Pattern Analysis and Machine Intelligence

Lecture Notes on Clustering (II)
2013-2014

Davide Eynard
davide.eynard@usi.ch

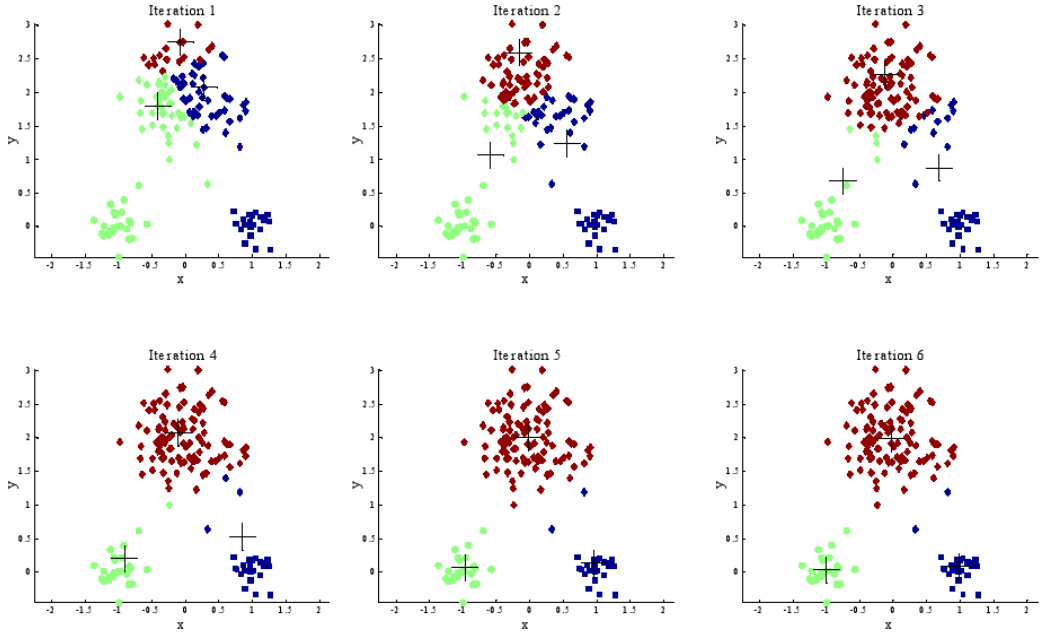
Department of Electronics and Information
Politecnico di Milano

Course Schedule [*Tentative*]

Date	Topic
25/11/2013	Clustering I: Introduction, K-means
29/11/2013	Clustering II: K-M alternatives, Hierarchical, SOM
02/12/2013	Clustering III: Mixture of Gaussians, DBSCAN, J-P
13/12/2013	Clustering IV: Spectral Clustering, cluster evaluation

