymbol{\theta}}$

- Also, let $\hat{P}_j$ denote the *prior* probability of ω_j ,
- To minimize J_{fuz}, the following necessary conditions: $\hat{P}(\omega_j)$

$$\partial J_{fuz} / \partial \boldsymbol{\mu}_i = 0, \quad \text{and}$$

$$\partial J_{fuz} / \partial \hat{P}_j = 0$$

- This leads to the following solutions:

$$\boldsymbol{\mu}_i = \frac{\sum_{j=1}^{n} \left[\hat{P}(\omega_i \mid \mathbf{x}_j)\right]^b \mathbf{x}_j}{\sum_{j=1}^{n} \left[\hat{P}(\omega_i \mid \mathbf{x}_j)\right]^b} \qquad (13)$$

and

$$\hat{P}(\omega_i \mid \mathbf{x}_j) = \frac{\left(1/d_{ij}\right)^{1/(b-1)}}{\sum_{l=1}^{k} \left(1/d_{lj}\right)^{1/(b-1)}} \qquad (14)$$

where $d_{ij} = \left\|\mathbf{x}_j - \boldsymbol{\mu}_i\right\|^2$ (squared Euclidean distance)

- In general, (13) and (14) do not have any analytical solution
- Then, we use an *iterative* updating rule…
- as follows…

Algorithm **Fuzzy *k*-Means**

Input: Number of clusters, k

A training dataset, $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

A blending factor, b

begin

Initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$, $\hat{P}(\omega_i \mid \mathbf{x}_j)$

where $i = 1, \ldots, k$ and $j = 1, \ldots, n$

repeat

Re-compute $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ by (13)

Re-compute $\hat{P}(\omega_i \mid \mathbf{x}_j)$ by (14), for all i, j

until *small* change in $\boldsymbol{\mu}_i$

return $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$

Notes on Fuzzy *k*-Means

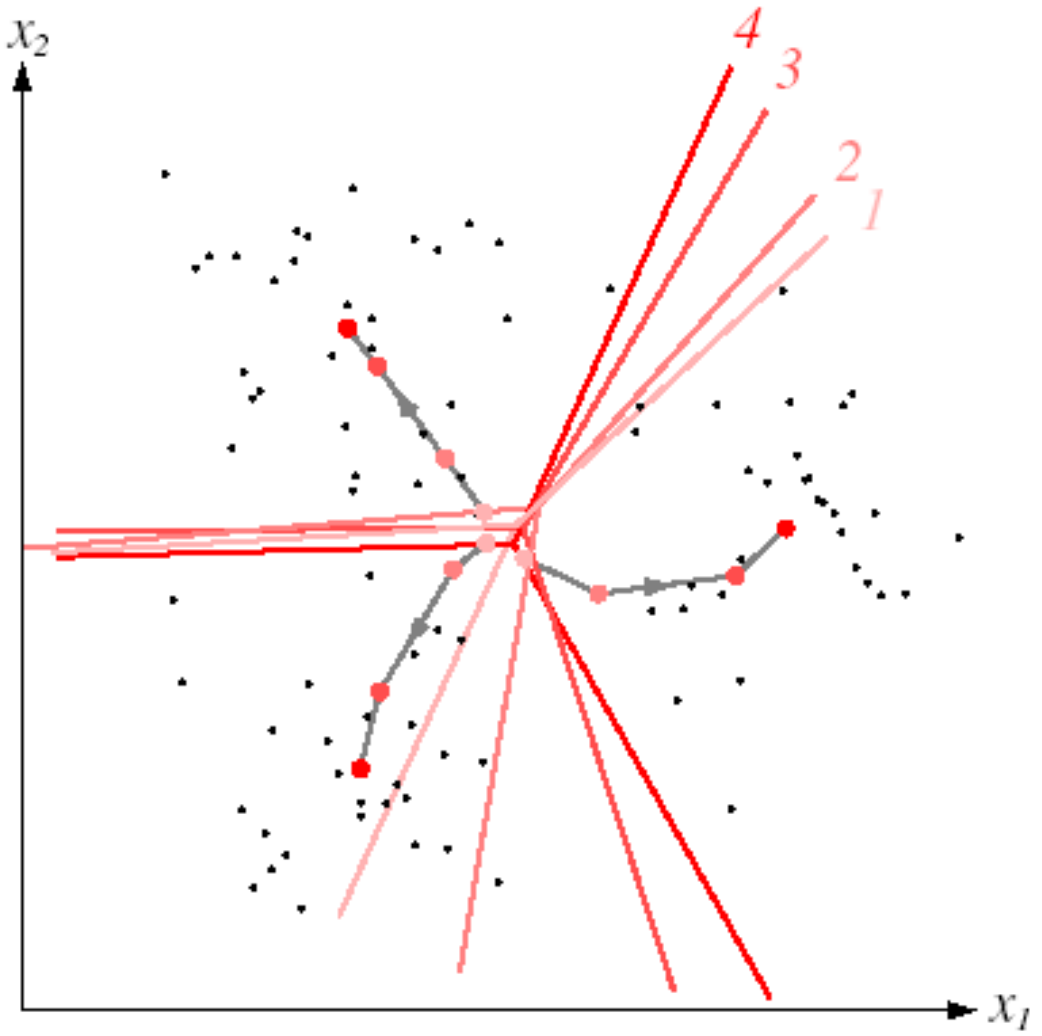
- J_{fuz} is minimized when cluster centers ($\boldsymbol{\mu}_i$'s)

  are close to the points that are estimated to more likely belong to class ω_i
- Considering *fuzzy* memberships usually improves convergence over *k*-Means

  but *not in all cases*
- *k*-Means can be seen as a *particular* case of Fuzzy *k*-Means, where:

$$\hat{P}(\omega_i \mid \mathbf{x}_j) = \begin{cases} 1 \text{ if } \left\|\mathbf{x}_j - \boldsymbol{\mu}_i\right\| < \left\|\mathbf{x}_j - \boldsymbol{\mu}_{i'}\right\| \quad \text{for all } i' \neq i \\ 0 \text{ otherwise} \end{cases}$$

Example: 3 clusters, 2D, $b = 2$



Fuzzy *k*-means usually needs more steps to converge (than *k*-means)

Figure from Duda et al.

Implementation of Fuzzy *k*-Means

- A typical implementation of Fuzzy *k*-Means maintains a $k \times n$ "membership matrix" **M**, where m_{ij} contains the probability of ω_i given $\mathbf{x}_j$ as in (14)

This implies:

- $O(kn)$ space
  - Think about a dataset of 1M samples and 20 clusters
- $O(kdn)$ time for each iteration
  assuming $\sum_{l=1}^{k} \left(1/d_{lj}\right)^{1/(b-1)}$ is computed once
- Also, k vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ are stored, and
  updated at each iteration, as in (13)
  - $O(kdn)$ time and space for each iteration

Example

3 clusters (classes), 6 samples

M:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$
ω_1	.60	.75	.04	.06	.02	.02
ω_2	.15	.10	.92	.89	.03	.01
ω_3	.25	.15	.04	.05	.95	.97

$$\boldsymbol{\mu} = \begin{bmatrix} \begin{bmatrix} \mu_{11} \\ \vdots \\ \mu_{1d} \end{bmatrix} \begin{bmatrix} \mu_{21} \\ \vdots \\ \mu_{2d} \end{bmatrix} \begin{bmatrix} \mu_{31} \\ \vdots \\ \mu_{3d} \end{bmatrix} \end{bmatrix}$$

- How to assign the samples to each cluster?
  - Pick the largest $\hat{P}(\omega_i \mid \mathbf{x}_j)$
  - Can use a threshold if values are too small

Note:

- In *k*-Means, **M** is a *binary* matrix
- Usually, *not stored* as it is *not used*

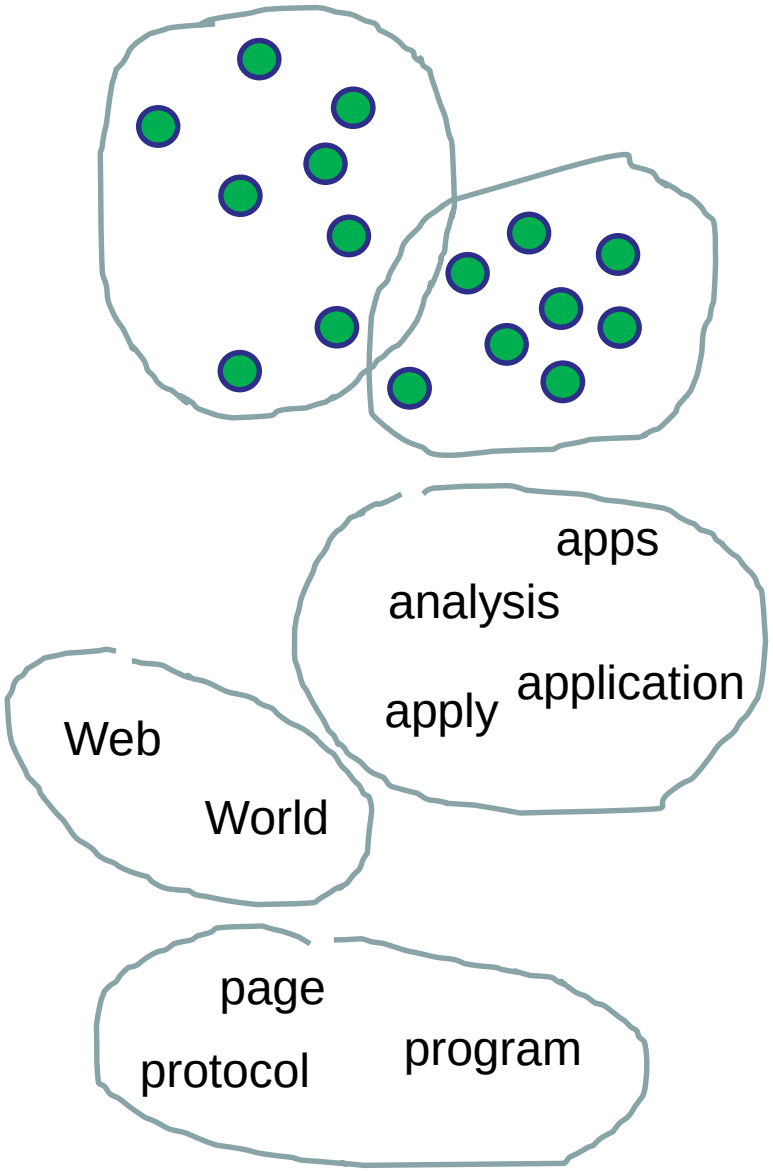
Hierarchical Clustering

Aim:

