Data Mining

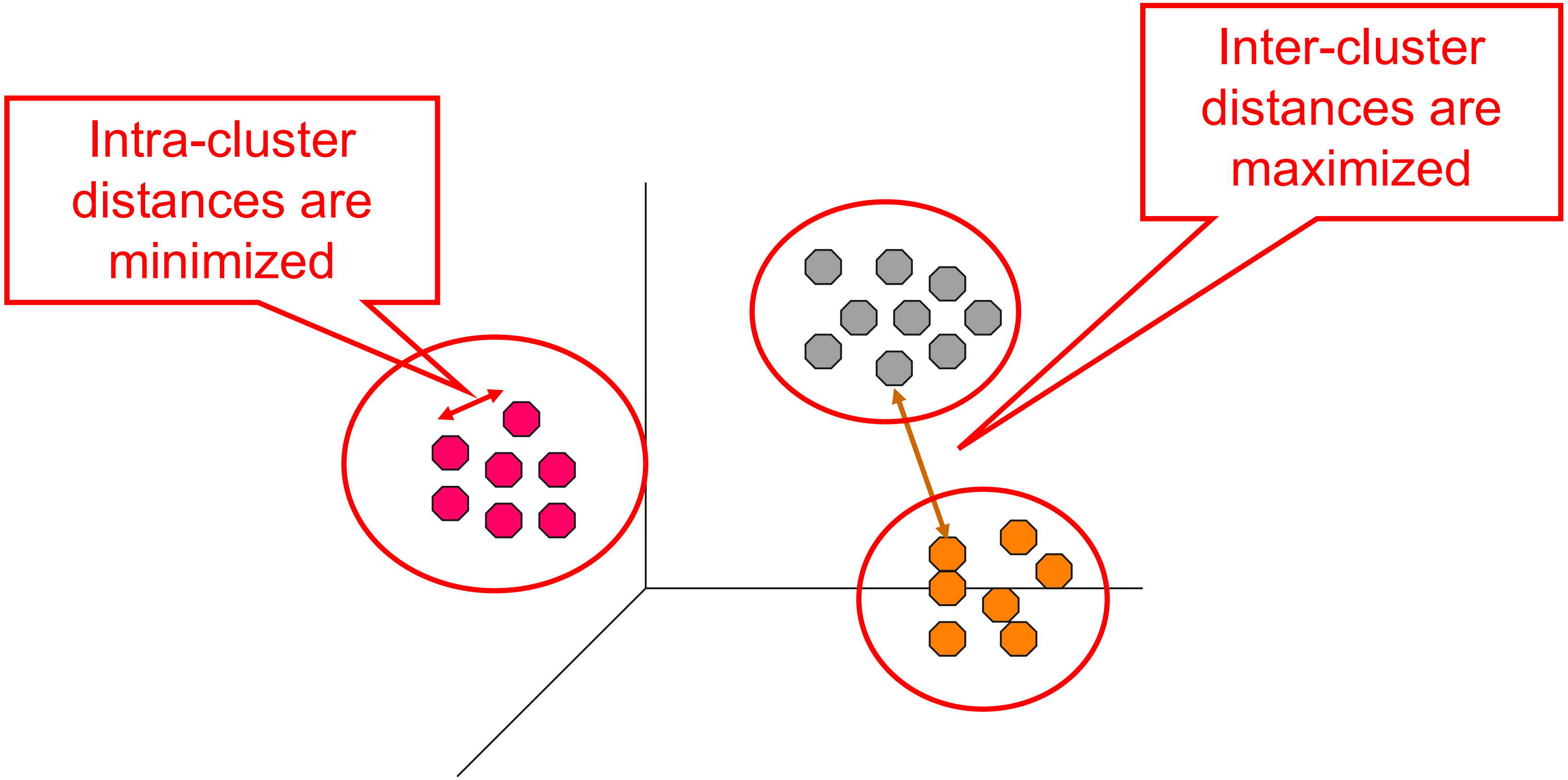
Cluster Analysis

Outline

1. What is Cluster Analysis?
2. K-Means Clustering
3. Density-based Clustering
4. Hierarchical Clustering
5. Proximity Measures

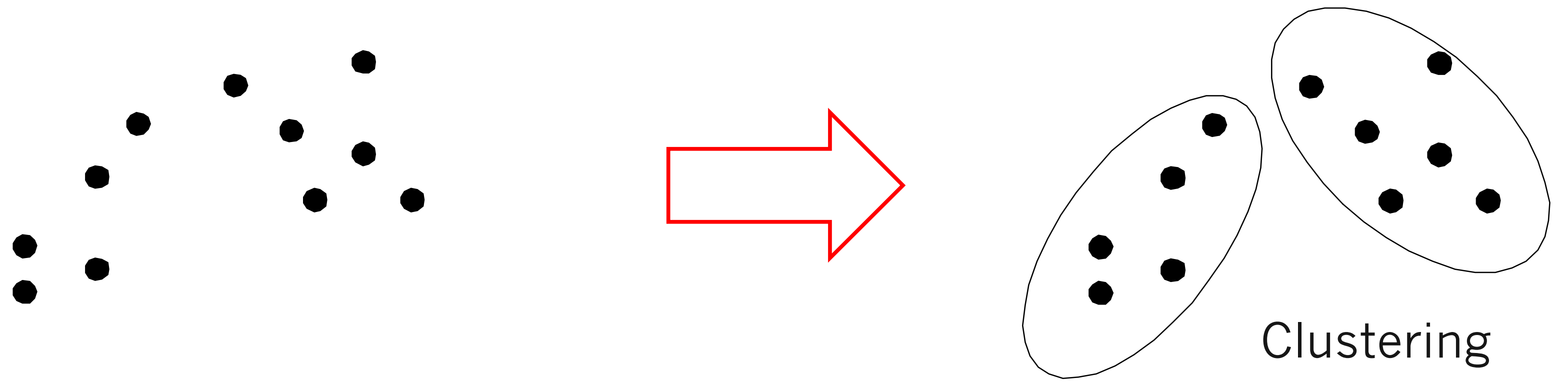
1. What is Cluster Analysis?

– Finding groups of objects such that
  - the objects in a group will be similar to one another
  - and different from the objects in other groups.
– Goal: Get a better understanding of the data

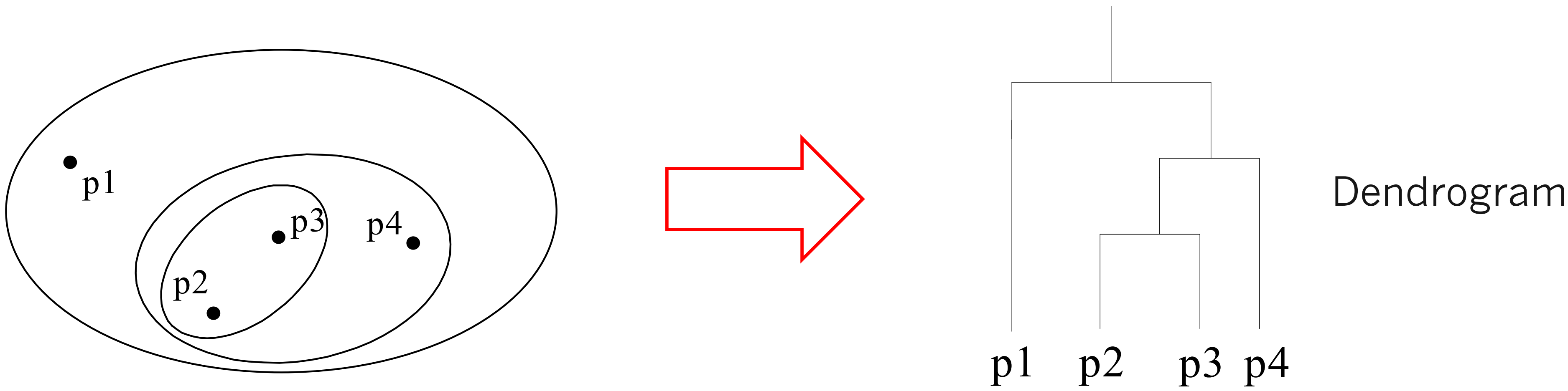


Types of Clusterings

- Partitional Clustering
  - A division of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset



- Hierarchical Clustering
  - A set of nested clusters organized as a hierarchical tree



Aspects of Cluster Analysis

- A clustering algorithm
  - Partitional algorithms
  - Density-based algorithms
  - Hierarchical algorithms
  - …
- A proximity (similarity, or dissimilarity) measure
  - Euclidean distance
  - Cosine similarity
  - Data type-specific similarity measures
  - Domain-specific similarity measures
- Clustering quality
  - Intra-clusters distance $\Rightarrow$ minimized
  - Inter-clusters distance $\Rightarrow$ maximized
  - The clustering should be useful with regard to the goal of the analysis

The Notion of a Cluster is Ambiguous

How many clusters do you see?

Six Clusters

Two Clusters

Four Clusters

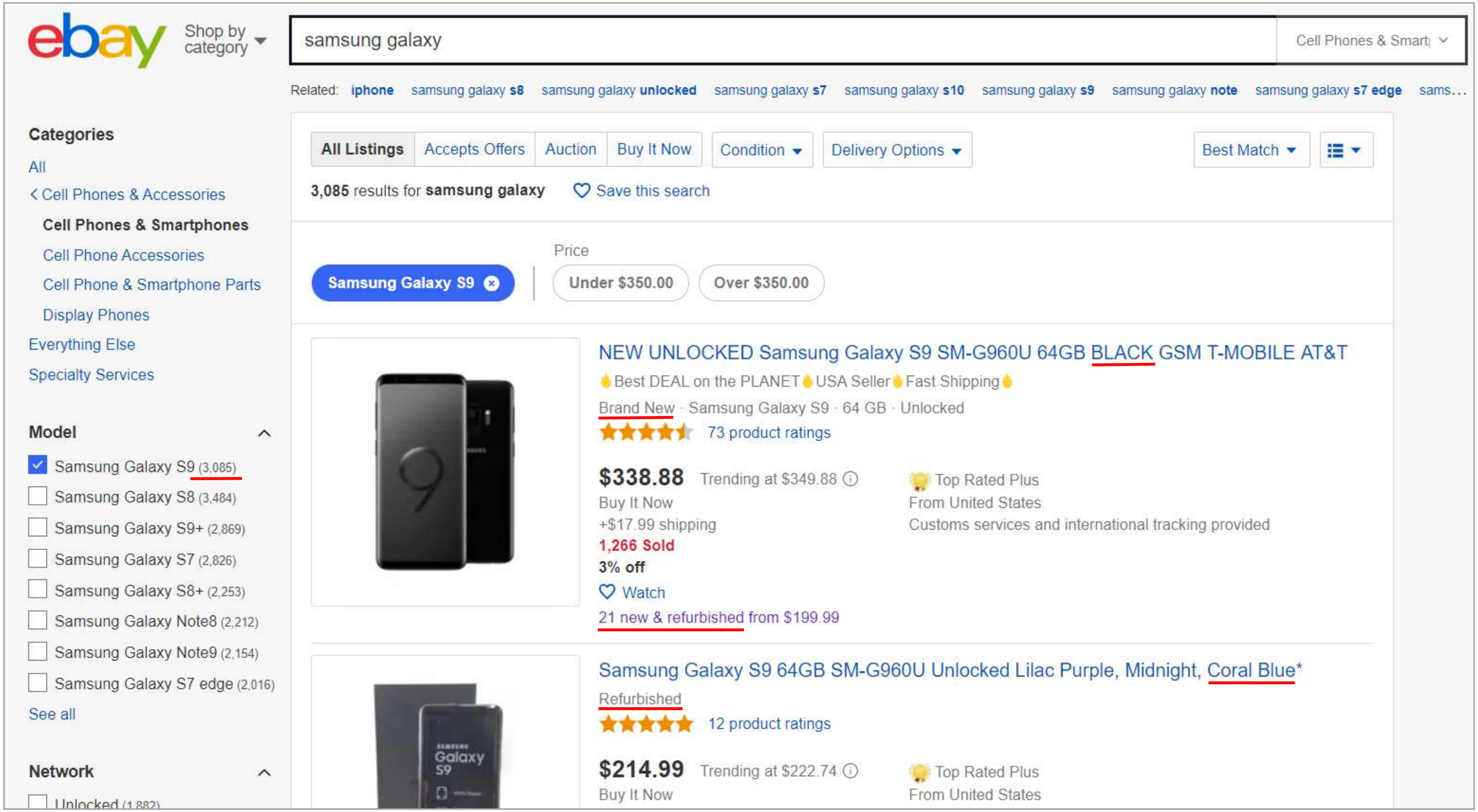
The usefulness of a clustering depends on the goal of the analysis

Example Application 1: Market Segmentation

- Goal: Identify groups of similar customers
- Level of granularity depends on the task at hand
- Relevant customer attributes depend on the task at hand

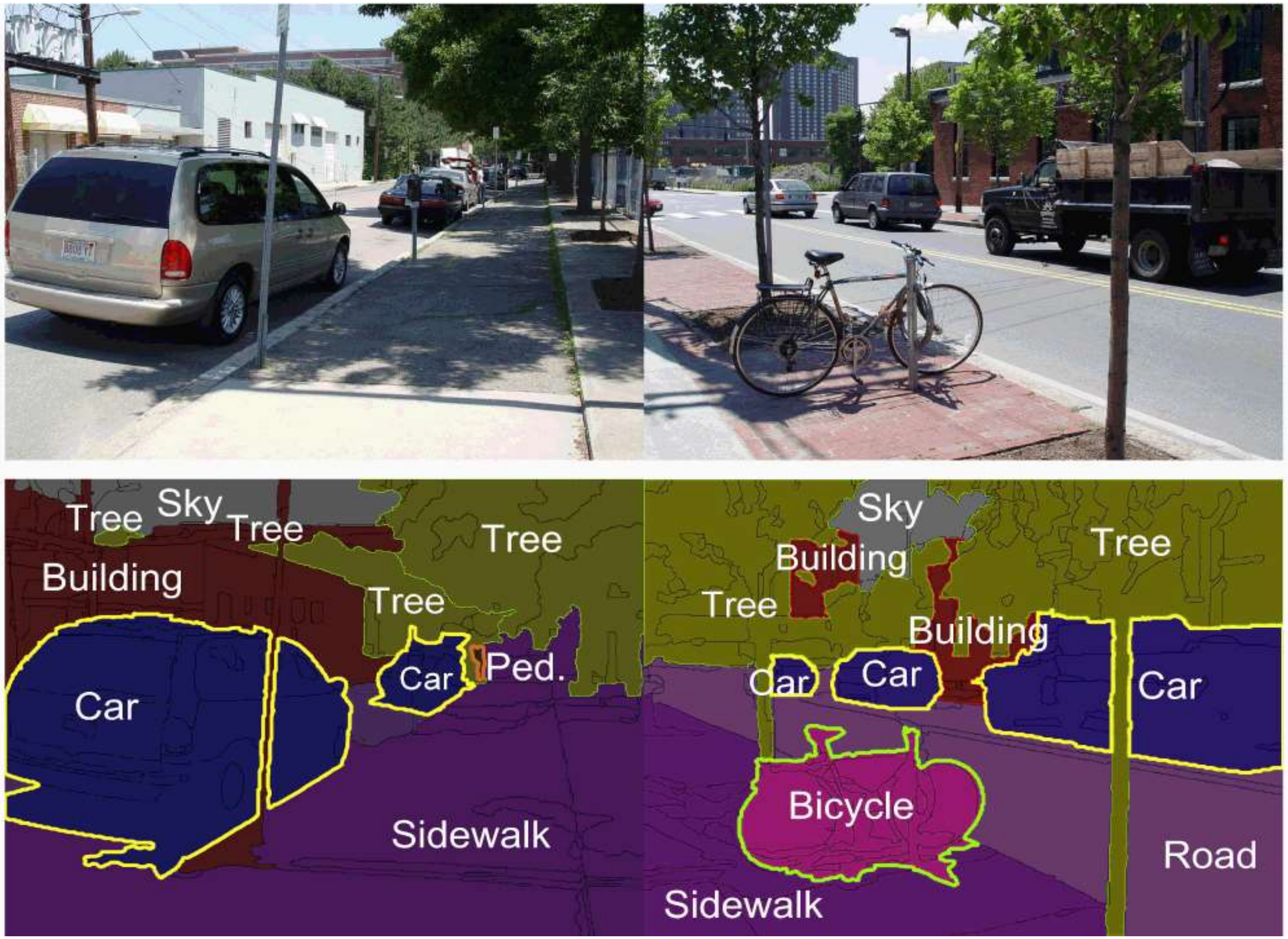
Example Application 2: E-Commerce

– Identify offers of the same product on electronic markets



Example Application 3: Image Recognition

– Identify parts of an image that belong to the same object



Cluster Analysis as Unsupervised Learning

- Supervised learning: Discover patterns in the data that relate data attributes with a target (class) attribute
  - these patterns are then utilized to predict the values of the target attribute in unseen data instances
  - the set of classes is known before
  - training data is often provided by human annotators
- Unsupervised learning: The data has no target attribute
  - we want to explore the data to find some intrinsic patterns in it
  - the set of classes/clusters is not known before
  - no training data is used
- Cluster Analysis is an unsupervised learning task