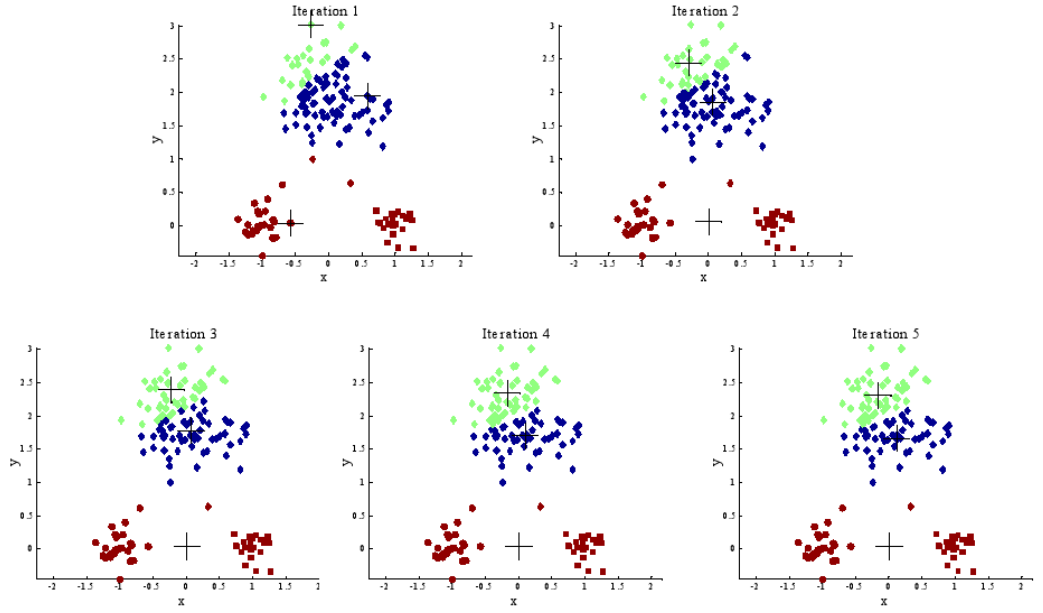
K-Means limits

Importance of choosing initial centroids



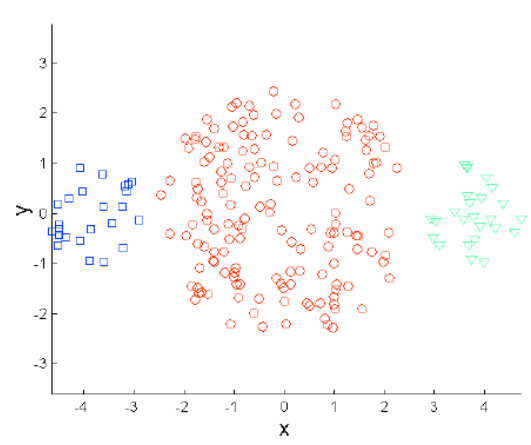
K-Means limits

Importance of choosing initial centroids

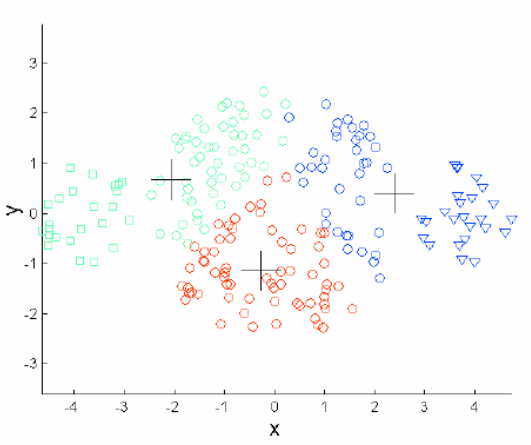


K-Means limits

Differing sizes



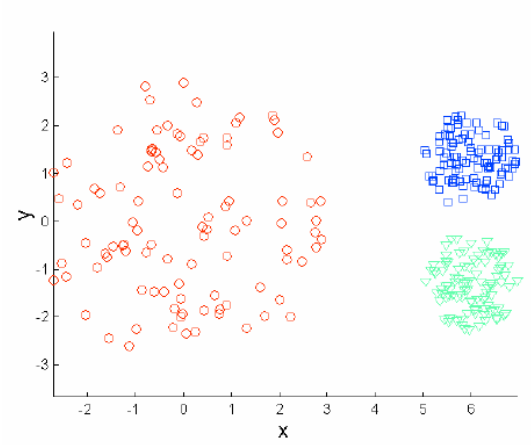
Original Points



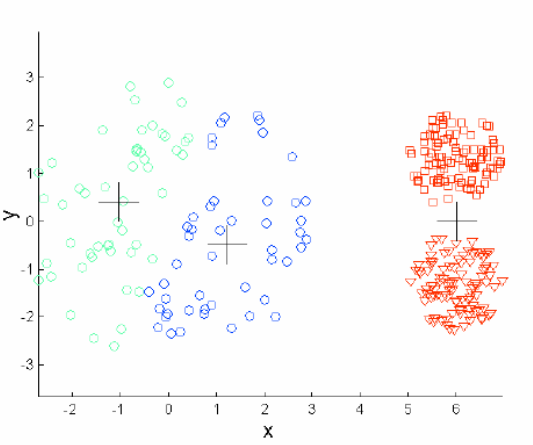
K-means Clusters

K-Means limits

Differing density



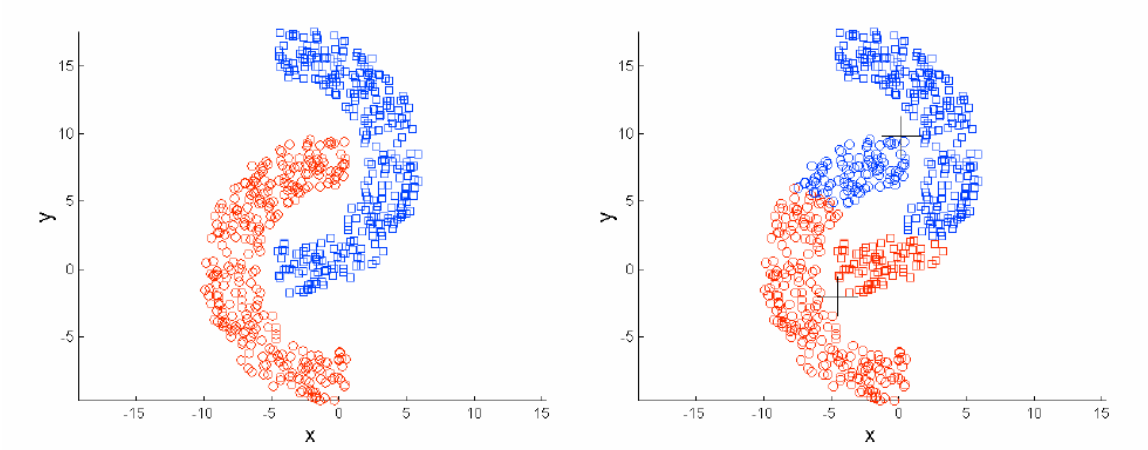
Original Points



K-means Clusters

K-Means limits

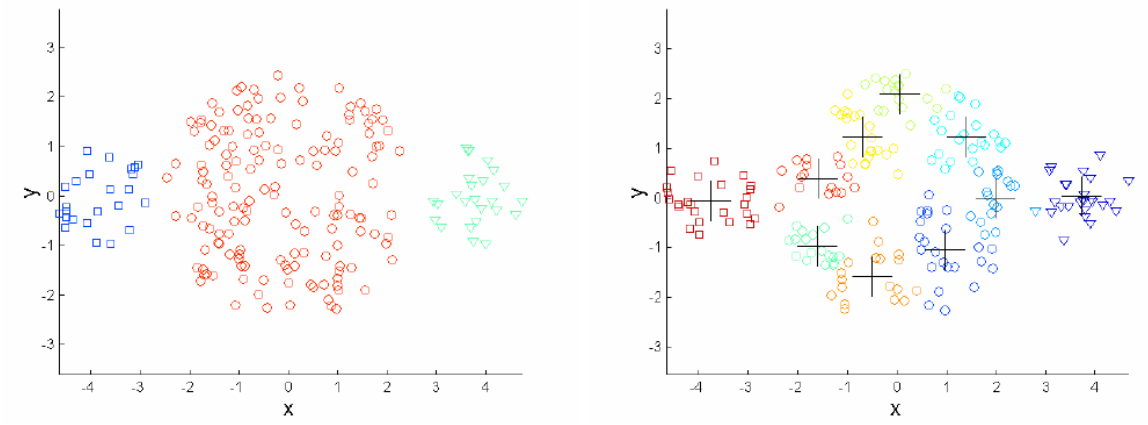
