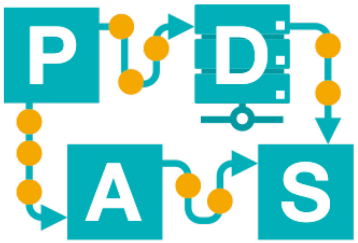
Clustering

Lecture 10

IDS-L10

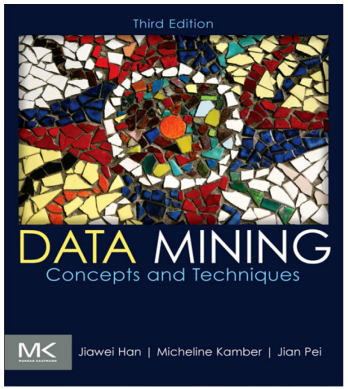


Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Outline of Today's Lecture

- Unsupervised learning
- Clustering: A few basics
- k-means clustering
- k-medoids clustering
- Agglomerative Hierarchical Clustering
- Density-based clustering using DBSCAN
- Self-organizing maps

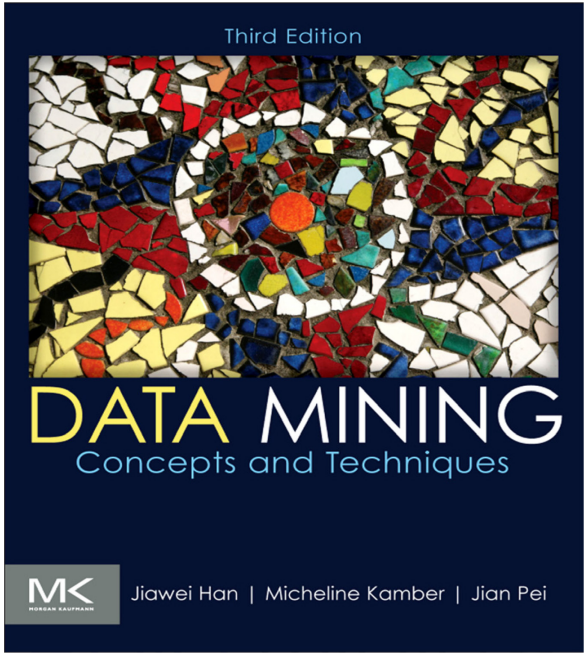


Chapter 8 (and Section 2.4) of Data Mining: Concepts and Techniques (3rd Edition) by Jiawei Han, Micheline Kamber, and Jian Pei. Morgan Kaufmann, June 2011.

Chair of Process and Data Science

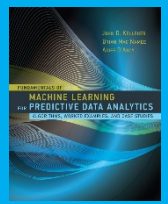
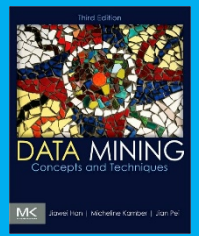
Acknowledgement

The lecture mostly based on Chapter 8 (and Section 2.4) of Data Mining: Concepts and Techniques (3rd Edition) by Jiawei Han, Micheline Kamber, and Jian Pei. Morgan Kaufmann, June 2011.



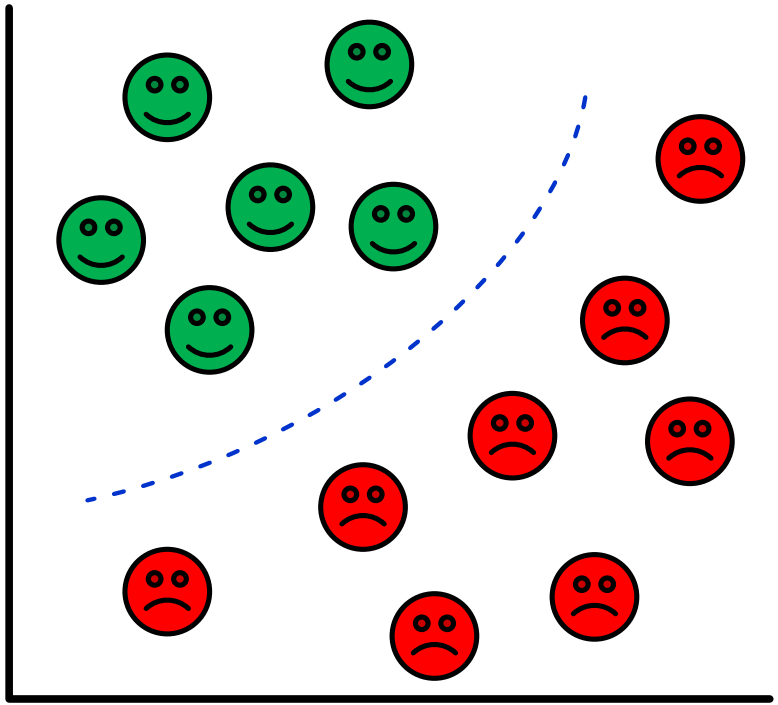
This book will be the main source of information for unsupervised learning (expect for process mining lectures).

Terminology: Don’t be confused

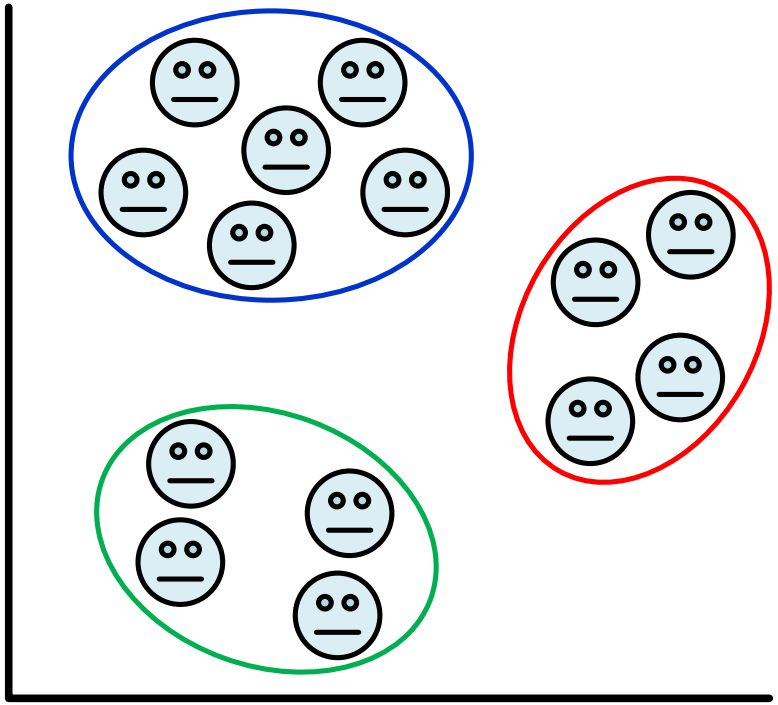
 Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies by John D. Kelleher, Brian Mac Namee and Aoife D’Arcy. MIT Press. ISBN: 9780262029445, 624 pages, July 2015	 Data Mining: Concepts and Techniques (3rd edition) by Jiawei Han , Micheline Kamber , Jian Pei. The Morgan Kaufmann Series in Data Management Systems, Elsevier. ISBN: 9780123814807, 744 pages, 2011
instance	object/tuple
feature	attribute
target feature	class label/target class
…	…

Unsupervised learning

Unsupervised learning: No target feature



supervised



unsupervised

Chair of Process and Data Science

Forms of unsupervised learning

- Clustering (grouping similar instances)
- Frequent item sets
- Association rules

static

- Sequence mining
- Process discovery (part of process mining)

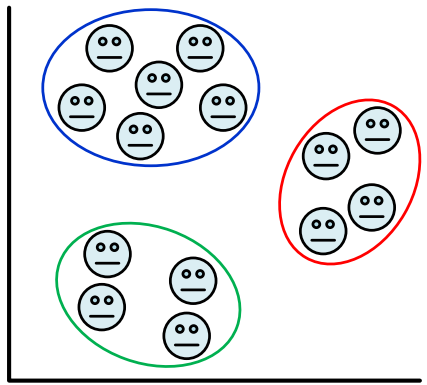
dynamic

Clustering: A few basics

Clustering

- **Find clusters such that:**
  - Instances within the cluster are similar.
  - Instances in different clusters are dissimilar.
- **Applications:**
  - To find unexpected groups.
  - As a preprocessing step.

    e.g., discover process models for each cluster



User	Song 1	Song 2	Song 3	Song 4	Song 5	…
User 1	4	0	2	1	1	…
User 2	0	3	1	2	0	…
…	…	…	…	…	…	…

Customer	Prod 1	Prod 2	Prod 3	Prod 4	Prod 5	…
Customer 1	1	0	1	1	1	…
Customer 2	0	1	1	1	0	…
…	…	…	…	…	…	…

sale	Butter	Bread	wine	beer	chips	…
sale 1	1	2	0	0	1	…
sale 2	0	0	5	10	3	…
…	…	…	…	…	…	…

Brand	Model	Top speed	Weight	HP	0-100	…
Porsche	911 GT3	319	1430	500	3.4	…
Porsche	911 GT2RS	340	1470	700	2.8	…
Suzuki	Alto	150	880	50	17.0	…
Renault	Kangoo	146	1364	50	20.3	…
…	….	…	…	…	…	…

Approaches

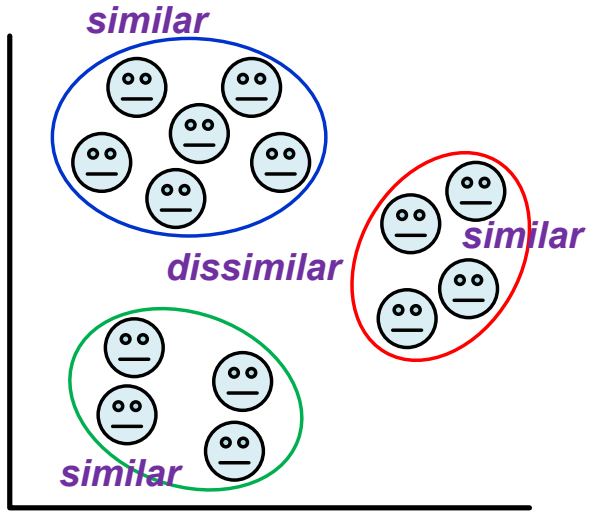
- Partitioning methods (split into subsets):
  - Centroid-based (e.g., *k*-means)
  - Medoids-based (e.g., *k*-medoids)
- Hierarchical methods (build dendrogram):
  - Agglomerative (bottom-up)
  - Divisive (top-down)
- Density-based methods
  - e.g. DBSCAN
- Grid-based methods

Approaches

- Partitioning methods (split into subsets):
  - Centroid-based (e.g., *k*-means)
  - Medoids-based (e.g., *k*-medoids)
- Hierarchical methods (build dendrogram):
  - Agglomerative (bottom-up)
  - Divisive (top-down)
- Density-based methods
  - e.g. DBSCAN
- Grid-based methods

Key concept: Similarity/Dissimilarity

- We aim to find clusters such that (1) instances within the cluster are similar and (2) instances in different clusters are dissimilar.
- Hence, we need a similarity/dissimilarity notion.
- Default is Euclidean distance.

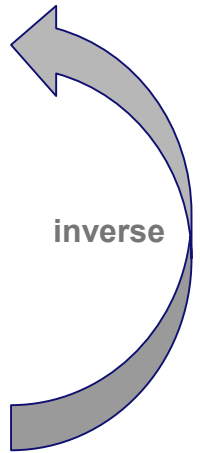


Chair of Process and Data Science

Key concept: Similarity/Dissimilarity

$$i = (x_{i1}, x_{i2}, \ldots, x_{ip}) \quad \Longleftrightarrow \quad j = (x_{j1}, x_{j2}, \ldots, x_{jp})$$

- **Similarity** (also called proximity)
  - Numerical measure of how alike two instances are.
  - Value is higher when instances are more alike.
  - Often falls in the range [0,1].
- **Dissimilarity** (e.g., distance)
  - Numerical measure of how different two instances are.
  - Lower when instances are more alike.
  - Minimum dissimilarity is often 0.
  - Upper limit varies.



Key concept: Similarity/Dissimilarity

- **Categorical features (examples):**
  - **Equal: distance = 0, similarity = 1**
  - **Different: distance = 1, similarity = 0**
    (Can be combined with one-hot encoding.)
- **Continuous features:**
  - **Euclidean distance** $d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}$
  - **Manhattan distance** $d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|$
  - **Minkowski distance** **(generalization of the two above)**
  - **Cosine similarity** **(non-metric measure)**

Metric Space

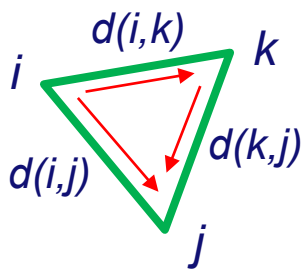
$$i = (x_{i1}, x_{i2}, \ldots, x_{ip}) \quad \text{distance} \quad j = (x_{j1}, x_{j2}, \ldots, x_{jp})$$

Non-negativity: $d(i, j) \geq 0$: Distance is a non-negative number.

Identity of indiscernibles: $d(i, i) = 0$: The distance of an object to itself is 0.

Symmetry: $d(i, j) = d(j, i)$: Distance is a symmetric function.

Triangle inequality: $d(i, j) \leq d(i, k) + d(k, j)$: Going directly from object i to object j in space is no more than making a detour over any other object k.



Chair of Process and Data Science

Minkowski distance

$$d(i,j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h}$$

$$h \geq 1$$

Manhattan distance (*h=1*)

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|$$

Euclidian distance (*h=2*)

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}$$

Supremum distance ($h \to \infty$)

Minkowski distance

$$d(i,j) = \sqrt[h]{|x_{i1} - x_{j1}|^h + |x_{i2} - x_{j2}|^h + \cdots + |x_{ip} - x_{jp}|^h}$$

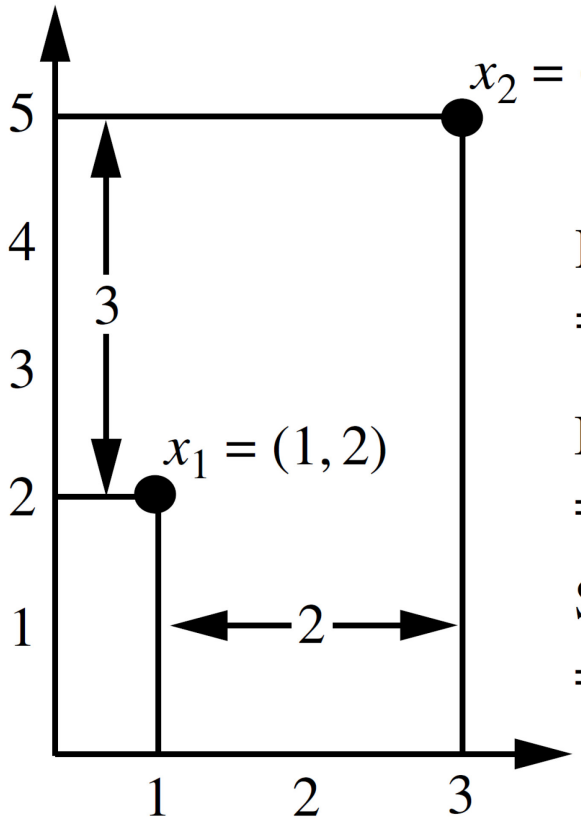
Manhattan distance (*h*=1)

$$d(i,j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \cdots + |x_{ip} - x_{jp}|$$

Euclidian distance (*h*=2)

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{ip} - x_{jp})^2}$$

Supremum distance (*h*→ ∞)



Euclidean distance
$= (2^2 + 3^2)^{1/2} = 3.61$

Manhattan distance
$= 2 + 3 = 5$

Supremum distance
$= 5 - 2 = 3$

Mixing different types of features

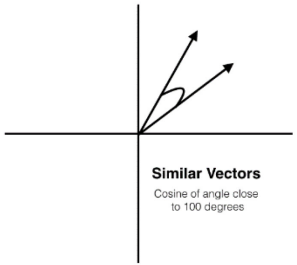
- Commonly used approach: normalize all feature ranges to [0,1].
- Not all features are equal:
  - Exclude features, e.g., because they should not / cannot play a role or because they would lead to obvious clusters (providing no insights).
  - One can give different weights to different features (i.e., distances are not the same in all dimensions).