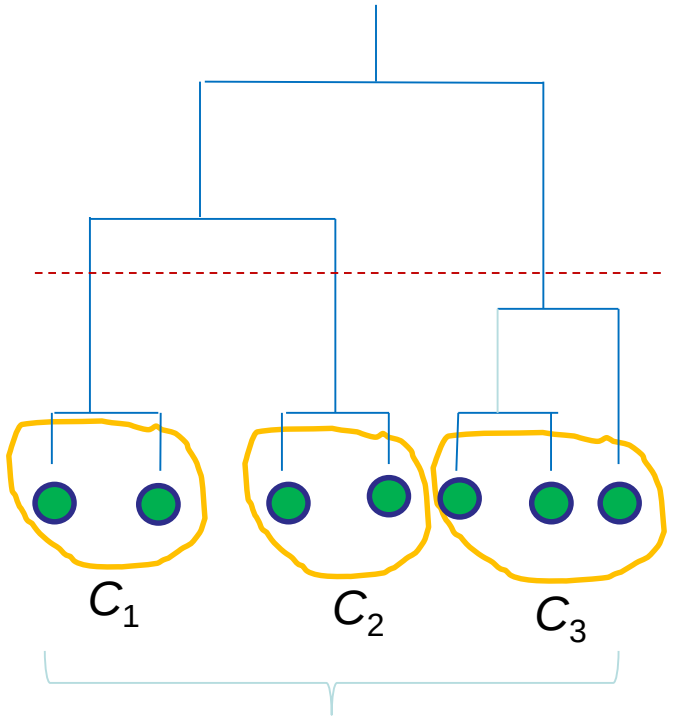
- Group data into classes, subclasses, sub-subclasses, and so on…
- Hierarchical clustering does not produce a “flat” clustering,
  - but a “hierarchy” of clusters
- Two kinds of approaches:
  - Agglomerative (bottom-up)
  - Divisive (top-down)

Flat vs Hierarchical clustering

Flat:



Hierarchical:



Cutoff used to
obtain 3 clusters

Hierarchical Clustering Representations

- Dendrogram: A tree in which
  - ➢ the root represents a *single* cluster (*n* samples),
  - ➢ the leaves represent *n* clusters (one sample)

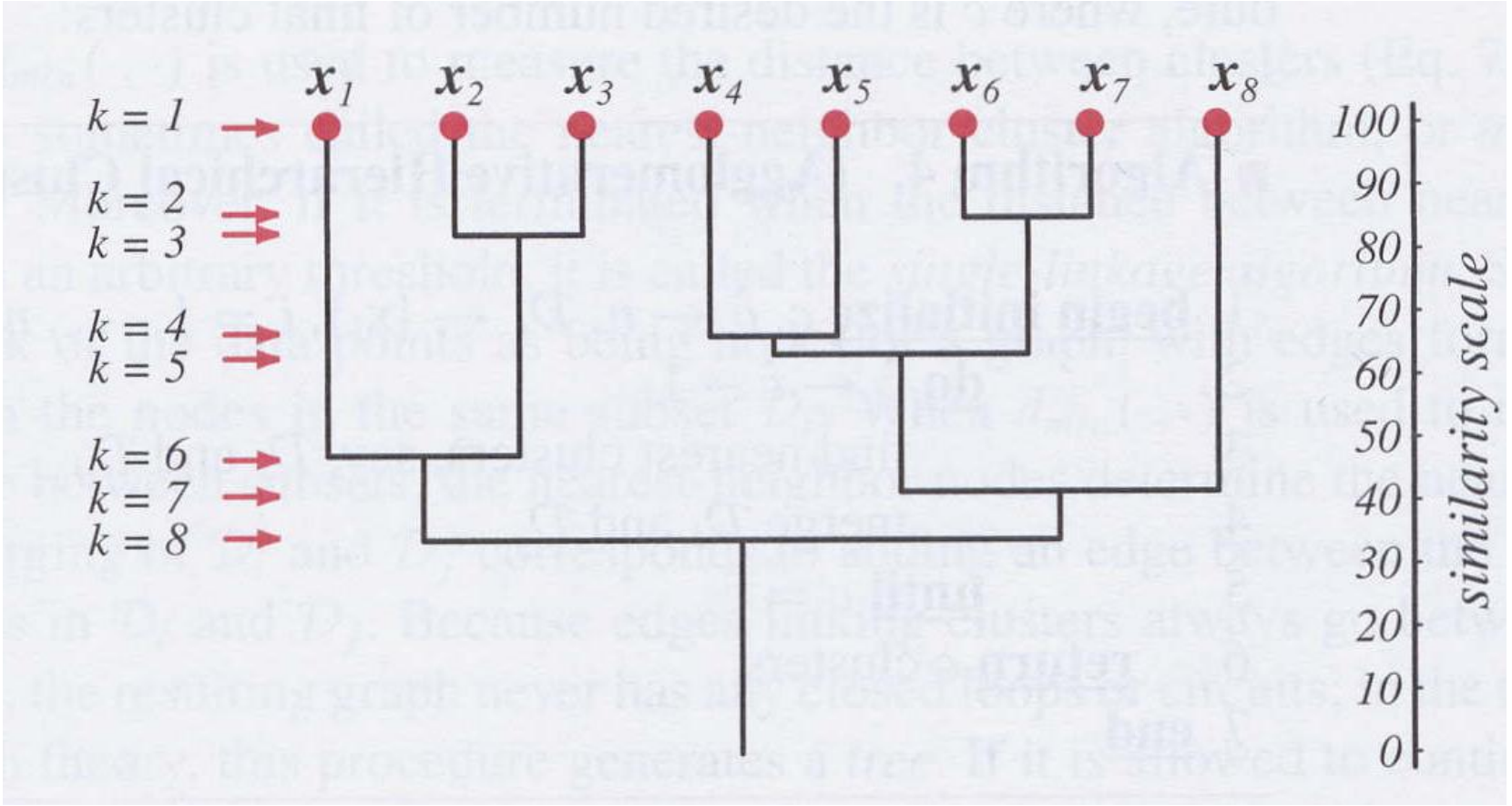


Figure from Duda et al.

Hierarchical Clustering Representations

- Venn diagram (set representation)
- Recursive definition: One set (cluster) contains
  - *two* sets (clusters), or
  - a *single* sample

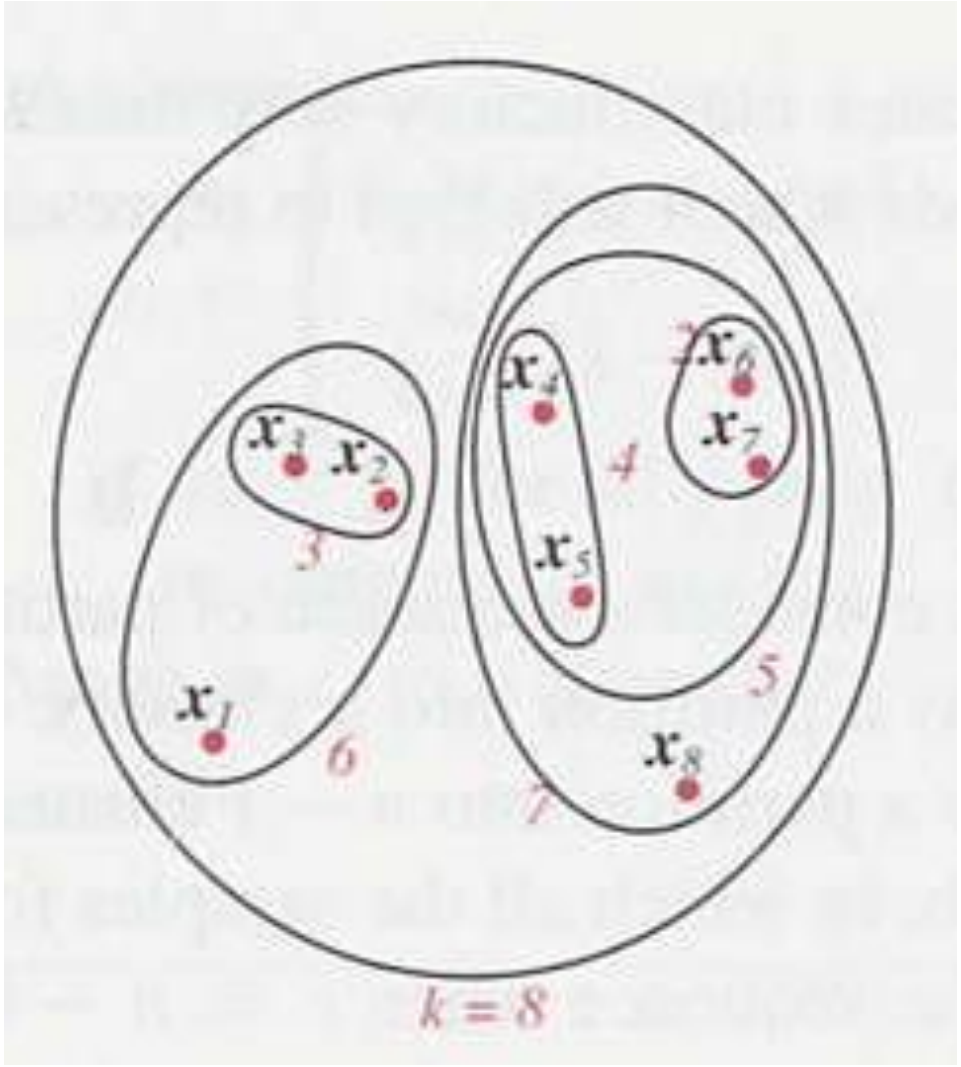


Figure from Duda et al.