Non-globular shapes



Original Points **K-means Clusters**

K-Means: higher K

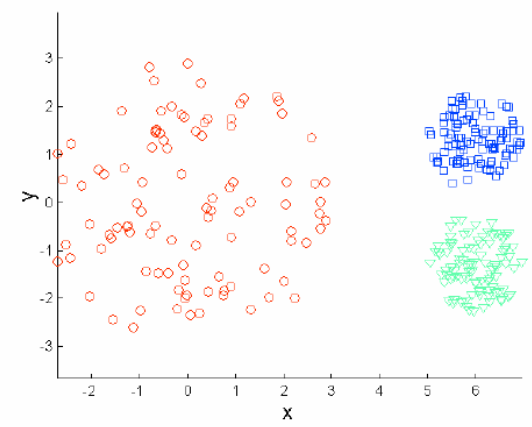
What if we tried to increase K to solve K-Means problems?



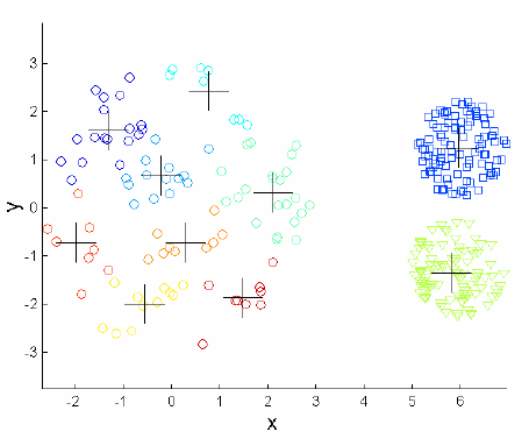
Original Points **K-means Clusters**

K-Means: higher K

What if we tried to increase K to solve K-Means problems?



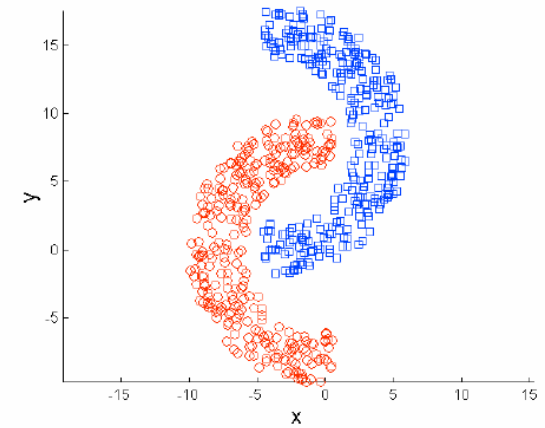
Original Points



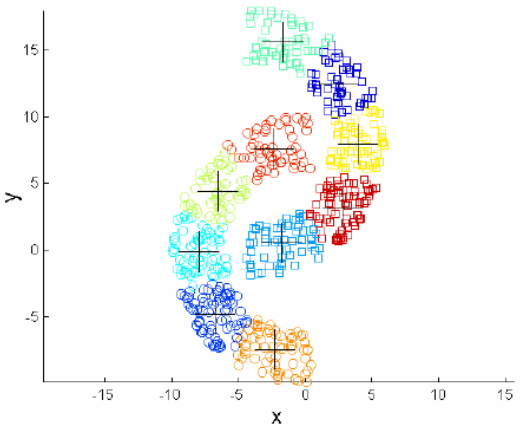
K-means Clusters

K-Means: higher K

What if we tried to increase K to solve K-Means problems?



