Clustering algorithms

Konstantinos Koutroumbas

Unit 9

– Hierarchical algs. for large data sets (ROCK, Chameleon)
– Clustering algs. Based on graph theory
– Competitive learning algorithms

koutroum@noa.gr

The ROCK (RObust Clustering using linKs) algorithm

It is best suited for **nominal** (**categorical**) features.

- Some preliminaries
  - Two points $\boldsymbol{x}, \boldsymbol{y} \in X$ are considered neighbors if $s(\boldsymbol{x}, \boldsymbol{y}) \geq \theta$, where $s(.)$ is a similarity function and θ a user-defined similarity threshold between two vectors ($0 \leq s(\boldsymbol{x}, \boldsymbol{y}) \leq 1$ and, consequently, $0 \leq \theta \leq 1$).
  - $link(\boldsymbol{x}, \boldsymbol{y})$ is the number of common neighbors between $\boldsymbol{x}$ and $\boldsymbol{y}$.

In the graph whose vertices correspond to data points and edges **connect** neighboring points, $link(\boldsymbol{x}, \boldsymbol{y})$ is the **number** of **distinct paths** of **length 2** that connect $\boldsymbol{x}, \boldsymbol{y}$.

- Assumption: There **exists** a function $f(\theta)$ (< 1) such that:

  "Each point assigned to a cluster C_i has approximately $n_i^{f(\theta)}$ neighbors in C_i (n_i is the number of points in C_i)"

  It can be proved that the expected total number of links **among** all pairs in C_i is $n_i^{1+2f(\theta)}$.

$$link(C_i) = \sum_{\boldsymbol{x} \in C_i} \sum_{\boldsymbol{y} \in C_i} link(\boldsymbol{x}, \boldsymbol{y})$$

The ROCK (RObust Clustering using linKs) algorithm

- ROCK is a **special case** of GAS where
  - The closeness between two clusters is defined as

$$link(C_i, C_j) = \sum_{x \in C_i} \sum_{y \in C_j} link(\boldsymbol{x}, \boldsymbol{y})$$

$$g(C_i, C_j) = \frac{link(C_i, C_j)}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

The denominator is the expected total number of links *between* the two clusters.
The larger the $g(\cdot)$, the more similar the clusters C_i and C_j *are*.

- The stopping criterion is:
  - the number of clusters becomes equal to a predefined number m *or*
  - $link(C_i, C_j) = 0$ for every pair in a clustering $\mathfrak{R}_t$.

- Time complexity for ROCK: Similar to CURE for large N.
- Prohibitive for very large data sets.
- *Solution:* **Adoption** of random sampling techniques.

The ROCK (RObust Clustering using linKs) algorithm

- ROCK utilizing Random Sampling
  - Identification of clusters
    - Select a subset X' of X via random sampling
    - Run the original ROCK algorithm on X'
  - Assignment of points to clusters
    - For each cluster C_i select a set L_i of n_{L_i} points
    - For each $\boldsymbol{z} \in X - X'$
      - **Compute** $t_i = N_i/(n_{L_i} + 1)^{f(\theta)}$, where N_i is the no of neighbors of $\boldsymbol{z}$ in L_i.
      - **Assign** $\boldsymbol{z}$ to the cluster with the maximum t_i.

Remarks:

- A choice for $f(\theta)$ is $f(\theta) = (1 - \theta)/(1 + \theta)$, with $(\theta < 1)$.
- $f(\theta)$ **depends** on the data set and the type of clusters we are interested in.
- The hypothesis about the existence of $f(\theta)$ is very strong. It may lead to poor results if the data do not satisfy it.
- It can be used for discrete-valued data sets.

The ROCK (RObust Clustering using linKs) algorithm

An application:

- Grouping the customers of supermarket according to their purchases.
- Each customer (entity) is represented by the set of goods he/she buys (categorical data representation).
- The similarity between two customers may be quantified via the **Jaccard coefficient**

For two finite sets T_i and T_j, the **Jaccard coefficient** is defined as

$$J(T_i, T_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}$$

- For example, assuming that $T_1 = \{A, B, C\}, T_2 = \{A, B, D\}, T_3 = \{A, B, D, E\}$ are the sets corresponding to three customers, it is

$$J(T_1, T_1) = \frac{3}{3} = 1, \quad J(T_1, T_2) = \frac{2}{4} = 0.5, \quad J(T_1, T_3) = \frac{2}{5} = 0.4,$$

$$J(T_2, T_3) = \frac{3}{4} = 0.75$$

Choosing $\theta = 0.45$, T_1 and T_2 are neighbors, T_2 and T_3 are neighbors but T_1 and T_3 are not neighbors. However, T_1 and T_3 share a common neighbor.

- For this application, a good choice for $f(\theta)$ is $f(\theta) = (1 - \theta)/(1 + \theta)$, with $(\theta < 1)$.

The ROCK (RObust Clustering using linKs) algorithm

Example: Consider a three-cluster clustering $\{C_1, C_2, C_3\}$, where the number of points in each one of them is $n_1 = 500$, $n_2 = 500$ and $n_3 = 100$, respectively.

$$g(C_i, C_j) = \frac{link(C_i, C_j)}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

Define $f(\theta)$ as $f(\theta) = \frac{1-\theta}{1+\theta}$, with $\theta = \frac{1}{3}$.

Let $link(C_1, C_2) = 100$ and $link(C_1, C_3) = 100$.

Compute $g(C_1, C_2)$ and $g(C_1, C_3)$ and draw your conclusions

Answer: It is $1 + 2f(\theta) = 1 + 2\frac{1-\theta}{1+\theta} = 1 + 2\frac{1-\frac{1}{3}}{1+\frac{1}{3}} = 2$,

$$(n_1 + n_2)^{1+2f(\theta)} - n_1^{1+2f(\theta)} - n_2^{1+2f(\theta)} = (500 + 500)^2 - 500^2 - 500^2 = 500000$$

$$(n_1 + n_3)^{1+2f(\theta)} - n_1^{1+2f(\theta)} - n_3^{1+2f(\theta)} = (500 + 100)^2 - 500^2 - 100^2 = 100000$$

Then $g(C_1, C_2) = \frac{100}{500000} = 0.0002$ and $g(C_1, C_3) = \frac{100}{100000} = 0.001$

Thus, among the clusters that have the same degree of similarity with C_1 wrt the $link(.)$ criterion, according to the normalized link criterion ($g(\cdot)$) C_1 is more similar with the smallest cluster (C_3), and not with the equally sized C_3.

The Chameleon algorithm

- This algorithm is not based on a "static" modeling of clusters like CURE (where each cluster is represented by the same number of representatives) and ROCK (where constraints are posed through the function $f(\theta)$).
- It enjoys both divisive and agglomerative features.

- *Some preliminaries:*
  Let $G = (V, E)$ be a graph where:
  - each vertex of V **corresponds** to a data point in X.
  - E is a set of edges connecting pairs of vertices in V. Each edge is weighted by the **similarity** of the corresponding points.

  - Edge cut set: Let C be a set of points corresponding to a subset of V.
    Assume that C is partitioned into two nonempty sets C_i and C_j.
    The subset E'_{ij} of the edges of E that connect points of C_i with points of C_j is called edge cut set.

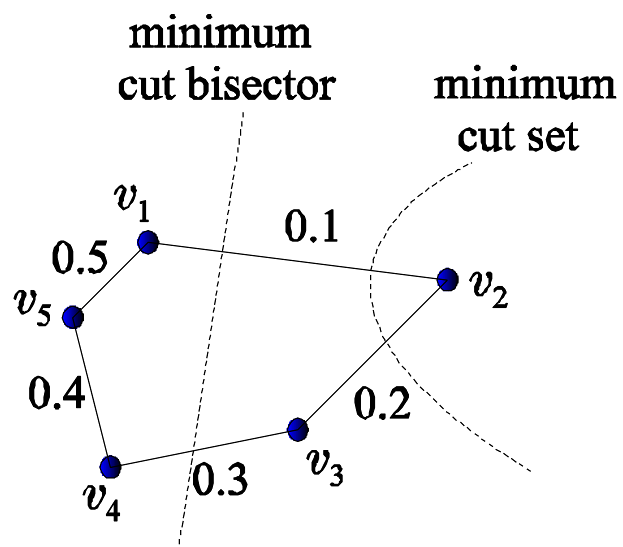
The Chameleon algorithm

- Minimum cut set: Let C be a set of points corresponding to a subset of V. If $|E'_{ij}| = min_{(C_u,C_v):\, C_u \cup C_v = C} |E'_{uv}|$, then (C_i, C_j) is the minimum cut set of C,

  where $|E'_{uv}|$ be the sum of weights of the edges in E'_{uv}.

- Minimum cut bisector: If C_i, C_j are constrained to be of approximate equal size, the minimum cut set (over all possible partitions of approximately equal size) is known as the minimum cut bisector.

Example: The graph in the following figure consists of the 5 vertices and the edges shown, each one weighted by the similarity of the points that correspond to the vertices it connects. The minimum cut set and the minimum cut bisector are shown.