Agglomerative (bottom-up)

Algorithm **Agglomerative Hierarchical Clustering**

Input: Desired number of clusters, k

A training dataset, $D = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$

begin

Initialize $\hat{k} \leftarrow n$, $D_i = \{\mathbf{x}_i\}$, $i = 1, \ldots, n$

repeat

$\hat{k} \leftarrow \hat{k} - 1$

find *most similar* clusters, say D_i and D_j

merge D_i and D_j into a single cluster

until $\hat{k} = k$

return k clusters

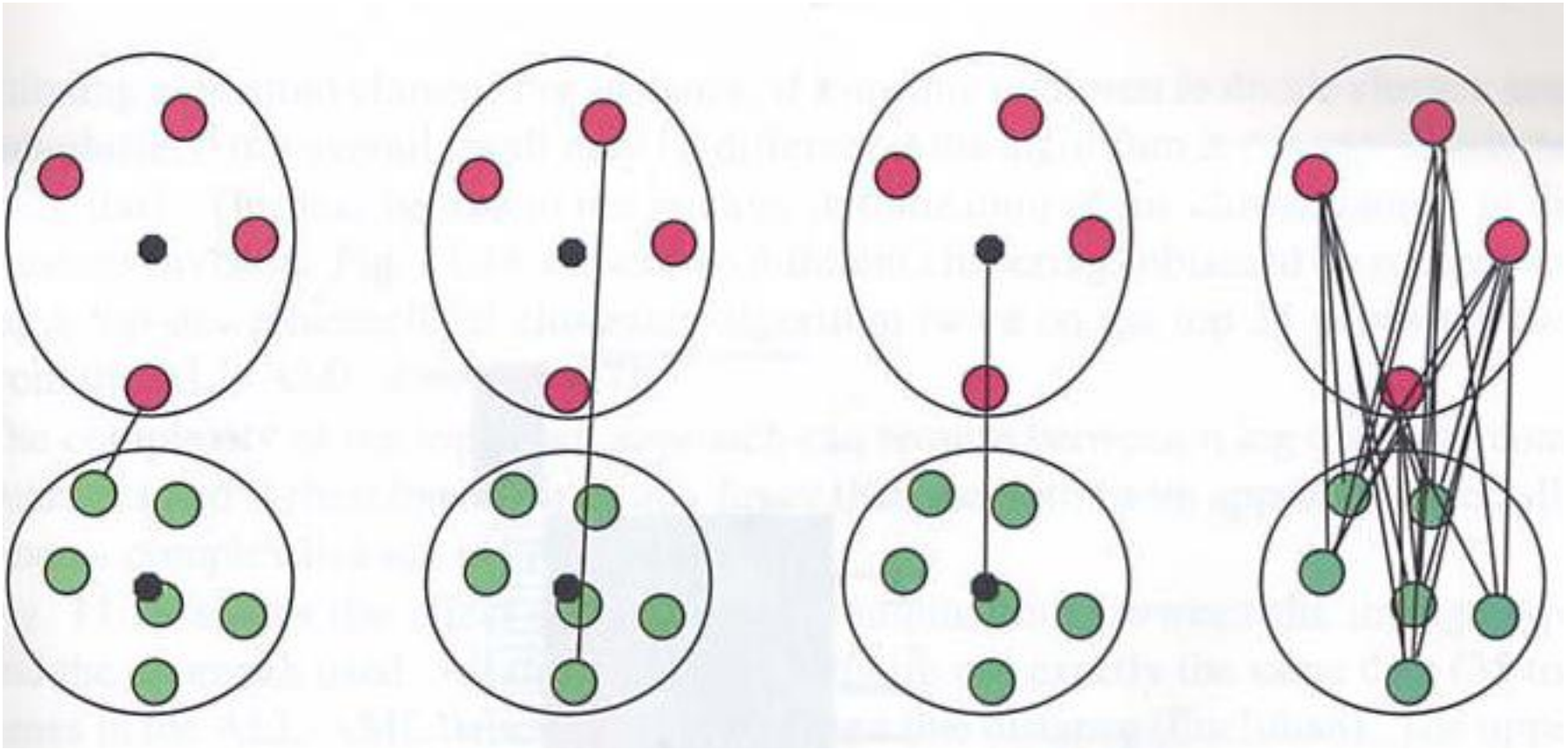
Inter-cluster Distance

- What is the meaning of “nearest”?
- Inter-cluster distances:
  - Minimum (nearest-neighbor)
    - Single linkage or shortest distance
  - Maximum (farthest-neighbor)
    - Complete linkage or longest distance
  - Average distance
    - Average linkage
  - Mean distance
    - Centroid linkage
  - … many more
- In general:
  - Using a **different** distance function gives a **different** clustering algorithm

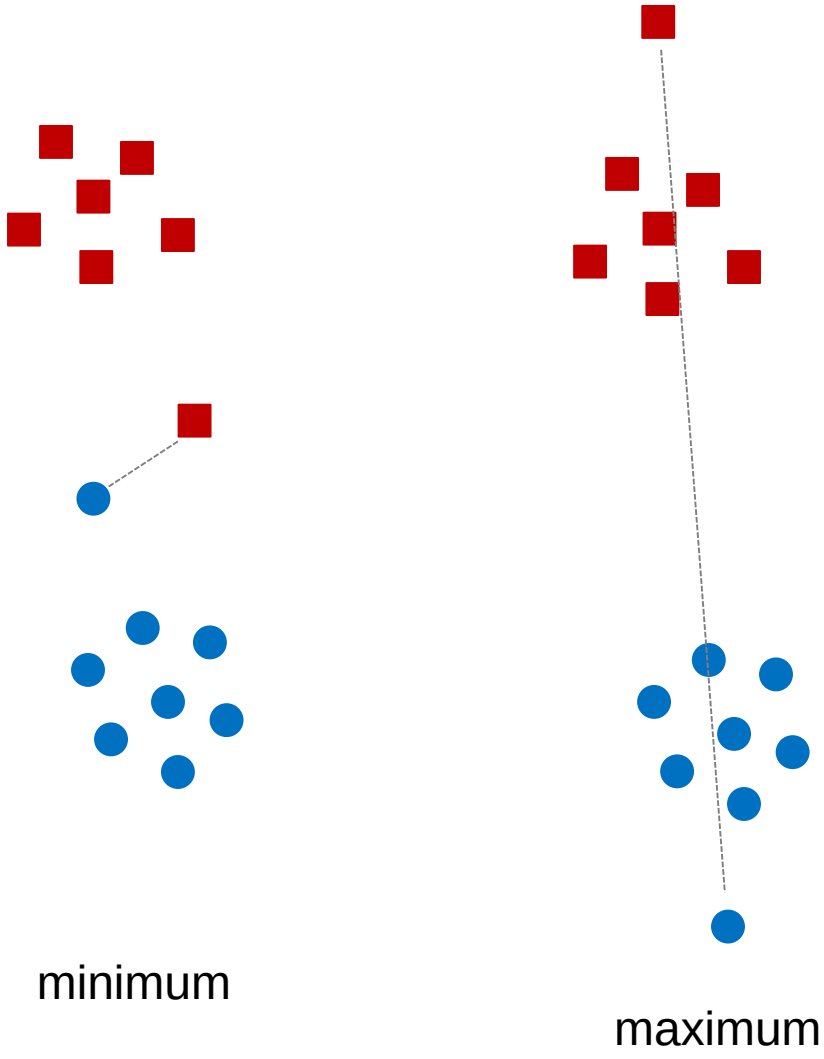
- Each corresponds to the following distance functions:
- Minimum: $d_{min}(D_i, D_j) = \min_{\substack{\mathbf{x} \in D_i \\ \mathbf{x}' \in D_j}} \|\mathbf{x} - \mathbf{x}'\|$
- Maximum: $d_{max}(D_i, D_j) = \max_{\substack{\mathbf{x} \in D_i \\ \mathbf{x}' \in D_j}} \|\mathbf{x} - \mathbf{x}'\|$
- Average: $d_{avg}(D_i, D_j) = \frac{1}{n_i n_j} \sum_{\mathbf{x} \in D_i} \sum_{\mathbf{x}' \in D_j} \|\mathbf{x} - \mathbf{x}'\|$
- Mean: $d_{mean}(D_i, D_j) = \|\mathbf{m}_i - \mathbf{m}_j\|$

Example

Minimum Maximum Mean Average



Example 2: minimum vs maximum



minimum

maximum

May under or over estimate the distance between two clusters

Divisive (top-down)