Original Points



K-means Clusters

K-Medoids

- K-Means algorithm is too sensitive to outliers
  - An object with an extremely large value may substantially distort the distribution of the data
- **Medoid**: the most centrally located point in a cluster, as a representative point of the cluster
- Note: while a medoid is always a point inside a cluster too, a centroid could be not part of the cluster
- Analogy to using *medians*, instead of *means*, to describe the representative point of a set
  - Mean of 1, 3, 5, 7, 9 is 5
  - Mean of 1, 3, 5, 7, 1009 is 205
  - Median of 1, 3, 5, 7, 1009 is 5

PAM

PAM means **P**artitioning **A**round **M**edoids. The algorithm follows:

1. Given k
2. Randomly pick k instances as initial medoids
3. Assign each data point to the nearest medoid x
4. Calculate the objective function
   - the sum of dissimilarities of all points to their nearest medoids. (squared-error criterion)
5. For each non-medoid point y
   - swap x and y and calculate the objective function
6. Select the configuration with the lowest cost
7. Repeat (3-6) until no change

PAM

- Pam is more robust than k-means in the presence of noise and outliers
  - A medoid is less influenced by outliers or other extreme values than a mean (can you tell why?)
- Pam works well for small data sets but does not scale well for large data sets
  - $O(k(n-k)^2)$ for each change where n is # of data objects, k is # of clusters
- NOTE: not having to calculate a *mean*, we do not need actual *positions* of points but just their *distances*!

Fuzzy C-Means

Fuzzy C-Means (FCM, developed by Dunn in 1973 and improved by Bezdek in 1981) is a method of clustering which allows one piece of data to belong to two or more clusters.

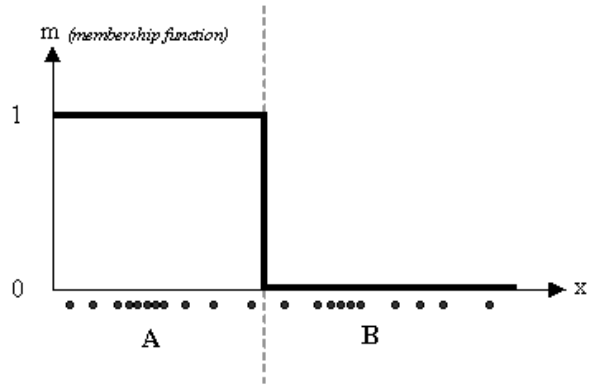
- frequently used in pattern recognition
- based on minimization of the following objective function:

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^m \|x_i - c_j\|^2, 1 \leq m < \infty$$

where:
m is any real number greater than 1 (*fuzziness coefficient*),
u_{ij} is the degree of membership of x_i in the cluster j,
x_i is the i-th of d-dimensional measured data,
c_j is the d-dimension center of the cluster,
$\|\cdot\|$ is any norm expressing the similarity between measured data and the center.

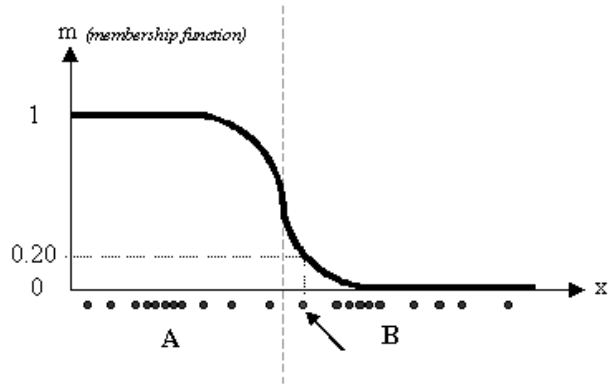
K-Means vs. FCM

- With K-Means, every piece of data either belongs to centroid A or to centroid B



K-Means vs. FCM

- With FCM, data elements do not belong exclusively to one cluster, but they may belong to several clusters (with different membership values)



Data representation

$$(KM)U_{N\times C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ \cdots & \cdots \\ 0 & 1 \end{bmatrix}$$

$$(FCM)U_{N\times C} = \begin{bmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \\ 0.6 & 0.4 \\ \cdots & \cdots \\ 0.9 & 0.1 \end{bmatrix}$$

FCM Algorithm

The algorithm is composed of the following steps:

1. Initialize $U = [u_{ij}]$ matrix, $U^{(0)}$
2. At t-step: calculate the centers vectors $C^{(t)} = [c_j]$ with $U^{(t)}$:

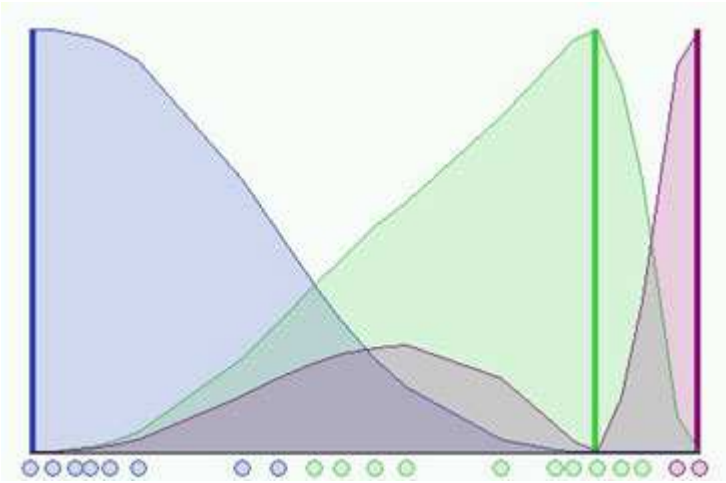
$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m \cdot x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