The Chameleon algorithm

Measuring the similarity between clusters

Relative interconnectivity:

–Let E_{ij} be the set of edges **connecting** points in C_i with points in C_j.

–Let E_i be the set of edges **corresponding** to the minimum cut bisector of C_i.

–Let $|E_i|$, $|E_{ij}|$ be the **sum** of the weights of the edges of E_i, E_{ij}, respectively.

–Absolute interconnectivity between C_i, $C_j = |E_{ij}|$

–Internal interconnectivity of $C_i = |E_i|$

–Relative interconnectivity between C_i, C_j:

$$RI_{ij} = \frac{|E_{ij}|}{\frac{|E_i| + |E_j|}{2}}$$

Relative closeness:

–Let S_{ij} be the **average** weight of the edges in E_{ij} .

–Let S_i be the **average** weight of the edges in E_i .

–Relative closeness between C_i and C_j:

$$RC_{ij} = \frac{S_{ij}}{\frac{n_i}{n_i + n_j} S_i + \frac{n_j}{n_i + n_j} S_j}$$

n_i, n_j: Number of points in C_i, C_j, resp.

The Chameleon algorithm

The Chameleon algorithm

Preliminary phase

Create a k-nearest neighbor graph $G = (V, E)$ such that:

- Each vertex of V corresponds to a data point.
- The edge between two vertices v_i and v_j is added to E if v_i is one of the k-nearest neighbors of v_j or vise versa.
- Each connected component of the resulting graph is **associated** with a cluster. Let $\Re$ be the clustering consisting of these clusters.

Divisive phase

Set $\mathscr{R}_0 = \mathscr{R}$

$t = 0$

Repeat

- $t = t + 1$
- **Select** the largest cluster C in $\mathscr{R}_{t-1}$.
- Referring to E, **partition** C into two sets so that:
  - –the sum of the weights of the edge cut set between the resulting clusters is minimized.
  - –each cluster contains at least 25% of the vertices of C.

Until each cluster in $\mathscr{R}_t$ **contains** fewer than q points.

The Chameleon algorithm

The Chameleon algorithm (cont)

Agglomerative phase

Set $\mathscr{R}'_0 = \mathscr{R}_t$

$t = 0$

Repeat

- $t = t + 1$
- **Merge** C_i, C_j in $\mathscr{R}'_{t-1}$ to a single cluster **if**

  $RI_{ij} \geq T_{RI}$ and $RC_{ij} \geq T_{RC}$ **(A)**

  (if more than one C_j satisfy the conditions for a given C_i, the C_j with the highest $|E_{ij}|$ is selected).

Until (A) does not hold for any pair of clusters in $\mathscr{R}'_{t-1}$.

Return $\mathscr{R}'_{t-1}$

NOTE: The internal structure of two clusters to be merged is of significant importance. The more similar the elements within each cluster the higher "their resistance" in merging with another cluster.

The Chameleon algorithm

Remarks:

- Condition **(A)** can be replaced by $\left(C_i, C_j\right) = max_{(C_u, C_v)} RI_{uv} \cdot RC_{uv}^{a}$

- Chameleon is not very sensitive to the choice of the user-defined parameters k (typically it is selected between 5 and 20), q (typically chosen in the range 1% to 5% of the total number of data points), T_{RI}, T_{RC} and/or a.

- Chameleon is well suited for **large data sets** (more accurate estimation of $\left|E_{ij}\right|$, $|E_i|$, S_{ij}, S_i)

- For **large** N, the worst-case time complexity of the algorithm is $O(N(\log_2 N + m))$, where m is the number of clusters formed by the divisive phase.

The Chameleon algorithm

Example: For the clusters shown in the figure we have:

$|E_1| = 0.48$, $|E_2| = 0.48$,

$|E_3| = 1.45$, $|E_4| = 1.45$,

$|S_1| = 0.48$, $|S_2| = 0.48$,

$|S_3| = 0.725$, $|S_4| = 0.725$,

$|E_{12}| = 0.4$, $|E_{34}| = 0.6$,

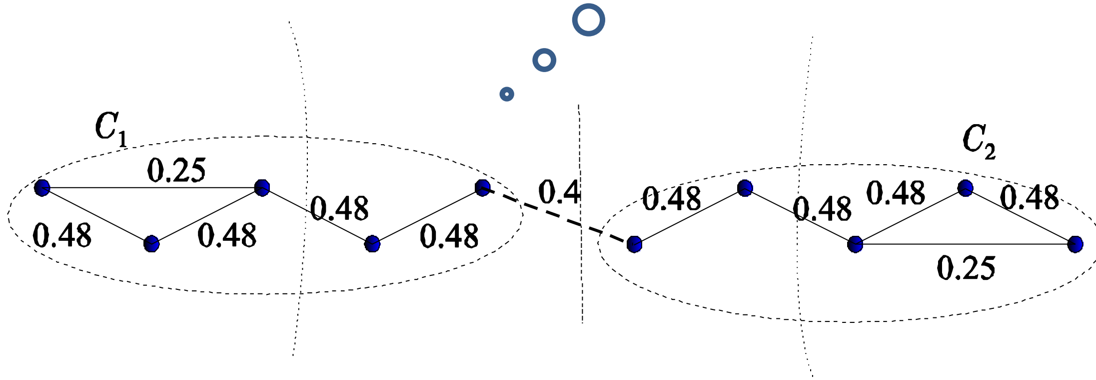
$|S_{12}| = 0.4$, $|S_{34}| = 0.6$.

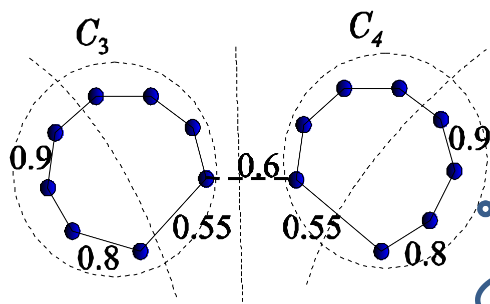
Thus,

$RI_{12} = 0.833$, $RI_{34} = 0.414$

$RC_{12} = 0.833$, $RC_{34} = 0.828$

The values in the figure stand for **similarities**.





All edge weights which are not denoted explicitly are equal to 0.9.

In conclusion: Both RI and RC **favor** the merging C_1 and C_2 **against** the merging of C_3 and C_4.

Note that the single-link algorithm would **merge** C_3 and C_4 instead of C_1 and C_2.

Other clustering algorithms

- The following types of algorithms will be considered:
  - Graph theory based clustering algorithms.
  - Competitive learning algorithms.
  - Valley seeking clustering algorithms.
  - Cost optimization clustering algorithms based on:
    - Branch and bound approach.
    - Simulated annealing methodology.
    - Deterministic annealing.
    - Genetic algorithms.
  - Density-based clustering algorithms.
  - Clustering algorithms for high dimensional data sets.

Graph theory based clustering algorithms

In principle, such algorithms are capable of detecting clusters of various shapes, at least when they are well separated.

In the sequel we discuss algorithms that are based on:

- The Minimum Spanning Tree (MST).
- Regions of influence.
- Directed trees.

Graph theory based clustering algorithms

Minimum Spanning Tree (MST) algorithms

Preliminaries: Let

- G be the complete graph, each node of which corresponds to a point of the data set X.
- $e = (\boldsymbol{x}_i, \boldsymbol{x}_j)$ denote an edge of G connecting $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.
- $w_e \equiv d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ denote the weight of the edge e.

Definitions:

- Two edges e_1 and e_2 are k steps away from each other if the minimum path that connects a vertex of e_1 and a vertex of e_2 contains $k-1$ edges.
- A Spanning Tree of G is a connected graph that:
  - Contains all the vertices of the graph.
  - Has no loops.
- The weight of a Spanning Tree is the sum of weights of its edges.
- A Minimum Spanning Tree (MST) of G is a spanning tree with minimum weight (when all w_e's are different from each other, the MST is unique).

Graph theory based clustering algorithms

Minimum Spanning Tree (MST) algorithms (cont)

Sketch of the algorithm:

- Determine the MST of G.

- Remove the edges that are "unusually" large compared with their neighboring edges (inconsistent edges).

- Identify as clusters the connected components of the MST, after the removal of the inconsistent edges.

Identification of inconsistent edges.

For a given edge e of the MST of G:

- Consider all the edges (except e) that lie k steps away (at the most) from e.

- Determine the mean m_e and the standard deviation σ_e of their weights.