Cosine similarity (non-metric)

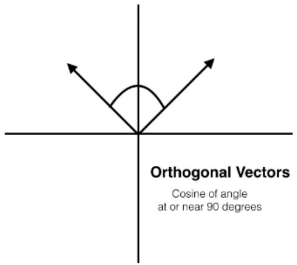
For sparse data (focus on angle rather than distance).

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| ||\boldsymbol{y}||}$$

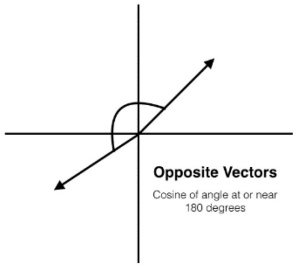
approaches 1.0



approaches 0.0



approaches -1.0



Document Vector or Term-Frequency Vector

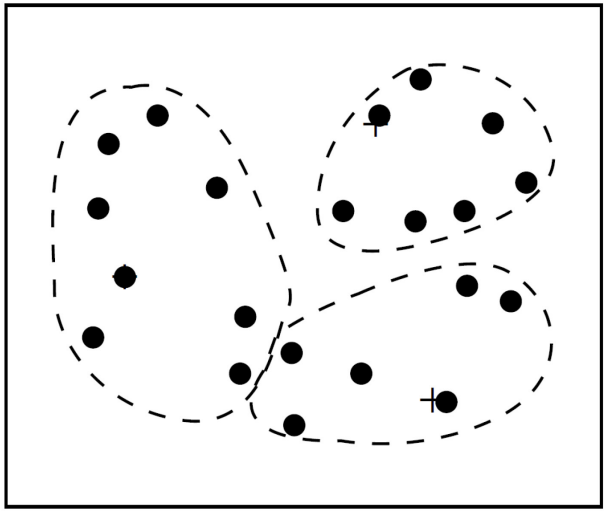
Document	team	coach	hockey	baseball	soccer	penalty	score	win	loss	season
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

k-means clustering

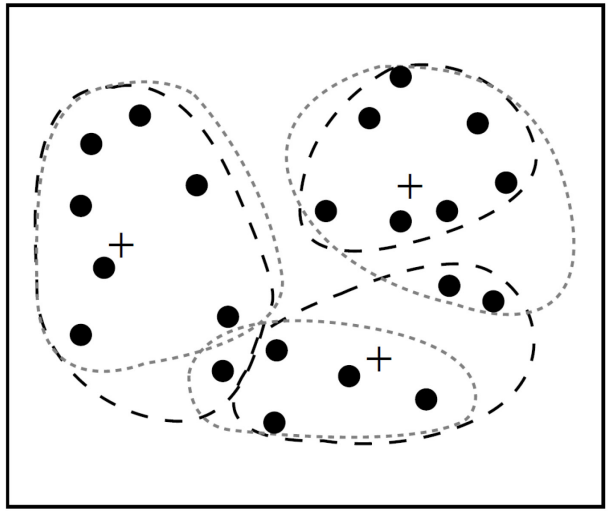
k-means: Idea

- Randomly pick *k* centriods
- Add each instance to the closest centroid.
- Recompute the centroids
  (the centroid is the center, i.e., “average” point, of the cluster)
- Add each instance to the closest centroid.
- Recompute the centroids (average).
- …
- Repeat until no changes occur anymore.

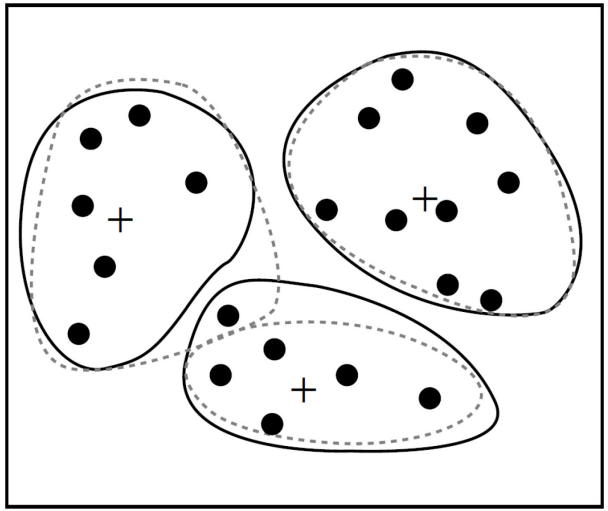
k-means: Example



(a) Initial clustering

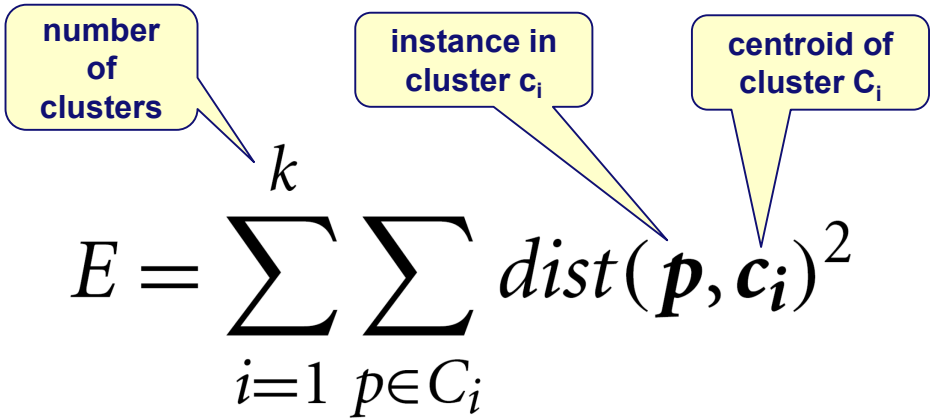


(b) Iterate



(c) Final clustering

k-means: Quality of clusters



- **Error is the sum of all squared errors between all instances in the cluster and the corresponding centroids.**
- **Other quality measures possible (e.g. sum).**

k-means: Algorithm

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
(5) **until** no change;

k-means: Design choices

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center is represented by the mean value of the objects in the cluster.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

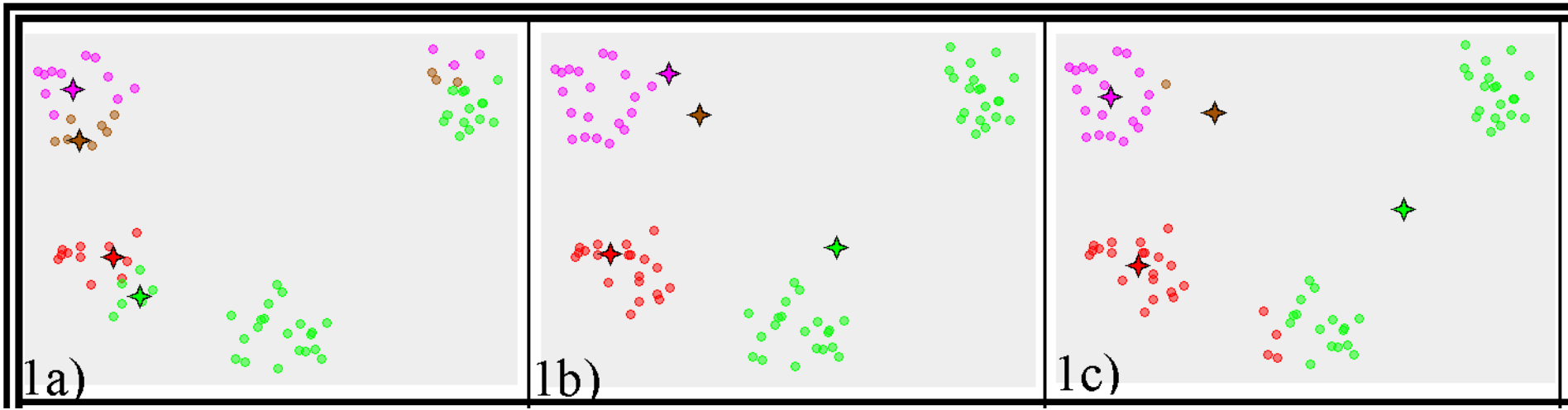
(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;
(2) **repeat**
(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;
(5) **until** no change;

number of clusters needs to be decided upfront

initialization of initial centroids

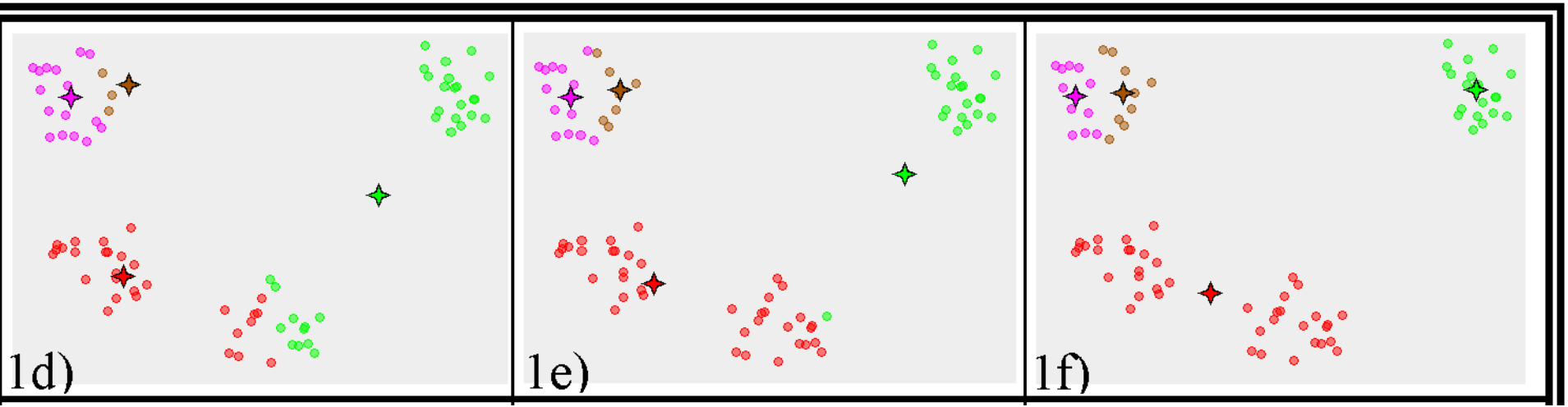
distance notion

Example *k*-means



The illustration was prepared with the Java applet, E.M. Mirkes, K-means and K-medoids: applet. University of Leicester, 2011.

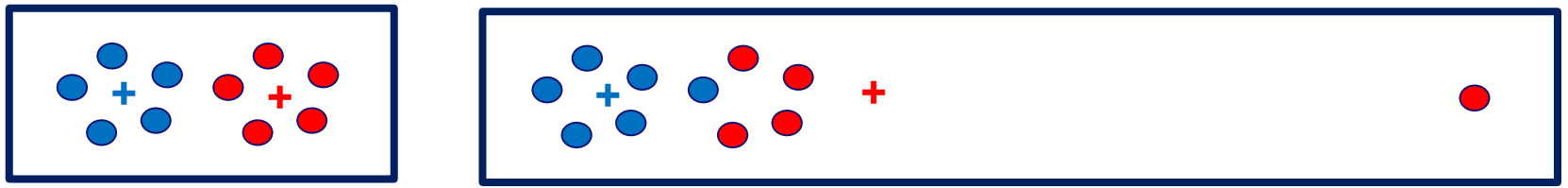
Example *k*-means



The illustration was prepared with the Java applet, E.M. Mirkes, K-means and K-medoids: applet. University of Leicester, 2011.

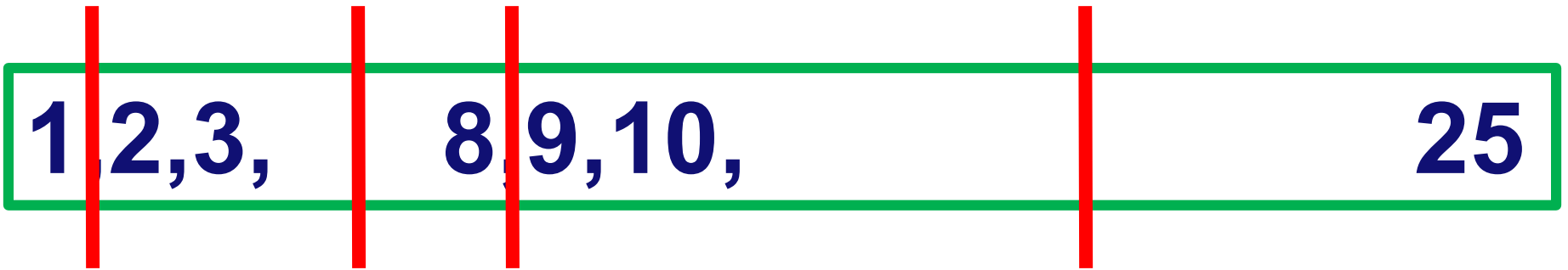
k-means: Limitations

- Typically finds a local optimum. Therefore, the algorithms is often run multiple times with random initializations and the best result is returned.
- Number of clusters needs to be decided beforehand.
- Sensitive to outliers.



Example (one feature)

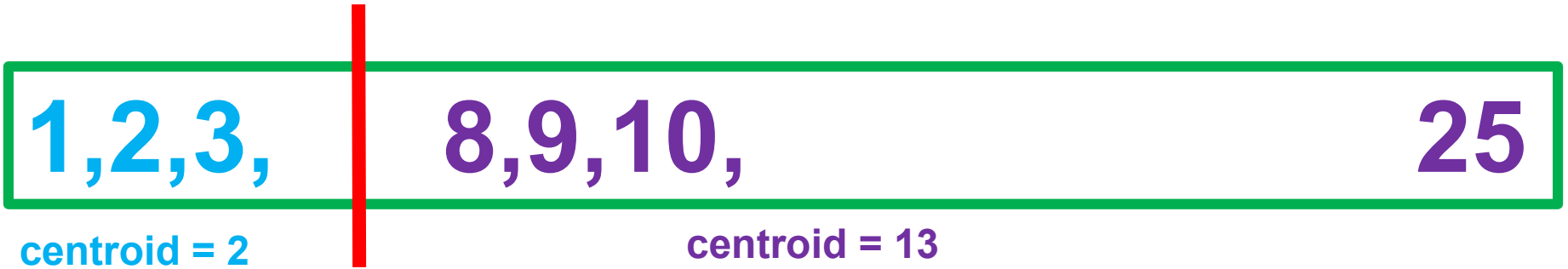
- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-means.



***k*-means**

Example (one feature)

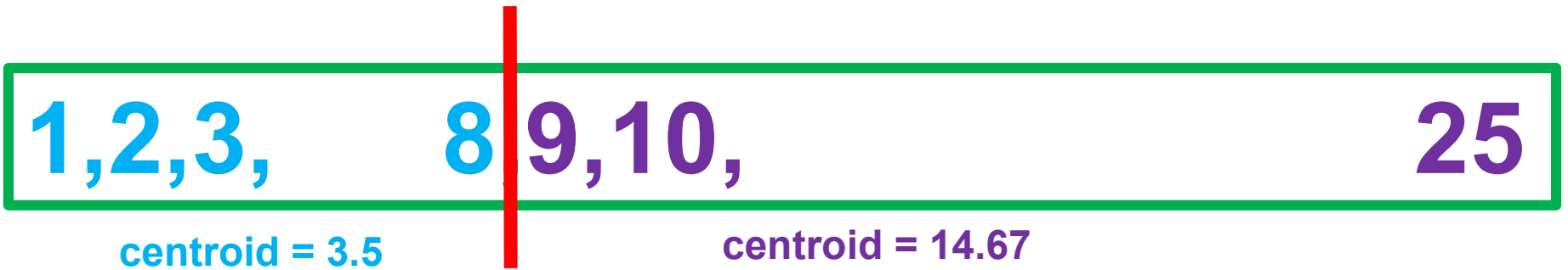
- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-means.



Chair of Process and Data Science

Example (one feature)

- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-means.