3. Update $U^{(t)}$, $U^{(t+1)}$:

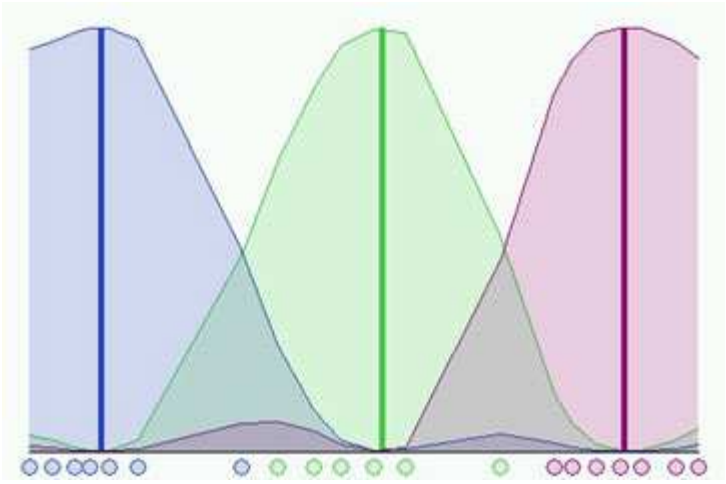
$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

4. If $\|U^{(k+1)} - U^{(k)}\| < \varepsilon$ then STOP; otherwise return to step 2.

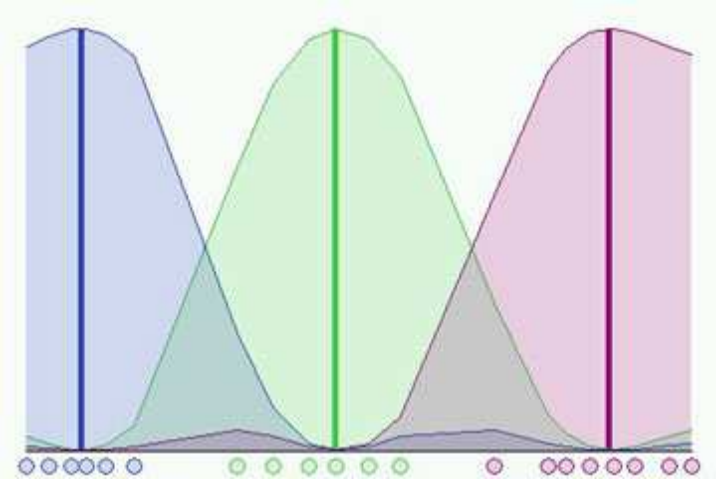
An Example



An Example



An Example



Hierarchical Clustering

- Top-down vs Bottom-up
- Top-down (or *divisive*):
  - Start with one universal cluster
  - Split it into two clusters
  - Proceed recursively on each subset
- Bottom-up (or *agglomerative*):
  - Start with single-instance clusters ("every item is a cluster")
  - At each step, join the two closest clusters
  - (design decision: distance between clusters)

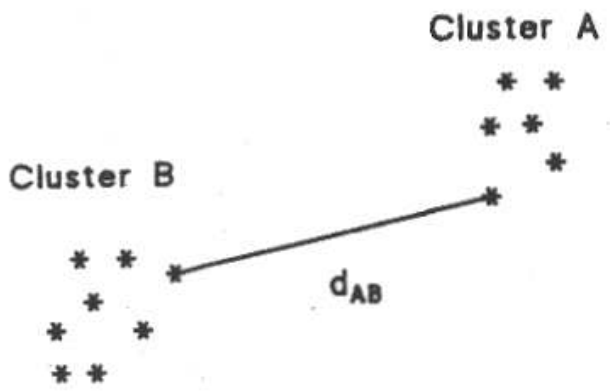
Agglomerative Hierarchical Clustering

Given a set of N items to be clustered, and an N*N distance (or dissimilarity) matrix, the basic process of agglomerative hierarchical clustering is the following:

1. Start by assigning each item to a cluster. Let the dissimilarities between the clusters be the same as the dissimilarities between the items they contain.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster. Now, you have one cluster less.
3. Compute dissimilarities between the new cluster and each of the old ones.
4. Repeat Steps 2 and 3 until all items are clustered into a single cluster of size N.