- If w_e lies more than q (typically $q = 2$) standard deviations σ_e away from m_e, then:
  - e is characterized as inconsistent.
- Else
  - e is characterized as consistent.
- End if

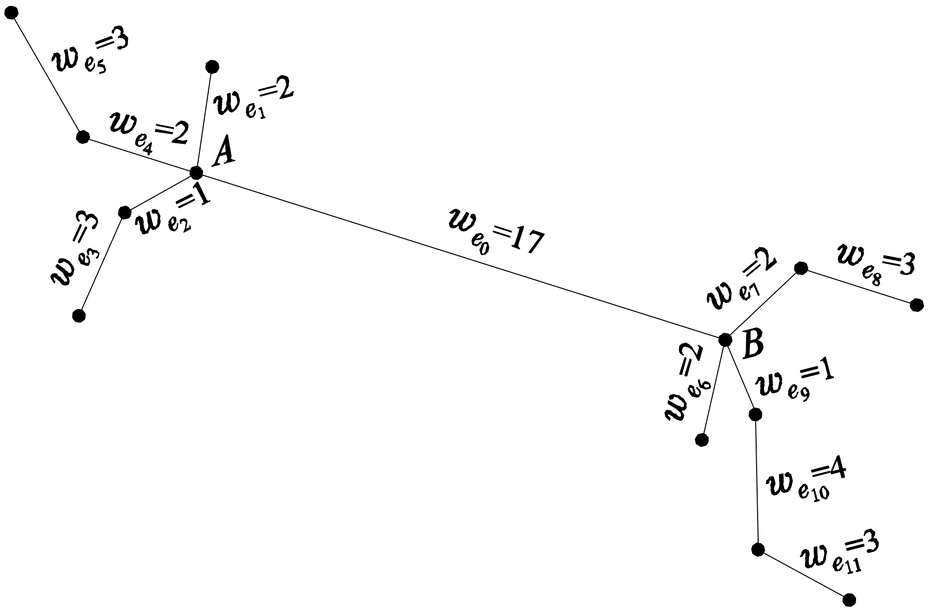
Graph theory based clustering algorithms

Minimum Spanning Tree (MST) algorithms (cont)

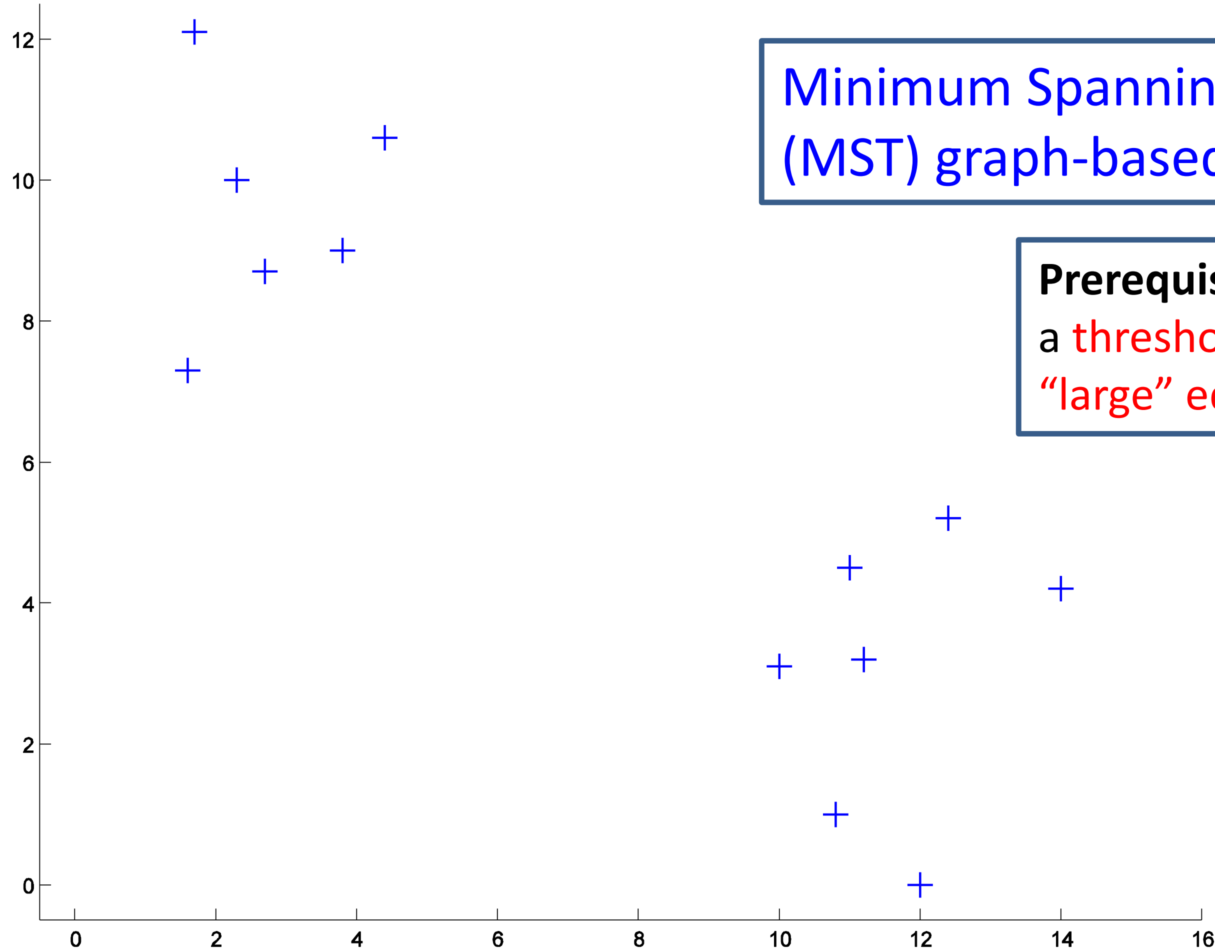
Example:

- For the MST in the figure and for $k = 2$ and $q = 3$ we have:
- For e_0: $w_{e_0} = 17, m_{e_0} = 2.3, \sigma_{e_0} = 0.95$. w_{e_0} lies 15.5 standard deviations σ_{e_0} away from m_{e_0}, hence it is inconsistent.

- For e_{11}: $w_{e_{11}} = 3, m_{e_{11}} = 2.5, \sigma_{e_{11}} = 2.12$. $w_{e_{11}}$ lies 0.24 standard deviations $\sigma_{e_{11}}$ away from $m_{e_{11}}$, hence it is consistent.



Graph theory based clustering algorithms

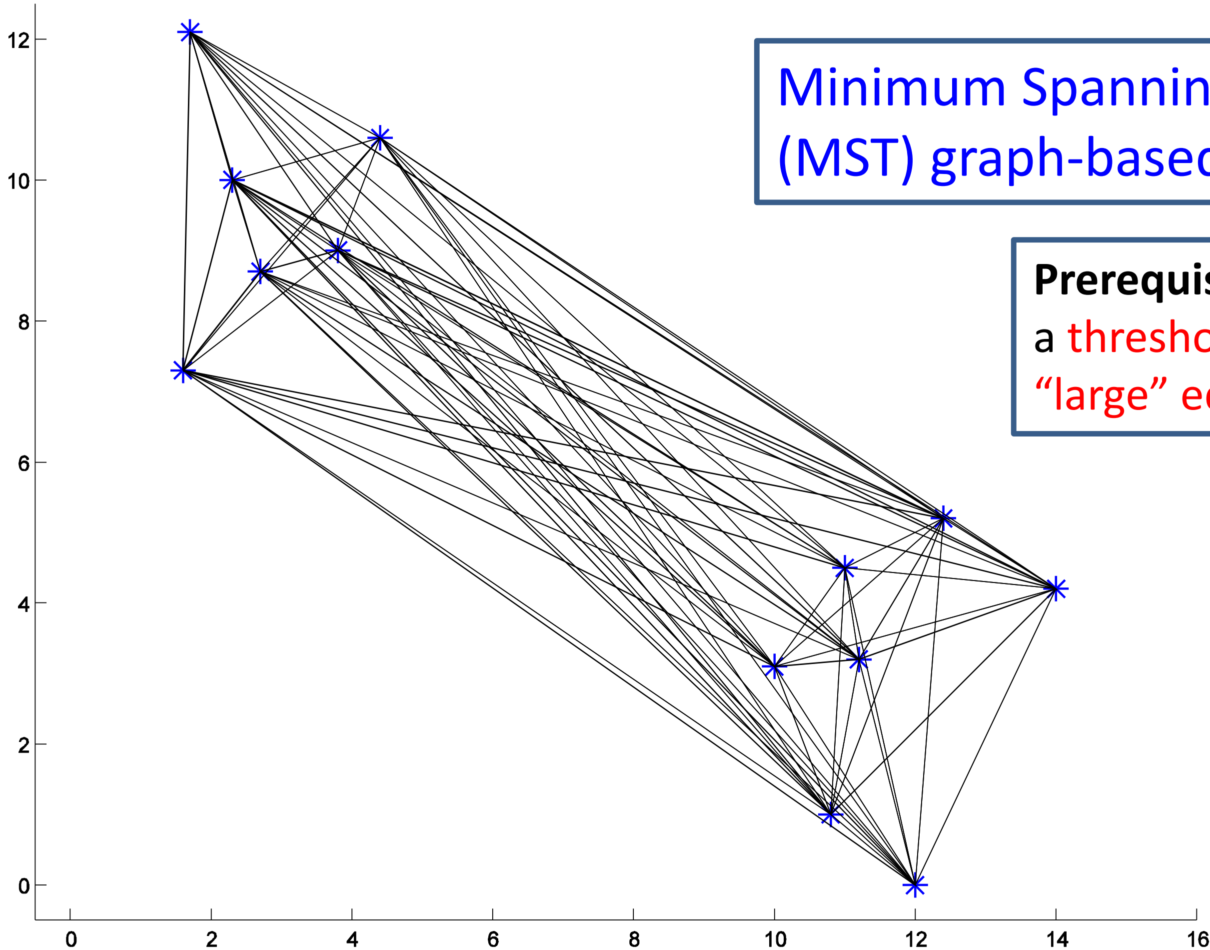


Minimum Spanning Tree (MST) graph-based algorithm

Prerequisite: Definition of a threshold for identifying "large" edges.