2. K-Means Clustering

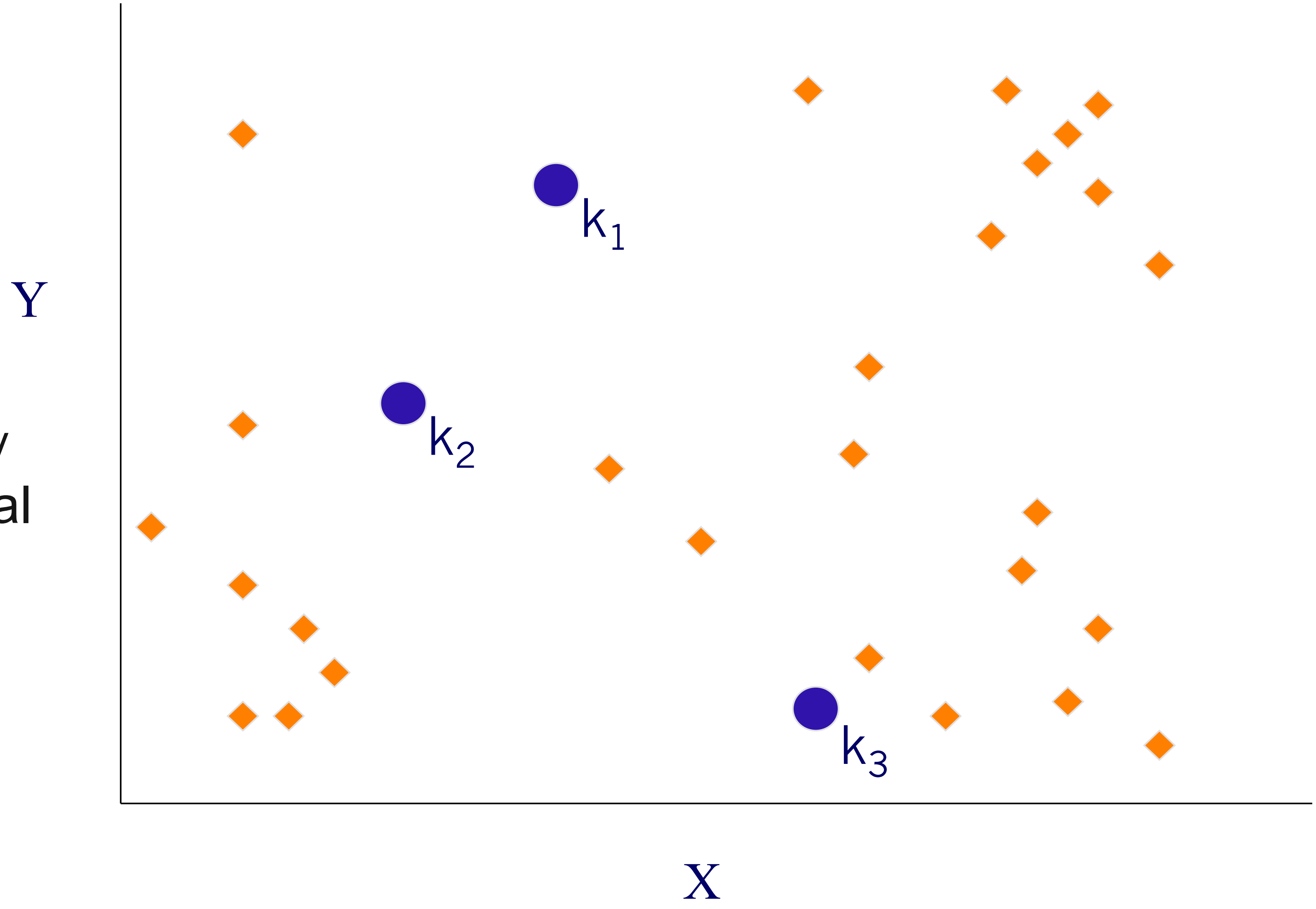
- Partitional clustering algorithm
- Each cluster is associated with a centroid (center point)
- Each point is assigned to the cluster with the closest centroid
- Number of clusters K must be specified beforehand

- The K-Means algorithm is very simple:

Select K points as the initial centroids.
repeat
    Form K clusters by assigning all points to the closest centroid.
    Recompute the centroid of each cluster.
until The centroids don't change