Example (one feature)

- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-means.
- The within cluster variation for {1,2,3},{8,9,10,25} is 196. $(1-2)^2+(2-2)^2+(3-2)^2+(8-13)^2+(9-13)^2+(10-13)^2+(25-13)^2=196$
- The within cluster variation for {1,2,3,8},{9,10,25} is 189.67.

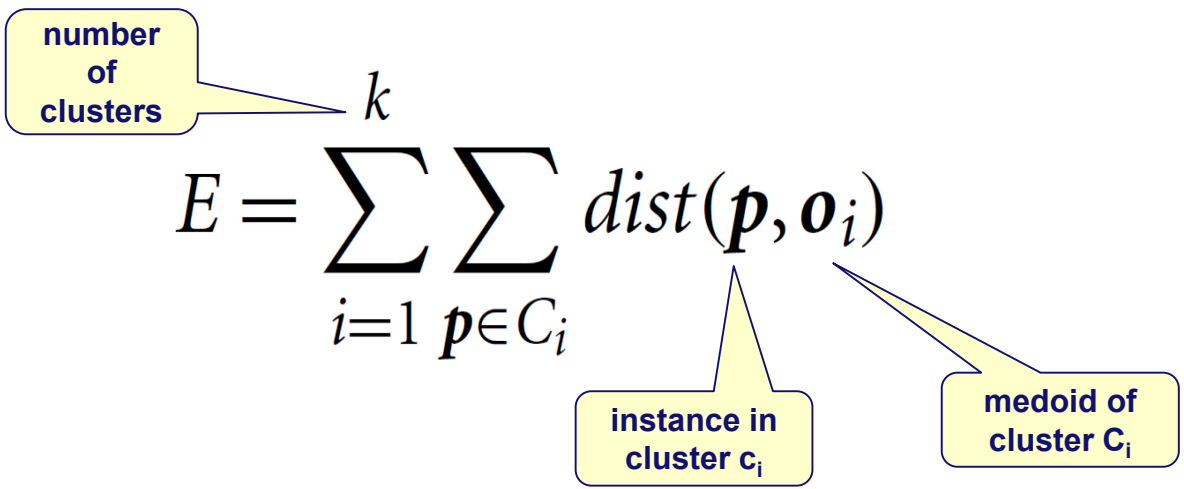
$$(1-3.5)^2+(2-3.5)^2+(3-3.5)^2+(8-3.5)^2+(9-14.67)^2$$
$$+(10-14.67)^2+(25-14.67)^2=189.67$$

lowest value, but not a good cluster (also the centroid 14.67 is far away from any value)

k-medoids clustering

Key idea

- **Use concrete instances (medoids) as “centers” rather than “averages” (centroids).**
- **Error is based on distances:**



Sum of all distances between instances are corresponding medoids.

Example (one feature)

- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-medoids.

1,2,3, 8,9,10, 25

Example (one feature)

- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-medoids.

0+1+2=3

0+1+2+17=20

E = 3+20 = 23

Example (one feature)

- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-medoids.

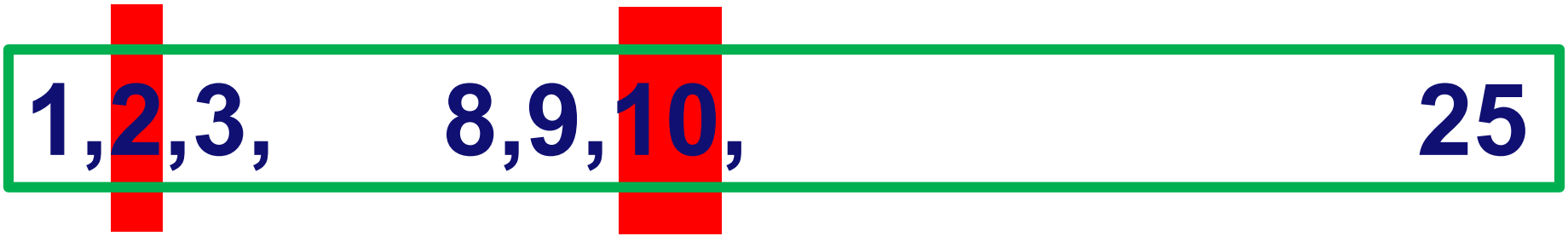
1+0+1=2

1+0+1+16=18

E = 2+18 = 20

Example (one feature)

- Consider 8 instances having one feature with values 1,2,3,8,9,10,25.
- Set *k*=2 and apply *k*-medoids.



1+0+1=2

2+1+0+15=18

E = 2+18 = 20

k-medoids clustering

- Randomly pick a set o_1, o_2, …, o_k instances (medoids).
- Assign all instances to the closest medoid.
- Pick a non-medoid o_{random}.
- Pick a current medoid o_j.
- Consider o_1, o_2, …, o_{j-1}, o_{random}, o_{j+1}, …, o_k as a candidate set of mediods.
- If this is an improvement, then keep the candidate set and pick another non-medoid o_{random} and current medoid o_j and repeat.
- If there is no improvement, pick another non-medoid o_{random} and current medoid o_j and repeat.
- Continue until there is no improvement possible.

k-medoids algorithm

Algorithm: *k*-**medoids.** PAM, a *k*-medoids algorithm for partitioning based on medoid or central objects.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects in *D* as the initial representative objects or seeds;
(2) **repeat**
(3) assign each remaining object to the cluster with the nearest representative object;
(4) randomly select a nonrepresentative object, o_{random};
(5) compute the total cost, *S*, of swapping representative object, o_j, with o_{random};
(6) **if** $S < 0$ **then** swap o_j with o_{random} to form the new set of *k* representative objects;
(7) **until** no change;

k-medoids algorithm

Algorithm: *k*-**medoids.** PAM, a *k*-medoids algorithm for partitioning based on medoid or central objects.

Input:

- *k*: the number of clusters,
- *D*: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects in *D* as the initial representative objects or seeds;
(2) **repeat**
(3) assign each remaining object to the cluster with the nearest representative object;
(4) randomly select a nonrepresentative object, o_{random};
(5) compute the total cost, *S*, of swapping representative object, o_j, with o_{random};
(6) **if** $S < 0$ **then** swap o_j with o_{random} to form the new set of *k* representative objects;
(7) **until** no change;

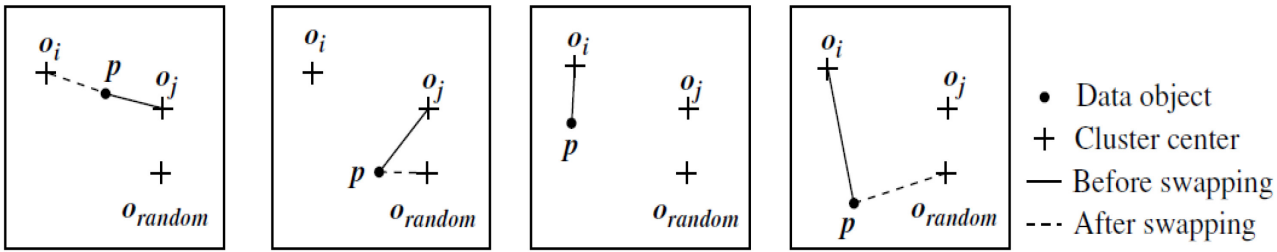
again the number of clusters needs to be decided upfront

initialization of initial medoids

distance notion

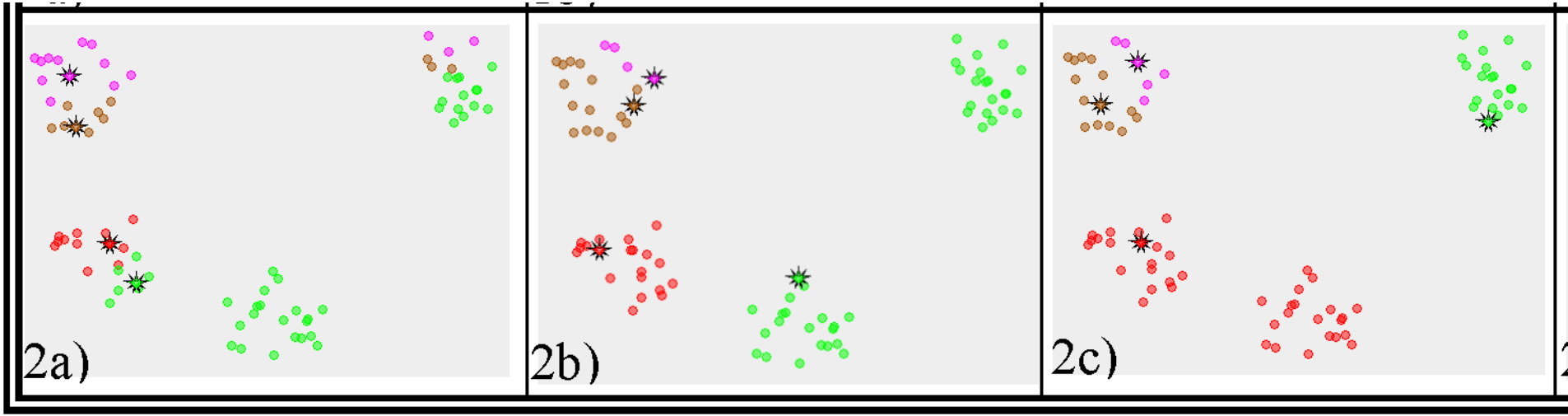
Updates can be done efficiently

- Suppose $o_1, o_2, \ldots, o_k$ is replaced by $o_1, o_2, \ldots, o_{j-1}, o_{random}, o_{j+1}, \ldots, o_k$.
- Updates only need to reconsider neighborhoods close to o_{j-1}, and o_{random}.



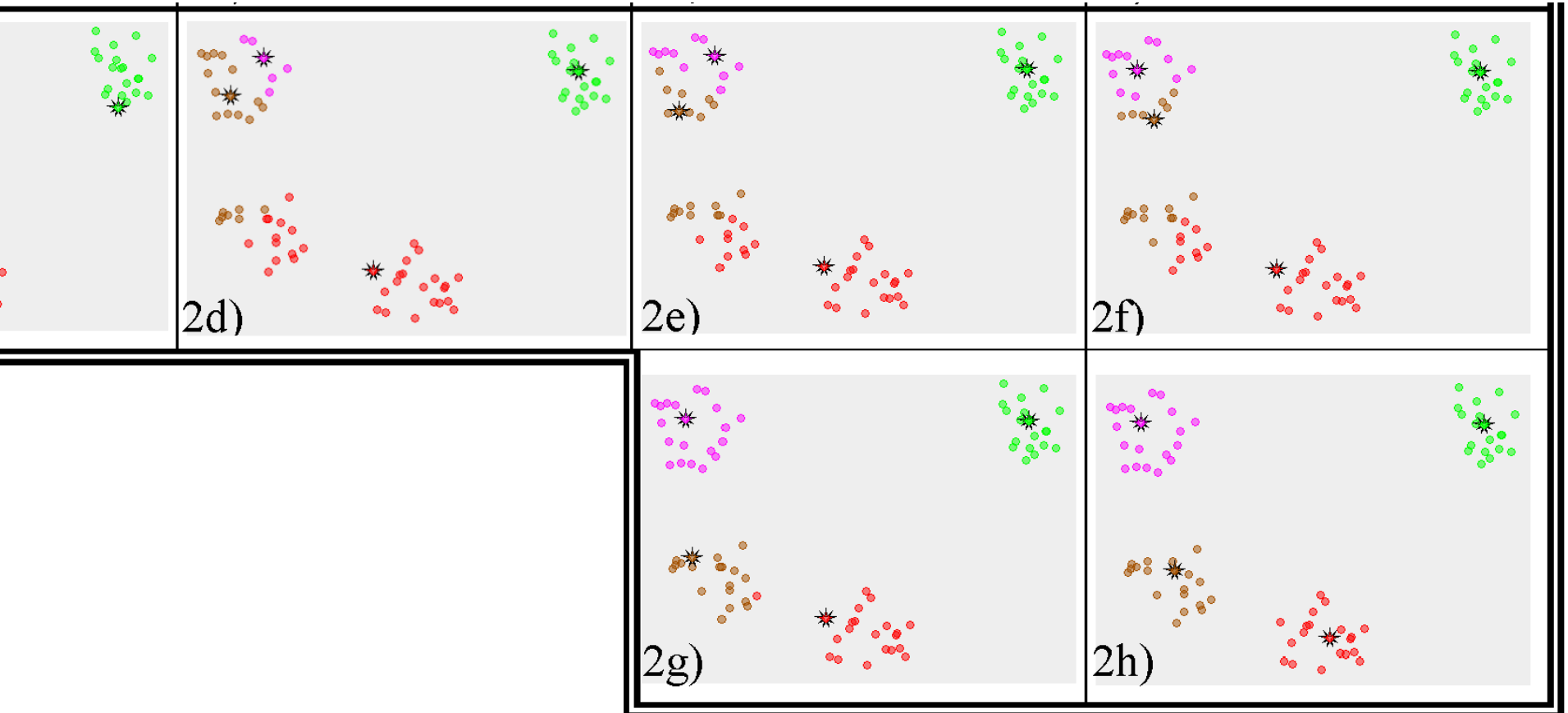
(a) Reassigned to o_i (b) Reassigned to o_{random} (c) No change (d) Reassigned to o_{random}

Example *k*-medoids algorithm

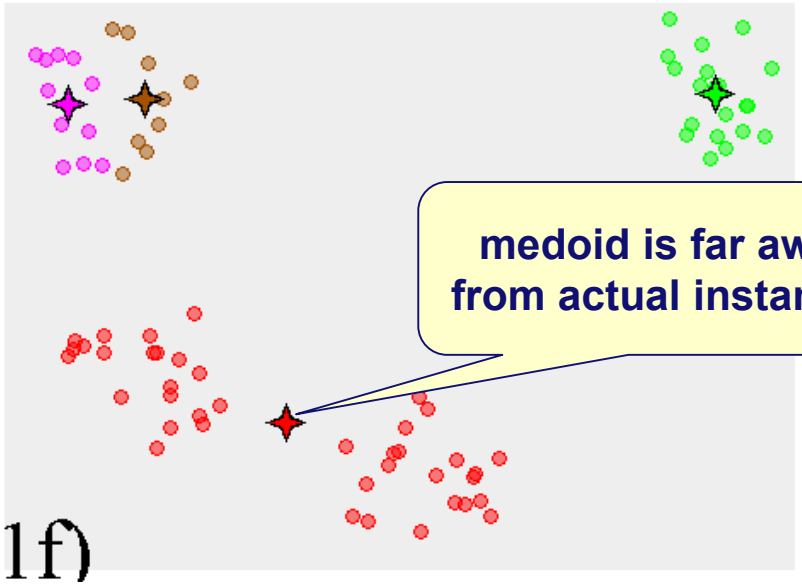


The illustration was prepared with the Java applet, E.M. Mirkes, K-means and K-medoids: applet. University of Leicester, 2011.