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph **and cut** the **"unusually large"** edges.
- The remaining sub-graphs **correspond** to the clusters.

Graph theory based clustering algorithms

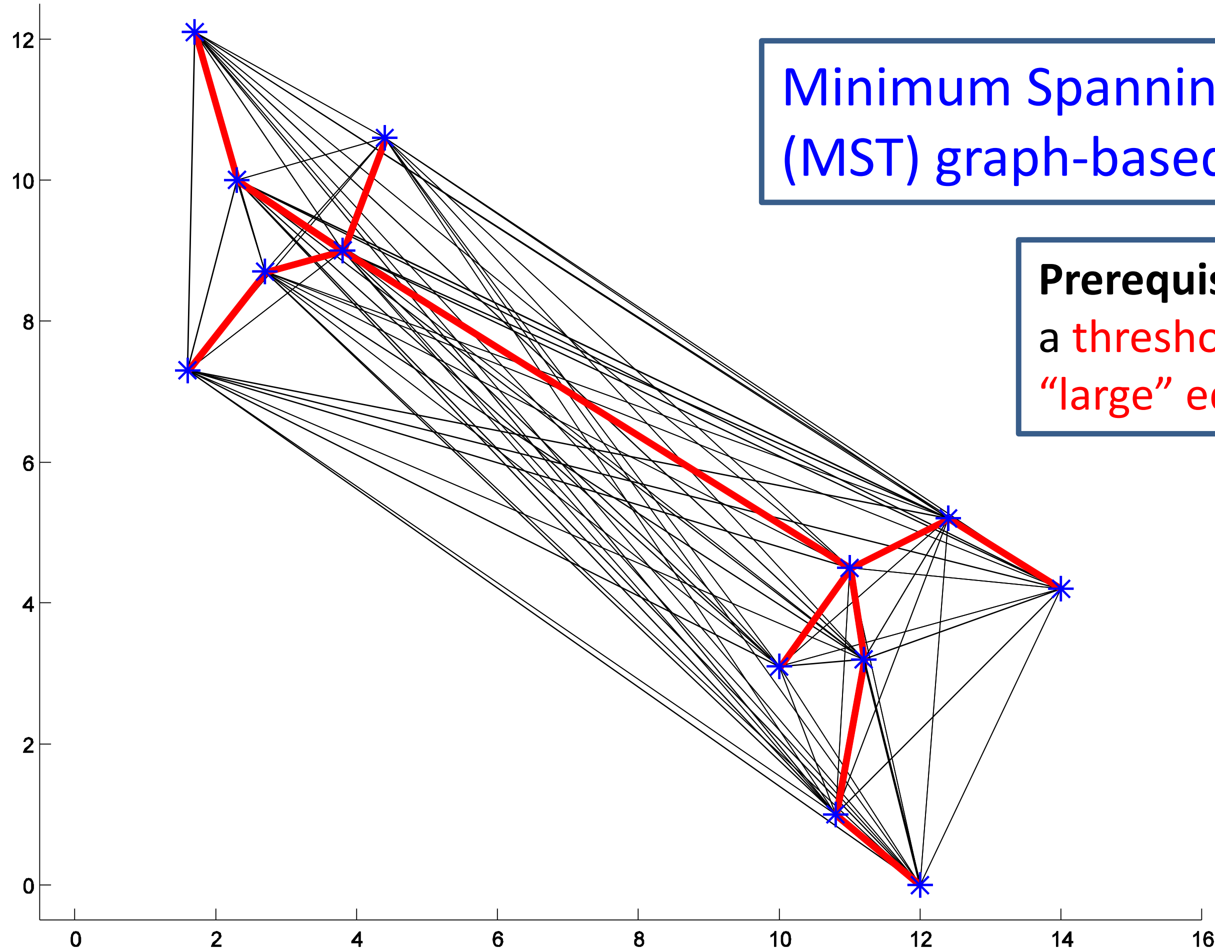


Minimum Spanning Tree (MST) graph-based algorithm

Prerequisite: Definition of a threshold for identifying "large" edges.

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph **and cut** the **"unusually large"** edges.
- The remaining sub-graphs **correspond** to the clusters.

Graph theory based clustering algorithms

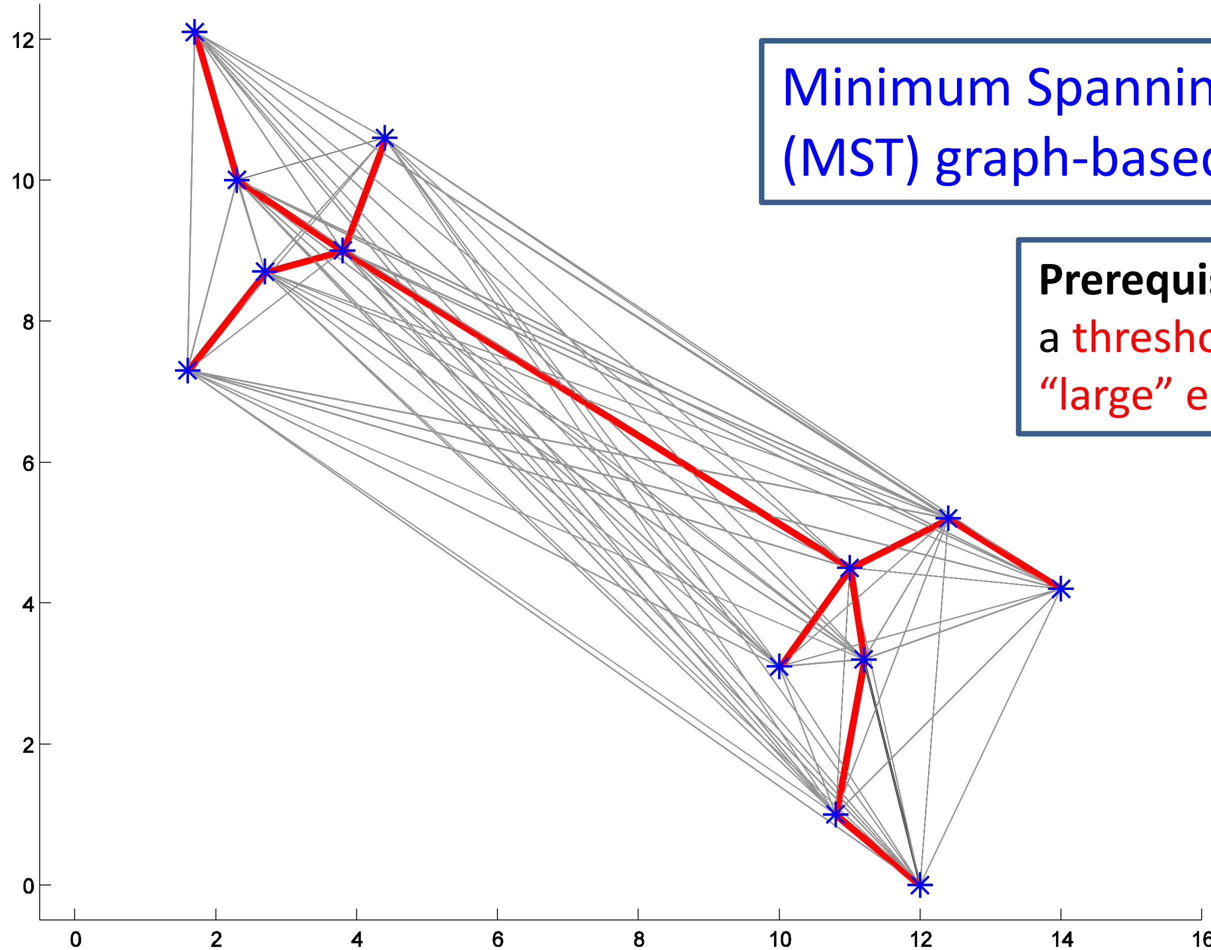


Minimum Spanning Tree (MST) graph-based algorithm

Prerequisite: Definition of a threshold for identifying "large" edges.

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph **and cut** the **"unusually large"** edges.
- The remaining sub-graphs **correspond** to the clusters.