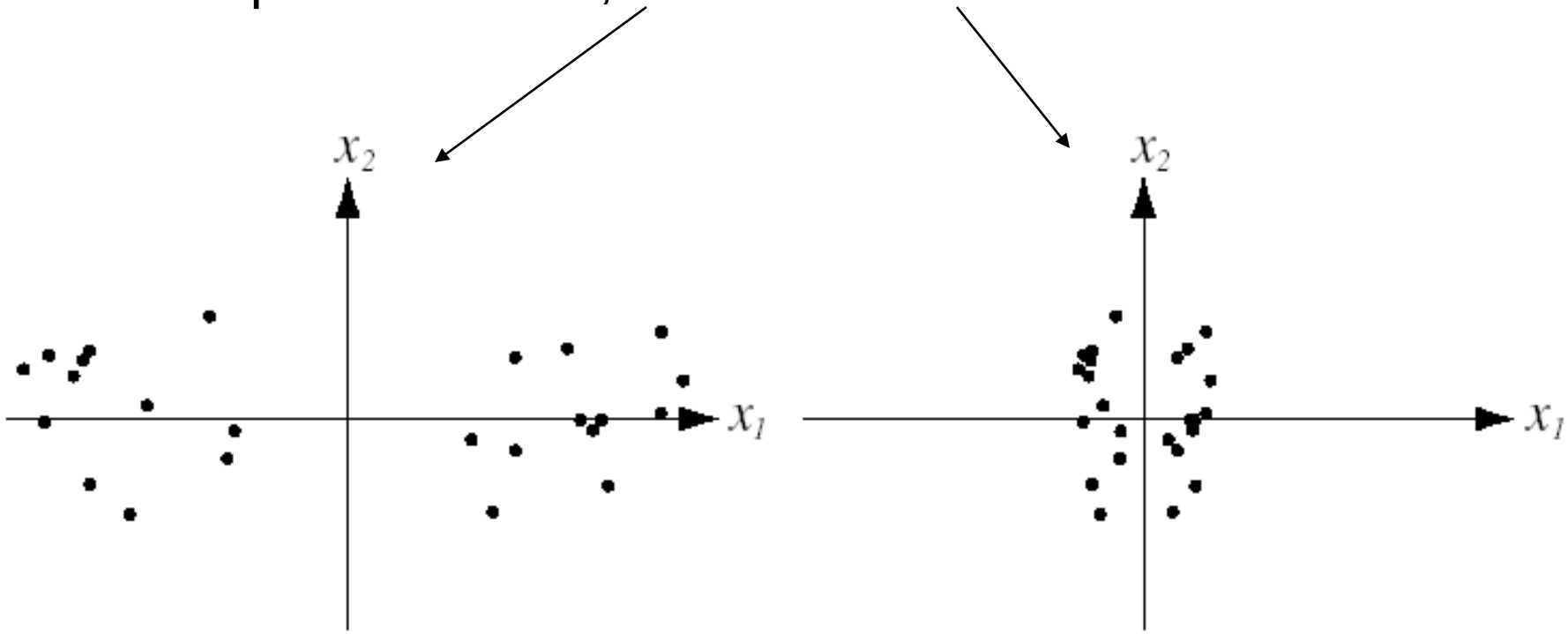
- Follows a top-down strategy
- Different approaches:
  - Graph-theoretic:
    - minimum spanning tree
    - graph connectivity (e.g., NetScan)
    - spectral clustering
    - other approaches exist
  - Hierarchically apply flat clustering methods:
    - *k*-Means on whole data, and then
    - recursively cluster clusters, and so on

Normalization & Clustering

- Using Euclidean distance,
  - features whose values spread out over a **large** range are *dominant* in computing the distance.
- To remedy this situation:
  - May want to “normalize” the data
- One way:
  - Rotate the axes so they coincide with the eigenvectors of the *scatter* matrix (a rotation transformation)
  - “Scale” the features so that covariance = **I** (whitening transformation)
  - These two transformations are known as *Diagonalization*
  - Generalizations of this: PCA, ICA, Kernel PCA, MDS
- Though this normalization may not be good for clustering

Simple normalization

- Simple normalization: convert each variable to a $N(0,1)$
- Example: 2 clusters, before and after normalization



Another issue:

- Are features of “real” objects correctly measured?

How good a clustering is?... Validity Indices

- Two questions are in place:
  - How good is the resulting clustering?
  - What is the correct number of clusters?
- Quite difficult to answer…
- However, we do have some answers…

  in terms of *validity indices*
- There are quite a few… many, indeed

  … five of them discussed here…
- See ref. [2]
- Also, ref. [3] has a comprehensive coverage of indices

good clustering vs bad clustering

- More compact clusters vs
- Clusters well separated from each other

Davies-Bouldin (DB) Index

- The scatter of the *i*th cluster is defined as:

$$S_i = \frac{1}{|D_i|} \sum_{x \in D_i} \left\{ \| x - \mu_i \| \right\}$$

  μ_i is the cluster center

- The distance between clusters D_i and D_j is:

$$d_{ij} = \| \mu_i - \mu_j \|$$

- The *DB* index aims to **minimize**:

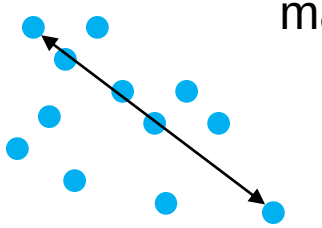
$$DB = \frac{1}{k} \sum_{i=1}^{k} R_i$$

where $$R_i = \max_{j, j \neq i} \left\{ \frac{S_i + S_j}{d_{ij}} \right\}$$

Dunn's Index

- The *diameter* of D_i defined as:

$$\Delta(D_i) = \max_{x,y \in D_i} \{d(x,y)\}$$



- The distance between D_i and D_j defined as:

$$\delta(D_i, D_j) = \min_{x \in D_i, y \in D_j} \{d(x,y)\}$$

- *Dunn's* index aims to **maximize**:

$$\upsilon_d = \min_{1 \le i \le k} \left\{ \min_{1 \le j \le k, j \ne i} \left\{ \frac{\delta(D_i, D_j)}{\max_{1 \le r \le k} \{\Delta(D_r)\}} \right\} \right\}$$

$d(x,y)$ is an arbitrary distance function (e.g. Euclidean)

Calinski-Harabasz (CH) Index

- Let the trace of $\mathbf{S}_B$ be computed as follows:.

$$\mathbf{S}_B = \sum_{i=1}^{k} |D_i| \|\mu - \mu_i\|^2$$

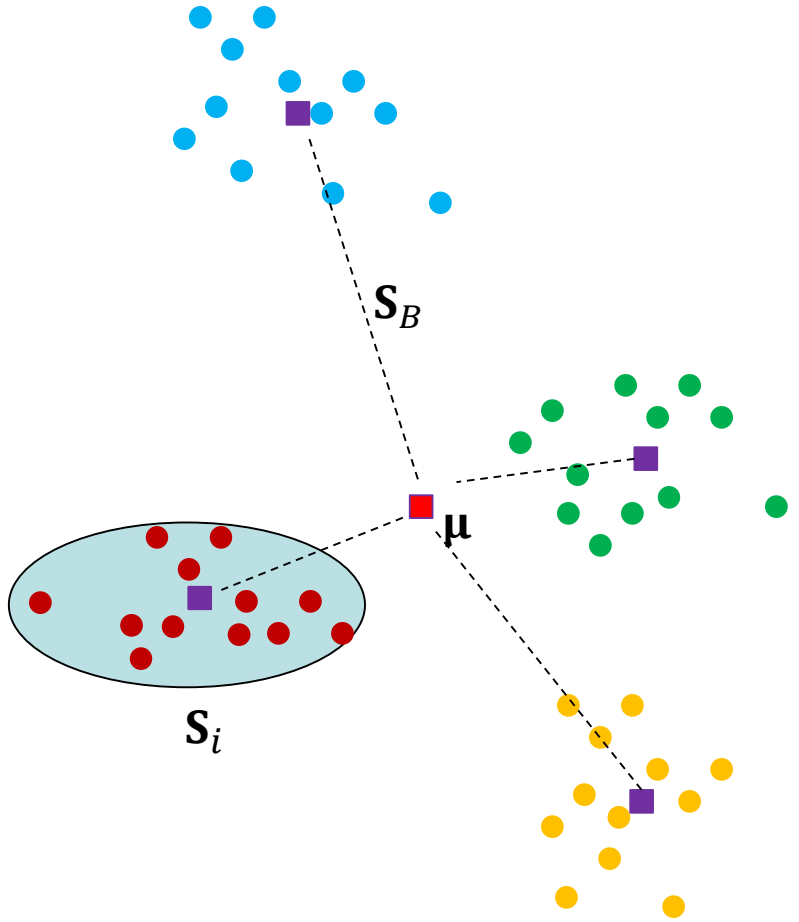
and the trace of $\mathbf{S}_W$ be computed as follows:

$$\mathbf{S}_W = \sum_{i=1}^{k} \sum_{j=1}^{|D_i|} \|x_j - \mu_i\|^2$$

- $\mathbf{S}_B$ and $\mathbf{S}_W$ are the *between* and *within* cluster scatter matrices
- The *CH* index aims to **maximize**:

$$CH = \frac{\mathbf{S}_B/(k-1)}{\mathbf{S}_W/(n-k)}$$

- μ is the mean of the means
- μ_i is the mean of cluster D_i



Index I

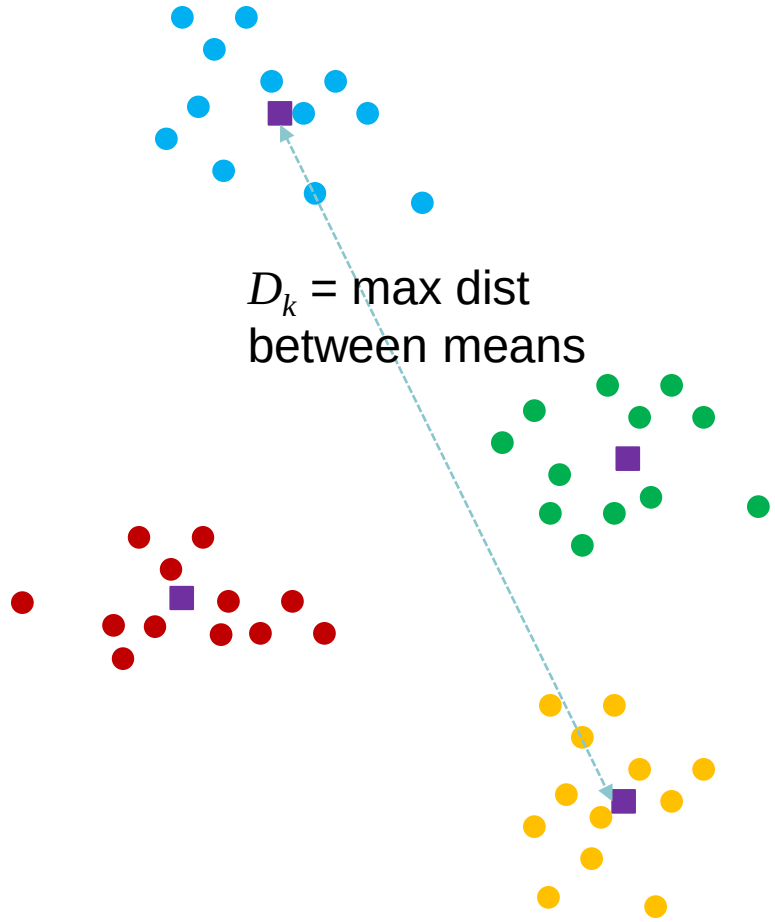
- It also applies to fuzzy *k*-means
- Aim is to **maximize**:

$$I(k) = \left\{ \frac{1}{k} \frac{E_1}{E_k} D_k \right\}^p$$

where:

$$E_k = \sum_{i=1}^{k} \sum_{j=1}^{|D_i|} u_{ij} \left\| x_j - \mu_i \right\|$$

$$D_k = \max_{i,j=1}^{k} \left\| \mu_i - \mu_j \right\|$$



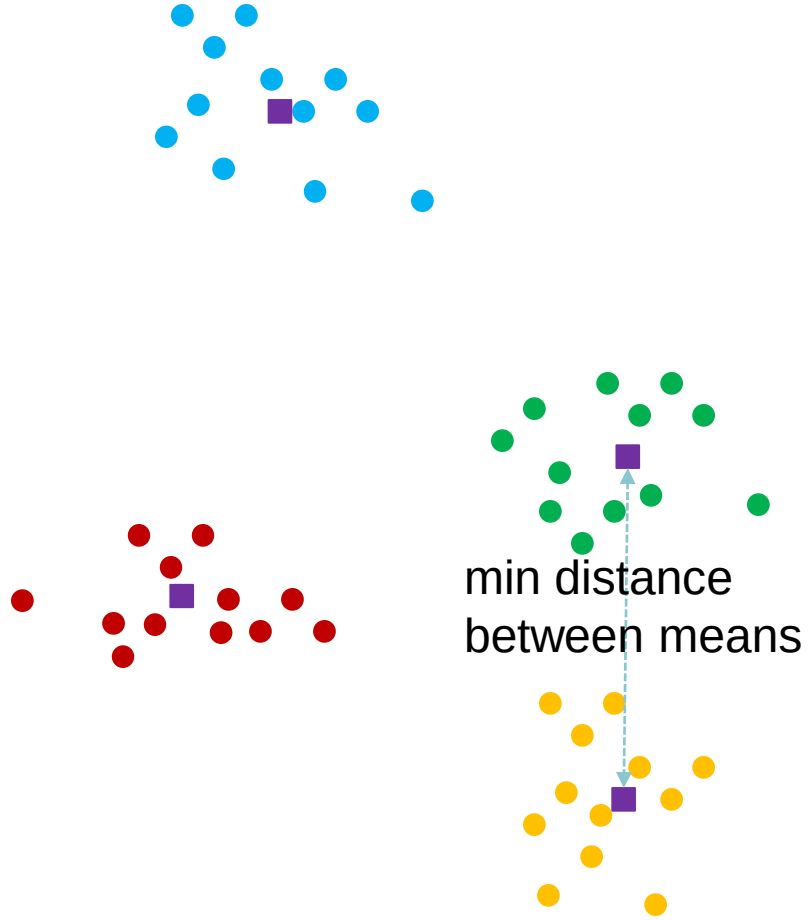
- u_{ij} is the membership of x_j to cluster D_i
- A typical value for p is 2
- E_1 computed as E_k, taking *all* samples in a *single* cluster

Xie-Beni Index

- It also applies to fuzzy *k*-means
- Aim is to **minimize**:

$$XB = \frac{\sum_{i=1}^{k}\sum_{j=1}^{n} u_{ij}^{2}\left\|x_j - \mu_i\right\|^2}{n \min_{i,j}\left\{\left\|\mu_i - \mu_j\right\|^2\right\}}$$

- u_{ij} is the membership of x_j to cluster D_i
- μ_i is the mean of cluster *i*



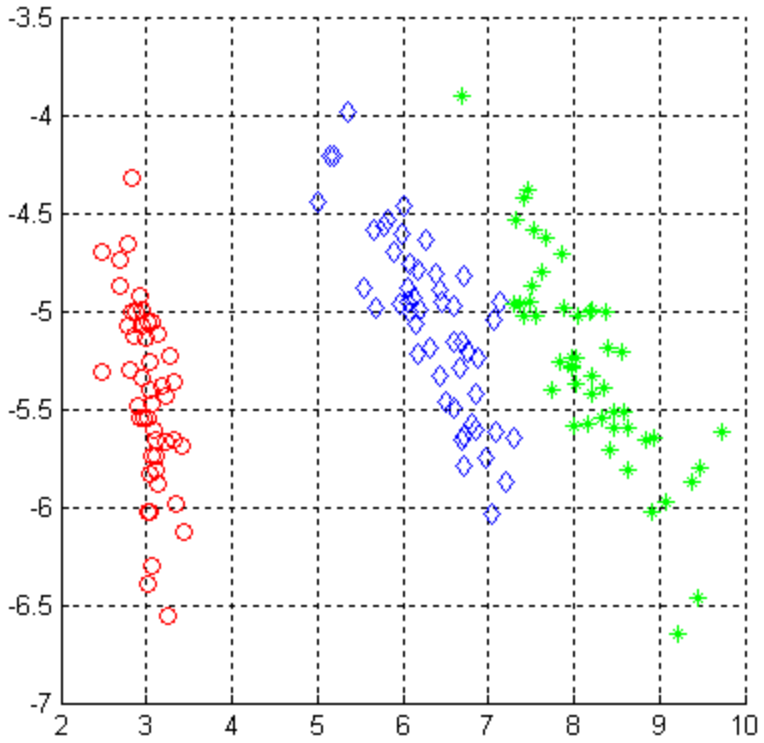
Experiments on Iris Data

Let:

- **DB** : Davies-Bouldin (DB) Index
- **DN** : Dunn's Index
- **CH** : Calinski-Harabasz (CH) Index
- **I** : Index I
- **XB** : Xie-Beni Index

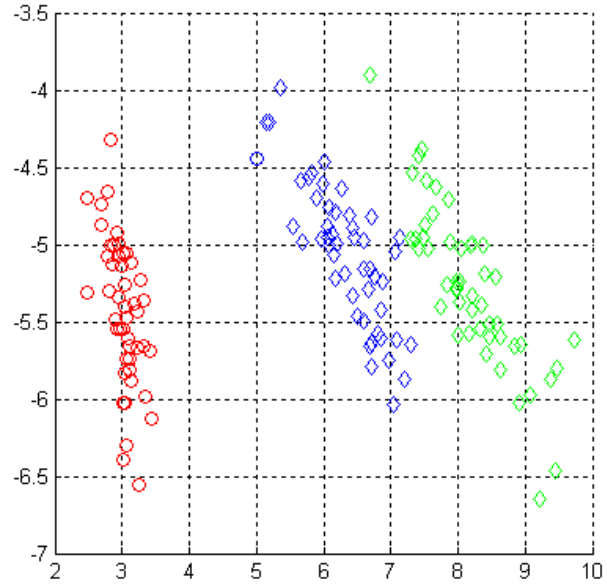
Visualization in 2D

- Iris dataset projected onto the 2D space using PCA:

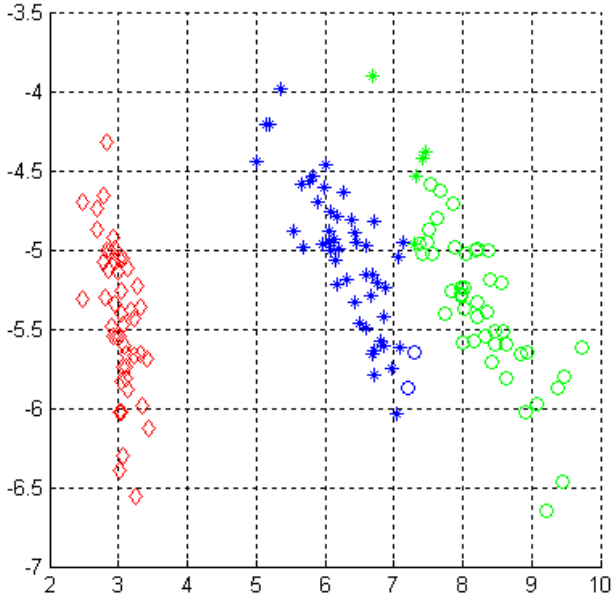


- Each color corresponds to a class of Iris dataset
- Each shape corresponds to a cluster *k*.
- Lets vary *k* = 2, …,10. Then…