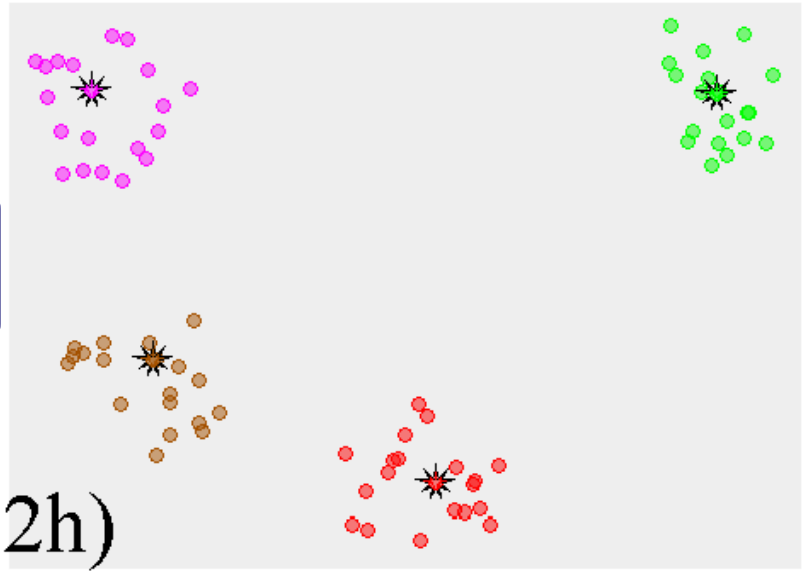
Example: *k*-medoids algorithm



Comparing *k*-means and *k*-medoids



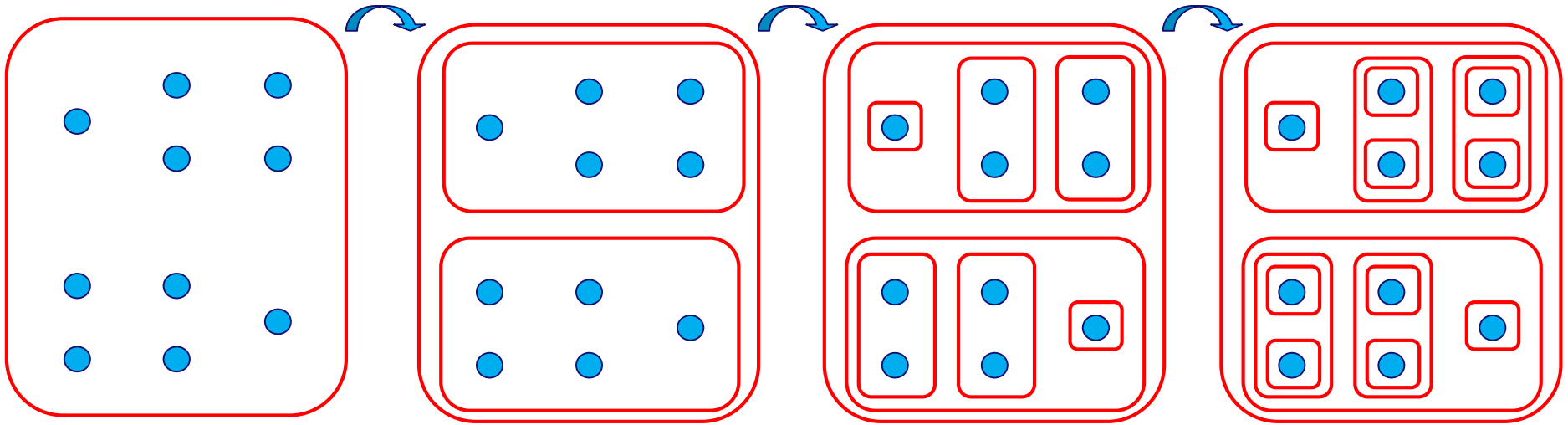
k-means



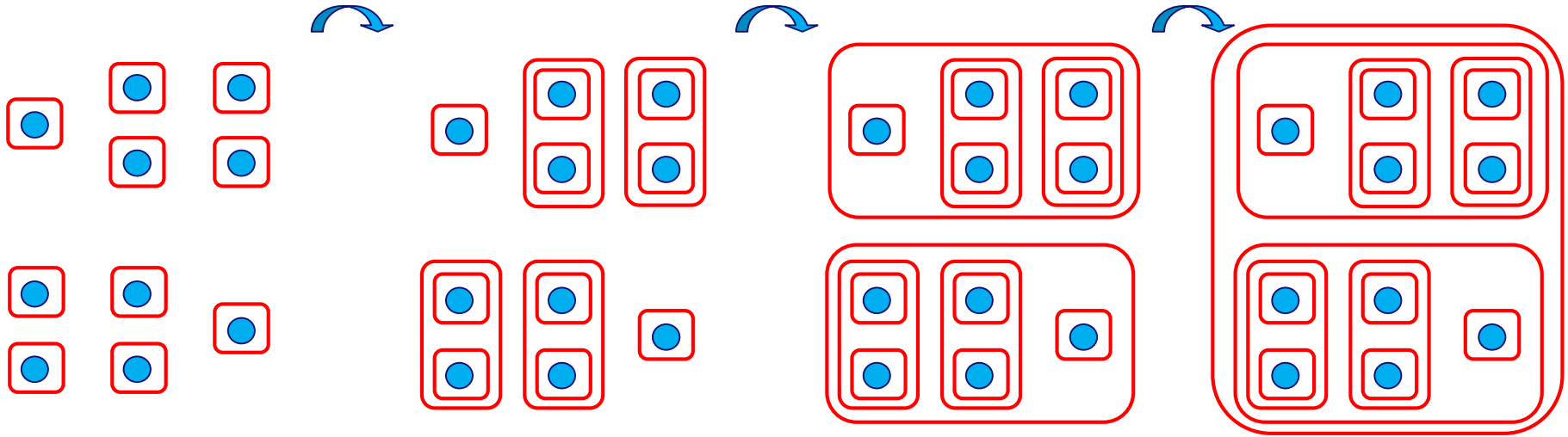
k-medoids

Agglomerative Hierarchical Clustering

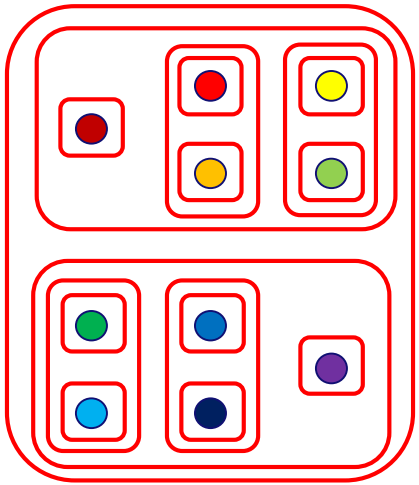
Divisive hierarchical clustering (top down)

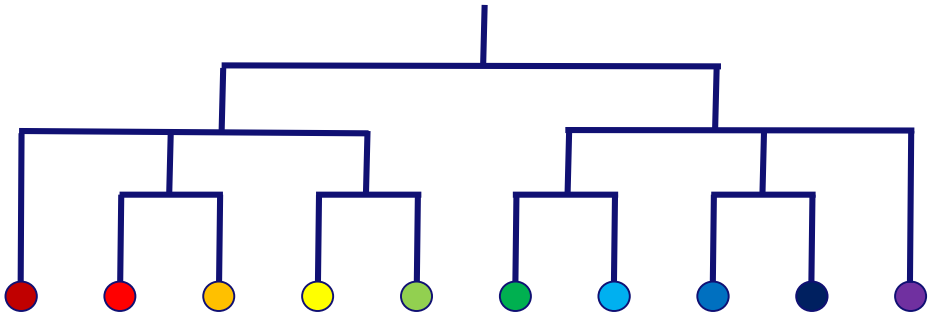


Agglomerative hierarchical clustering (bottom up)

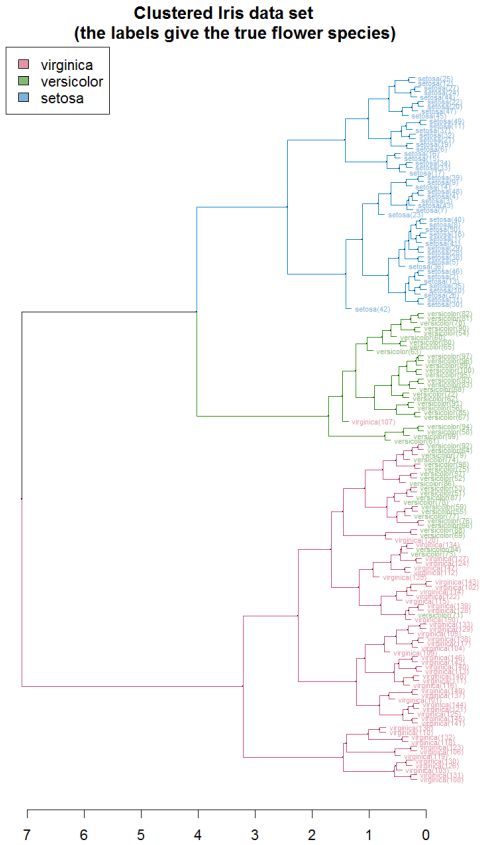


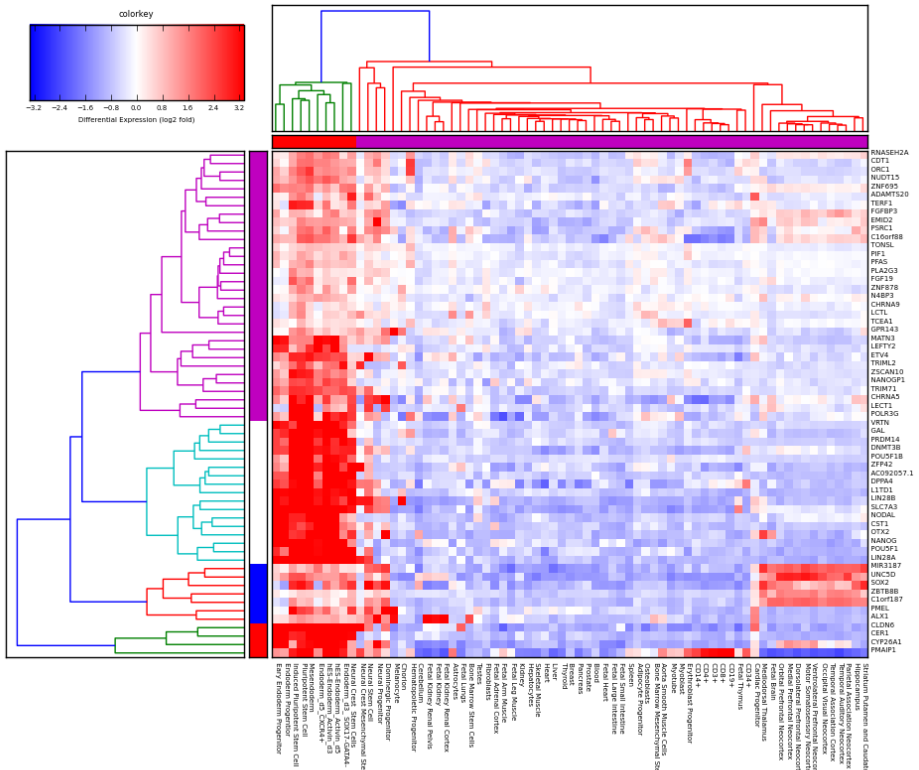
Dendrogram





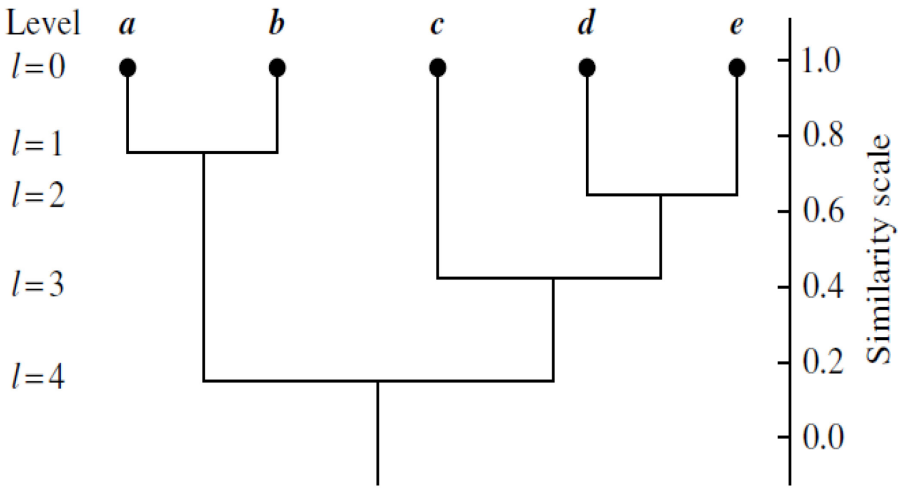
Comes in many different flavors …





Chair of Process and Data Science

Dendrograms have a scale



A new similarity/distance notion is needed: between clusters

Process: Look for two clusters that are most similar and then create a new cluster by merging them. Value depicted when merged is the similarity before merging.

Chair of Process and Data Science

Linkage measures

Four widely used measures for distance between clusters are as follows, where $|p - p'|$ is the distance between two objects or points, p and p'; m_i is the mean for cluster, C_i; and n_i is the number of objects in C_i. They are also known as *linkage measures.*

Minimum distance:
$$dist_{min}(C_i, C_j) = \min_{p \in C_i, p' \in C_j} \{|p - p'|\} \quad (10.3)$$

Maximum distance:
$$dist_{max}(C_i, C_j) = \max_{p \in C_i, p' \in C_j} \{|p - p'|\} \quad (10.4)$$

Mean distance:
$$dist_{mean}(C_i, C_j) = |m_i - m_j| \quad (10.5)$$

Average distance:
$$dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p \in C_i, p' \in C_j} |p - p'| \quad (10.6)$$

Sketch of algorithm for agglomerative hierarchical clustering (simplistic version, many variants possible)

- Create a singleton cluster for each instance.
- Repeat until just one cluster is left:
  - Compute the pairwise distance between any two clusters (several measures possible).
  - Merge the two closest clusters.
- Return dendrogram

Properties of agglomerative hierarchical clustering

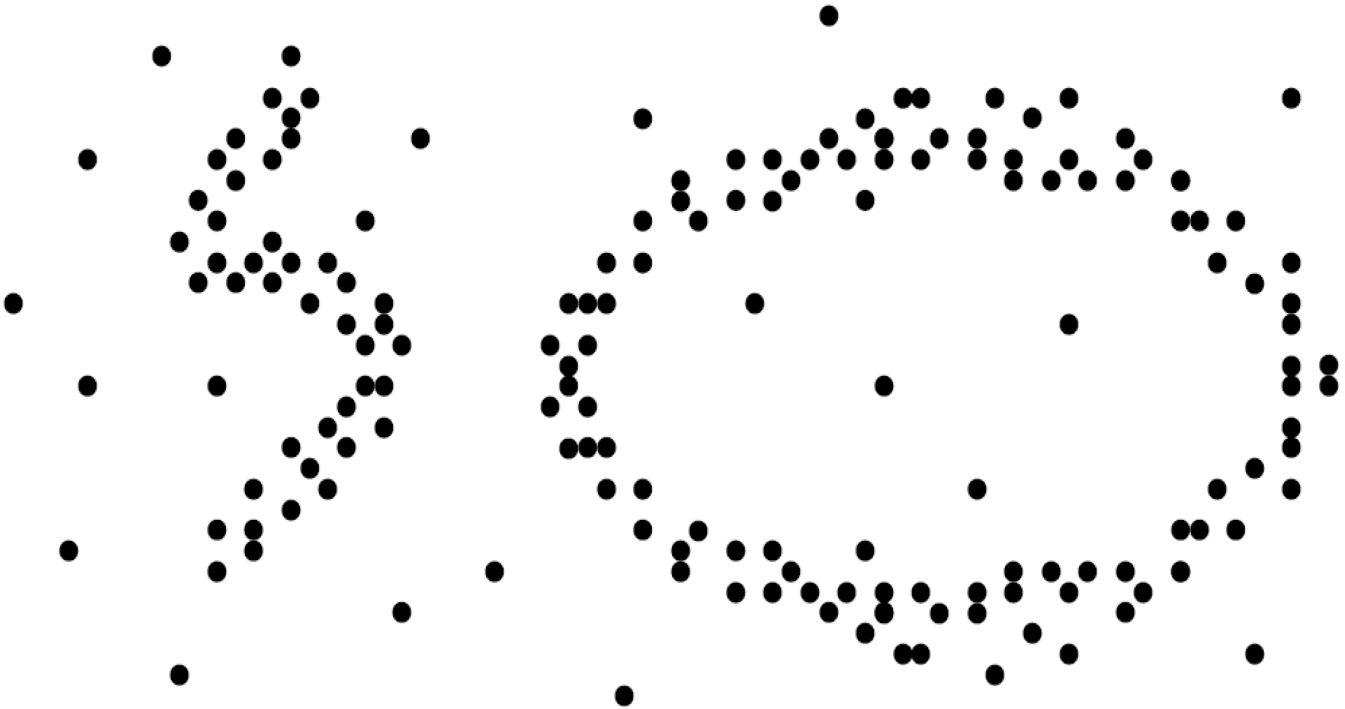
- No a priori information/decision about the number of clusters is required.
- Dendrogram allows analyst to “play” with abstraction level.
- Algorithm cannot undo joins that turn out to be undesirable and there is no approach to objectively minimize some well-defined error.