Graph theory based clustering algorithms

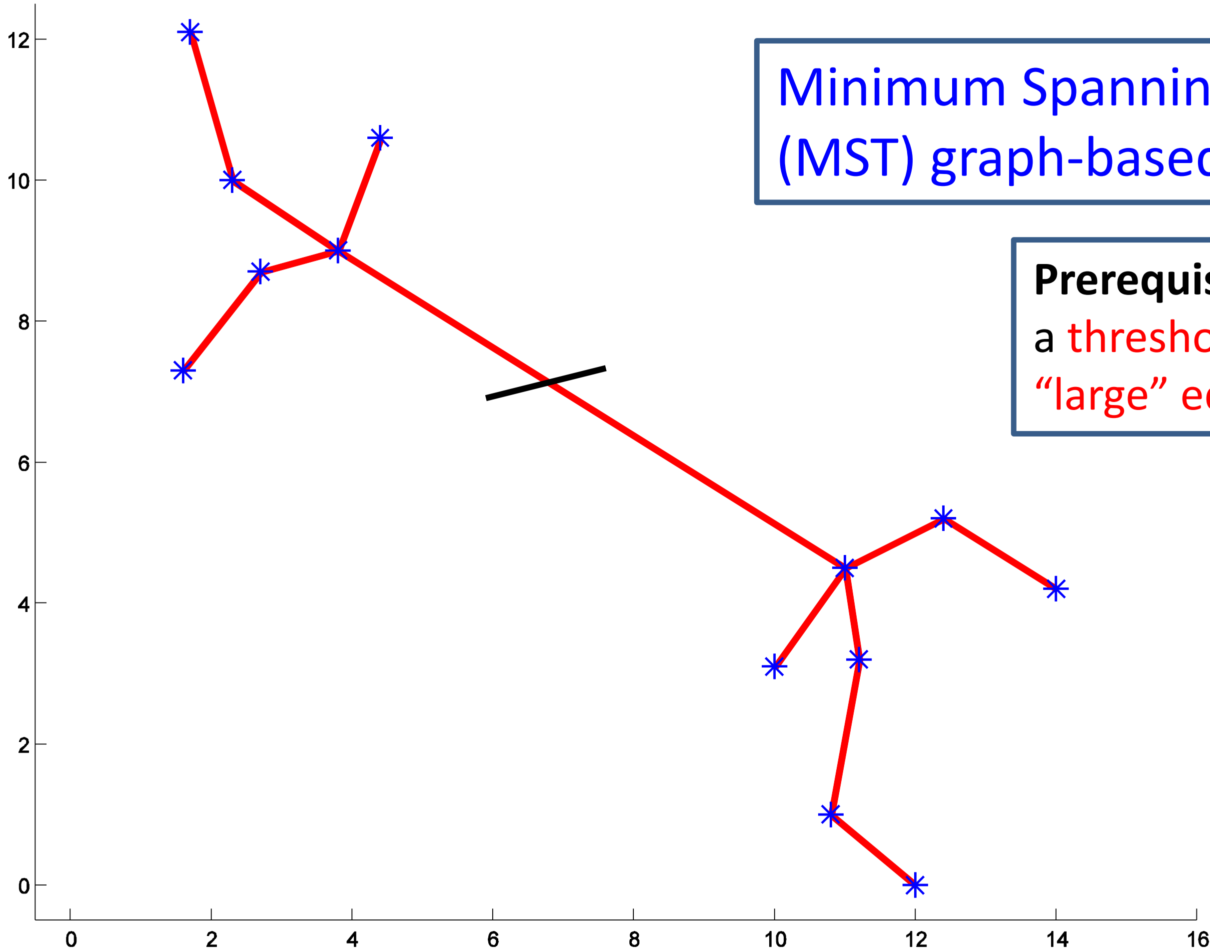


Minimum Spanning Tree (MST) graph-based algorithm

Prerequisite: Definition of a threshold for identifying "large" edges.

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph **and cut** the **"unusually large"** edges.
- The remaining sub-graphs **correspond** to the clusters.

Graph theory based clustering algorithms

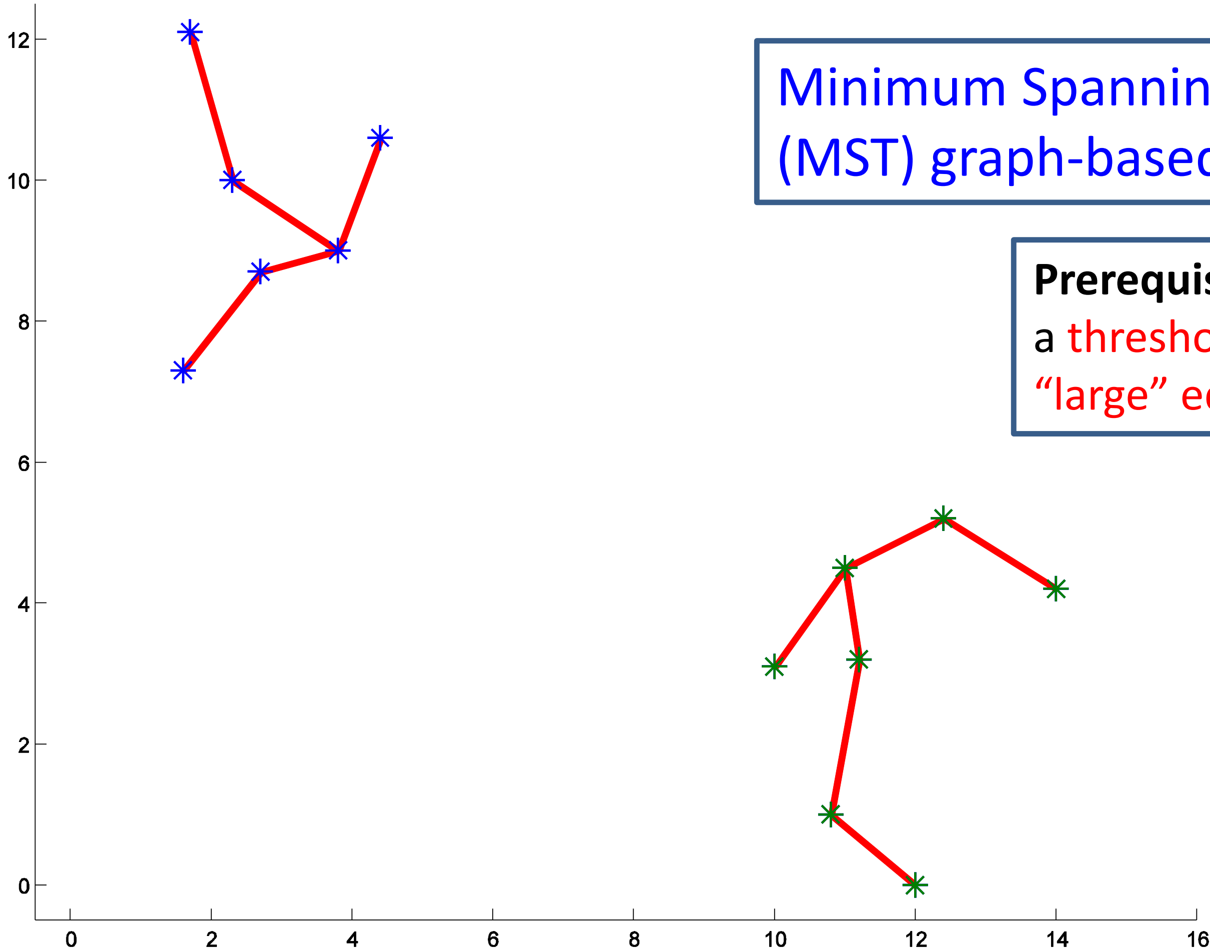


Minimum Spanning Tree (MST) graph-based algorithm

Prerequisite: Definition of a threshold for identifying "large" edges.

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph **and cut** the **"unusually large"** edges.
- The remaining sub-graphs **correspond** to the clusters.

Graph theory based clustering algorithms



Minimum Spanning Tree (MST) graph-based algorithm

Prerequisite: Definition of a threshold for identifying “large” edges.

- **Define** a **complete** graph with vertices the data points and edges the segments connecting every pair of vertices.
- **Weight** each edge by the distance between its two end-points.
- **Define** the MST of the graph **and cut** the **“unusually large”** edges.
- The remaining sub-graphs **correspond** to the clusters.