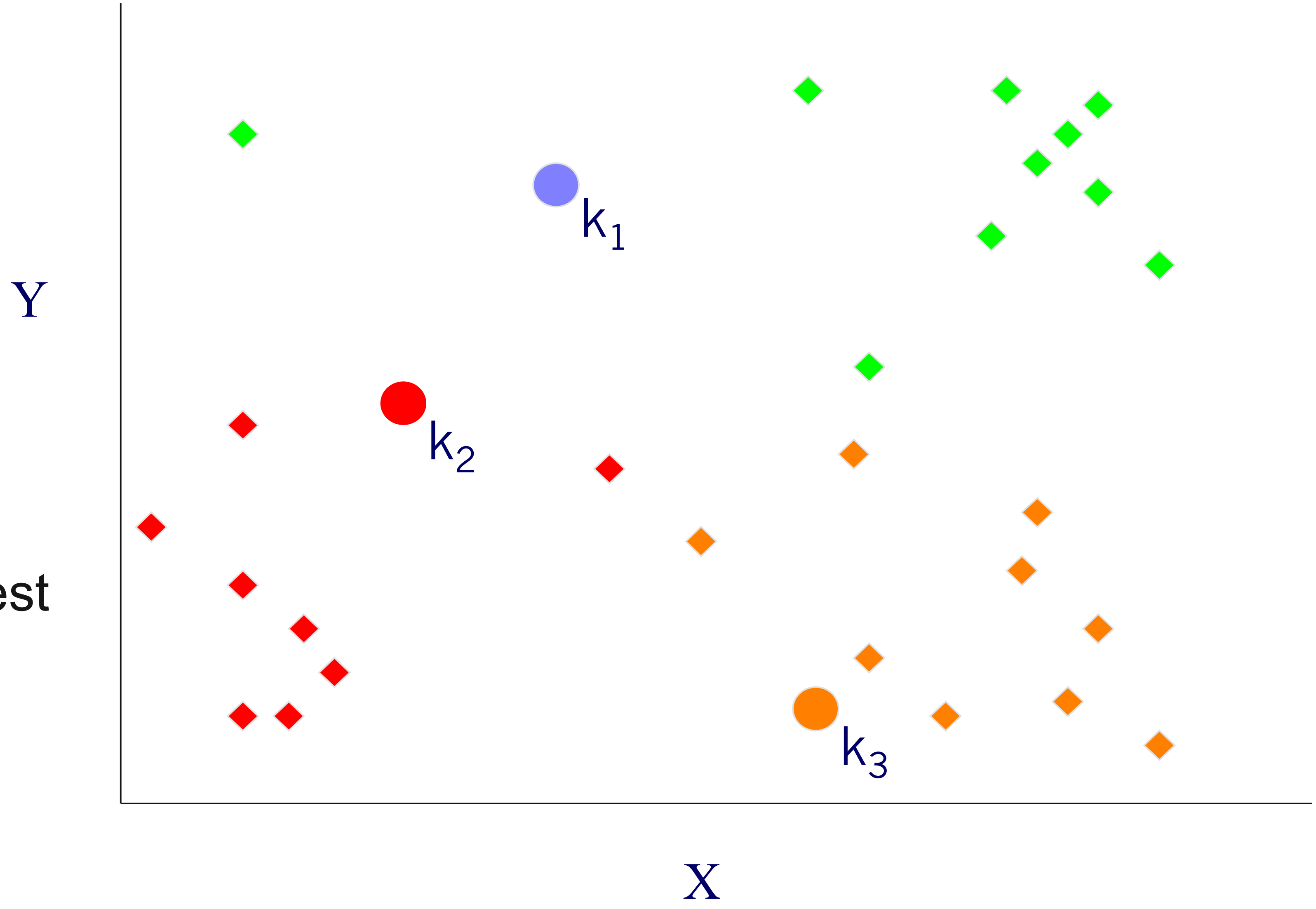
K-Means Example, Step 1

Randomly pick 3 initial centroids

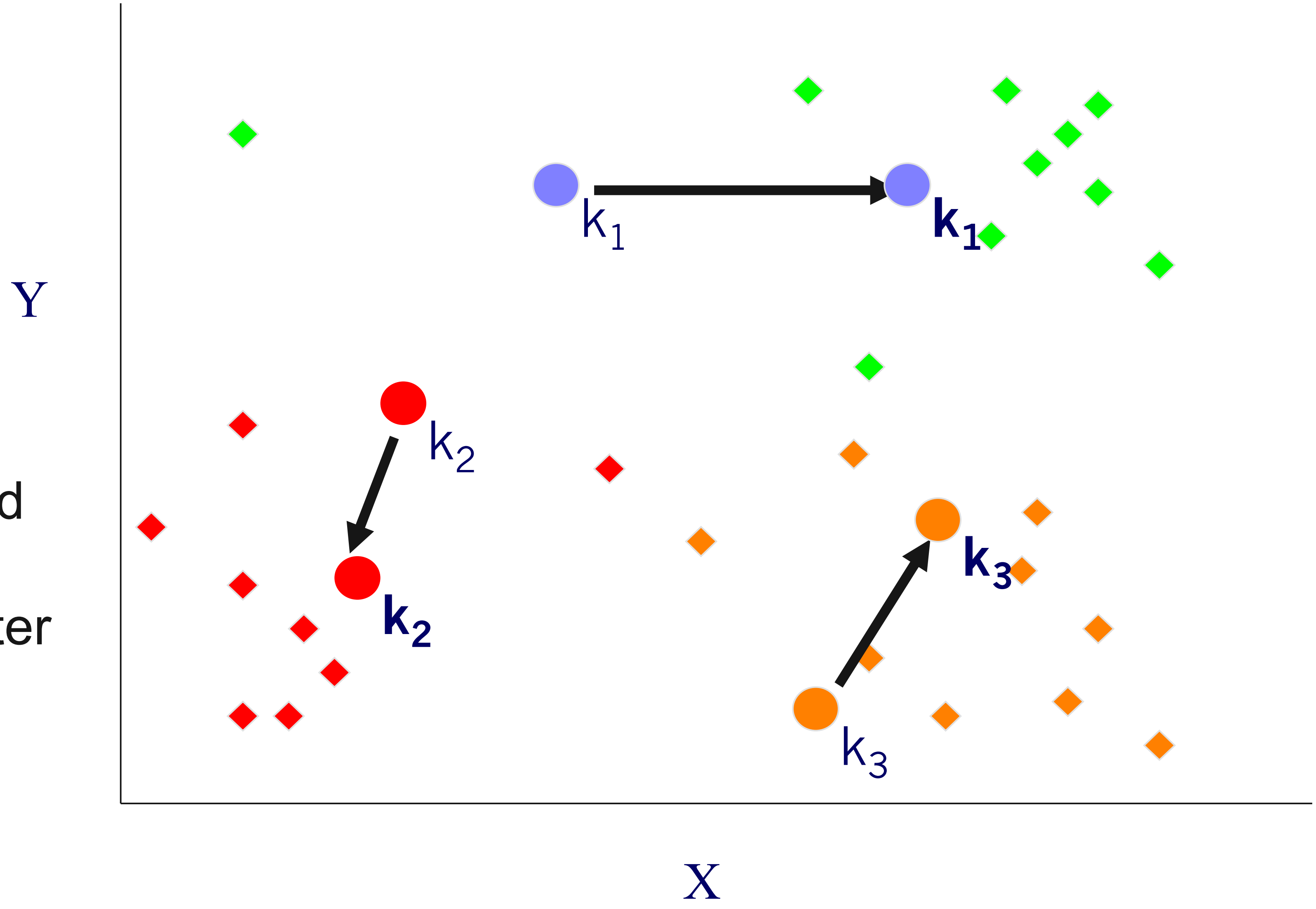


K-Means Example, Step 2

Assign each point to the closest centroid



K-Means Example, Step 3

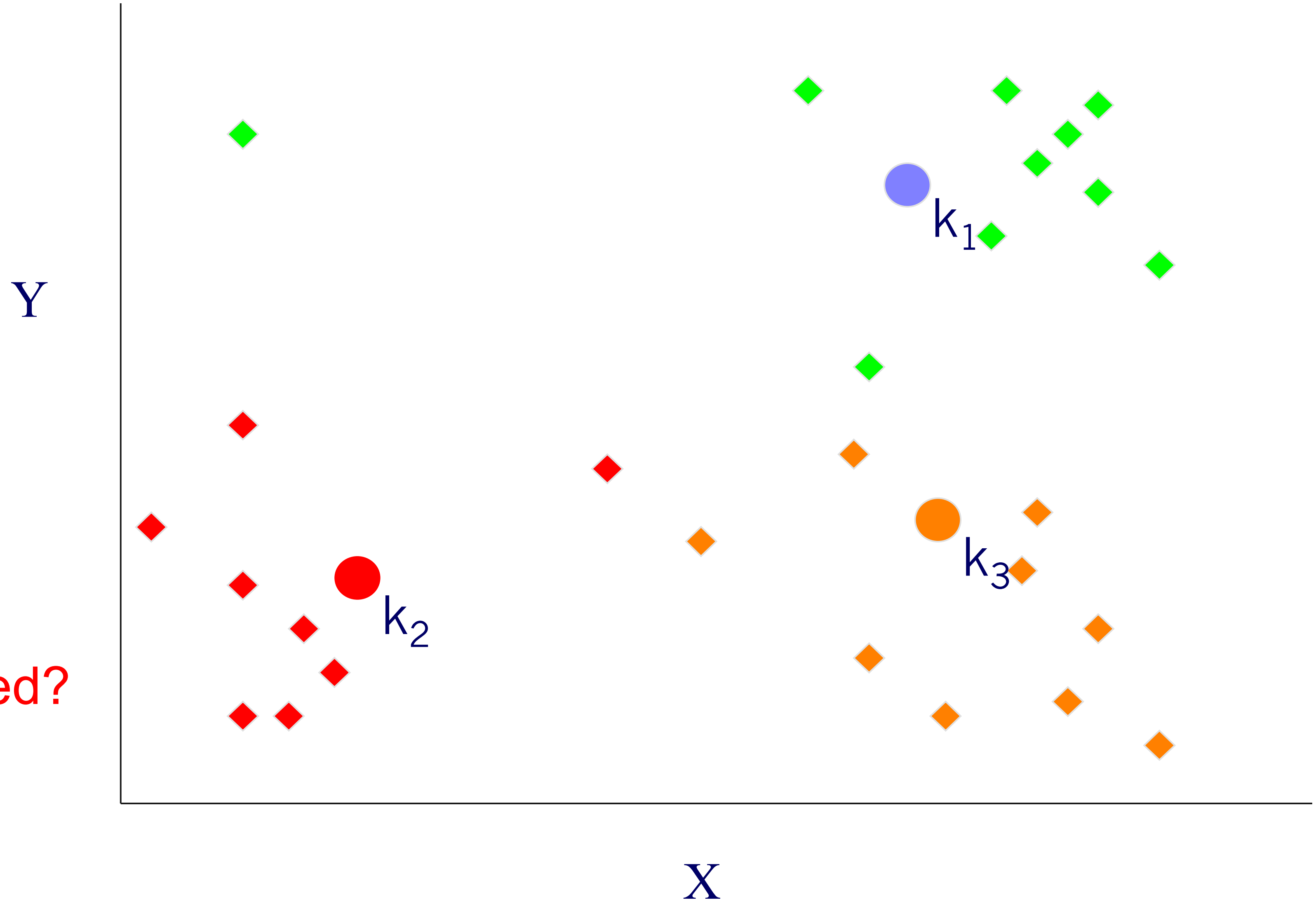


Move
each centroid
to the mean
of each cluster

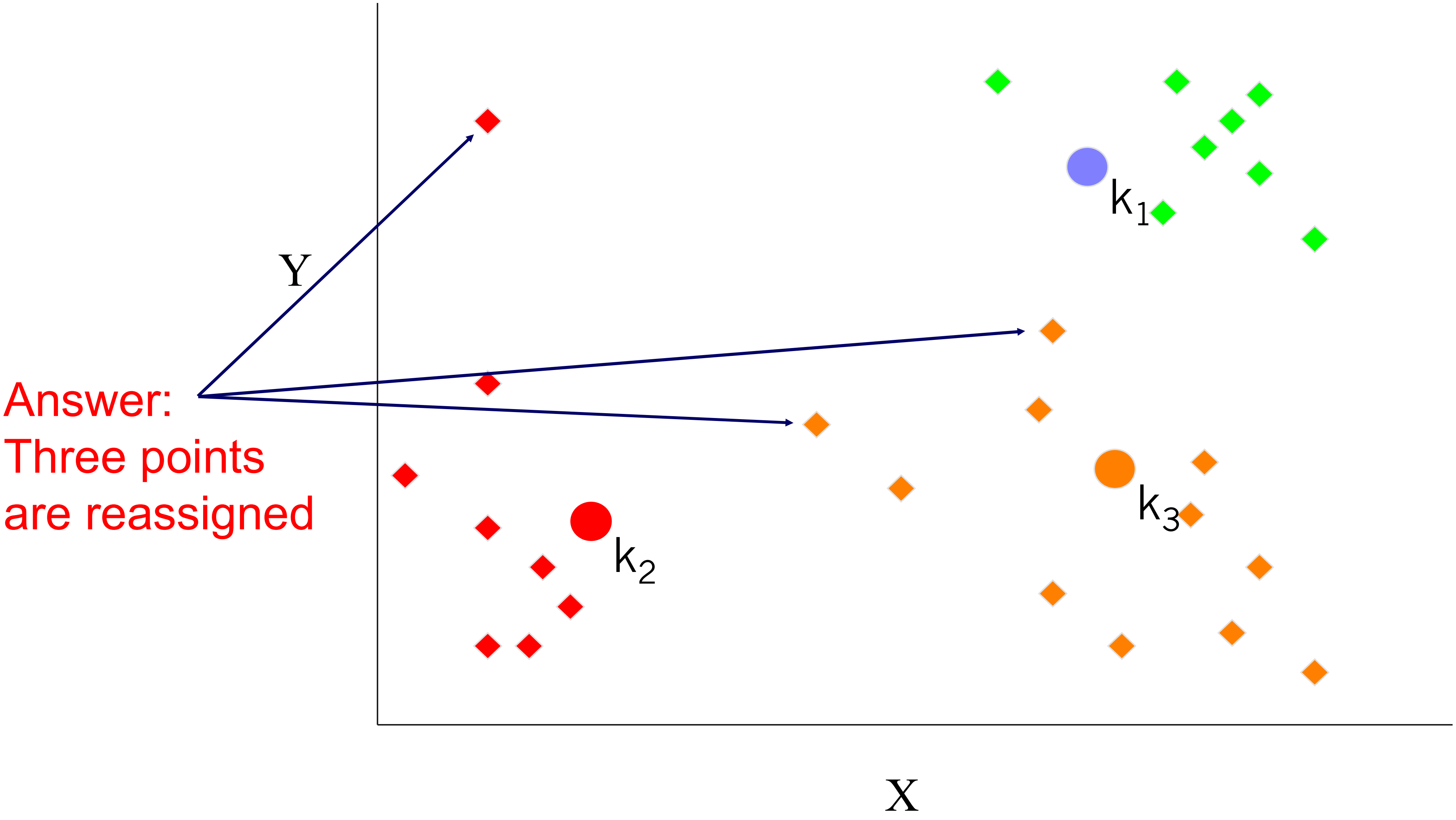
K-Means Example, Step 4

Reassign points if they are now closer to a different centroid

Question: Which points are reassigned?

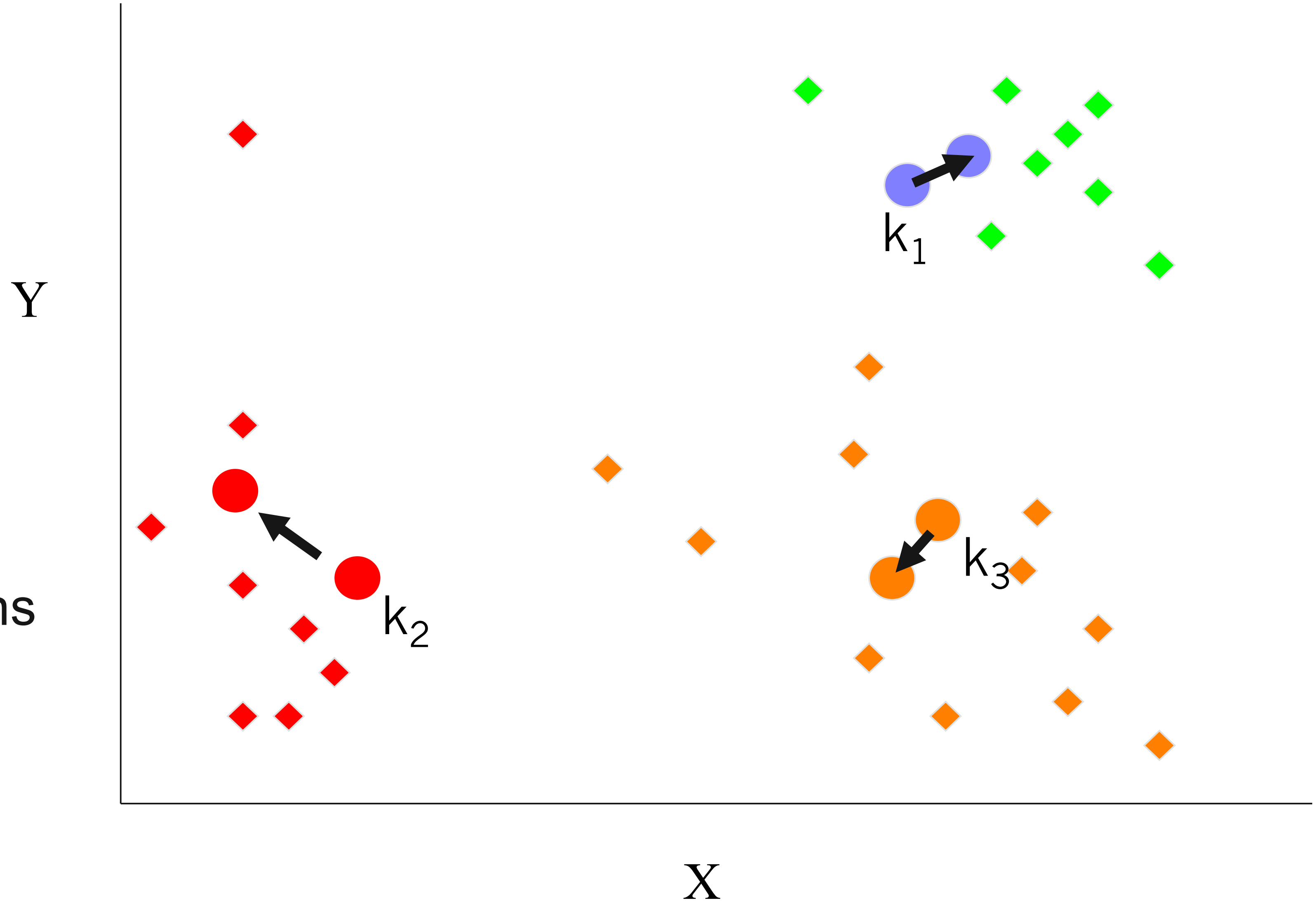


K-Means Example, Step 4



K-Means Example, Step 5

1. Re-compute cluster means
2. Move centroids to new cluster means



Convergence Criteria

Default convergence criterion

– no (or minimum) change of centroids

Alternative convergence criteria

1. no (or minimum) re-assignments of data points to different clusters
2. stop after x iterations
3. minimum decrease in the sum of squared error (SSE)
   - see next slide

Evaluating K-Means Clusterings

– Widely used cohesion measure: Sum of Squared Error (SSE)
  - For each point, the error is the distance to the nearest centroid
  - To get SSE, we square these errors and sum them

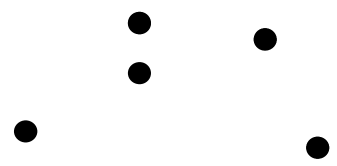
$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2$$

  - C_j is the j-th cluster
  - m_j is the centroid of cluster C_j (the mean vector of all the data points in C_j)
  - $dist(x, m_j)$ is the distance between data point x and centroid m_j

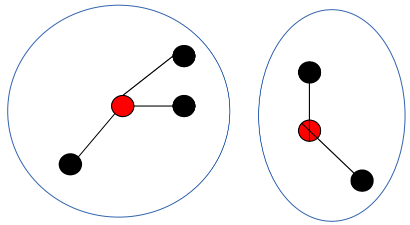
– Given several clusterings (=groupings), we should prefer the one with the smallest SSE

Illustration: Sum of Squared Error

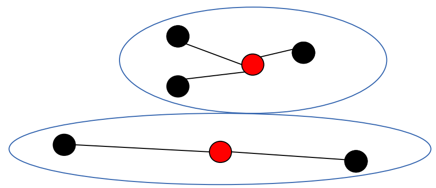
– Cluster analysis problem



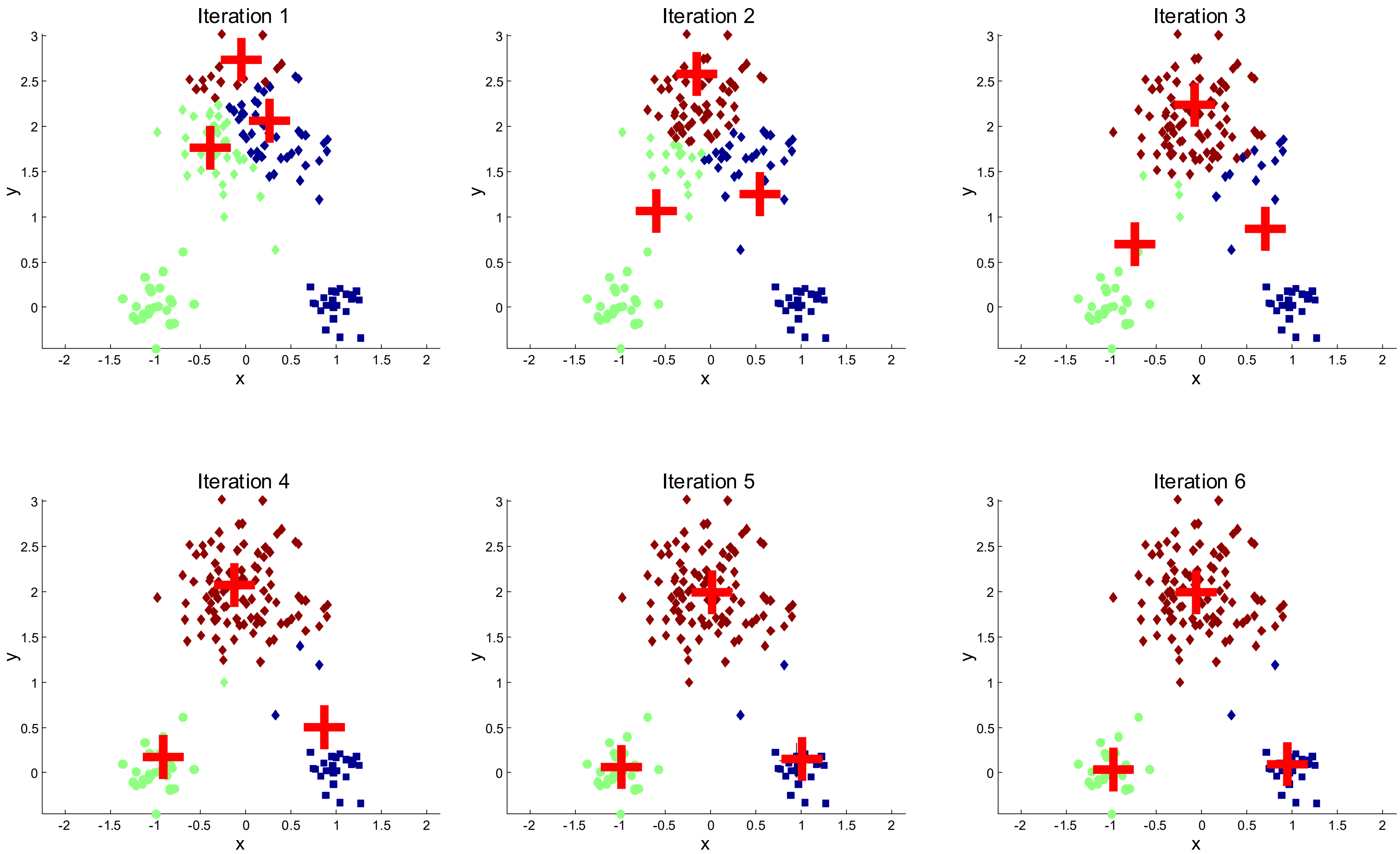
– Good clustering
  - small distances to centroids