Density-based clustering using DBSCAN

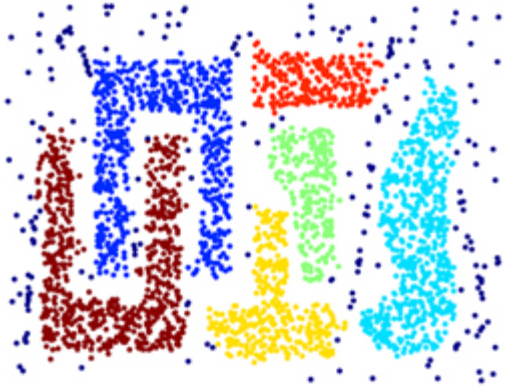
Density-based clustering

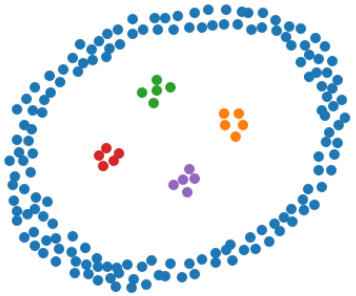
- Clusters are areas having a higher density than other areas.
- Instances in sparse areas are considered to be outliers.
- Used to find clusters of any shape (partitioning and hierarchical methods tend to find spherical clusters due to distance function used).
- Example: Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

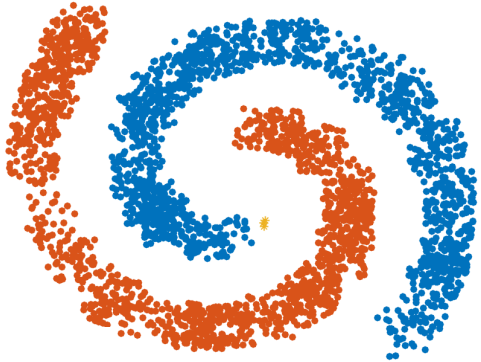
How to cluster?



Examples of density-based clustering







Diagrams taken from Wikipedia

DBSCAN

- Two parameters: ϵ (fixed neighborhood size) and $MinPts$ (density threshold for dense regions).
- An instance p is a core point if at least $MinPts$ points are within distance ϵ (ϵ is the maximum radius of the neighborhood from p) of it (including p).

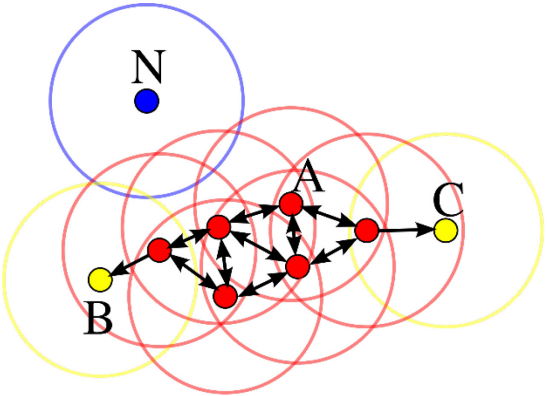
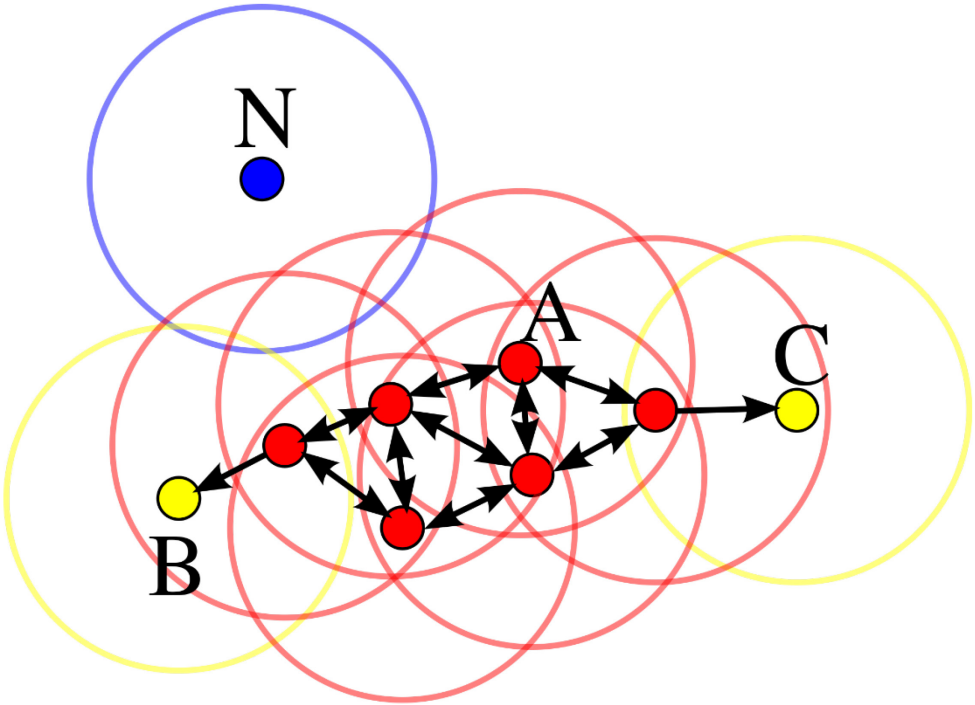


Diagram taken from Wikipedia

Reachable in DBSCAN terms

- A point q is directly reachable from p if point q is within distance ϵ from point p and p is a core point.
- A point q is reachable from p if there is a path $\langle p_1, p_2, \ldots p_n \rangle$ with $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i (all the points on the path must be core points, with the possible exception of q, i.e., $p_1, p_2, \ldots p_{n-1}$ are core points).
- All points not reachable from any other point are outliers.

Example



- Red instances are core points.
- B is reachable from A.
- A is not reachable from B.
- C is reachable from A.
- A is not reachable from C.
- N is an outlier.

ϵ is indicated by circles and $MinPts = 4$.

Clusters in DBSCAN

- A two instances p_1 and p_2 are density-connected if there is a point q such that both p_1 and p_2 are reachable from q.
- Density-connectedness is symmetric (unlike reachability).
- A cluster satisfies the following two properties:
  - All points within the cluster are mutually density-connected.
  - If a point is density-connected to any point of the cluster, it is part of the cluster as well (it cannot be outside).

DBSCAN Algorithm

Algorithm: DBSCAN: a density-based clustering algorithm.

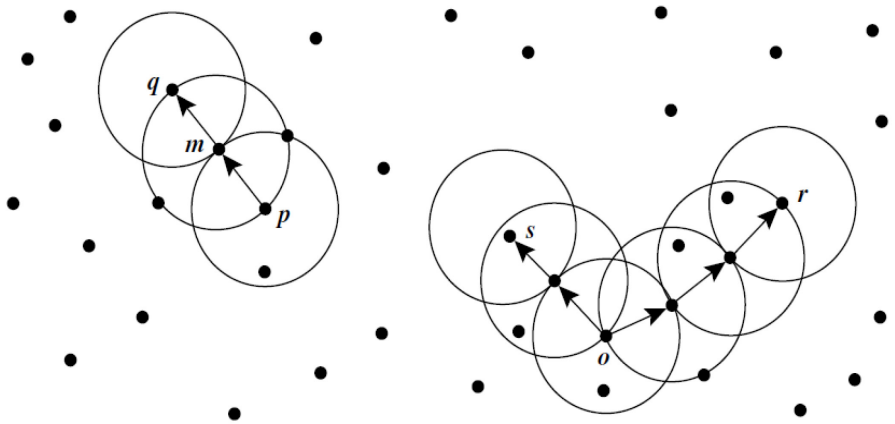
Input:

- D: a data set containing n objects,
- ϵ: the radius parameter, and
- *MinPts*: the neighborhood density threshold.

Output: A set of density-based clusters.

Method:

```
(1)  mark all objects as unvisited;
(2)  do
(3)      randomly select an unvisited object p;
(4)      mark p as visited;
(5)      if the ε-neighborhood of p has at least MinPts objects
(6)          create a new cluster C, and add p to C;
(7)          let N be the set of objects in the ε-neighborhood of p;
(8)          for each point p' in N
(9)              if p' is unvisited
(10)                 mark p' as visited;
(11)                 if the ε-neighborhood of p' has at least MinPts points,
                     add those points to N;
(12)             if p' is not yet a member of any cluster, add p' to C;
(13)         end for
(14)         output C;
(15)     else mark p as noise;
(16) until no object is unvisited;
```



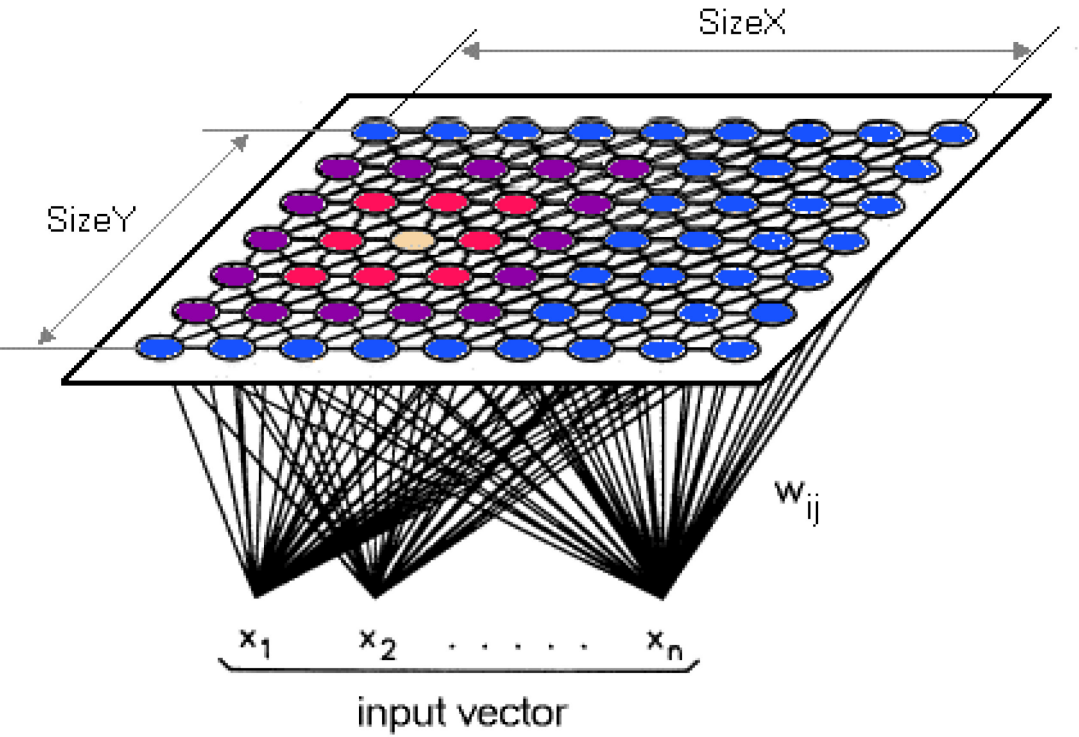
Non-deterministic wrt to border points that potentially belong two clusters.

Self-Organizing Maps (SOMs)

Self-Organizing Maps (SOM)

- **Data clustering** and **dimension reduction** technique based on (another type of) Artificial Neural Networks (ANNs).
- Introduced by Teuvo Kohonen in the 1980s (also called **Kohonen map**).
- **Unsupervised**, aims to capture the "topology" of the input data set.
- **Insightful visualizations** possible by coloring using specific features (the same map can be used to inspect the distribution of different features).

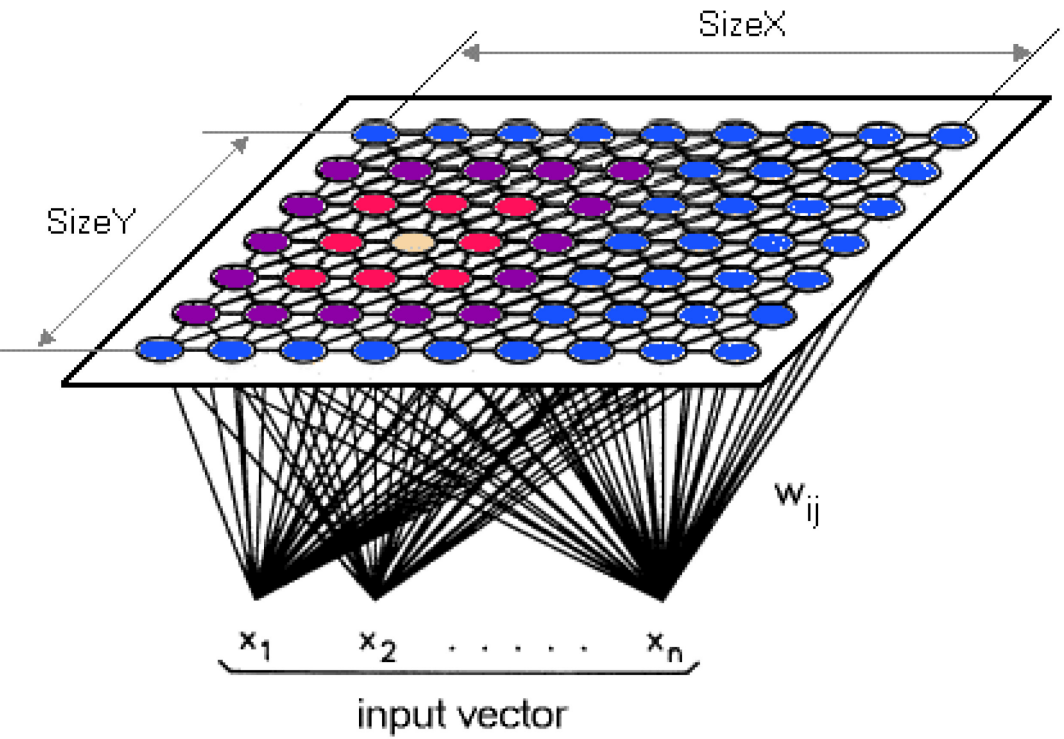
Key idea



- Nodes are arranged into a 2D (or 1D) structure.
- Each instance i has n features: $(x_{i1}, x_{i2}, \dots x_{in})$.
- Each node j has a similar structure: $(w_{j1}, w_{j2}, \dots w_{jn})$ (like a centroid).
- Each node node j has a distance to instance i: $d(i,j)$ (like an error).

diagram taken from wikipedia

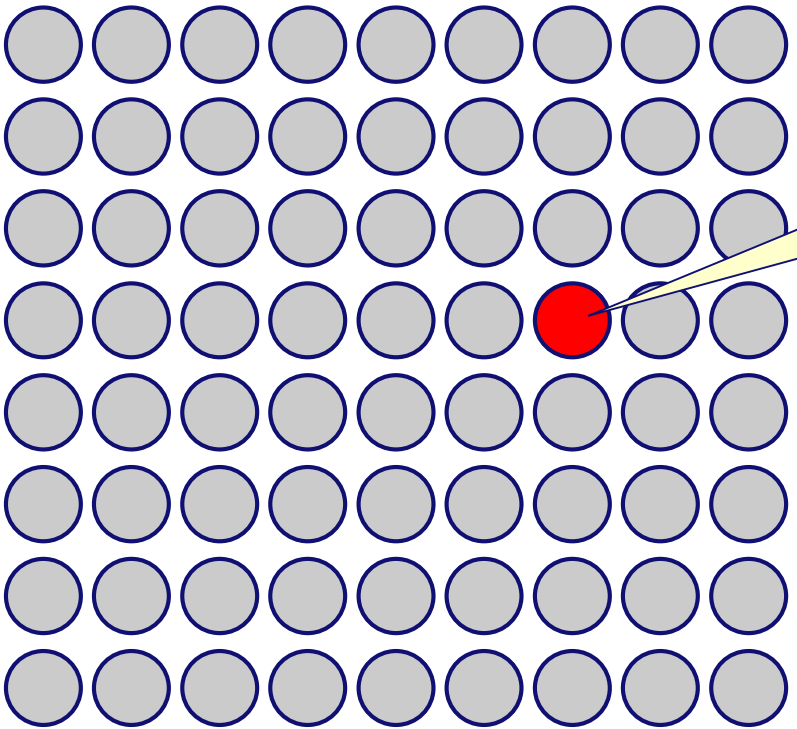
Key idea



- Pick an instance $i = (x_{i1}, x_{i2}, \ldots x_{in})$.
- Nodes are competing for instance i based on $d(i, j)$.
- Given an instance i we can determine the **Best Matching Unit (BMU)**, say node $j = (w_{j1}, w_{j2}, \ldots w_{jn})$.
- Update winning node j to be closer to instance i.
- Update environment of j.
- Repeat.