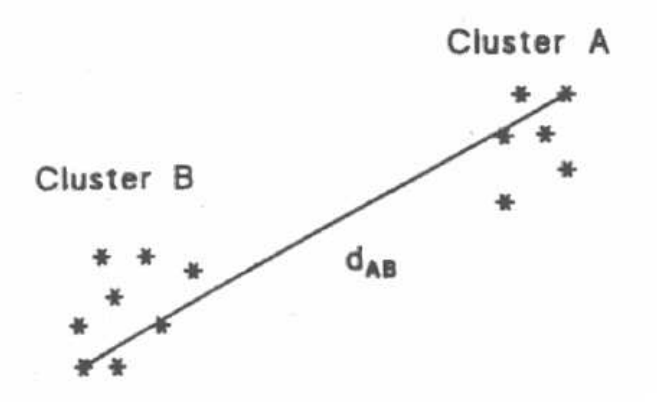
Single Linkage (SL) clustering

- We consider the distance between two clusters to be equal to the **shortest** distance from any member of one cluster to any member of the other one (**greatest** similarity).



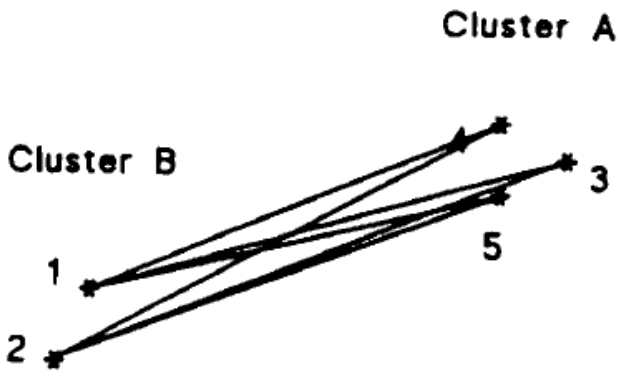
Complete Linkage (CL) clustering

- We consider the distance between two clusters to be equal to the **greatest** distance from any member of one cluster to any member of the other one (**smallest** similarity).



Group Average (GA) clustering

- We consider the distance between two clusters to be equal to the **average** distance from any member of one cluster to any member of the other one.



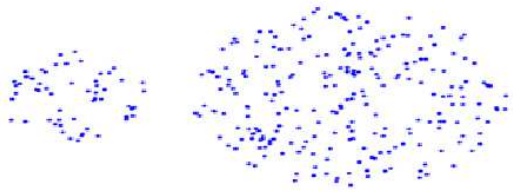
About distances

If the data exhibit strong clustering tendency, all 3 methods produce similar results.

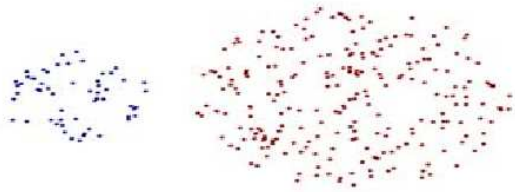
- **SL**: requires only a single dissimilarity to be small. Drawback: produced clusters can violate the “compactness” property (cluster with large diameters)
- **CL**: opposite extreme (compact clusters with small diameters, but can violate the “closeness” property)
- **GA**: compromise, it attempts to produce relatively compact clusters and relatively far apart. BUT it depends on the dissimilarity scale.

Hierarchical algorithms limits

Strength of MIN



Original Points

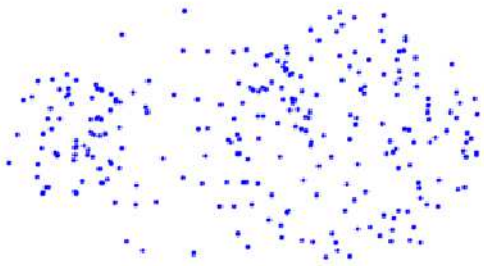


Two Clusters

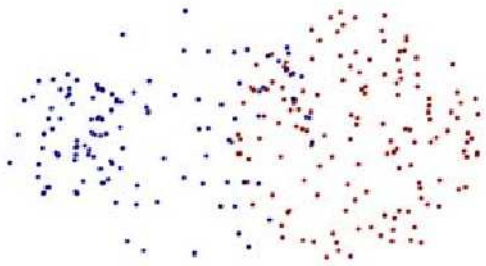
- Easily handles clusters of different sizes
- Can handle non elliptical shapes