– Not so good clustering
  - larger distances to centroids

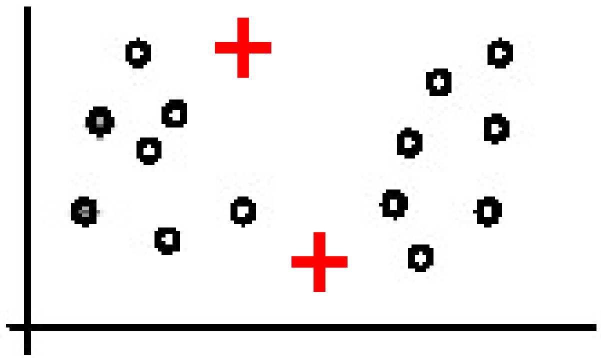


K-Means Clustering – Second Example

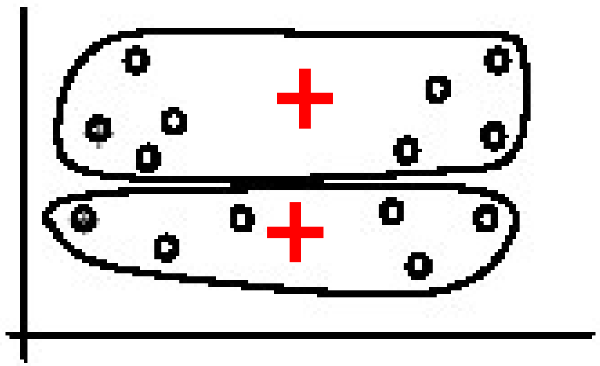


Weaknesses of K-Means: Initial Seeds

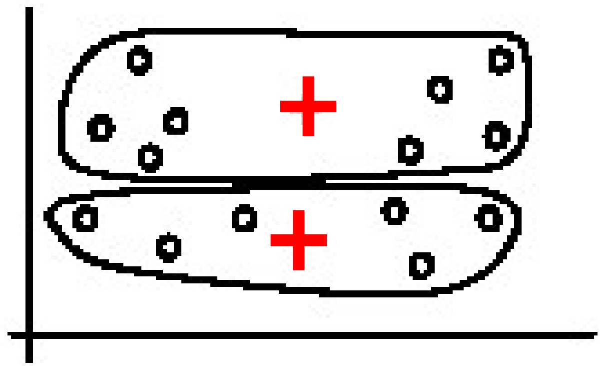
Clustering results may vary significantly depending on initial choice of seeds (number and position of seeds)



(A). Random selection of seeds (centroids)



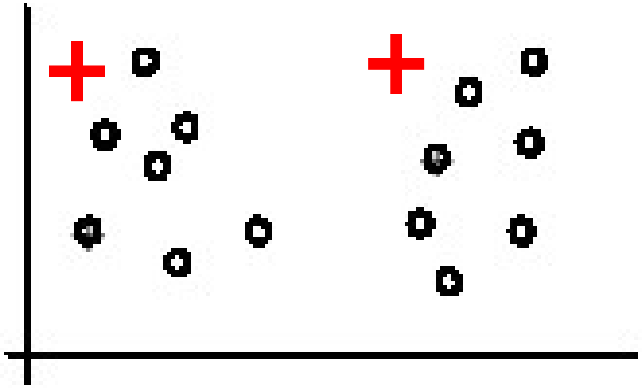
(B). Iteration 1



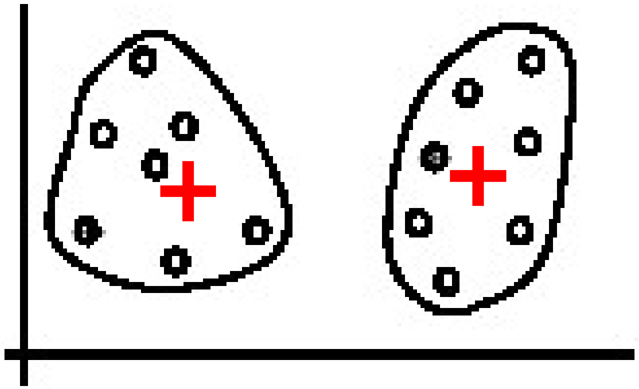
(C). Iteration 2

Weaknesses of K-Means: Initial Seeds

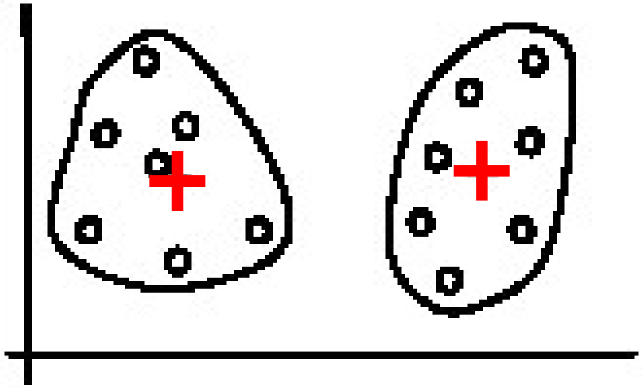
If we use different seeds, we get good results



(A). Random selection of *k* seeds (centroids)

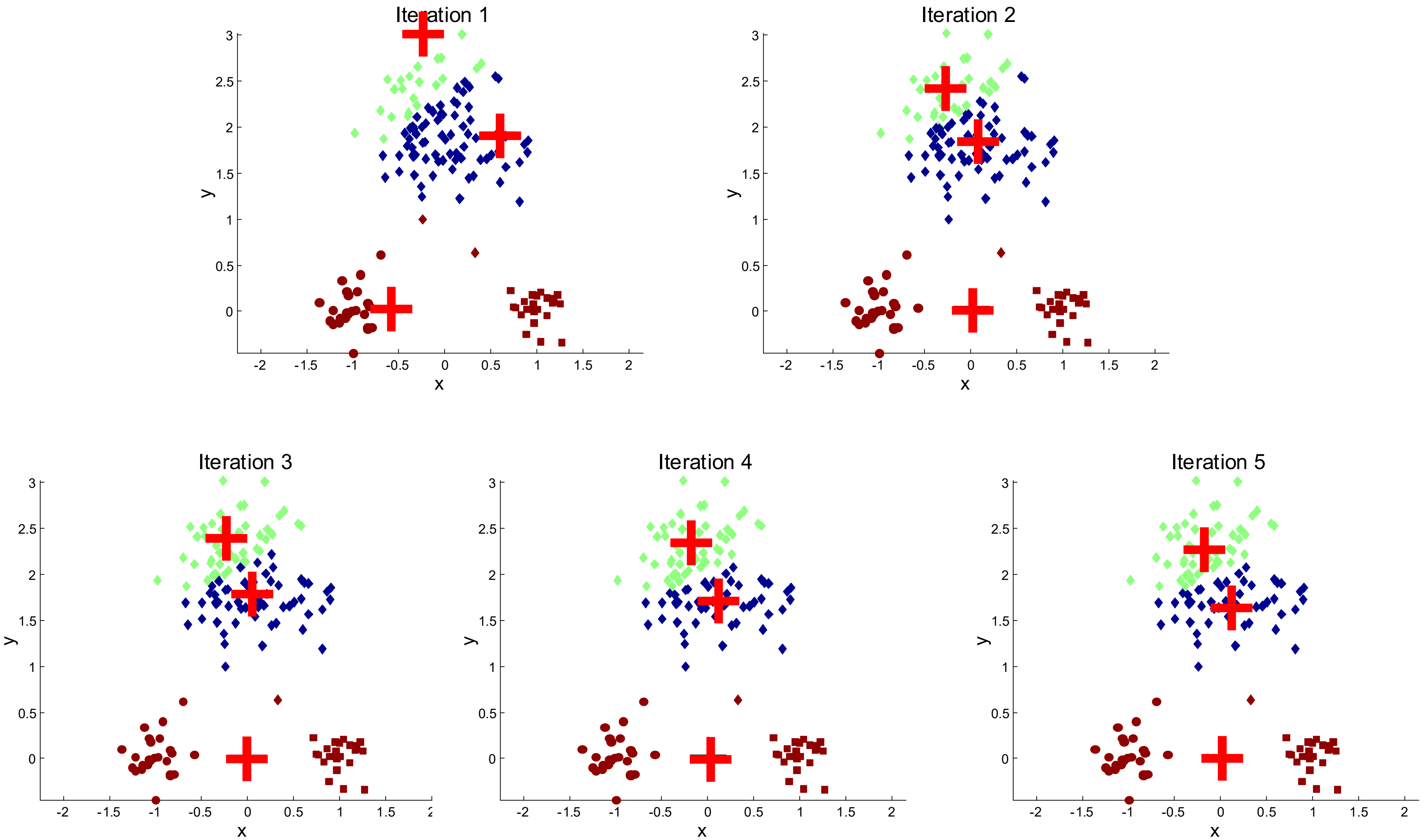


(B). Iteration 1



(C). Iteration 2

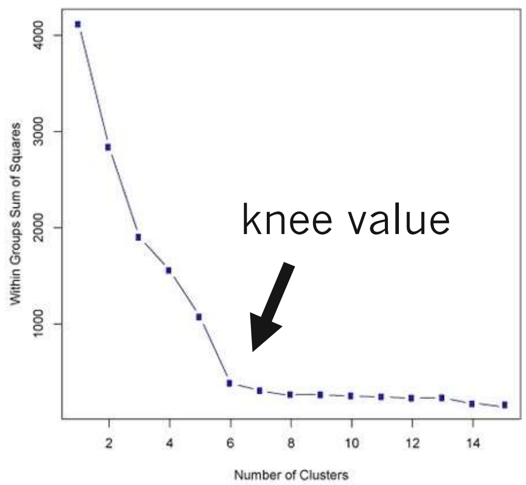
Bad Initial Seeds – Second Example



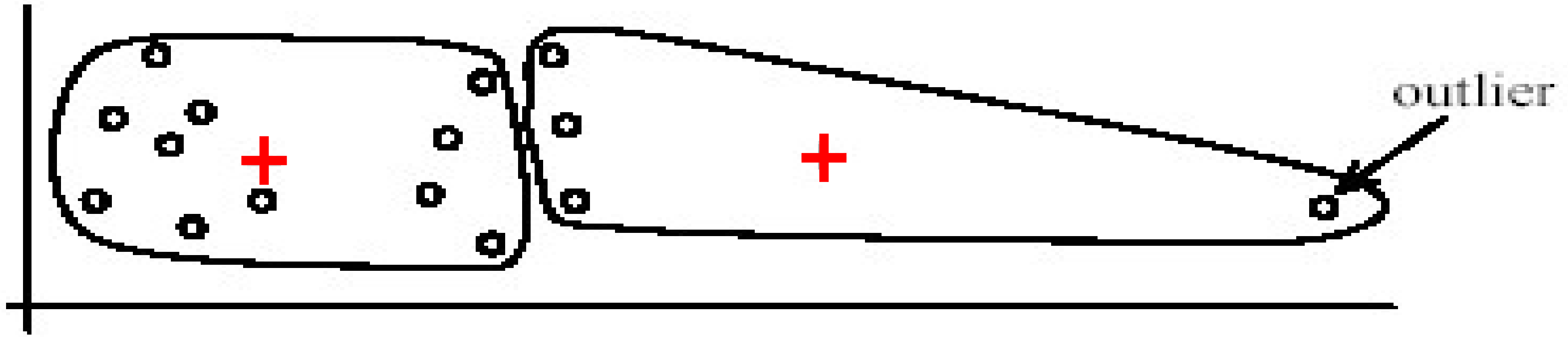
Increasing the Chance of Finding Good Clusters

1. Restart a number of times with different random seeds
   – chose the resulting clustering with the smallest sum of squared error (SSE)

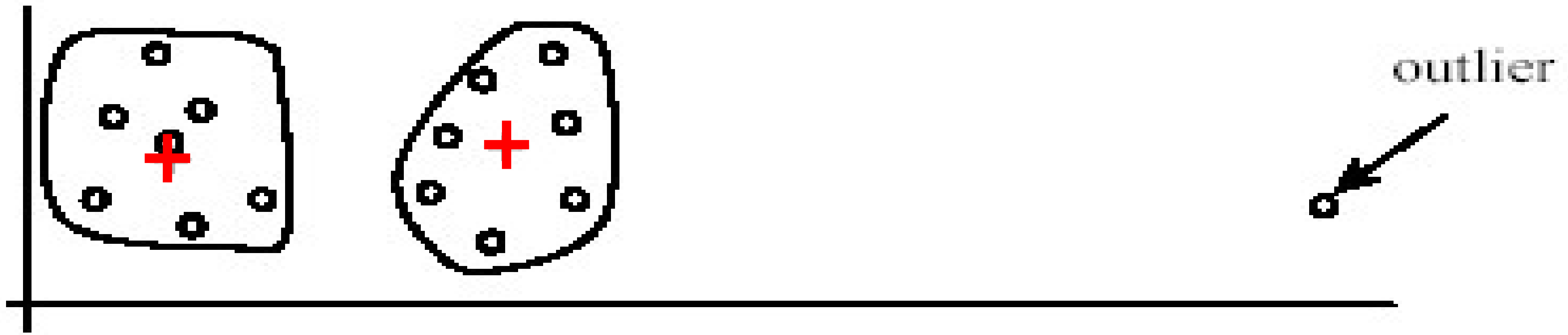
2. Run k-means with different values of k
   – The SSE for different values of k cannot directly be compared
     – think: what happens for k → number of examples?
   – Workarounds
     1. Choose k where SSE improvement decreases (knee value of k)
     2. Employ X-Means
        – variation of K-Means algorithm that automatically determines k
        – starts with small k, then splits large clusters until improvement decreases



Weaknesses of K-Means: Problems with Outliers



(A): Undesirable clusters



(B): Better clusters

Weaknesses of K-Means: Problems with Outliers

Approaches to deal with outliers:

1. K-Medoids
   - K-Medoids is a K-Means variation that uses the median of each cluster instead of the mean
   - Medoids are the most central existing data points in each cluster
   - K-Medoids is more robust against outliers as the median is less affected by extreme values:
     - Mean and Median 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
2. DBSCAN
   - Density-based clustering method that removes outliers
     - see next section

K-Means Clustering Summary

Advantages

– Simple, understandable

– Efficient time complexity: O(n * K * I * d)

  where

  - *n = number of points*
  - *K = number of clusters*
  - *I = number of iterations*
  - *d = number of attributes*