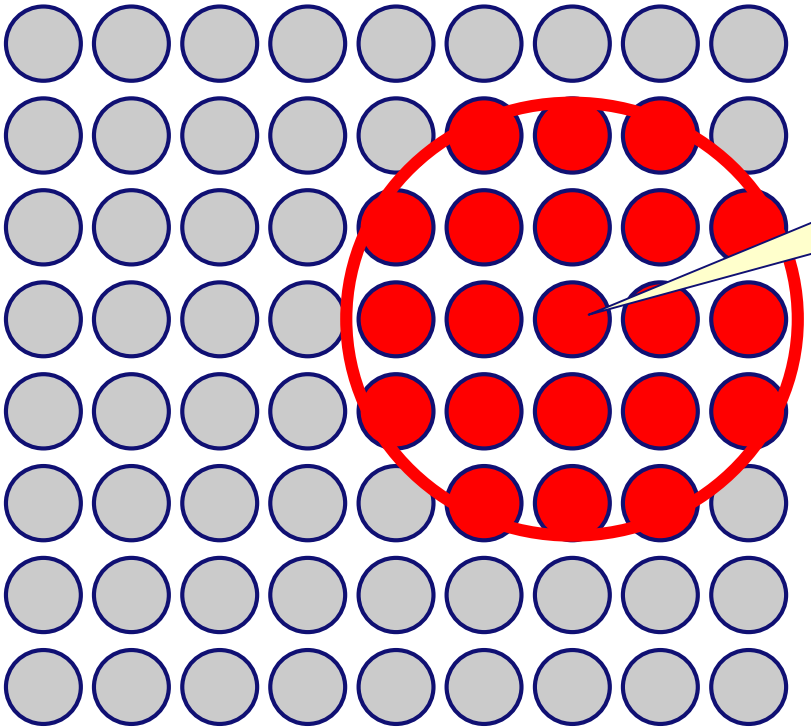
diagram taken from wikipedia

Updating the neighborhood of the BMU



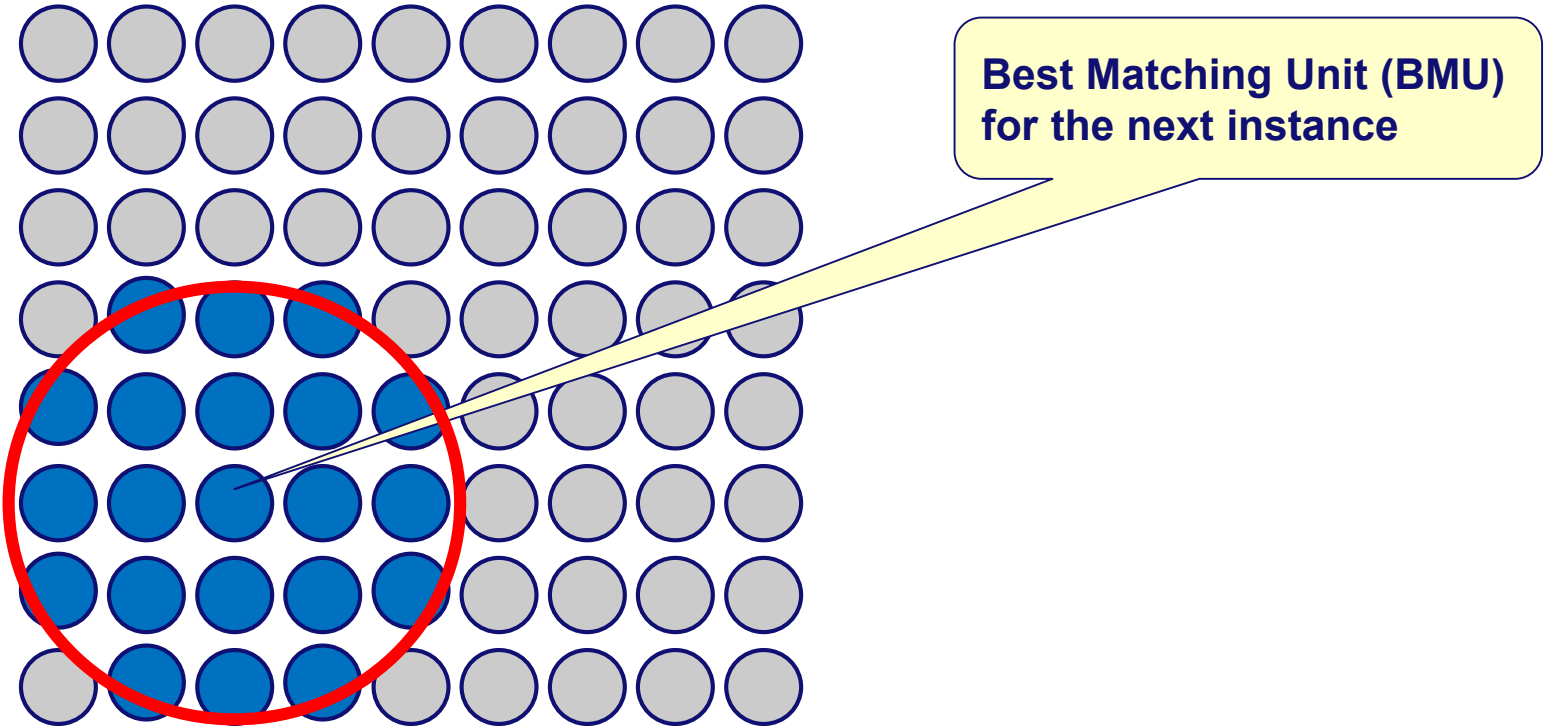
Best Matching Unit (BMU) $j = (w_{j1}, w_{j2}, \ldots w_{jn})$ **closest to the selected instance** $i = (x_{i1}, x_{i2}, \ldots x_{in})$

Updating the neighborhood of the BMU



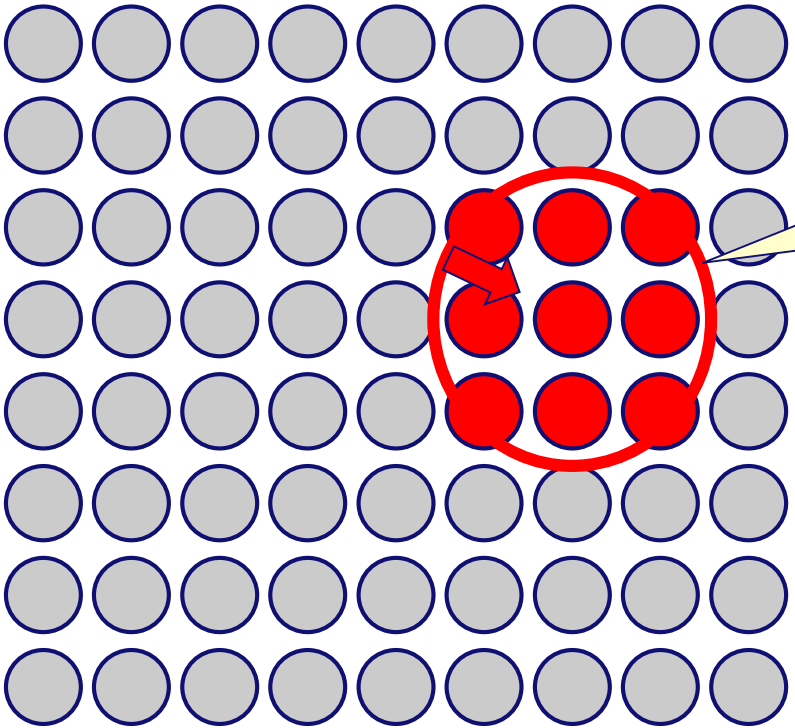
Best Matching Unit (BMU) $j = (w_{j1}, w_{j2}, \ldots w_{jn})$ **closest to the selected instance** $i = (x_{i1}, x_{i2}, \ldots x_{in})$

Updating the neighborhood of the BMU



Chair of Process and Data Science

Updating the neighborhood of the BMU

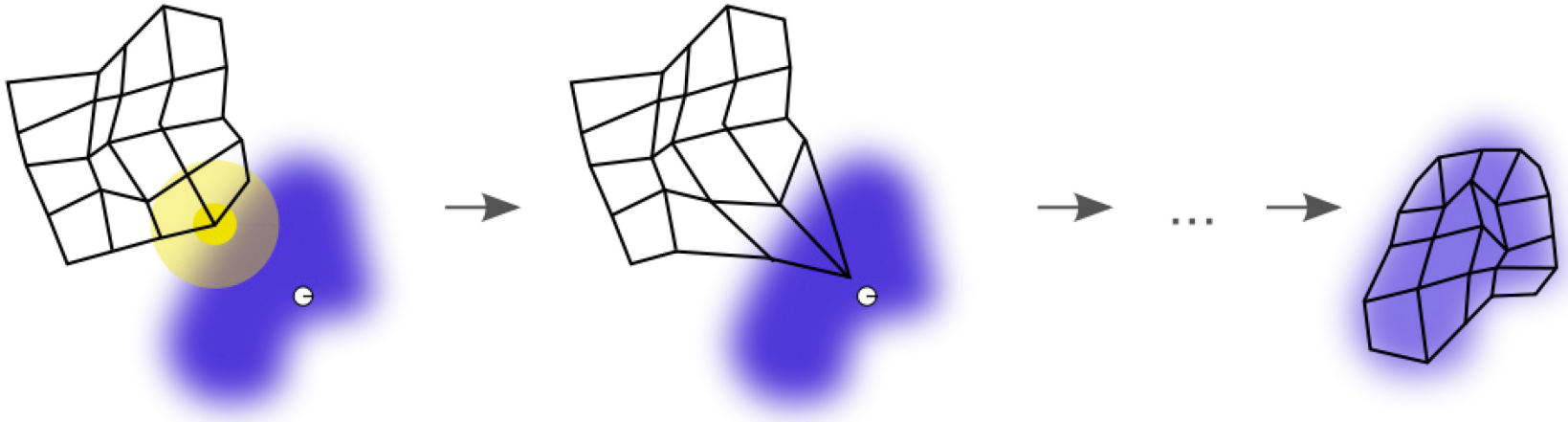


Over time the neighborhood of the Best Matching Unit (BMU) gets smaller and updates are more local

Sketch of a SOM algorithm (many variants possible)

1. The weight of each node j is initialized (e.g., randomly).
2. An instance i is chosen randomly from the input data.
3. Every node j is examined to calculate which one's weights are most like the input vector. The "winning node" is commonly known as the **Best Matching Unit (BMU)**.
4. Then the neighborhood of the BMU is determined (over time the size of the neighborhood shrinks).
5. The BMU is updated to be closer to the selected instance i. The BMU's neighboring nodes are also updated. The closer a node is to the BMU, the more its weights get altered and the farther away the neighbor is from the BMU, the smaller the update.
6. Repeat step 2 for N iterations.

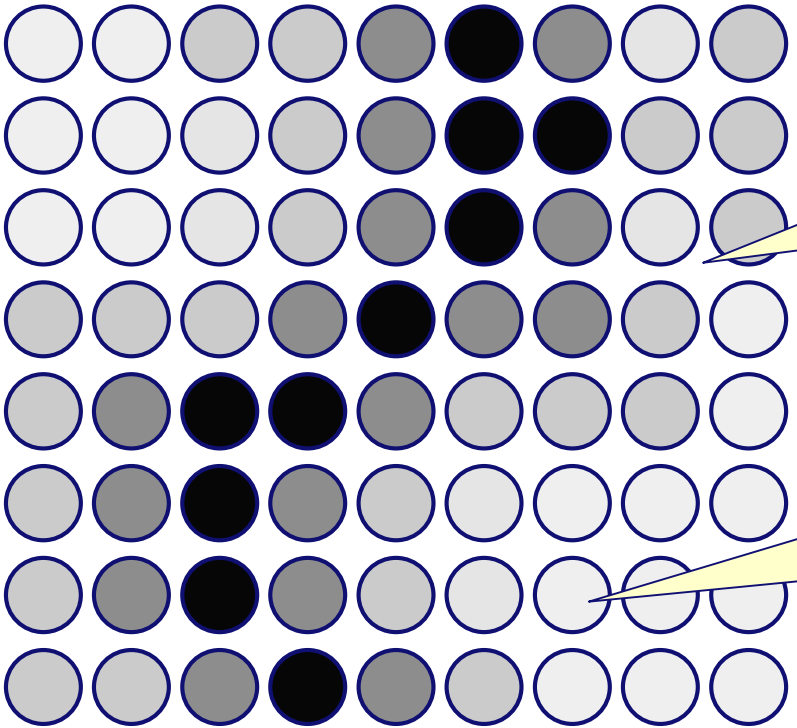
Training process of a SOM



The blue blob is the distribution of the input data, and the small white disc is the instance $i = (x_{i1}, x_{i2}, \ldots x_{in})$ under consideration. At first (left) the SOM nodes are arbitrarily positioned in the data space. The node nearest to the training node (highlighted in yellow) is selected, and is moved towards the training datum, as (to a lesser extent) are its neighbors on the grid. After many iterations the grid tends to approximate the data distribution (right).