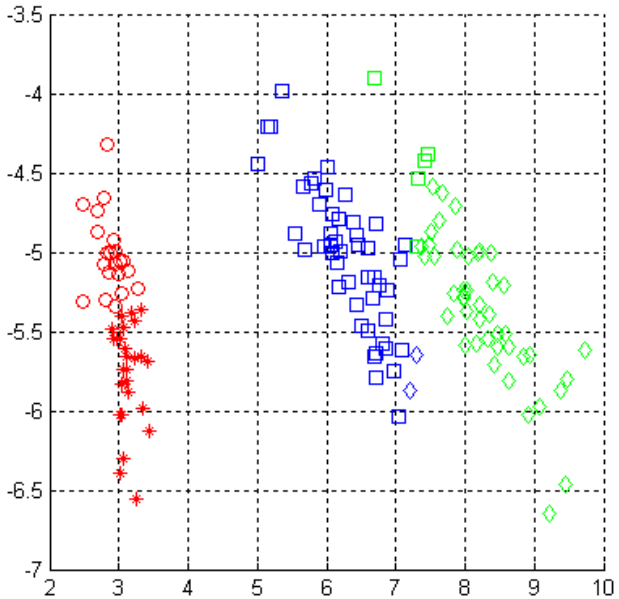
$k = 2$


-3.5
-4
-4.5
-5
-5.5
-6
-6.5
-7
2
3
4
5
6
7
8
9
10

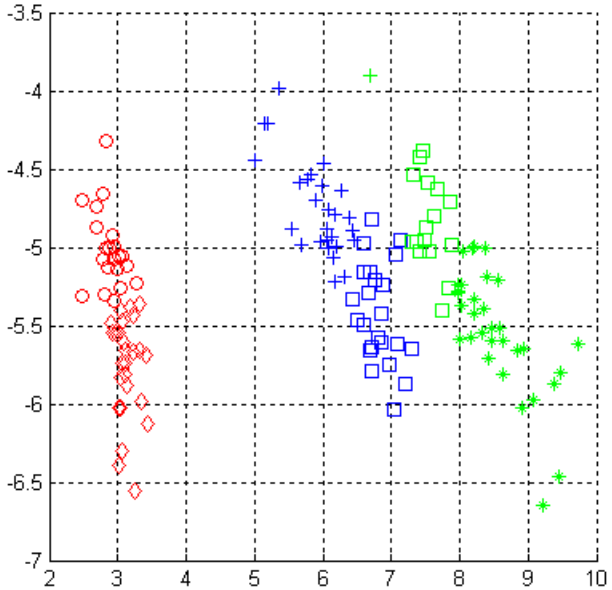
$k = 3$


-3.5
-4
-4.5
-5
-5.5
-6
-6.5
-7
2
3
4
5
6
7
8
9
10

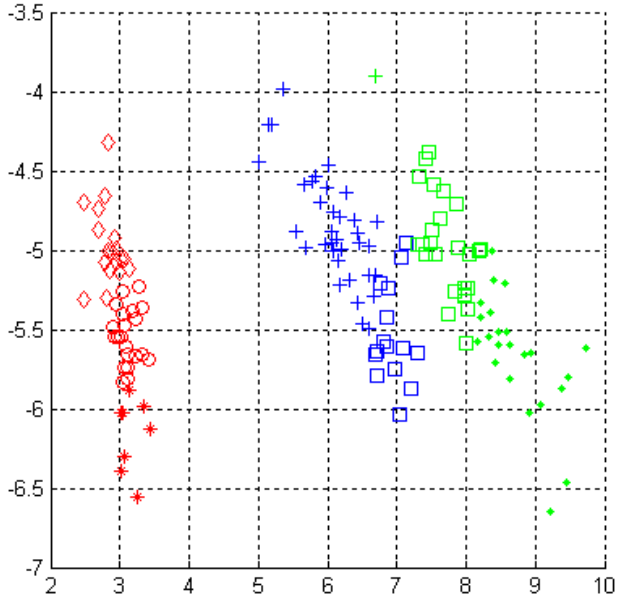
$k = 4$


-3.5
-4
-4.5
-5
-5.5
-6
-6.5
-7
2
3
4
5
6
7
8
9
10

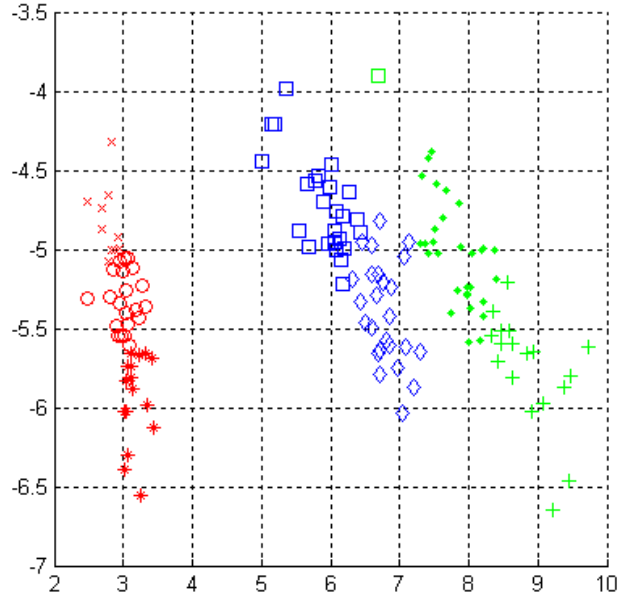
$k = 5$


-3.5
-4
-4.5
-5
-5.5
-6
-6.5
-7
2
3
4
5
6
7
8
9
10

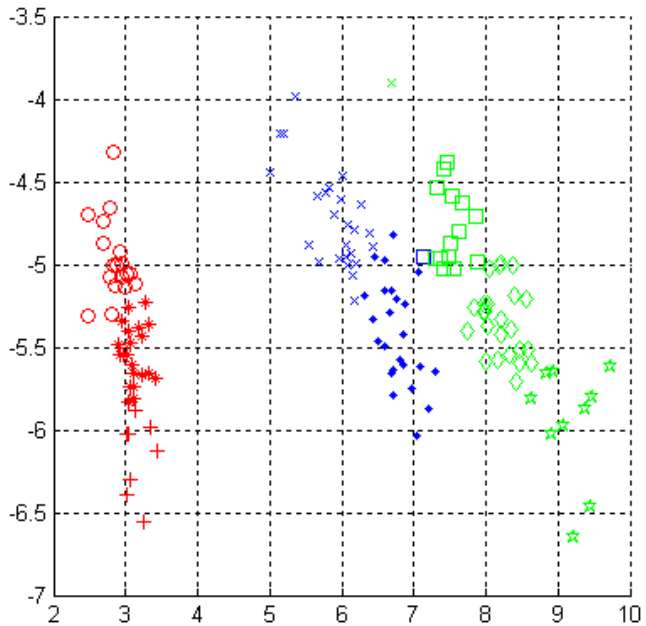
$k = 6$



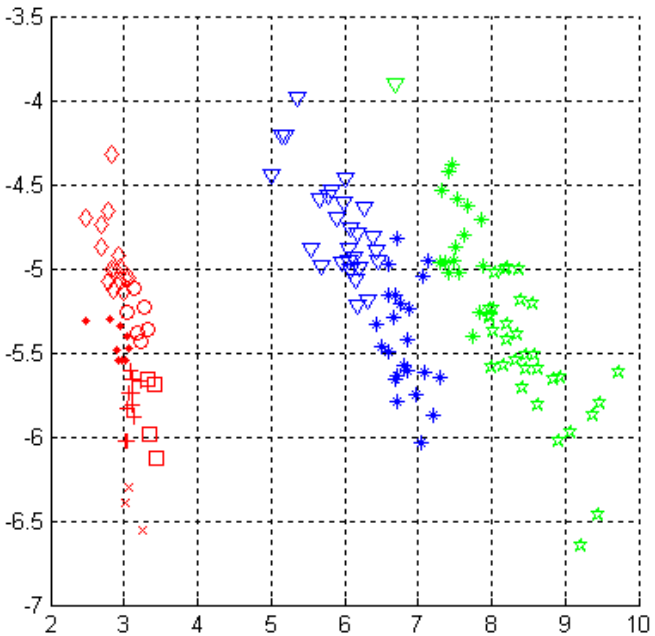
$k = 7$



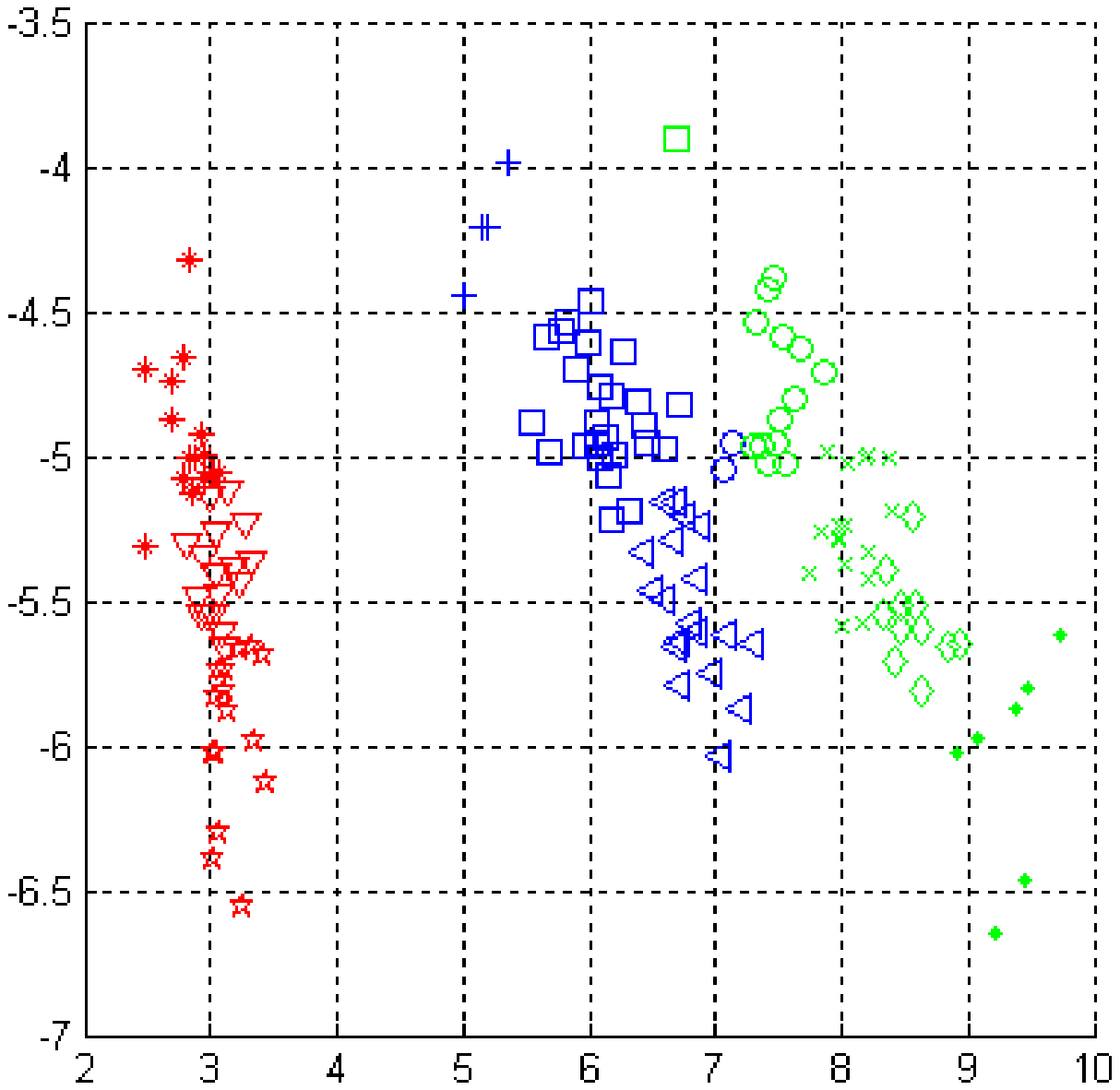
$k = 8$



$k = 9$



$k = 10$



Index values for each *k*

k	DB	DN	CH	I	XB
2	**4,96**	**0,0560**	**153,41**	0,0000004	323857,22
3	**11,01**	0,0306	**76,07**	**0,0003216**	220,41
4	41,72	0,0306	50,76	0,0000031	19111,79
5	39,49	**0,0460**	37,46	0,0000017	24864,68