Disadvantages

– Need to determine number of clusters

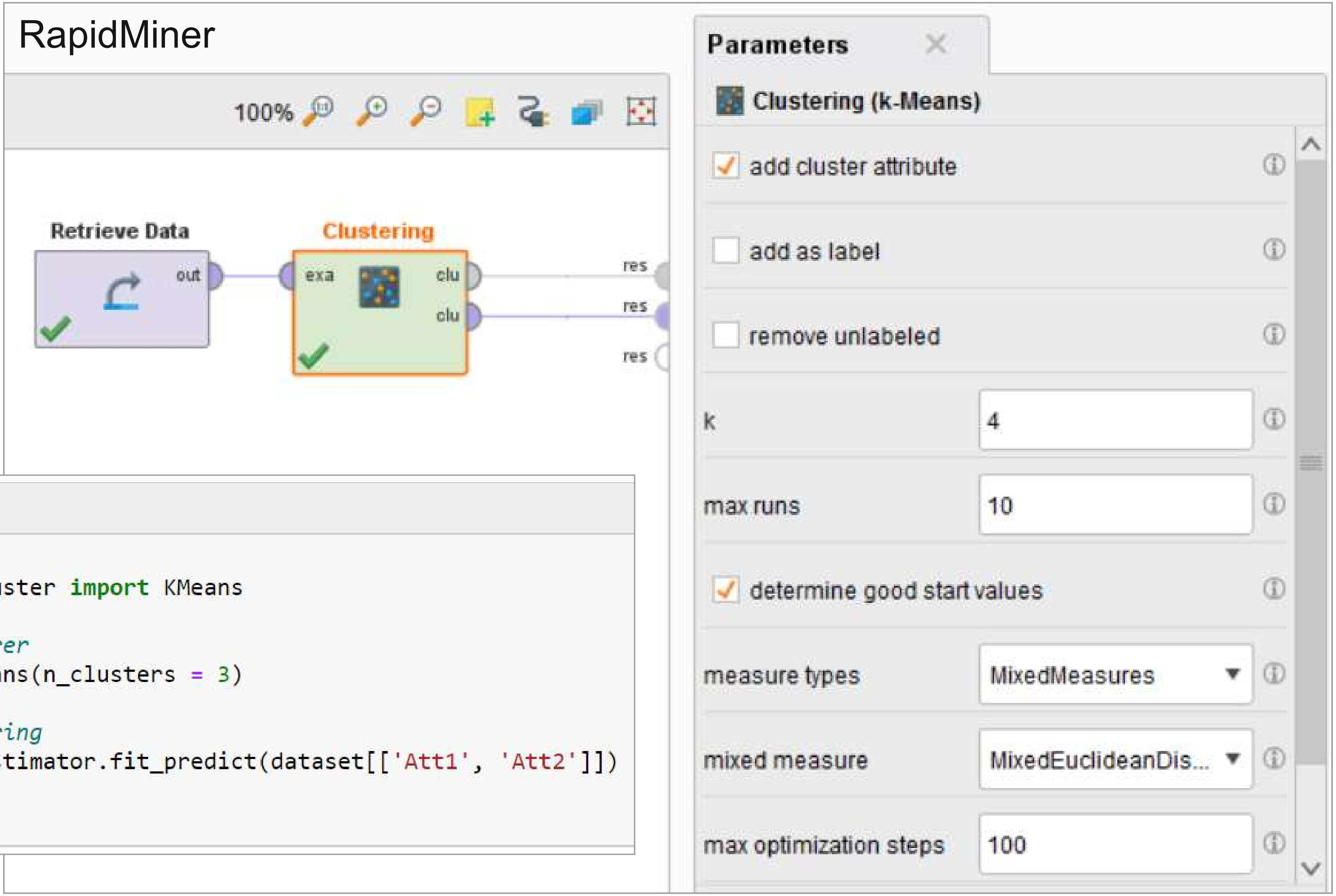
– All items are forced into a cluster

– Sensitive to outliers

– Does not work for non-globular clusters

K-Means Clustering in RapidMiner and Python

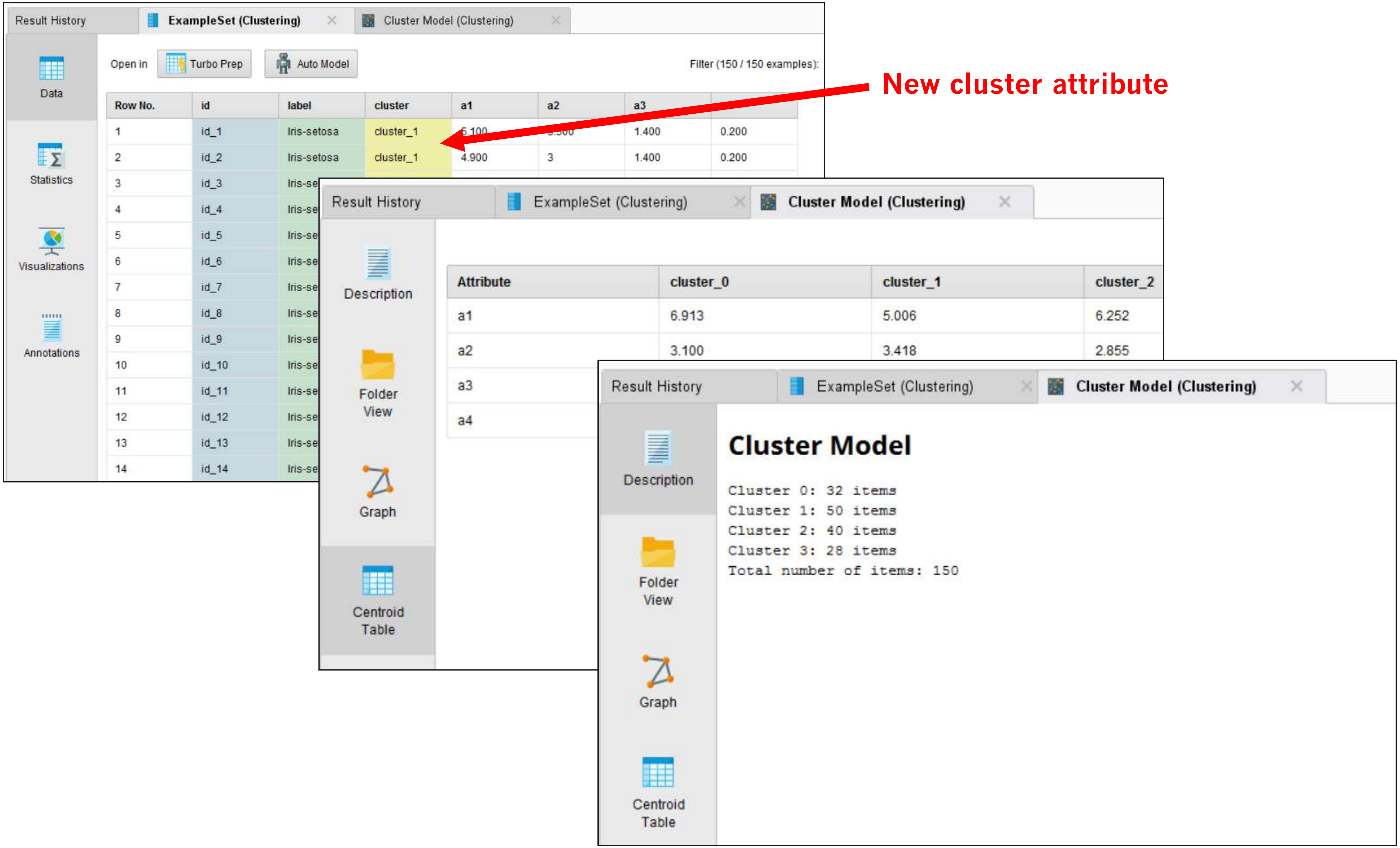
RapidMiner



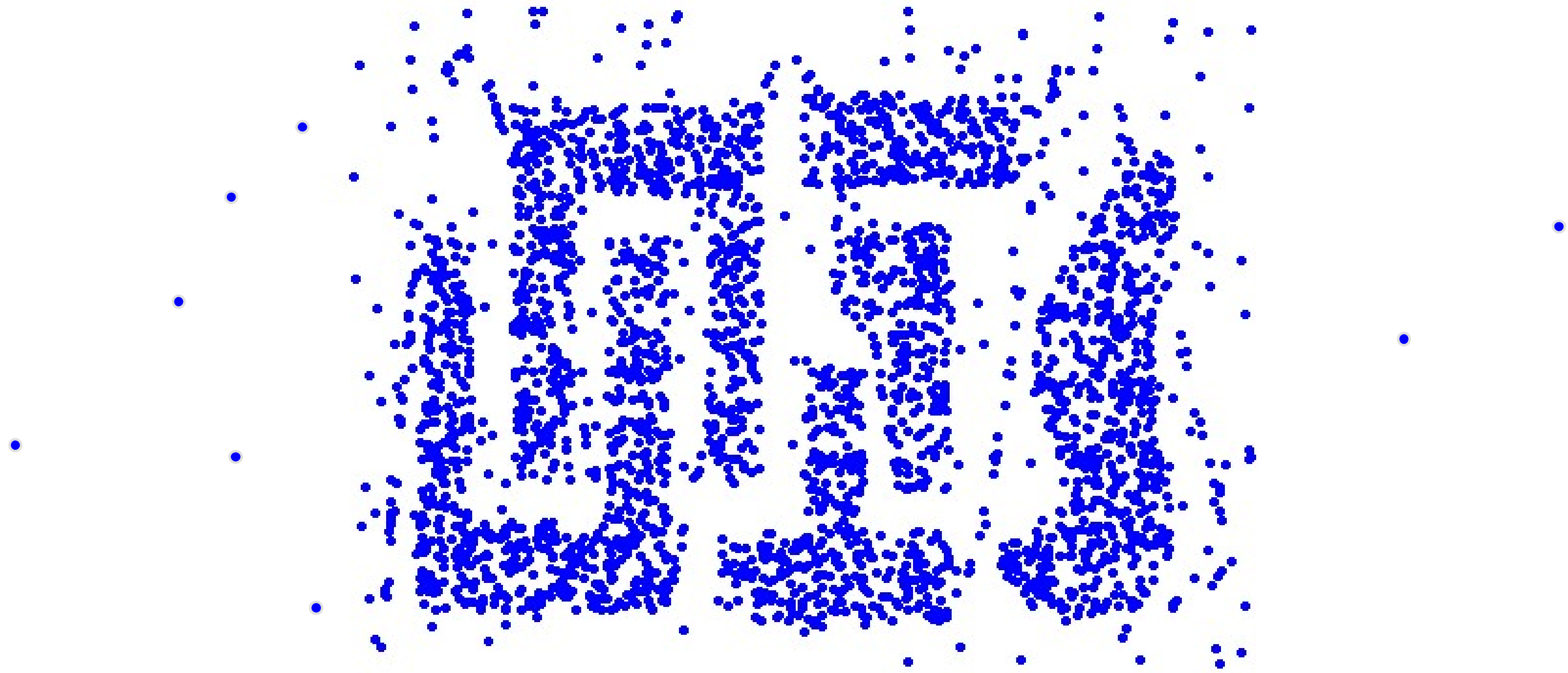
Python

```python
# import KMeans
from sklearn.cluster import KMeans

# create clusterer
estimator = KMeans(n_clusters = 3)

# create clustering
cluster_ids = estimator.fit_predict(dataset[['Att1', 'Att2']])
```

K-Means Clustering Results



3. Density-based Clustering



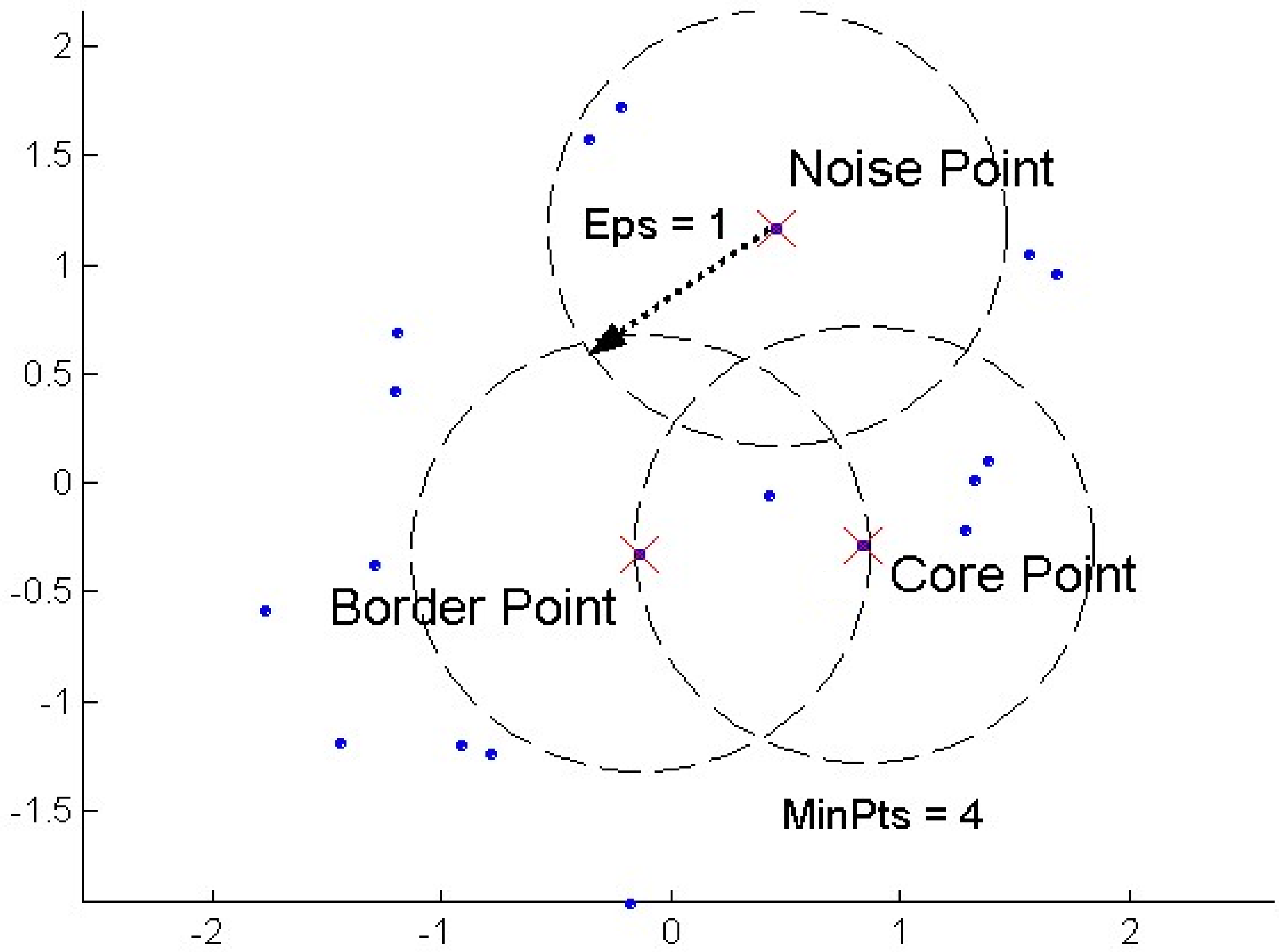
Challenging use case for K-Means because

- Problem 1: Non-globular shapes
- Problem 2: Outliers / noise points

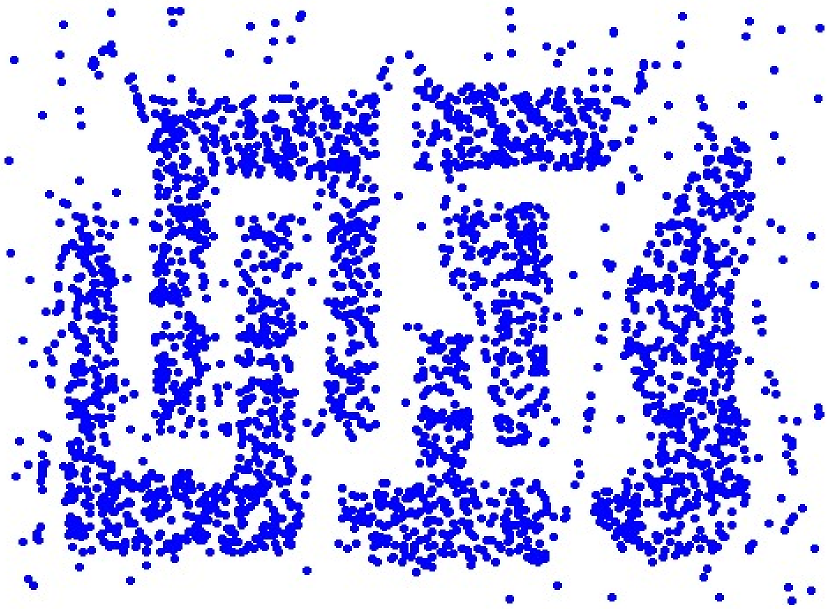
DBSCAN

- DBSCAN is a density-based algorithm
  - Density = number of points within a specified radius Epsilon (Eps)
- Divides data points into three classes:
  1. A point is a core point if it has at least a specified number of neighboring points (MinPts) within the specified radius Eps
     - the point itself is counted as well
     - these points form the interior of a dense region (cluster)
  2. A border point has fewer points than MinPts within Eps, but is in the neighborhood of a core point
  3. A noise point is any point that is not a core point or a border point