Hierarchical algorithms limits

Limitations of MIN



Original Points

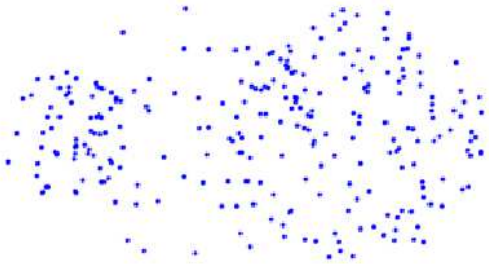


Two Clusters

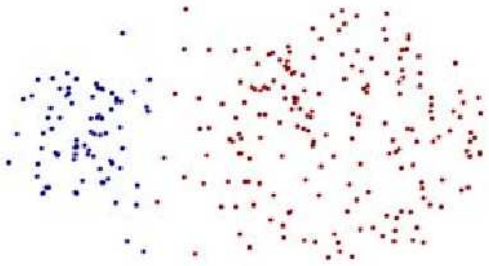
- Sensitive to noise and outliers

Hierarchical algorithms limits

Strength of MAX



Original Points

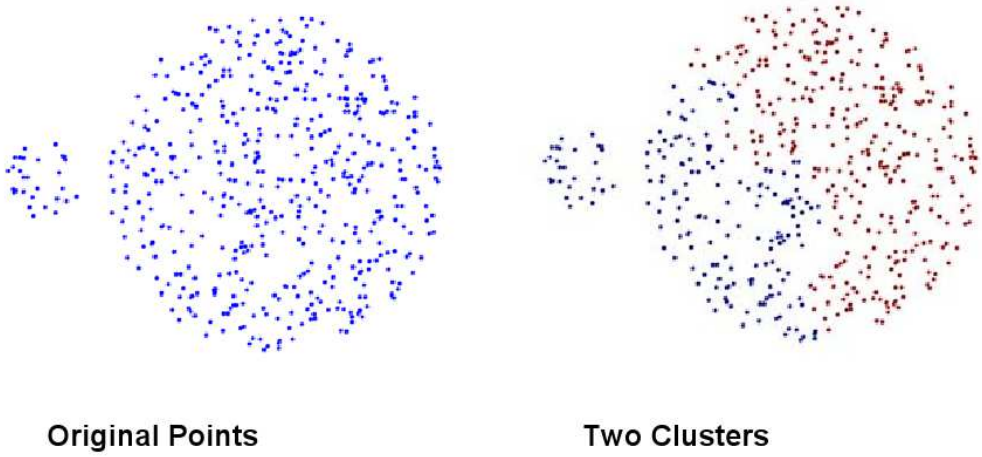


Two Clusters

- Less sensitive to noise and outliers

Hierarchical algorithms limits

Limitations of MAX



Original Points Two Clusters

- Tends to break large clusters
- Biased toward globular clusters

Hierarchical clustering: Summary

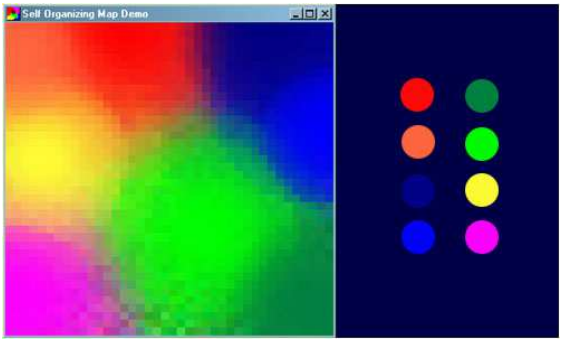
- Advantages
  - It's nice that you get a hierarchy instead of an amorphous collection of groups
  - If you want k groups, just cut the $(k-1)$ longest links
- Disadvantages
  - It doesn't scale well: time complexity of at least $O(n^2)$, where n is the number of objects

Self Organizing Features Maps

Kohonen Self Organizing Features Maps (a.k.a. SOM) provide a way to represent multidimensional data in much lower dimensional spaces.

- They implement a data compression technique similar to *vector quantization*
- They store information in such a way that any topological relationships within the training set are maintained

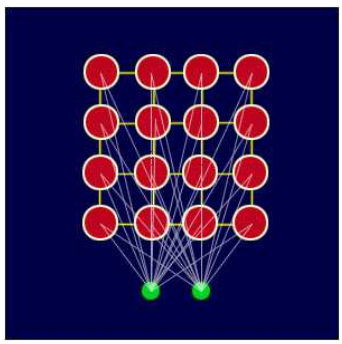
Example: Mapping of colors from their three dimensional components (i.e., red, green and blue) into two dimensions.



Self Organizing Feature Maps: The Topology

- The network is a lattice of "nodes", each of which is fully connected to the input layer
- Each node has a specific topological position and contains a vector of weights of the same dimension as the input vectors
- There are no lateral connections between nodes within the lattice

A SOM does not need a target output to be specified; instead, where the node weights match the input vector, that area of the lattice is selectively optimized to more closely resemble the data vector



Self Organizing Features Maps: The Algorithm

Training occurs in several steps over many iterations:

1. Initialize each node's weights
2. Given a random vector from the training set to the lattice
3. Examinate every node to calculate which one's weights are most similar to the input vector (the winning node is commonly known as the Best Matching Unit)
4. Calculate the radius of the neighborhood of the BMU (this is a value that starts large, typically set to the 'radius' of the lattice, but diminishes each time-step). Any nodes found within this radius are deemed to be inside the BMU's neighborhood
5. Each neighboring node's weights are adjusted to make them more similar to the input vector. The closer a node is to the BMU, the more its weights get altered
6. Repeat step 2 for N iterations

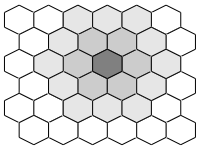
Practical Learning of Self Organizing Features Maps

There are few things that have to be specified in the previous algorithm:

- Choosing the weights initialization
- We select the Best Matching Unit according to the distance between its weights and the input vector:

$$||\mathbf{x} - \mathbf{w}_i|| = \sqrt{\sum_{k=1}^{p}(\mathbf{x}[k] - \mathbf{w}_i[k])^2}$$