Graph theory based clustering algorithms

Minimum Spanning Tree (MST) algorithms (cont)

Remarks:

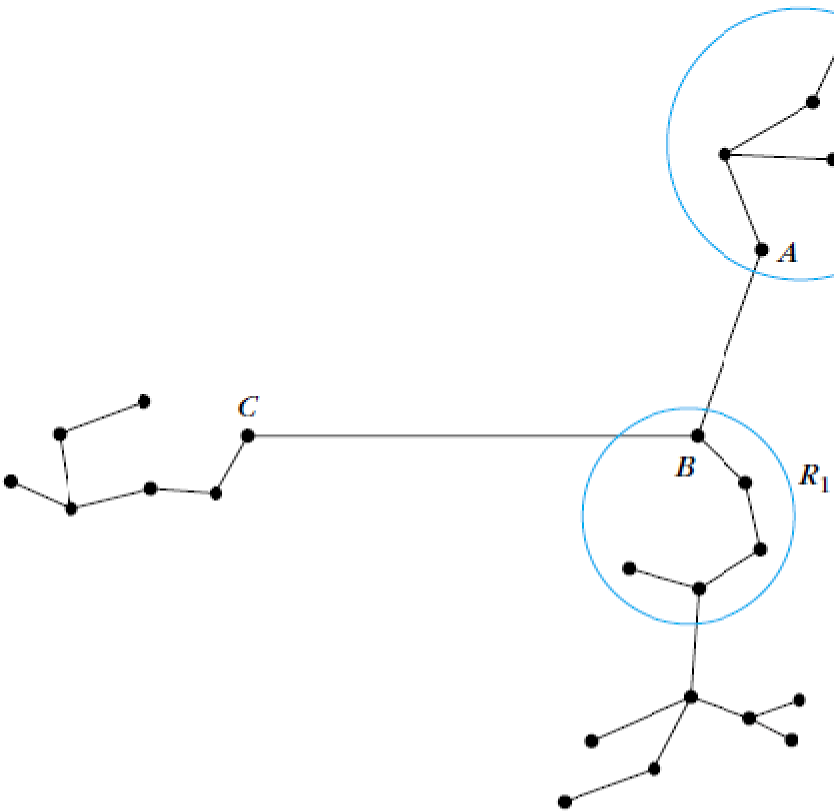
- The algorithm **depends** on the choices of k and q.
- The algorithm is insensitive to the order of consideration of the data points.
- No initial conditions are required, no convergence issues are arised.
- The algorithm works well for many cases where the clusters are well separated.

Graph theory based clustering algorithms

Minimum Spanning Tree (MST) algorithms (cont)

Remarks:

- A **problem** may occur when a "large" edge e has another "large" edge as its neighbor. In this case, e is likely not to be characterized as inconsistent and the algorithm may fail to unravel the underlying clustering structure correctly.



Example: The vectors of the regions R_1 and R_2 will probably be assigned to the same cluster.

Graph theory based clustering algorithms

Algorithms based on Regions of Influence (ROI)

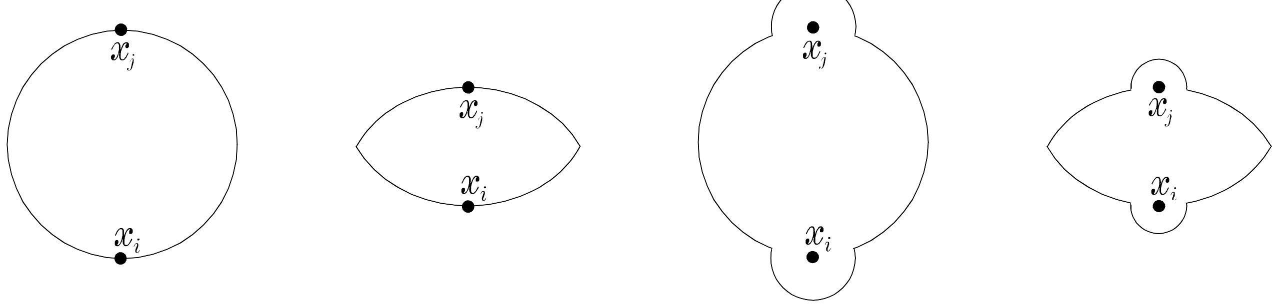
<u>Definition</u>: The region of influence of two distinct vectors $\boldsymbol{x}_i, \boldsymbol{x}_j \in X$ is defined as:

$$R(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left\{\boldsymbol{x}\colon\ cond\left(d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j), d(\boldsymbol{x}_i, \boldsymbol{x}_j)\right), \boldsymbol{x}_i \neq \boldsymbol{x}_j\right\}$$

where $cond\left(d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j), d(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)$ may be defined as:

a) $d^2(\boldsymbol{x}, \boldsymbol{x}_i) + d^2(\boldsymbol{x}, \boldsymbol{x}_j) < d^2(\boldsymbol{x}_i, \boldsymbol{x}_j)$,
b) $\max\left\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\right\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)\}$,
c) $\left(d^2(\boldsymbol{x}, \boldsymbol{x}_i) + d^2(\boldsymbol{x}, \boldsymbol{x}_j) < d^2(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)\ OR\ \left(\sigma \min\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)$,
d) $\left(\max\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)\}\right)\ OR\ \left(\sigma \min\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)$

where σ affects the size of the ROI defined by $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ and is called relative edge consistency.



(a) (b) (c) (d)

Graph theory based clustering algorithms

Algorithms based on Regions of Influence (cont)

<u>Algorithm based on ROI</u>

- For $i = 1$ to N
  - For $j = i + 1$ to N
    - **Determine** the region of influence $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$
    - If $R(\boldsymbol{x}_i, \boldsymbol{x}_j) \cap \left(X - \{\boldsymbol{x}_i, \boldsymbol{x}_j\}\right) = \emptyset$ then
      - o Add the edge connecting $\boldsymbol{x}_i, \boldsymbol{x}_j$.
    - –End if
  - End For
- End For

Determine the connected components of the resulted graph and **identify** them as clusters.

In words:

- The edge $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is **added** to the graph **if** no other $\boldsymbol{x}_q \in X$ lies in $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

- Since for $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ close to each other it is likely that $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$ contains no other vectors in X, it is expected that close to each other points will be assigned to the same cluster.

Graph theory based clustering algorithms

Algorithms based on Regions of Influence (cont)

Remarks:

- The algorithm is insensitive to the order in which the pairs are considered.
- In order to exclude (possible) edges connecting distant points, one could use a procedure like the one described previously for removing "unusually large" edges.
- In the choices of $cond$ in (c) and (d), σ must be chosen *a priori*.
- For the resulting graphs:
  - if the choice (a) is used for $cond$, they are called relative neighborhood graphs (RNGs)
  - if the choice (b) is used for $cond$, they are called Gabriel graphs (GGs)
- Experimental results show that better clusterings are produced when (c) and (d) conditions are used in the place of $cond$, instead of (a) and (b).

Graph theory based clustering algorithms

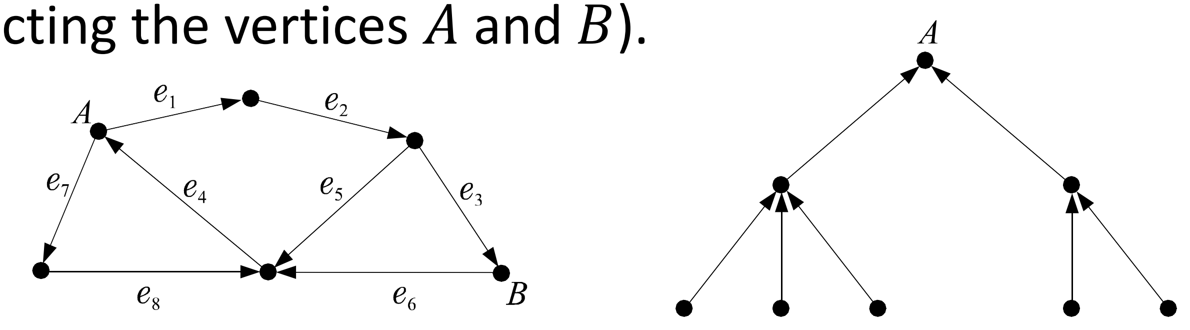
Algorithms based on Directed Trees

Definitions:

- A directed graph is a graph whose edges are directed.

- A set of edges $e_{i_1}, \dots, e_{i_q}$ **constitute** a directed path from a vertex A to a vertex B, if,
  - A is the initial vertex of e_{i_1}
  - B is the final vertex of e_{i_q}
  - The destination vertex of the edge $e_{i_j}, j = 1, \dots, q-1$, **is** the departure vertex of the edge $e_{i_{j+1}}$.

  (In figure (a) the sequence e_1, e_2, e_3 constitute a directed path connecting the vertices A and B).



(a) (b)

Graph theory based clustering algorithms

Algorithms based on Directed Trees (cont)

- A directed tree is a directed graph with a specific node A, known as root, such that,
  - From every node $B \neq A$ of the tree **departs** exactly one edge.
  - No edge **departs** from A.
  - No circles are encountered (see figure (b) in the previous slide).

- The neighborhood of a point $\boldsymbol{x}_i \in X$ is defined as

$$\rho_i(\theta) = \{\boldsymbol{x}_j \in X : d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \theta, \boldsymbol{x}_i \neq \boldsymbol{x}_j\}$$

where θ determines the neighborhood size.

- Also let
  - $n_i = |\rho_i(\theta)|$ be the number of points of X lying within $\rho_i(\theta)$
  - $g_{ij} = (n_j - n_i)/d(\boldsymbol{x}_i, \boldsymbol{x}_j)$

Main philosophy of the algorithm

Identify the directed trees in a graph whose vertices are points of X, so that each directed tree corresponds to a cluster.

Graph theory based clustering algorithms

Algorithms based on Directed Trees (cont.)