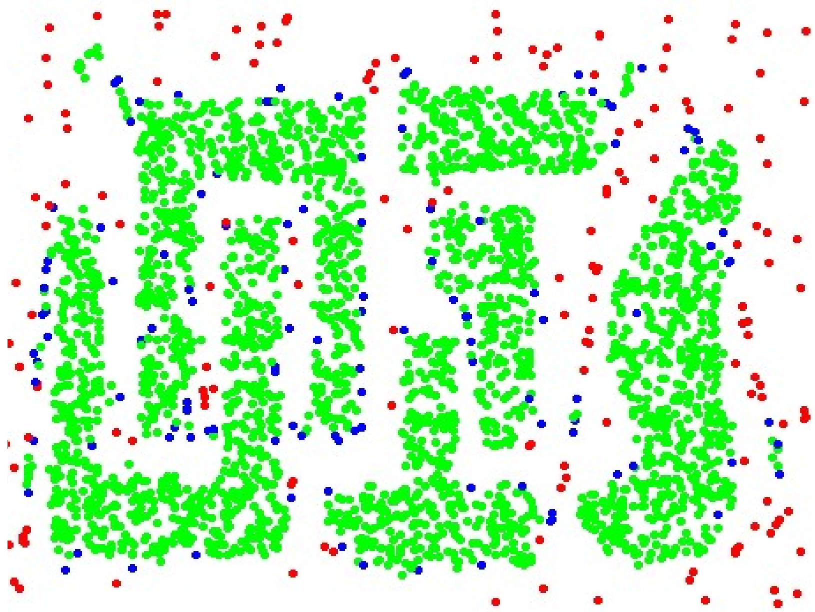
Examples of Core, Border, and Noise Points 1



Examples of Core, Border, and Noise Points 2



Original Points



Point types: core, border and noise

The DBSCAN Algorithm

Eliminates noise points and returns clustering of the remaining points:

1. Label all points as core, border, or noise points
2. Eliminate all noise points
3. Put an edge between all core points that are within Eps of each other
4. Make each group of connected core points into a separate cluster
5. Assign each border point to one of the clusters of its associated core points
   - as a border point can be at the border of multiple clusters
   - use voting if core points belong to different clusters
   - if equal vote, than assign border point randomly

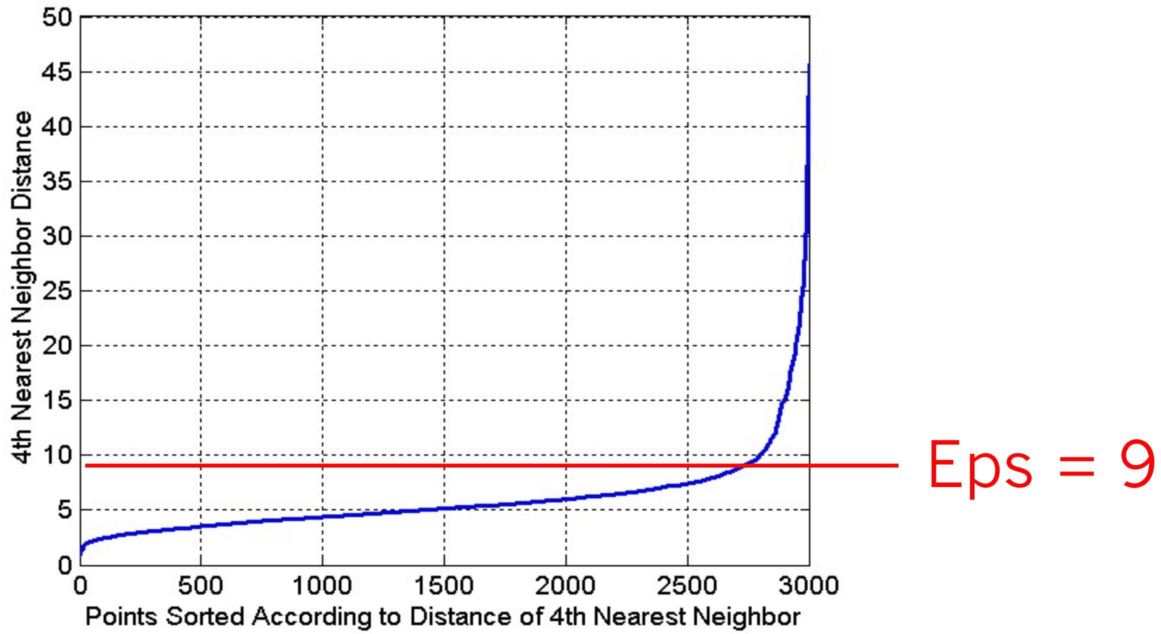
Time complexity: O(n log n)

- dominated by neighborhood search for each point using an index

How to Determine Suitable Eps and MinPts Values?

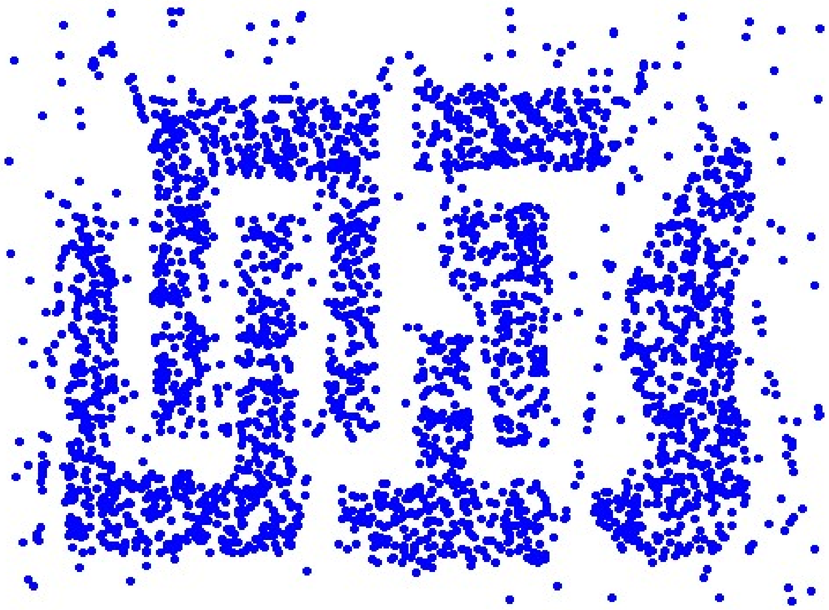
For points in a cluster, their k^{th} nearest neighbor (single point) should be at roughly the same distance. Noise points have their k^{th} nearest neighbor at farther distance

1. Start with setting MinPts = 4 (rule of thumb)
2. Plot sorted distance of every point to its k^{th} nearest neighbor:

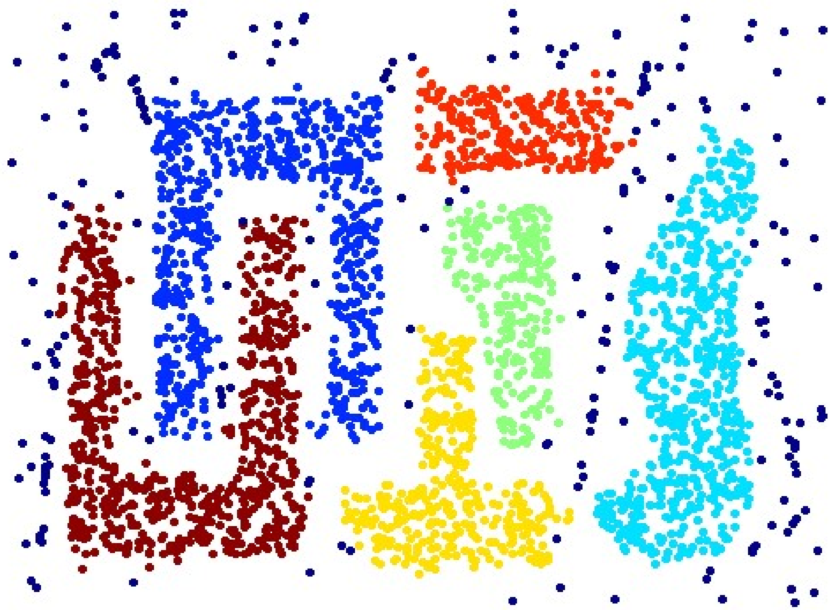


3. Set Eps to the sharp increase of the distances (start of noise points)
4. Decrease k if small clusters are labeled as noise (subjective decision)
5. Increase k if outliers are included into the clusters (subjective decision)

When DBSCAN Works Well



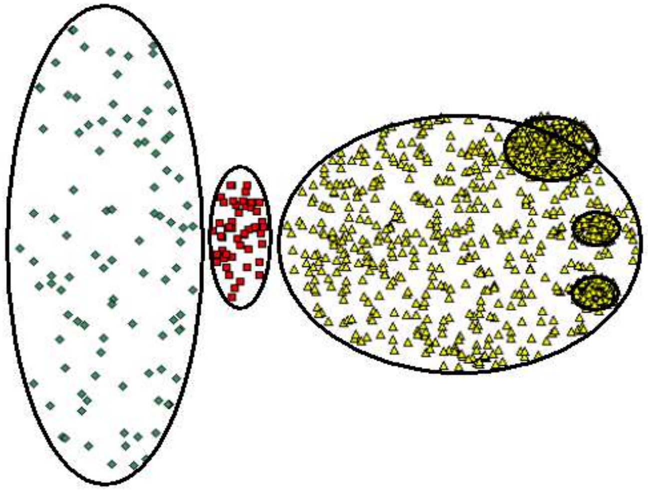
Original Points



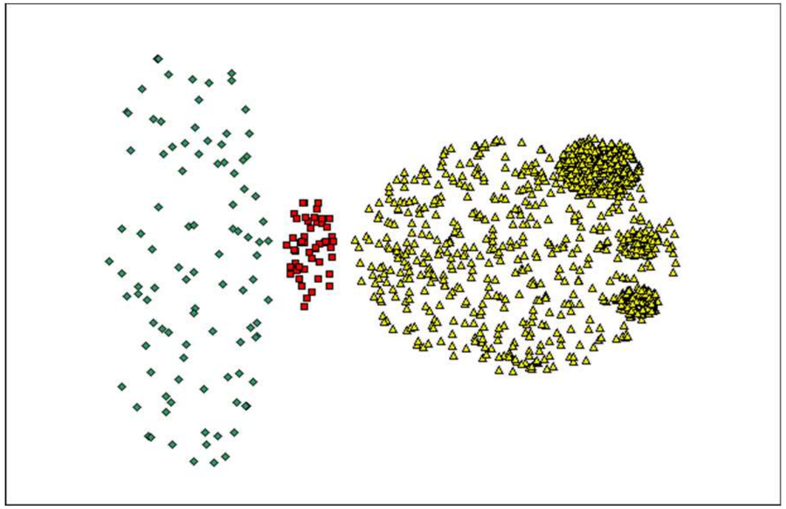
Clusters

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

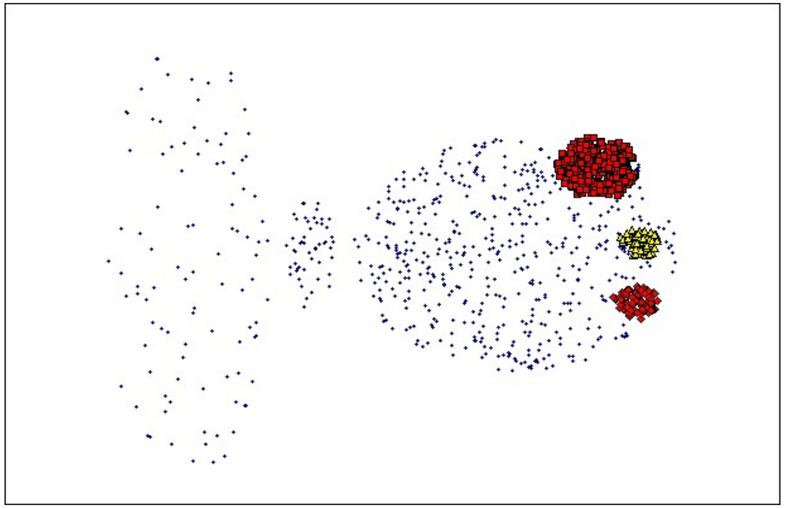
When DBSCAN Does NOT Work Well



Original Points



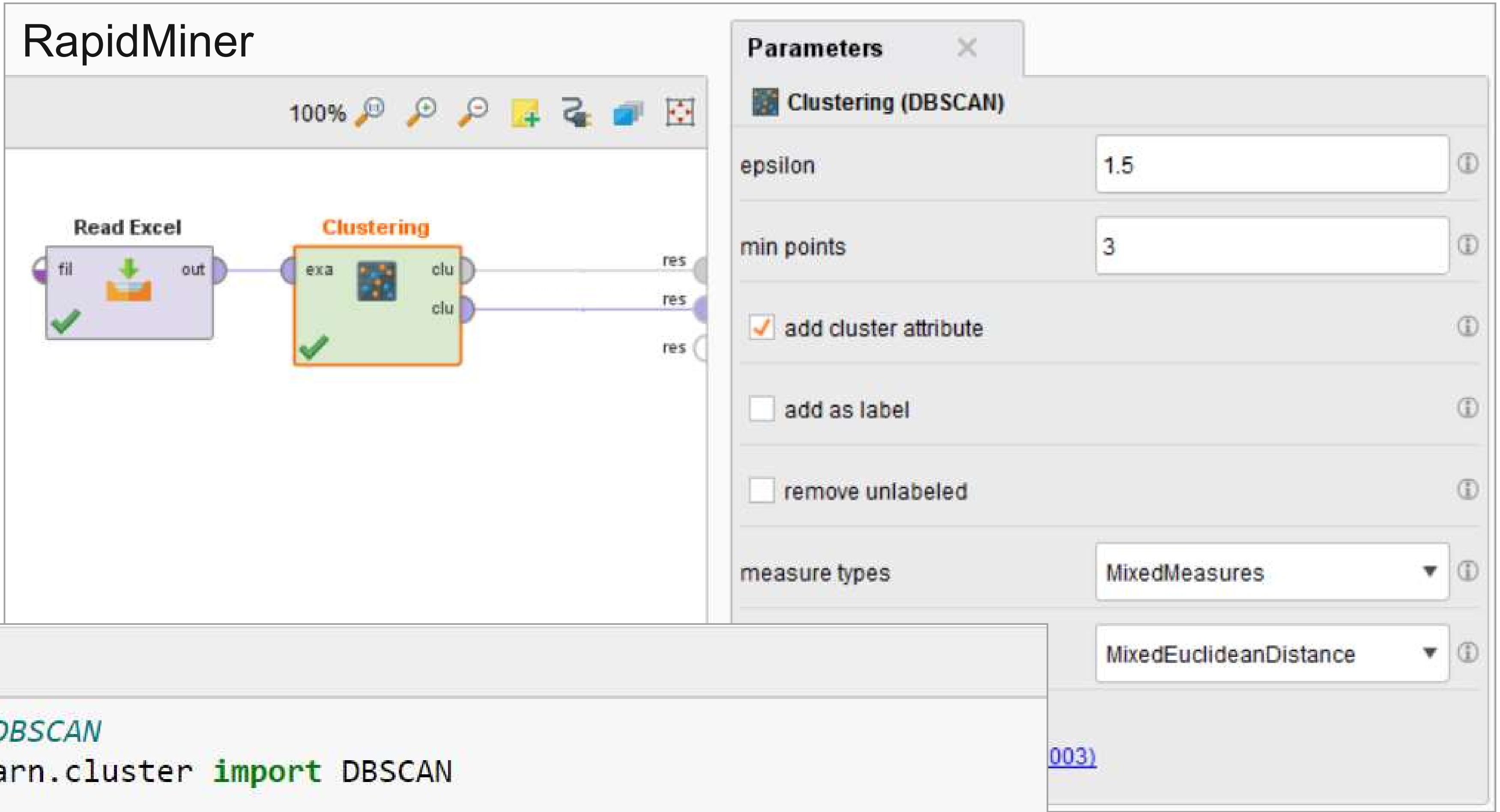
(MinPts=4, Eps=9.92)



(MinPts=4, Eps=9.75)

DBSCAN has problems with datasets of varying densities.