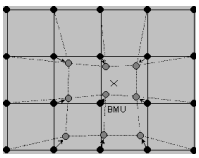
- Select the neighborhood according to some decreasing function



$$h_{ij} = e^{-\frac{(i-j)^2}{2\sigma^2}}$$

- Define the updating rule

$$\mathbf{w}_i(t+1) = \begin{cases} \mathbf{w}_i + \alpha(t)[\mathbf{x}(t) - \mathbf{w}_i(t)], & i \in N_i(t) \\ \mathbf{w}_i, & i \notin N_i(t) \end{cases}$$



Clustering on text files: the Vector Space Model

Search Engines

How do search engines work?

- document retrieval and indexing
- query language that allows to search for Web pages that contain (or not) given words and phrases
- SE have their roots in *information retrieval* systems, which prepare a keyword index for the given corpus and respond to keyword queries with a ranked list of documents
- some queries:
  - docs containing the word “Java”
  - docs containing “Java” but not “coffee”
  - docs containing the phrase “Java Beans” and the word “API”
  - docs where “Java” and “island” occur in the same sentence

Search Engines - A naive approach

tid	did	pos
my	1	1
care	1	2
is	1	3
⋮		
new	2	8
care	2	9
won	2	10

1. `select did from POSTING where tid = 'java'`
2. `(select did from POSTING where tid = 'java') except (select did from POSTING where tid = 'coffee')`
3.
```
with
  D_JAVA (did, pos) as (select did, pos from POSTING where tid = 'java'),
  D_BEANS(did, pos) as (select did, pos from POSTING where tid = 'beans'),
  D_JAVABEANS(did) as
    (select D_JAVA.did from D_JAVA, D_BEANS
      where D_JAVA.did = D_BEANS.did
      and D_JAVA.pos + 1 = D_BEANS.pos),
  D_API(did) as (select did from POSTING where tid = 'api'),
(select did from D_JAVABEANS) union (select did from D_API)
```


Search Engines - Not naive at all

Can we always think about text search in terms of sets?

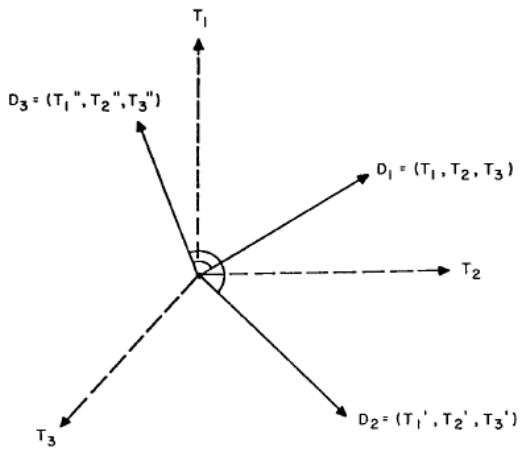
- The index.of approach
- The epanaleptical approach

Main problems:

- Index creation, compression and update
- Stopwords and stemming
- Relevance ranking
  - recall vs precision
- Relevance feedback

VSM

In the Vector Space Model, documents are represented as vectors in a multidimensional Euclidean space.



VSM

The coordinate of document d in the direction corresponding to the term t is determined by two quantities:

- *Term Frequency* $TF(d,t)$: this is simply $n(d,t)$, the number of times term t occurs in document d, scaled to normalize document length
- *Inverse Document Frequency* $IDF(t)$: this is a weight factor used to scale down the coordinates of terms which occur in many documents

$$IDF(t) = log\frac{|D|}{1+|D_t|}$$

VSM

TF and IDF are combined into the complete vector-space model in the obvious way: the coordinate of document d in axis t is given by

$$d_t = TF(d,t)IDF(T)$$

Then, the *cosine distance* between the two vectors

$$v_{d1} = [w_{1,d1}, w_{2,d1}, \ldots, w_{N,d1}]^T$$
$$v_{d2} = [w_{1,d2}, w_{2,d2}, \ldots, w_{N,d2}]^T$$

is calculated as

$$cos\theta = \frac{v_1 \cdot v_2}{||v_1||||v_2||}$$

Note: one of the two vectors might be the query itself!

Limits of VSM

- Long documents are poorly represented, because they have poor similarity values (a small scalar product and a large dimensionality)
- Search keywords must precisely match document terms; word substrings might result in a "false positive" match
- Semantic sensitivity: documents with similar context but different term vocabulary won't be associated, resulting in a "false negative" match

Bibliography

- A Tutorial on Clustering Algorithms Online tutorial by M. Matteucci
- K-means and Hierarchical Clustering
  Tutorial Slides by A. Moore
- "Metodologie per Sistemi Intelligenti" course - Clustering
  Tutorial Slides by P.L. Lanzi
- K-Means Clustering Tutorials
  Online tutorials by K. Teknomo
- Salton, G., Wong, A., and Yang, C. S. (1975).
  A Vector Space Model for Automatic Indexing.