diagram taken from wikipedia

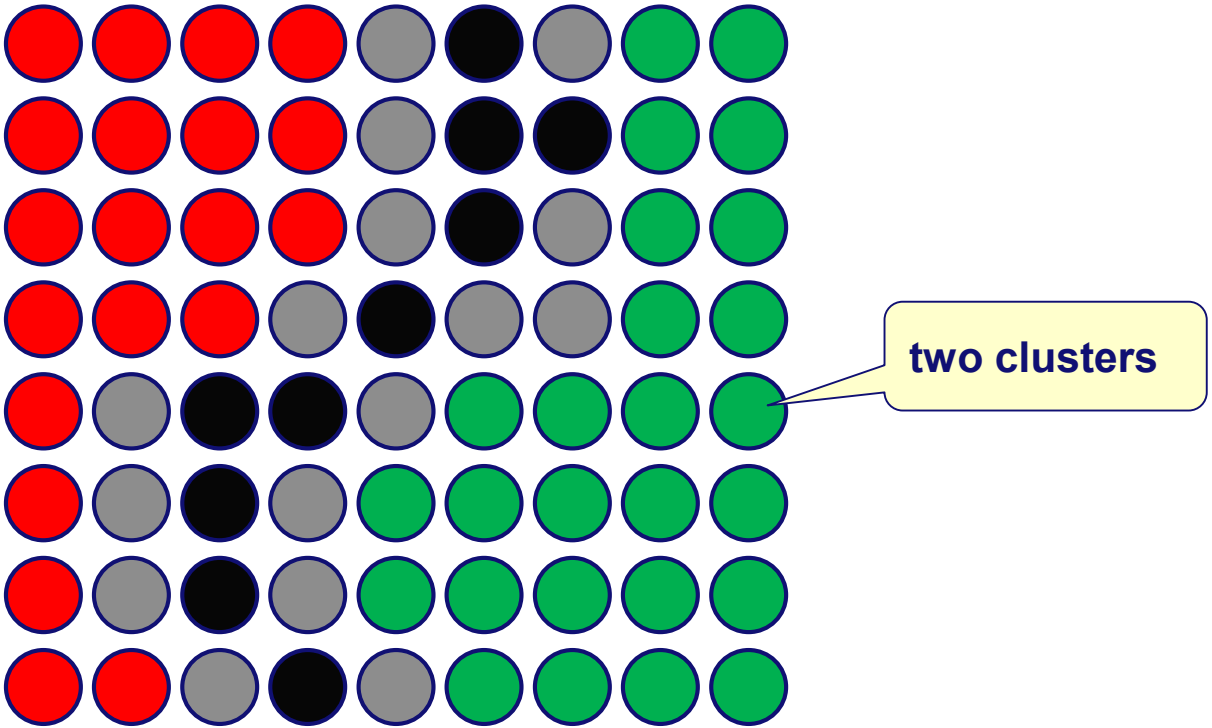
Visualizing SOMs: Boundaries



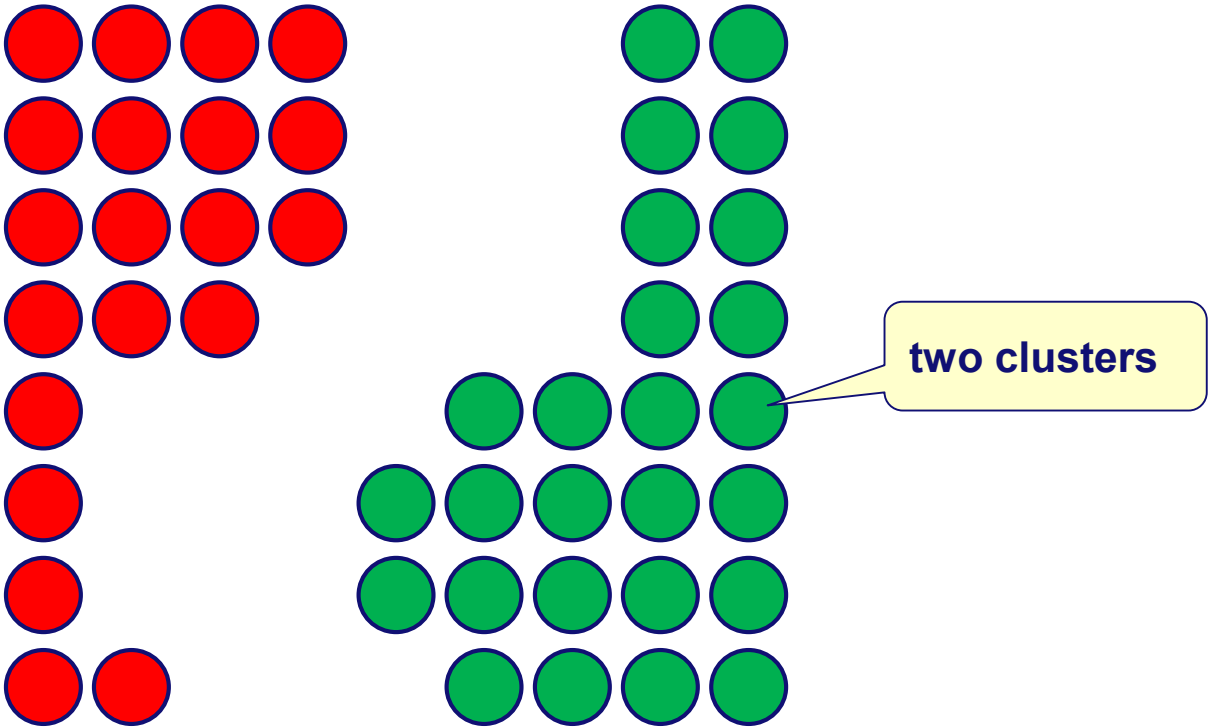
The darker the node the more different the neighboring nodes are.

Groups of light colors can be considered as clusters, and the dark parts as the boundaries between the clusters.

Visualizing SOMs: Clusters

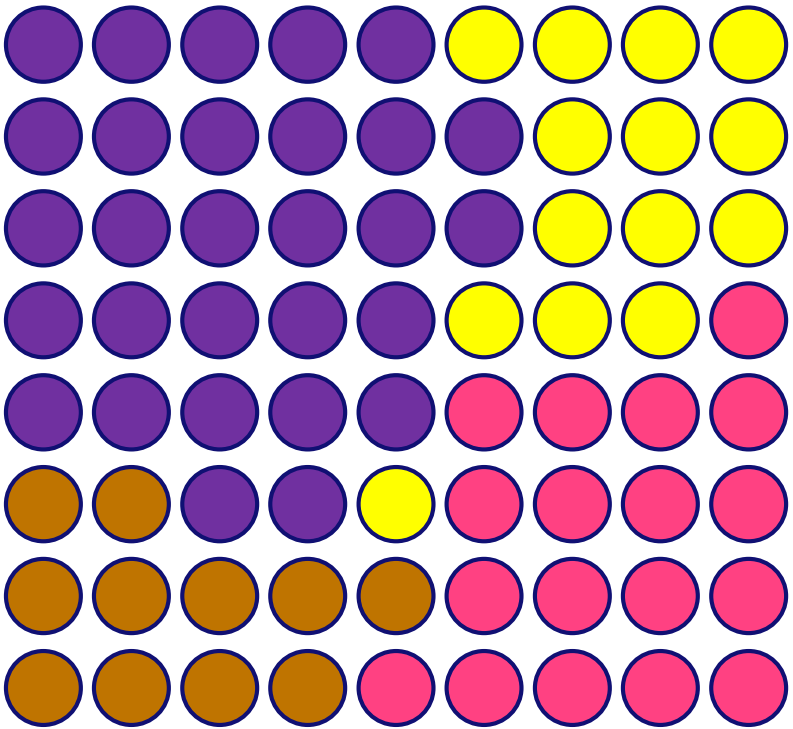


Visualizing SOMs: Clusters



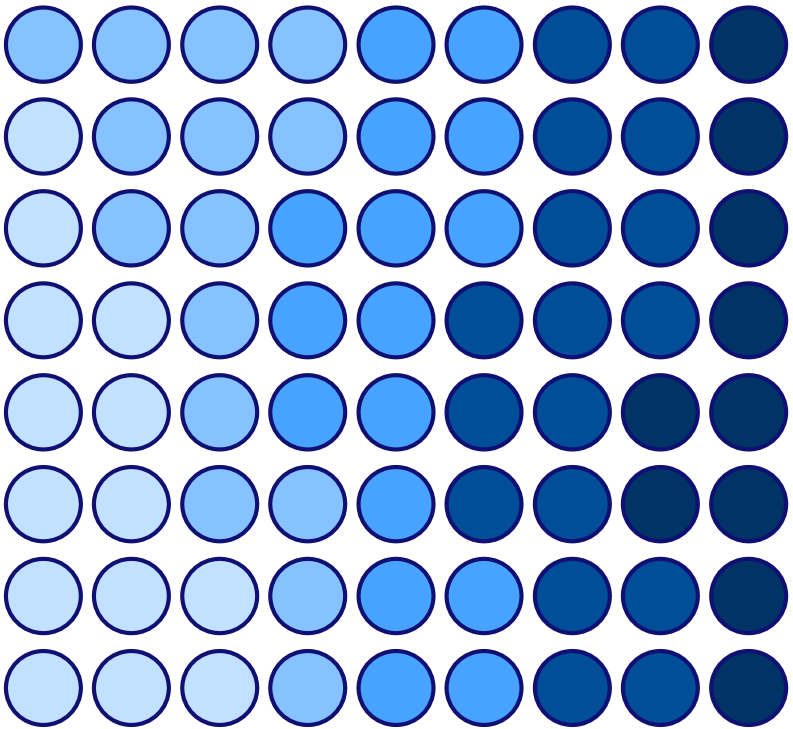
Visualizing SOMs: Color using any attribute

(Remember: each node is connected to the set of instances through weights, so any computation is possible using input data)

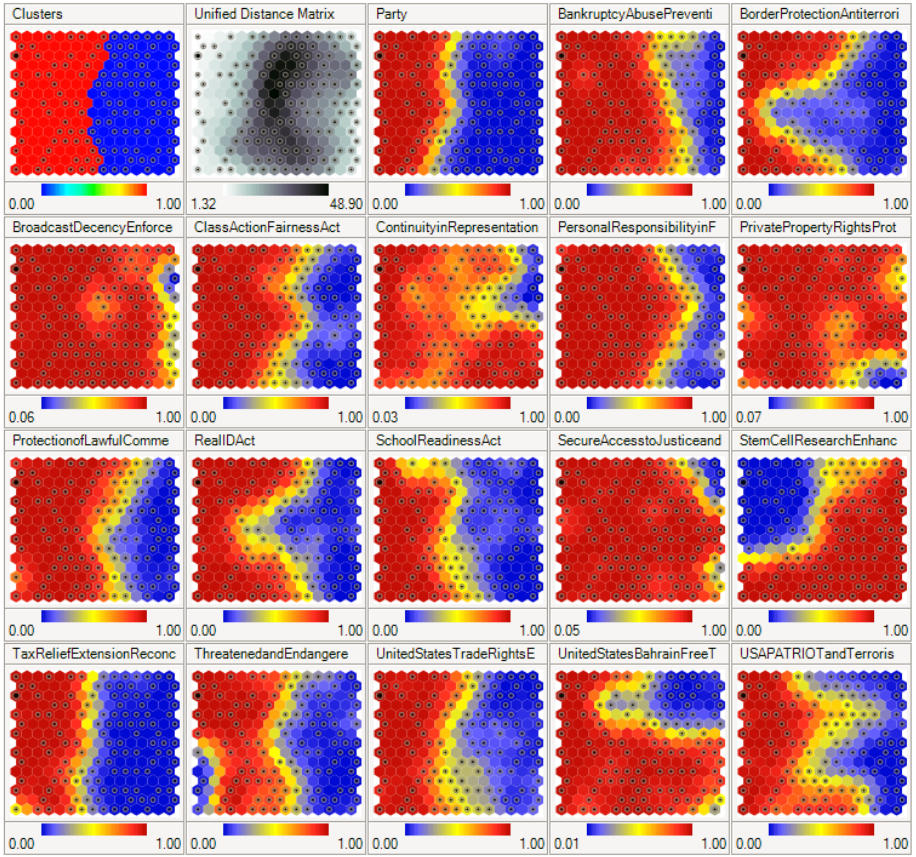


Visualizing SOMs: Color using any attribute

(Remember: each node is connected to the set of instances through weights, so any computation is possible using input data)

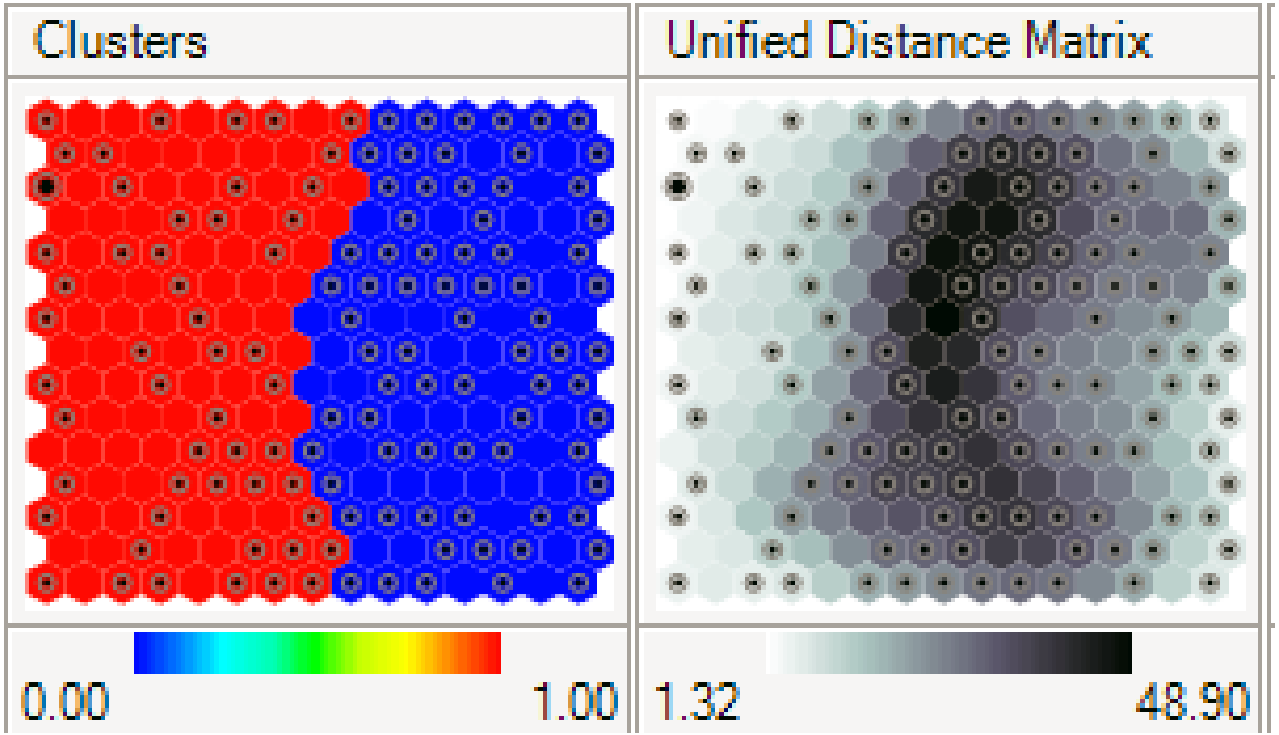


Example: US congress voting patterns



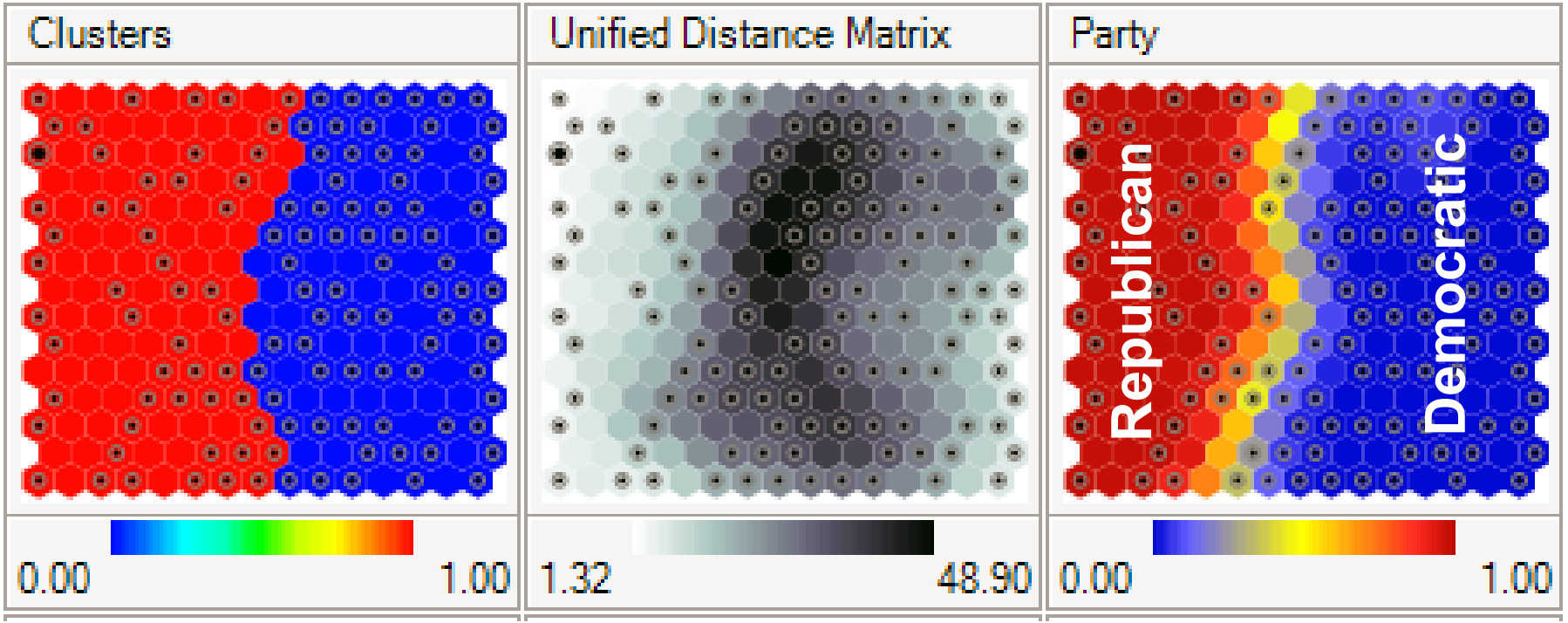
From https://en.wikipedia.org/wiki/Self-organizing_map:
A self-organizing map showing U.S. Congress voting patterns. The input data was a table with a row for each member of Congress, and columns for certain votes containing each member's yes/no/abstain vote. The SOM algorithm arranged these members in a two-dimensional grid placing similar members closer together. The first plot shows the grouping when the data are split into two clusters. The second plot shows average distance to neighbors: larger distances are darker. The third plot predicts Republican (red) or Democratic (blue) party membership. The other plots each overlay the resulting map with predicted values on an input dimension: red means a predicted 'yes' vote on that bill, blue means a 'no' vote. The plot was created in Synapse.