DBSCAN in RapidMiner and Python



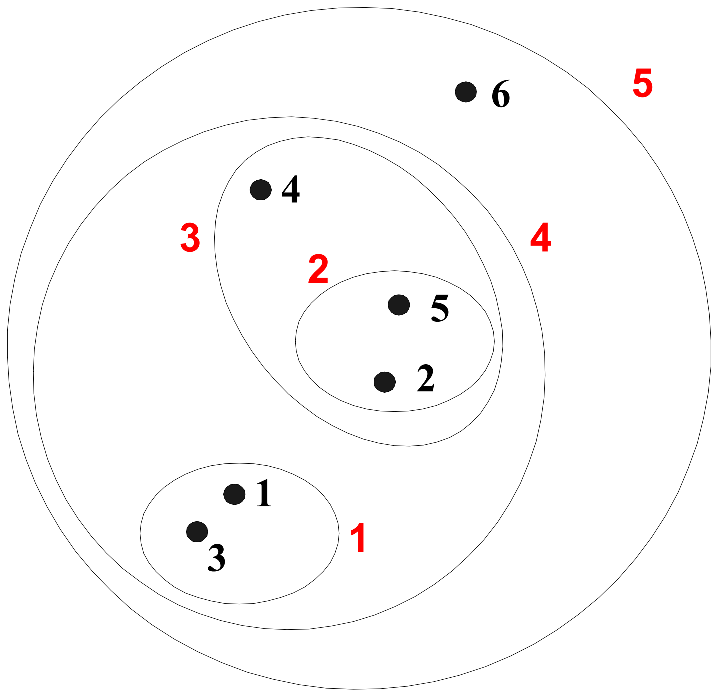
Python

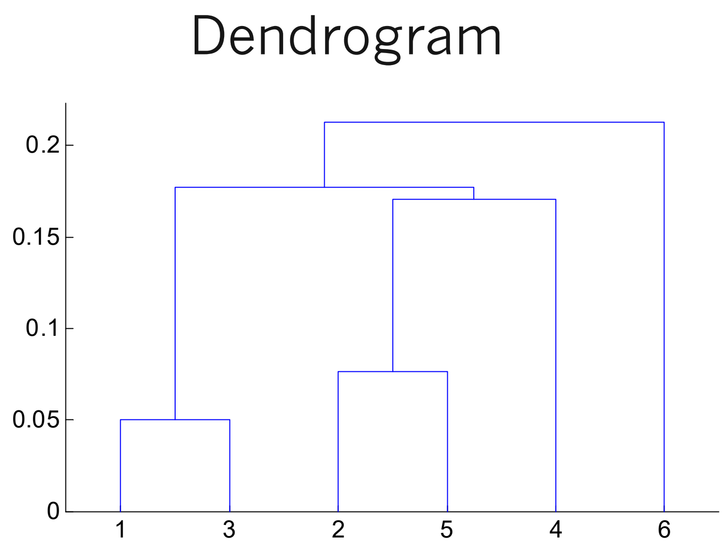
```python
# import DBSCAN
from sklearn.cluster import DBSCAN

# create the clusterer
clusterer = DBSCAN(min_samples=3, eps=1.5, metric='euclidean')

# create the clusters
clusters = clusterer.fit_predict(dataset[['Att1', 'Att2']])
```

4. Hierarchical Clustering

– Produces a set of nested clusters organized as a hierarchical tree

– Can be visualized as a dendrogram
  - A tree like diagram that records the sequences of merges or splits
  - The y-axis displays the former distance between merged clusters

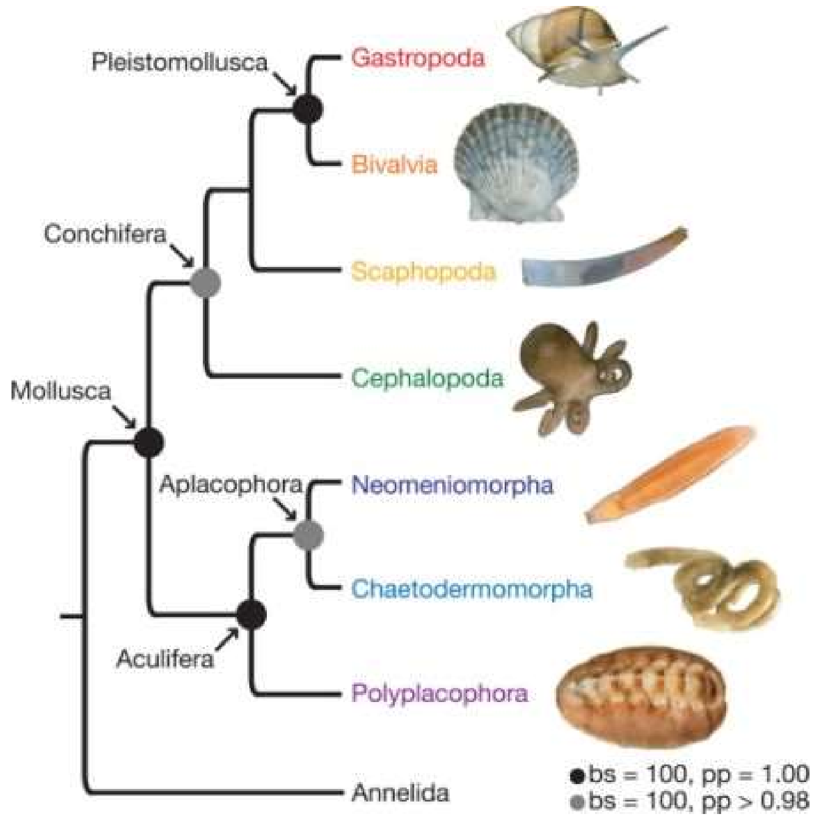




Strengths of Hierarchical Clustering

– We do not have to assume any particular number of clusters
  - any desired number of clusters can be obtained by ‘cutting’ the dendogram at the proper level

– May be used to discover meaningful taxonomies
  - taxonomies of biological species
  - taxonomies of different customer groups



Two Main Types of Hierarchical Clustering

- Agglomerative
  - start with the points as individual clusters
  - at each step, merge the closest pair of clusters until only one cluster (or k clusters) is left
- Divisive
  - start with one, all-inclusive cluster
  - at each step, split a cluster until each cluster contains a single point (or there are k clusters)
- Agglomerative Clustering is more widely used

Agglomerative Clustering Algorithm

The basic algorithm is straightforward:

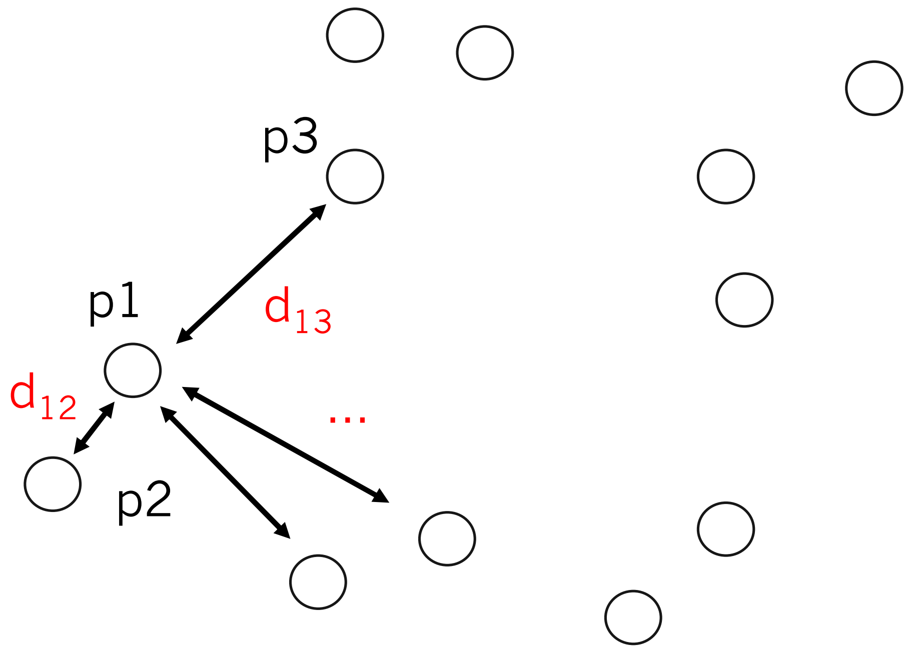
1. Compute the proximity matrix
2. Let each data point be a cluster
3. **Repeat**
    1. Merge the two closest clusters
    2. Update the proximity matrix

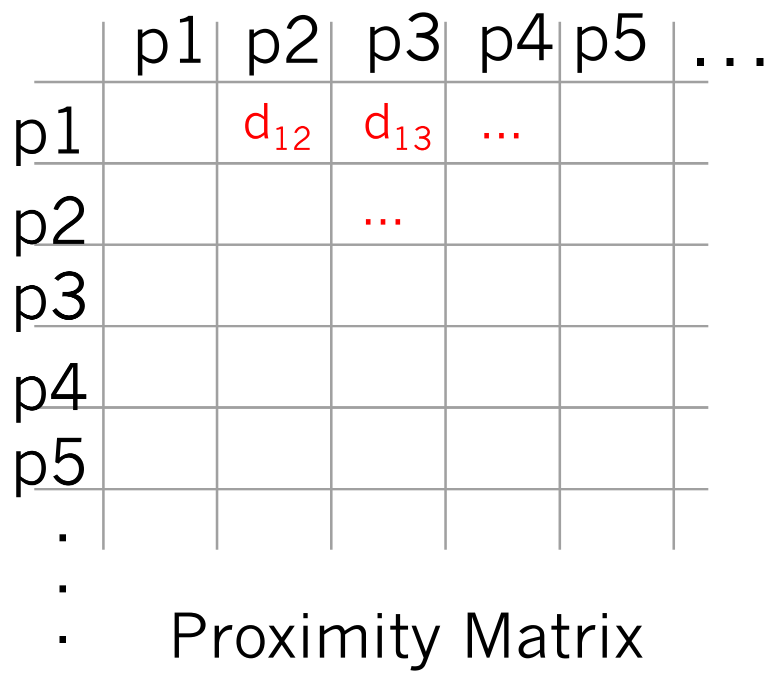
    Until only a single cluster remains

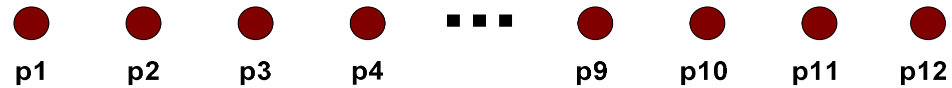
- The key operation is the computation of the proximity of two clusters
- The different approaches to defining the distance between clusters distinguish the different algorithms

Starting Situation

Start with clusters of individual points and a proximity matrix

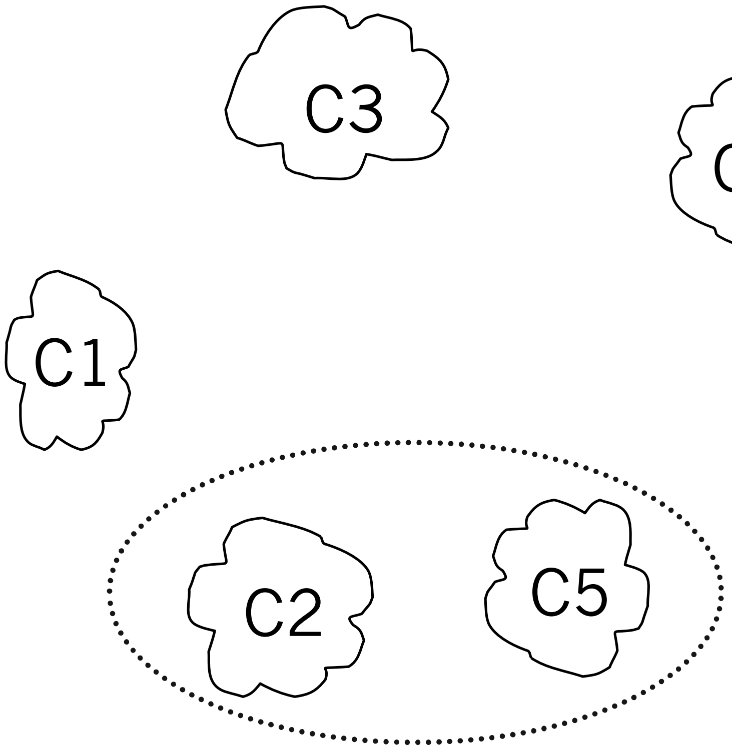






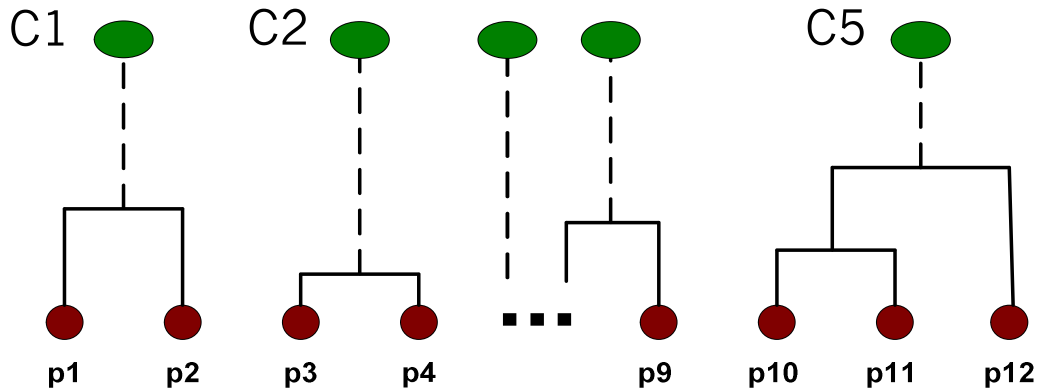
Intermediate Situation

- After some merging steps, we have larger clusters.
- We want to keep on merging the two closest clusters (C2 and C5?)

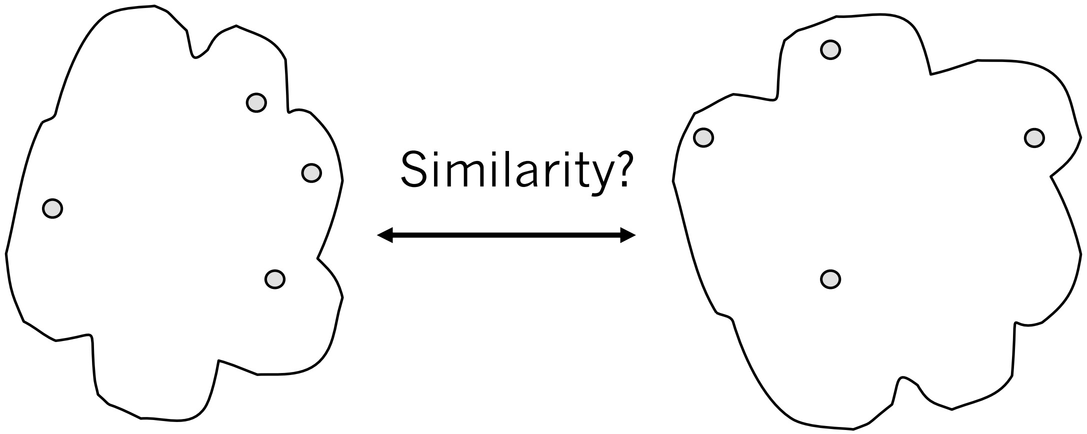


	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?	?	?	?
C3		?		
C4		?		