Clustering Algorithm based on Directed Trees

$g_{ij} = (n_j - n_i)/d(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- **Set** θ to a specific value.
- **Determine** n_i, $i = 1, \dots, N$.
- **Compute** g_{ij}, $i, j = 1, \dots, N$, $i \neq j$.
- For $i = 1$ to N
  - If $n_i = 0$ then
    - $\boldsymbol{x}_i$ is the root of a new directed tree.
  - Else
    - Determine $\boldsymbol{x}_r$ such that $g_{ir} = max_{x_j \in \rho_i(\theta)} g_{ij}$
    - If $g_{ir} < 0$ then
      - $\boldsymbol{x}_i$ is the root of a new directed tree.
    - Else if $g_{ir} > 0$ then
      - $\boldsymbol{x}_r$ is the parent of $\boldsymbol{x}_i$ (there exists a directed edge from $\boldsymbol{x}_i$ to $\boldsymbol{x}_r$).

Graph theory based clustering algorithms

Algorithms based on Directed Trees (cont.)

Clustering Algorithm based on Directed Trees

- Else if $g_{ir} = 0$ then
  - o Define $T_i = \{\boldsymbol{x}_j : \boldsymbol{x}_j \in \rho_i(\theta), g_{ij} = 0\}$.
  - o Eliminate all the elements $\boldsymbol{x}_j \in T_i$, for which there exists a directed path from $\boldsymbol{x}_j$ to $\boldsymbol{x}_i$.
  - o If the resulting T_i is empty then
    - * $\boldsymbol{x}_i$ is the root of a new directed tree
  - o Else
    - * The parent of $\boldsymbol{x}_i$ is $\boldsymbol{x}_q$ such that $d(\boldsymbol{x}_i, \boldsymbol{x}_q) = min_{\boldsymbol{x}_s \in T_i} d(\boldsymbol{x}_i, \boldsymbol{x}_s)$.
  - o End if
- End if

• End if

➢ End for

➢ **Identify** as clusters the directed trees formed above.

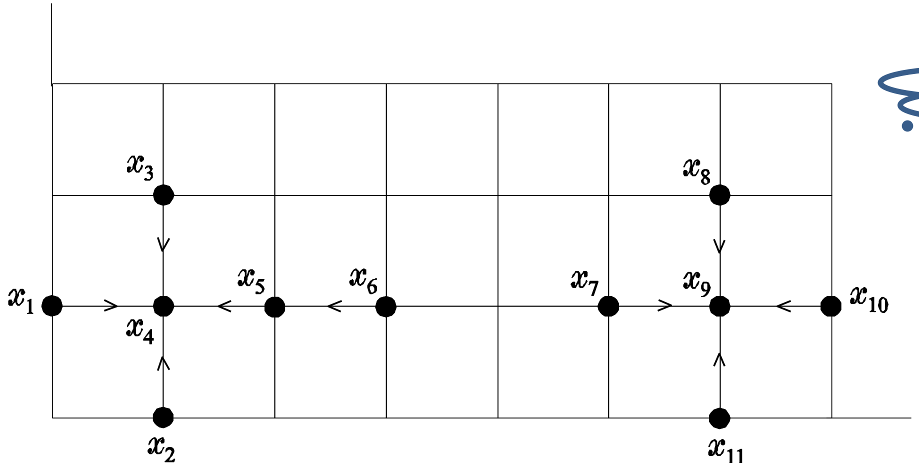
Graph theory based clustering algorithms

Algorithms based on Directed Trees (cont.)

Remarks:

- The root $\boldsymbol{x}_i$ of a directed tree is the point in $\rho_i(\theta)$ with the most dense neighborhood.
- The branch that handles the case $g_{ir} = 0$ **ensures** that no circles occur.
- The algorithm is sensitive to the order of consideration of the data points.
- For proper choice of θ and large N, this scheme **behaves** as a mode-seeking algorithm (see below).

Example: In the figure below, the size of the edge of the grid is 1 and $\theta = 1.1$. The above algorithm gives the directed trees shown in the figure.



$$g_{ij} = (n_j - n_i)/d(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

Competitive learning clustering algorithms

The main idea

- **Employ** a set of representatives $\boldsymbol{w}_j$ (in the sequel we consider only point representatives).
- **Move** them to regions of the vector space that are "dense" in vectors of X.

Comments

- In general, representatives are **updated** each time a new vector $\boldsymbol{x} \in X$ is presented to the algorithm (pattern mode algorithms).
- These algorithms do not necessarily stem from the optimization of a cost function.

The strategy

- For a given vector $\boldsymbol{x}$
  - All representatives **compete** to each other
  - The winner (representative that lies closest to $\boldsymbol{x}$) **moves** towards $\boldsymbol{x}$.
  - The losers (the rest of the representatives) either remain unchanged **or** they move towards $\boldsymbol{x}$ but at a much slower rate.

Competitive learning clustering algorithms

Generalized Competitive Learning Scheme (GCLS)

$t = 0$

$m = m_{init}$ (initial number of representatives)

(A) **Initialize** any other necessary parameters (depending on the specific algorithm).

Repeat

- $t = t + 1$
- **Present** a new randomly selected $\boldsymbol{x} \in X$ to the algorithm.
- (B) **Determine** the winning representative $\boldsymbol{w}_j$.
- (C) *If* $((\boldsymbol{x}$ *is* ***not "similar"*** *to* $\boldsymbol{w}_j(t-1))$ OR $(other\ condition))$ AND $(m < m_{max})$ *then*
  - $m = m + 1$
  - $\boldsymbol{w}_m = \boldsymbol{x}$

  Else
  - (D) *Parameter updating*

$$\boldsymbol{w}_q(t) = \begin{cases} \boldsymbol{w}_q(t-1) + \eta h\left(\boldsymbol{x}, \boldsymbol{w}_q(t-1)\right), & if\ \boldsymbol{w}_q \equiv \boldsymbol{w}_j\ (winner) \\ \boldsymbol{w}_q(t-1) + \eta' h\left(\boldsymbol{x}, \boldsymbol{w}_q(t-1)\right), & otherwise \end{cases}$$

  End

(E) **Until** (convergence occurred) OR $(t > t_{max})$

Assign each $\boldsymbol{x} \in X$ to the cluster whose representative $\boldsymbol{w}_j$ lies closest to $\boldsymbol{x}$.

maximum allowable number of clusters

maximum allowable number of iterations

Competitive learning clustering algorithms

Remarks:

- $h(\boldsymbol{x}, \boldsymbol{w}_q)$ is an appropriately defined function (see below).

- η and η' are the learning rates controlling the updating of the winner and the losers, respectively (η' may differ from looser to looser).

- A threshold of similarity Θ (carefully chosen) controls the similarity between $\boldsymbol{x}$ and its closest representative $\boldsymbol{w}_j$.
  –If $d(\boldsymbol{x}, \boldsymbol{w}_j) > \Theta$, for some distance measure, $\boldsymbol{x}$ and $\boldsymbol{w}_j$ are considered as dissimilar.

- A termination criterion may be the small variation of $\boldsymbol{W} = [\boldsymbol{w}_1^T, \dots, \boldsymbol{w}_m^T]^T$ for at least N iterations (N is the cardinality of X), i.e., for any pair of t_1, t_2, with $(p-1) \cdot N \leq t_1, t_2 \leq p \cdot N, p \in Z$, to hold $||\boldsymbol{W}(t_1) - \boldsymbol{W}(t_2)|| < \varepsilon$.

- With appropriate choices of (A), (B), (C) and (D), most competitive learning algorithms may be viewed as special cases of GCLS.

Competitive learning clustering algorithms

Basic Competitive Learning Algorithm

Here the number of representatives m is **constant**.

<u>*The algorithm*</u>

- $t = 0$
- **Repeat**
  - $t = t + 1$
  - **Present** a new randomly selected $\boldsymbol{x} \in X$ to the algorithm.
  - (B) **Determine** the winning representative $\boldsymbol{w}_j$ on $\boldsymbol{x}$ as the one for which

    $$d(\boldsymbol{x}, \boldsymbol{w}_j(t-1)) = min_{k=1,\dots,m} d(\boldsymbol{x}, \boldsymbol{w}_k(t-1)) \ (*).$$

  - (D) *Parameter updating*.

    $\eta \in (0,1)$

    $$\boldsymbol{w}_q(t) = \begin{cases} \boldsymbol{w}_q(t-1) + \eta\left(\boldsymbol{x} - \boldsymbol{w}_q(t-1)\right), & if\, \boldsymbol{w}_q \equiv \boldsymbol{w}_j \ (winner) \\ \boldsymbol{w}_q(t-1), & otherwise \end{cases}$$

  - End
- (E) **Until** (convergence occurred) OR $(t > t_{max})$
- **Assign** each $\boldsymbol{x} \in X$ to the cluster whose representative $\boldsymbol{w}_j$ lies closest to $\boldsymbol{x}$.

(*) $d(\cdot)$ may be any distance (e.g., Euclidean dist., Itakura-Saito distortion). Also, similarity measures **may** be **used** (in this case min is **replaced by** max).