6	143,89	0,0394	30,58	0,0000009	42985,73
7	57,36	0,0323	24,82	0,0000013	23929,74
8	120,86	0,0323	21,39	0,0000064	3859,84
9	403,57	**0,0460**	19,16	**0,0004101**	**90,86**
10	197,72	0,0336	16,37	0,0002199	**102,53**

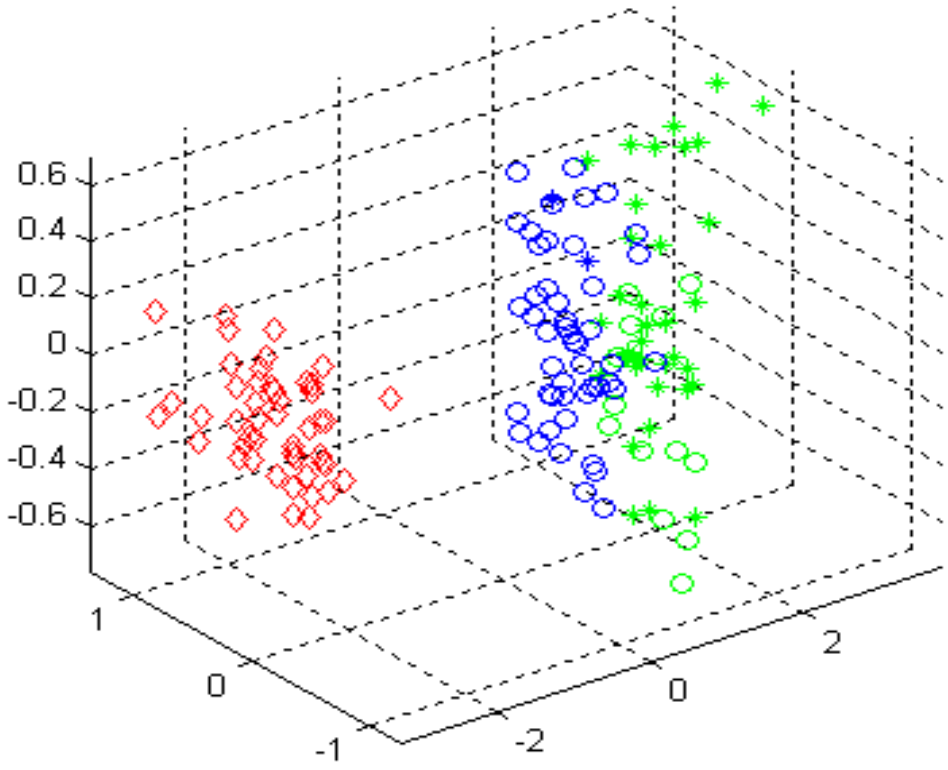
Red value represents the best choice for the index and

Blue value represents second best choice

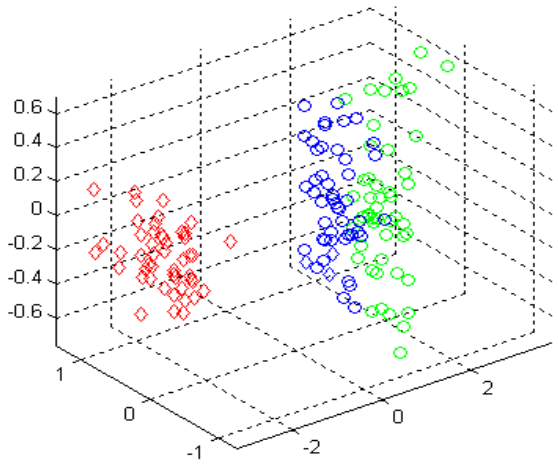
Visualization in 3D

- Iris dataset projected onto the 3D space

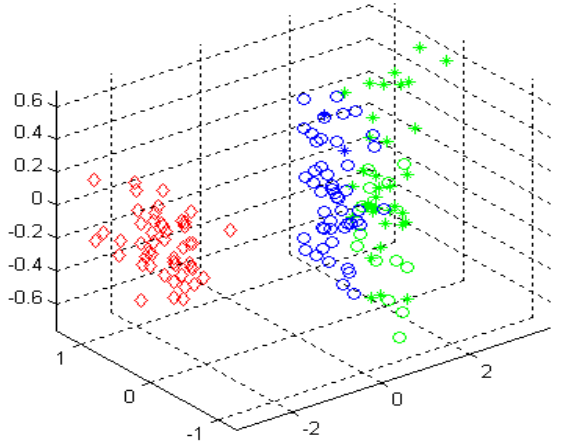
- Each color corresponds to a class of Iris dataset
- Each shape corresponds to a cluster *k*.
- Lets vary *k* = 2, …,10. Then…



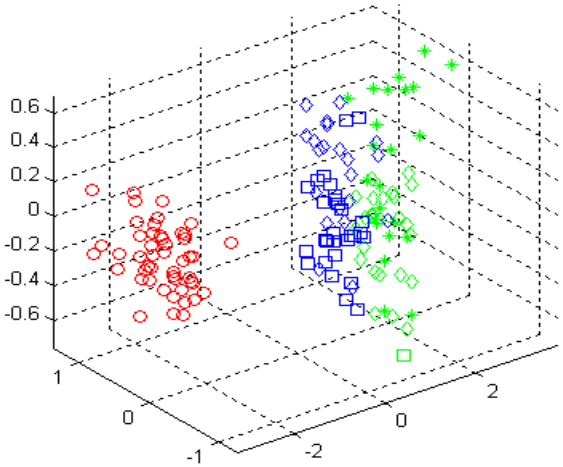
$k = 2$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

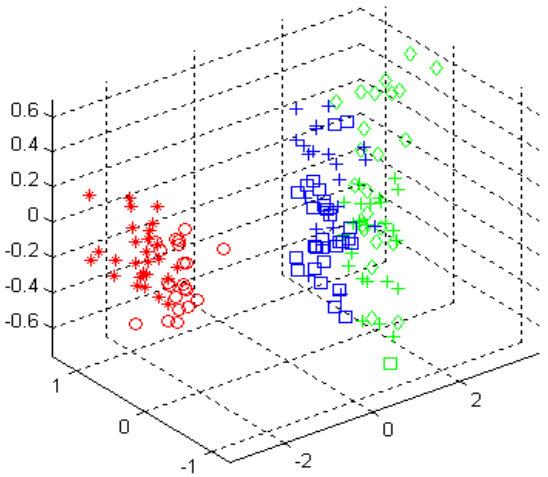
$k = 3$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