Example: US congress voting patterns



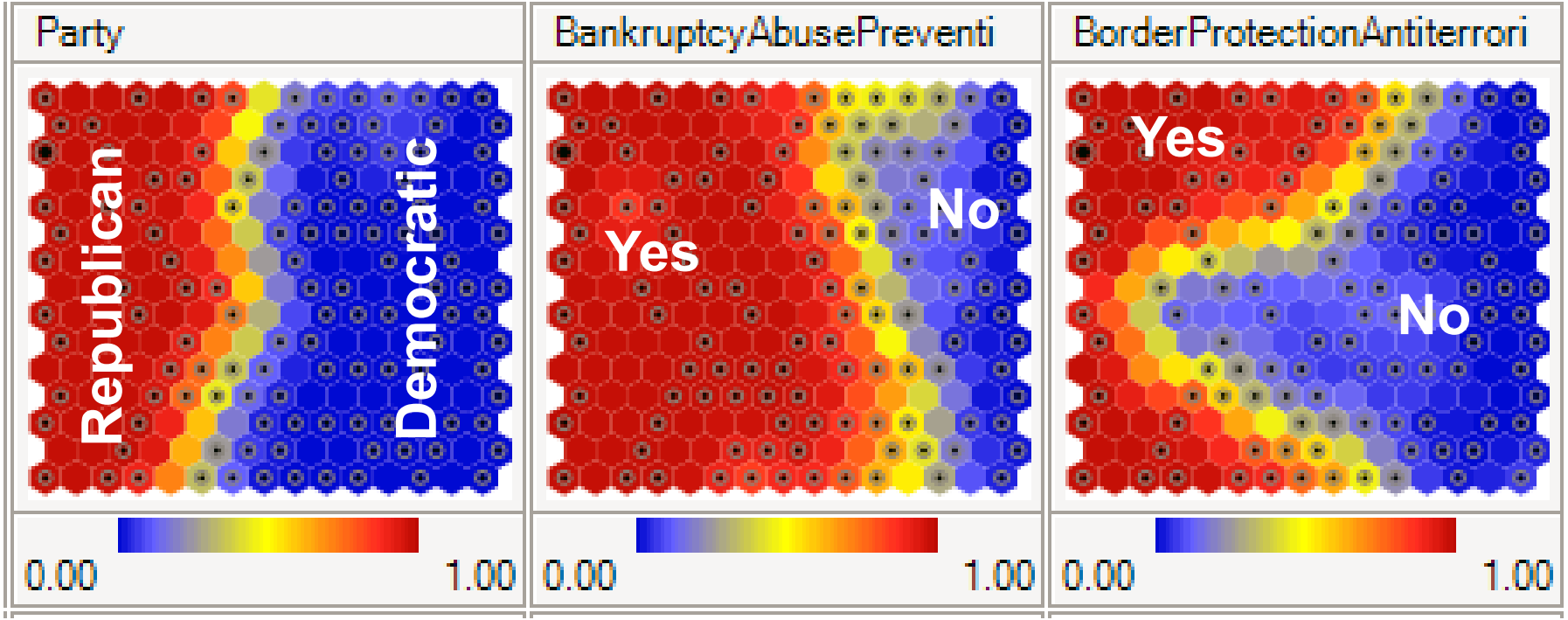
Taken from https://en.wikipedia.org/wiki/Self-organizing_map

Example: US congress voting patterns



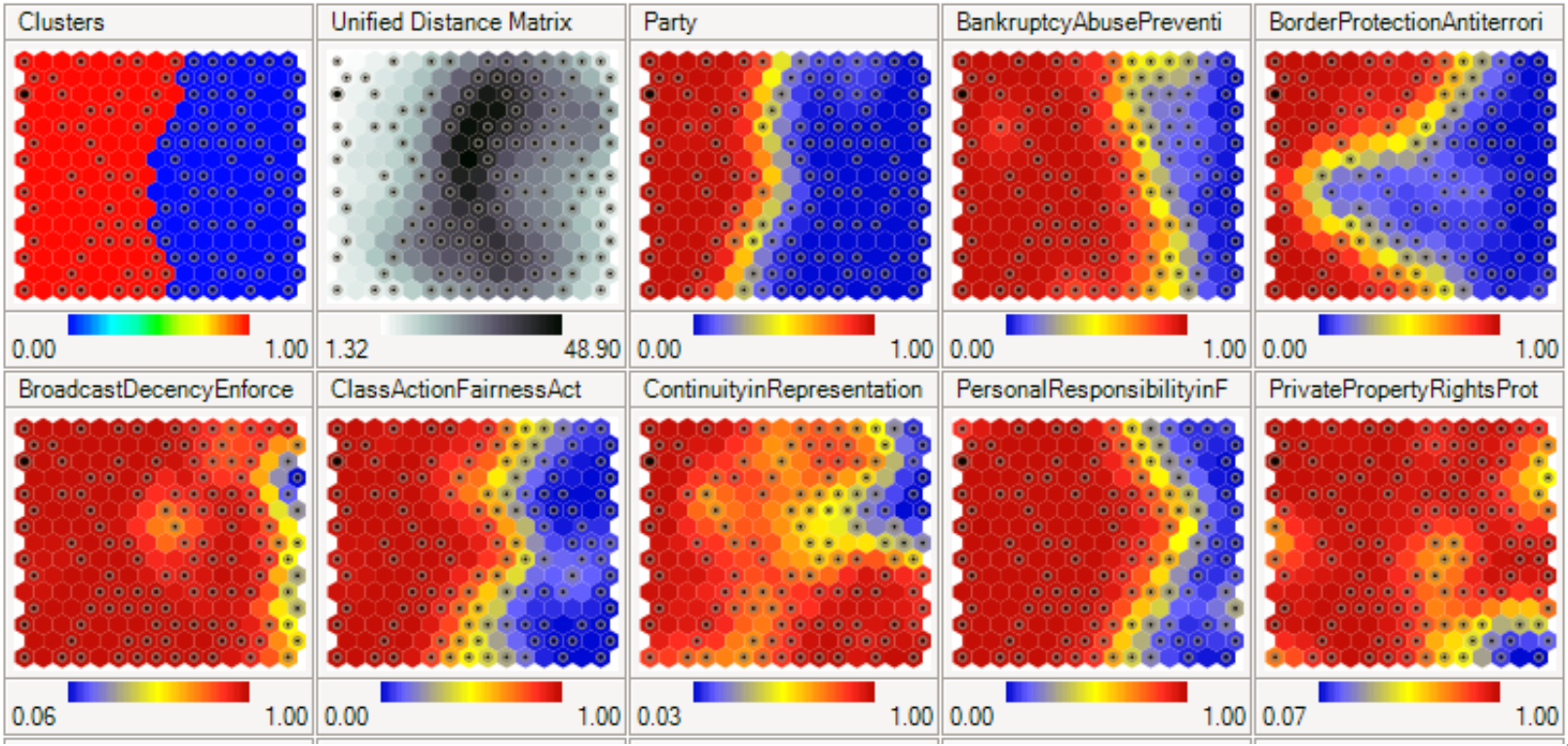
Taken from https://en.wikipedia.org/wiki/Self-organizing_map

Example: US congress voting patterns



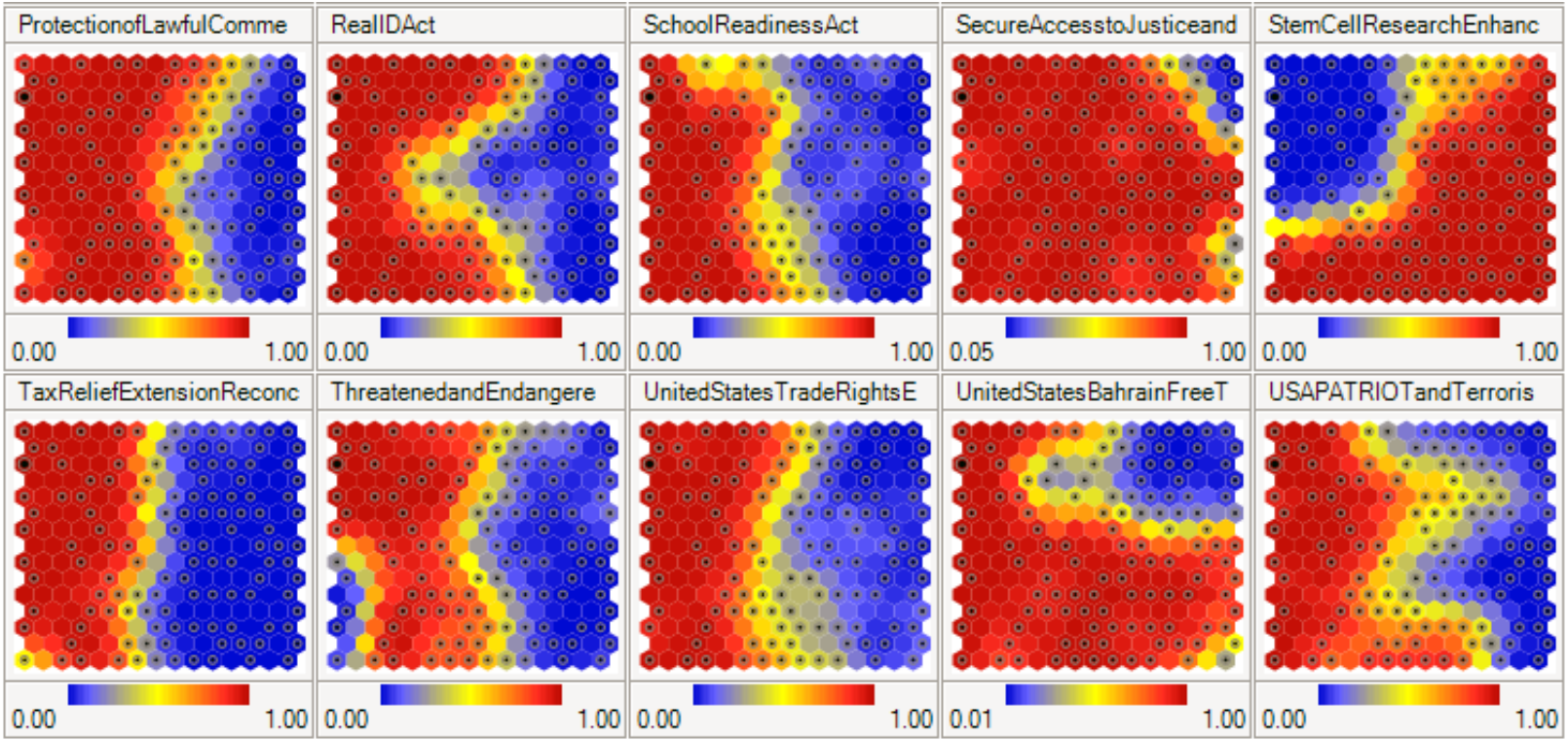
Taken from https://en.wikipedia.org/wiki/Self-organizing_map

Example: US congress voting patterns



Taken from https://en.wikipedia.org/wiki/Self-organizing_map

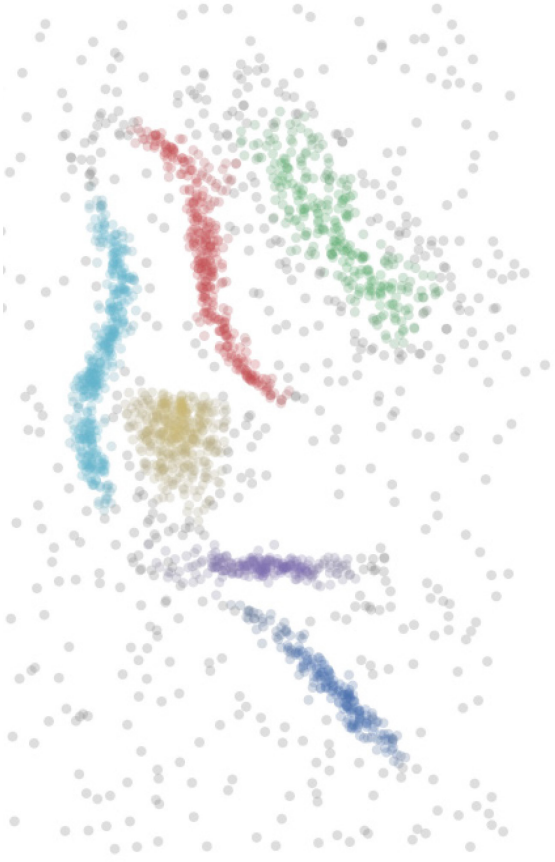
Example: US congress voting patterns



Taken from https://en.wikipedia.org/wiki/Self-organizing_map

General problem: How to interpret clusters?

- Cluster quality may be good, but this does not imply that the clusters actually reveal new insights.
- Describe clusters in terms of their features (e.g., compare centroids).
- Use simple visualization techniques like boxplots to compare clusters.



Final remark: Data science is like a making cocktail. Mix the proper ingredients in the right way!

Python
Visualization
Decision trees
Regression
Support vector machines
Neural networks
Evaluation
Clustering
Frequent items sets
Association rules
Sequence mining
Process mining
Process discovery
Conformance checking
Text mining
Preprocessing
Visual analytics
Encryption
Anonymization
Big data infra
Distribution

Decision trees

Clustering

Visualization

Preprocessing

Conclusion

Short summary of lecture

- **Clustering as the first example of an unsupervised learning technique** (many will follow)
- **Four approaches were presented**
  1. k-means clustering
  2. k-medoids clustering
  3. Agglomerative Hierarchical Clustering
  4. Density-based clustering using DBSCAN
- **Related to clustering: Self-organizing maps**

Other unsupervised learning techniques that will follow in later lectures

- Frequent items sets
- Association rules
- Sequence mining
- Process mining
- Text mining
- Visualization

Some of these techniques can also be used for supervised learning.

#	Lecture	date	day
Lecture 1	Introduction	10/10/2018	Wednesday
Lecture 2	Crash Course in Python	11/10/2018	Thursday
Instruction 1	*Python*	*12/10/2018*	*Friday*
Lecture 3	Basic data visualisation/exploration	17/10/2018	Wednesday
Lecture 4	Decision trees	18/10/2018	Thursday
Instruction 2	*Decision trees and data visualization/exploration*	*19/10/2018*	*Friday*
Lecture 5	Regression	24/10/2018	Wednesday
Lecture 6	Support vector machines	25/10/2018	Thursday
Instruction 3	*Regression and support vector machines*	*26/10/2018*	*Friday*
Lecture 7	Neural networks (1/2)	31/10/2018	Wednesday
Instruction 4	*Neural networks and supervised learning*	*02/11/2018*	*Friday*
Lecture 8	Neural networks (2/2)	07/11/2018	Wednesday
Lecture 9	Evaluation of supervised learning problems	08/11/2018	Thursday
Instruction 5	*Neural networks and supervised learning*	*09/11/2018*	*Friday*
Lecture 10	Clustering	14/11/2018	Wednesday
Lecture 11	Frequent items sets	15/11/2018	Thursday
Lecture 12	Association rules	21/11/2018	Wednesday
Lecture 13	Sequence mining	22/11/2018	Thursday
Instruction 6	*Clustering, frequent items sets, association rules*	*23/11/2018*	*Friday*
Lecture 14	Process mining (unsupervised)	28/11/2018	Wednesday
Lecture 15	Process mining (supervised)	29/11/2018	Thursday
Instruction 7	*Process mining and sequence mining*	*30/11/2018*	*Friday*
backup		31/01/2019	Thursday
extra	*Question hour*	*01/02/2019*	*Friday*