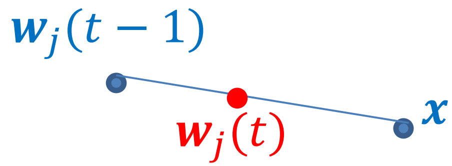
Competitive learning clustering algorithms

Basic Competitive Learning Algorithm (cont.)

Remarks:

- In this scheme losers **remain** unchanged. The winner, after the updating, **lies** in the line segment formed by $\boldsymbol{w}_j\ (t-1)$ and $\boldsymbol{x}$.

$$\boldsymbol{w}_j(t) = \boldsymbol{w}_j(t-1) + \eta\left(\boldsymbol{x} - \boldsymbol{w}_j(t-1)\right)$$
$$\Leftrightarrow \boldsymbol{w}_j(t) = (1-\eta)\boldsymbol{w}_j(t-1) + \eta\boldsymbol{x}$$



- *A priori* knowledge of the number of clusters m is required.

- If a representative is initialized far away from the regions where the points of X lie, it will never win.
  Possible solution: **Initialize** all representatives using vectors of X.

- Versions of the algorithm with variable learning rate have also been studied. Specifically, $\eta_t \rightarrow 0$, as $t \rightarrow \infty$, but not too fast(*)

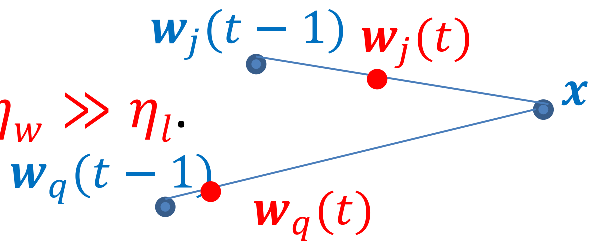
(*) $\sum_{t=1}^{\infty} \eta_t = \infty$ and $\sum_{t=1}^{\infty} {\eta_t}^2 < \infty$ (**stochastic** algorithms)

Competitive learning clustering algorithms

Leaky Learning Algorithm

The same with the Basic Competitive Learning Algorithm except part (D), the updating equation of the representatives, which becomes

$$\boldsymbol{w}_q(t) = \begin{cases} \boldsymbol{w}_q(t-1) + \eta_w\left(\boldsymbol{x} - \boldsymbol{w}_q(t-1)\right), & if\ \boldsymbol{w}_q \equiv \boldsymbol{w}_j\ (winner) \\ \boldsymbol{w}_q(t-1) + \eta_l\left(\boldsymbol{x} - \boldsymbol{w}_q(t-1)\right), & otherwise \end{cases}$$



where η_w and η_l are the learning rates in $(0, 1)$ and $\eta_w \gg \eta_l$.

Remarks:

- All representatives **move** towards $\boldsymbol{x}$ but the losers move at a much slower rate than the winner does.
- The algorithm does not suffer from the problem of poor initialization of the representatives (why?).
- An algorithm in the same spirit is the "neural-gas" algorithm, where η_l varies from loser to loser and decays as the corresponding representatives lie away from $\boldsymbol{x}$. This algorithm **results** from the optim. of a cost function.

Competitive learning clustering algorithms

Conscientious Competitive Learning Algorithms

Main Idea: **Discourage** a representative $\boldsymbol{w}_q$ from winning *if it has won many times in the past*. Do this by assigning a "conscience" to each representative.

A simple implementation

- **Equip** each representative $\boldsymbol{w}_q$, $q = 1, \ldots, m$, with a counter f_q that **counts** the times that $\boldsymbol{w}_q$ wins.
- At part (A) (initialization stage) of GCLS set $f_q = 1, q = 1, \ldots, m$.
- Define the distance $d^*(\boldsymbol{x}, \boldsymbol{w}_q)$ as

$$d^*(\boldsymbol{x}, \boldsymbol{w}_q) = d(\boldsymbol{x}, \boldsymbol{w}_q) f_q.$$

  (the distance is penalized to discourage representatives that have won many times)

- Part (B) becomes
  - The representative $\boldsymbol{w}_j$ is the winner on $\boldsymbol{x}$ if

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = min_{q=1,\ldots,m} d^*(\boldsymbol{x}, \boldsymbol{w}_q)$$

  - Set $f_j(t) = f_j(t-1) + 1$

- Parts (C) and (D) are the same as in the Basic Competitive Learning Algorithm
- Also $m = m_{init} = m_{max}$

Competitive learning clustering algorithms

Conscientious Competitive Learning Algorithms

The algorithm

- Set $f_q = 1, q = 1, \dots, m$
- $t = 0$
- **Repeat**
  - $t = t + 1$
  - **Present** a new randomly selected $\boldsymbol{x} \in X$ to the algorithm.
  - (B) **Compute** $d^*\left(\boldsymbol{x}, \boldsymbol{w}_q(t-1)\right) = d\left(\boldsymbol{x}, \boldsymbol{w}_q(t-1)\right) f_q\ ,\ q = 1, \dots, m.$

    Determine the winning representative $\boldsymbol{w}_j$ on $\boldsymbol{x}$ as the one for which

    $$d^*\left(\boldsymbol{x}, \boldsymbol{w}_j(t-1)\right) = min_{q=1,\dots,m} d^*\left(\boldsymbol{x}, \boldsymbol{w}_q(t-1)\right).$$

    Set $f_j(t) = f_j(t-1) + 1$
  - (D) *Parameter updating*

    $$\boldsymbol{w}_q(t) = \begin{cases} \boldsymbol{w}_q(t-1) + \eta\left(\boldsymbol{x} - \boldsymbol{w}_q(t-1)\right), & if\ \boldsymbol{w}_q \equiv \boldsymbol{w}_j\ (winner) \\ \boldsymbol{w}_q(t-1), & otherwise \end{cases}$$
  - End
- (E) **Until** (convergence occurred) OR $(t > t_{max})$
- **Assign** each $\boldsymbol{x} \in X$ to the cluster whose representative $\boldsymbol{w}_j$ lies closest to $\boldsymbol{x}$.

Supervised Learning Vector Quantization (VQ)

In this case

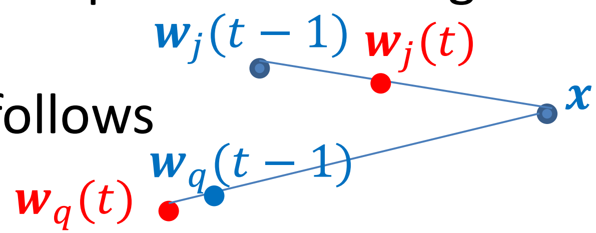
- each cluster is **treated** as a class (m compact classes are assumed)
- the available vectors have known class labels.

The goal:

Use a set of m representatives and place them in such a way so that each class is “optimally” represented.

The simplest version of VQ (LVQ1) may be obtained from GCLS as follows:

- Parts (A), (B) and (C) are the same with the basic competitive learning scheme.
- In part (D) the updating for $\boldsymbol{w}_j$’ s is carried out as follows



$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta(t)\left(\boldsymbol{x} - \boldsymbol{w}_j(t-1)\right), & if\ \boldsymbol{w}_j\ correctly\ wins\ on\ \boldsymbol{x} \\ \boldsymbol{w}_j(t-1) - \eta(t)\left(\boldsymbol{x} - \boldsymbol{w}_j(t-1)\right), & if\ \boldsymbol{w}_j wrongly\ wins\ on\ \boldsymbol{x} \\ \boldsymbol{w}_j(t-1), & otherwise \end{cases}$$

Supervised Learning Vector Quantization (VQ)

The algorithm

- $t = 0$
- **Repeat**
  - $t = t + 1$
  - **Present** a new randomly selected $\boldsymbol{x} \in X$ to the algorithm.
  - (B) **Determine** the winning representative $\boldsymbol{w}_j$ on $\boldsymbol{x}$ as the one for which

$$d\big(\boldsymbol{x}, \boldsymbol{w}_j(t-1)\big) = \min_{k=1,\ldots,m} d(\boldsymbol{x}, \boldsymbol{w}_k(t-1))$$

  - (D) *Parameter updating*

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta(t)\big(\boldsymbol{x} - \boldsymbol{w}_j(t-1)\big), & if\ \boldsymbol{w}_j\ correctly\ wins\ on\ \boldsymbol{x} \\ \boldsymbol{w}_j(t-1) - \eta(t)\big(\boldsymbol{x} - \boldsymbol{w}_j(t-1)\big), & if\ \boldsymbol{w}_j wrongly\ wins\ on\ \boldsymbol{x} \\ \boldsymbol{w}_j(t-1), & otherwise \end{cases}$$

- (E) **Until** (convergence occurred) OR $(t > t_{max})$ (max allowable no of iter.)

In words:

- $\boldsymbol{w}_j$ is moved:
  - Towards $\boldsymbol{x}$ if $\boldsymbol{w}_j$ wins and $\boldsymbol{x}$ belongs to the j-th class.
  - Away from $\boldsymbol{x}$ if $\boldsymbol{w}_j$ wins and $\boldsymbol{x}$ does not belong to the j-th class.
- All other representatives remain unaltered.