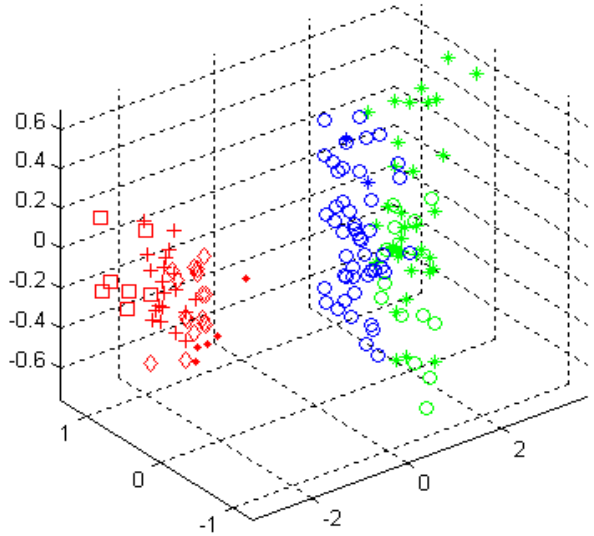
$k = 4$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

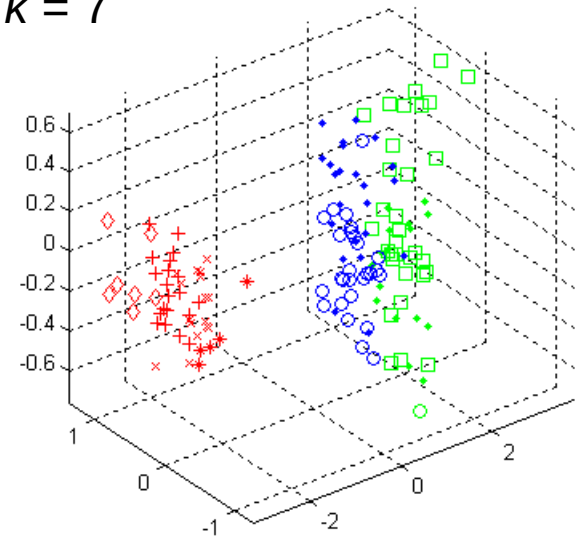
$k = 5$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

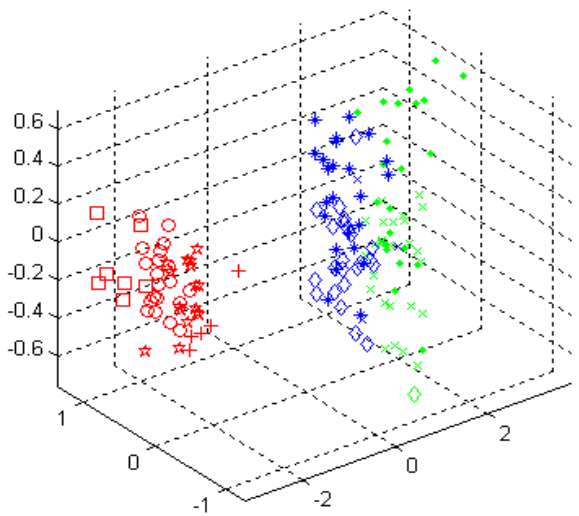
$k = 6$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

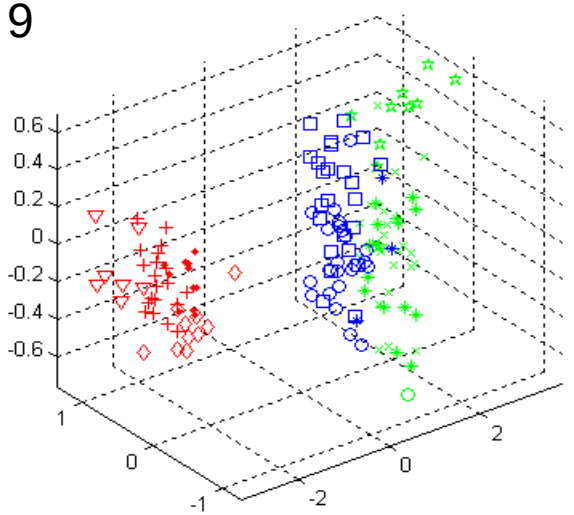
$k = 7$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

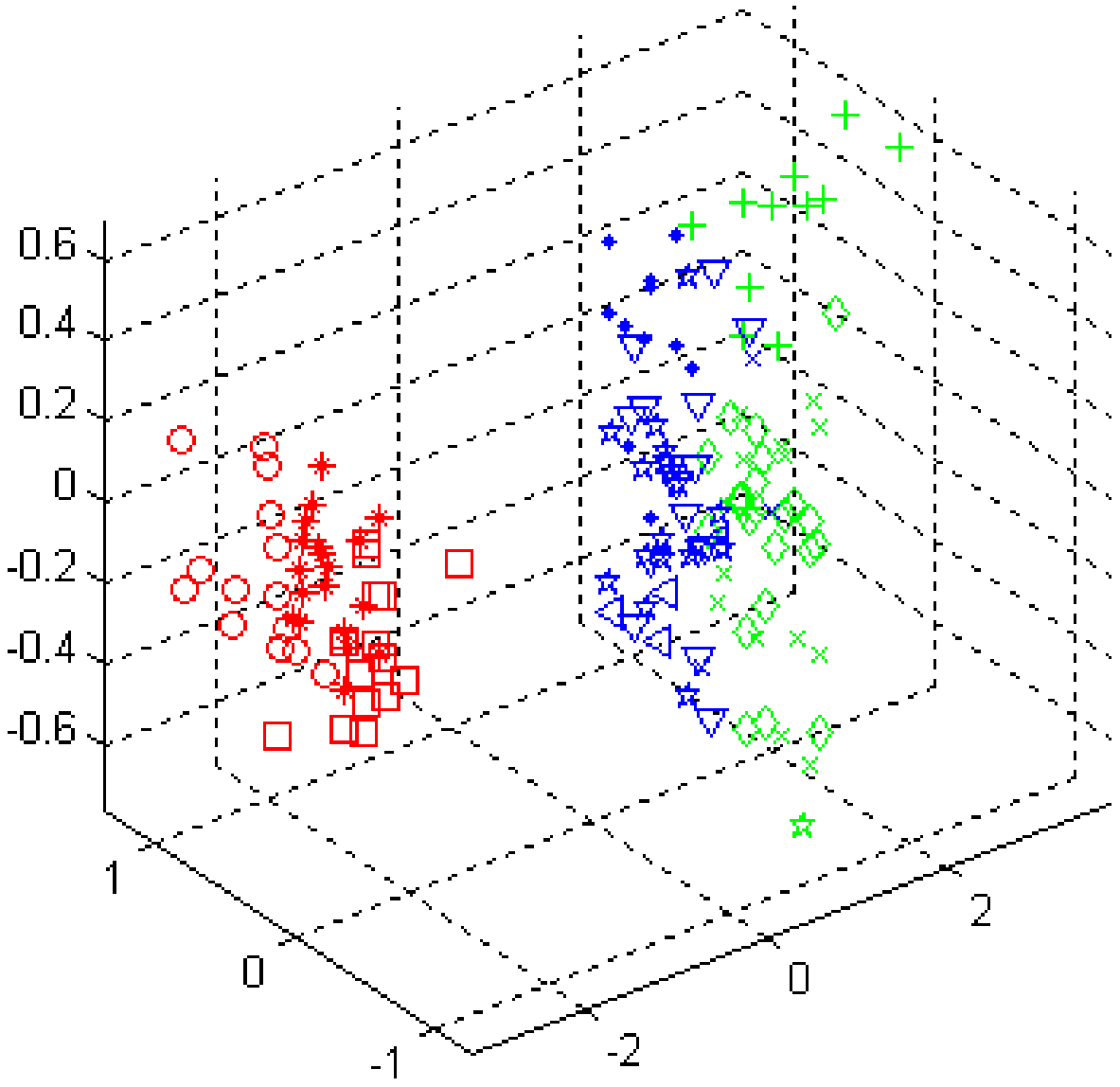
$k = 8$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

$k = 9$


0.6
0.4
0.2
0
-0.2
-0.4
-0.6
1
0
-1
-2
0
2

$k = 10$



Index values for each *k*:

k	DB	DN	CH	I	XB
2	**1,527**	0,0767	**192,641**	0,0105	205,965
3	**2,344**	**0,0985**	**100,447**	**0,0371**	**38,454**
4	3,159	**0,0788**	69,359	0,0008	1465,005
5	14,128	0,0561	51,382	0,0003	2653,271
6	63,097	0,0435	45,004	**0,0172**	67,649
7	54,810	0,0513	36,261	0,0158	**53,391**
8	48,996	0,0613	29,705	0,0009	711,588
9	47,294	0,0730	25,366	0,0036	140,846
10	26,621	0,0525	22,341	0,0005	798,878

Red value represents the best choice for the index and

Blue value represents second best choice

Tools for Clustering

Scikit:

- Provides a number clustering algorithms via the sklearn.cluter module
- Clustering algorithms:
  - k-means, EM (Gaussian mixtures)
  - Affinity propagation
  - Spectral clustering
  - DBSCAN (for large datasets)
  - Others
- Indices of validity:
  - Several measures, if known class labels
  - CH index
- Documentation:
  - http://scikit-learn.org/stable/modules/clustering.html#clustering

References

1. S. Fortunato, Community detection in networks: A user guide, Physics Report, Volume 659:1–44, Elsevier, 2016
2. U. Maulik et al., “Performance Evaluation of Some Clustering Algorithms and Validity Indices”, IEEE-PAMI, Vol. 24, No. 12, Dec. 2002, pp. 1650-1654.
3. S. Theodoridis and K. Koutroumbas. Pattern Recognition.
4. R. Duda et al. Pattern Classification. Second Edition. Wiley, 2000.