Proximity Matrix



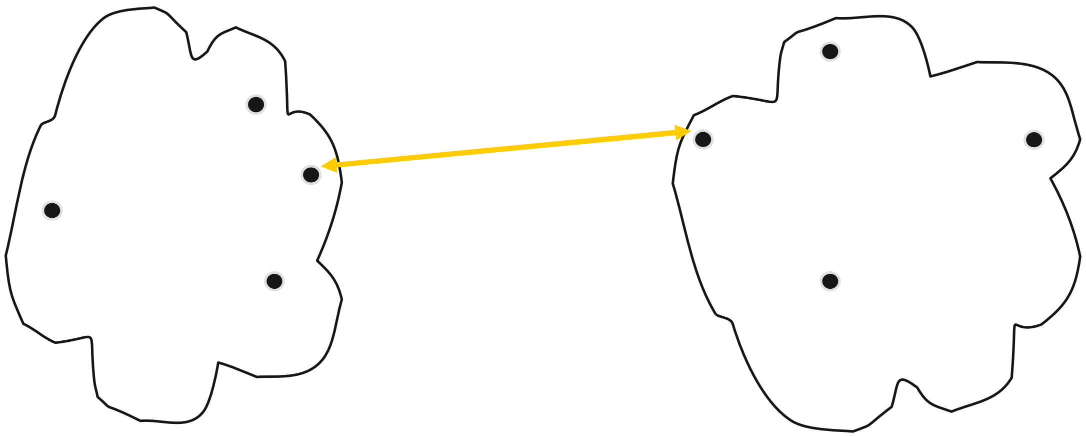
How to Define Inter-Cluster Similarity?



Different approaches are used:

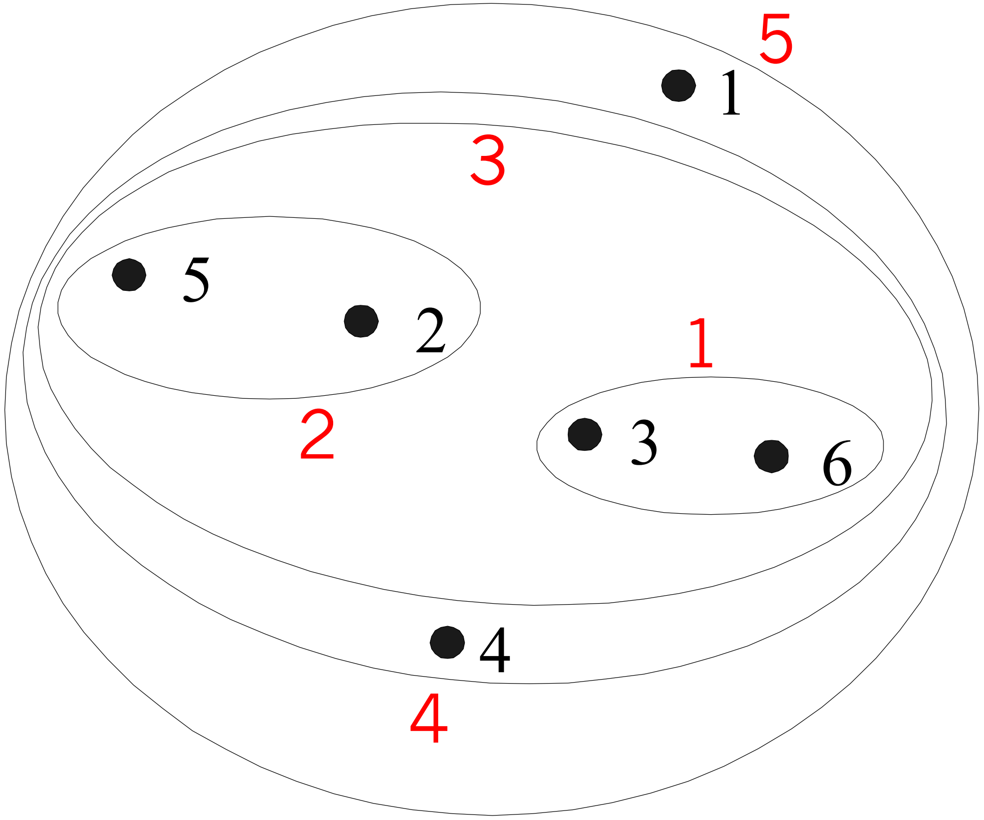
1. Single Link
2. Complete Link
3. Group Average
4. Distance Between Centroids

Cluster Similarity: Single Link

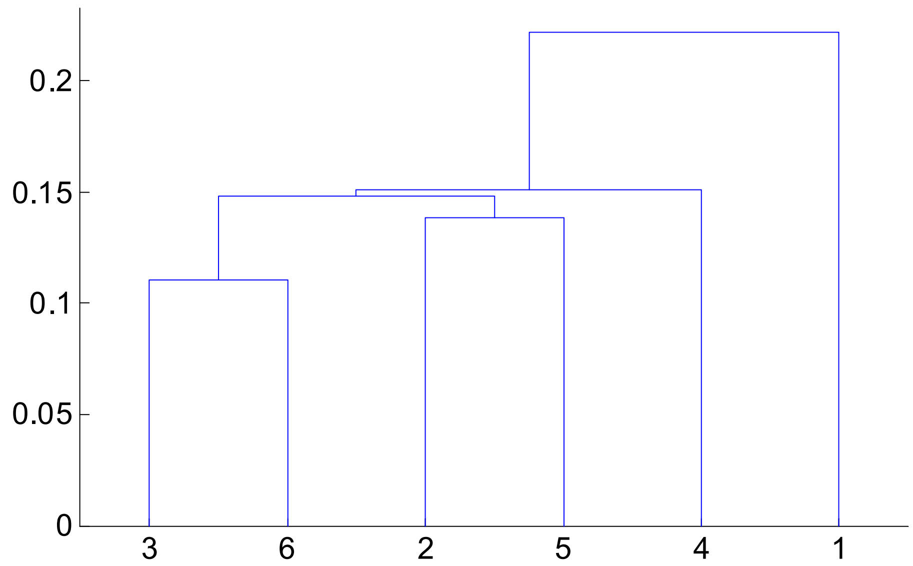


- Similarity of two clusters is based on the
  two most similar (closest) points in the different clusters
- Determined by one pair of points,
  i.e. by one link in the proximity graph

Example: Single Link

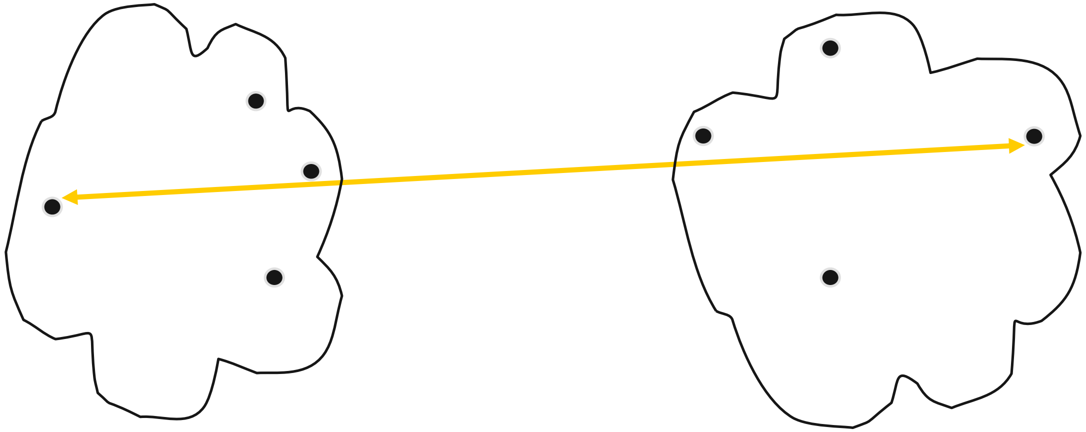


Nested Clusters



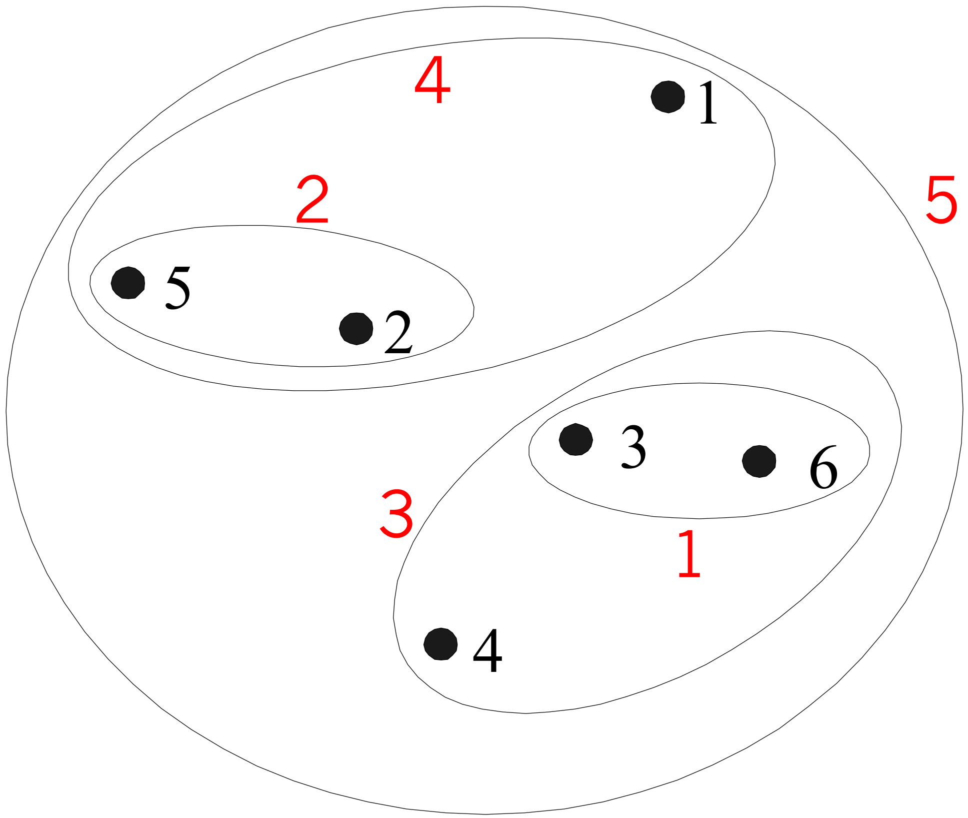
Dendrogram

Cluster Similarity: Complete Linkage

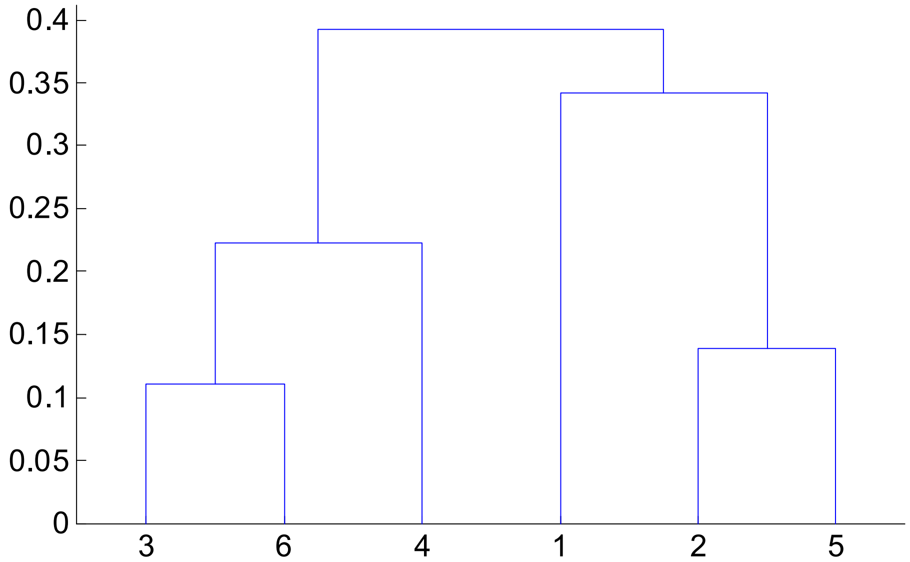


- Similarity of two clusters is based on the two least similar (most distant) points in the different clusters
- Determined by all pairs of points in the two clusters

Example: Complete Linkage



Nested Clusters

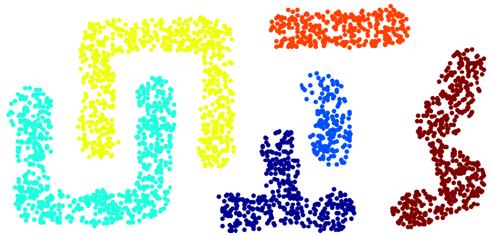


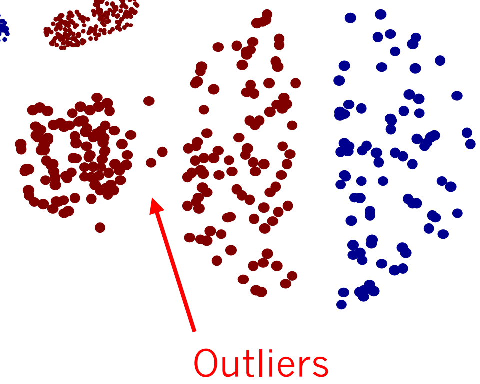
Dendrogram

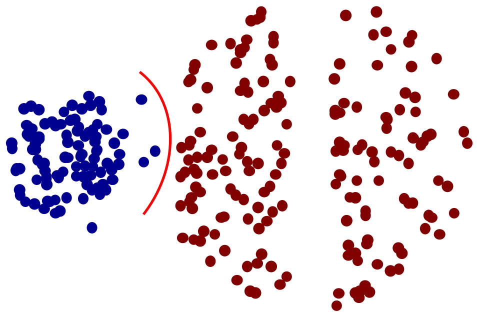
Single Link vs. Complete Linkage

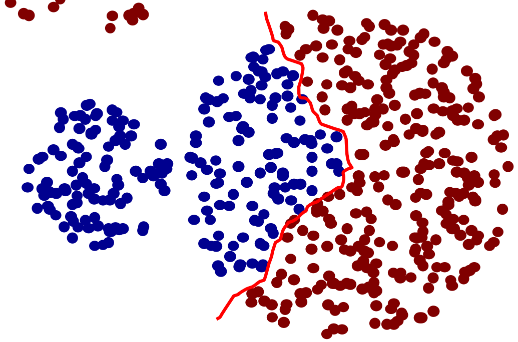
- Single Link
  - Strength: Can handle non-elliptic shapes
  - Limitation: Sensitive to noise and outliers

- Complete Linkage
  - Strength: Less sensitive to noise and outliers
  - Limitation: Biased towards globular clusters
  - Limitation: Tends to break large clusters,
    as decisions can not be undone.

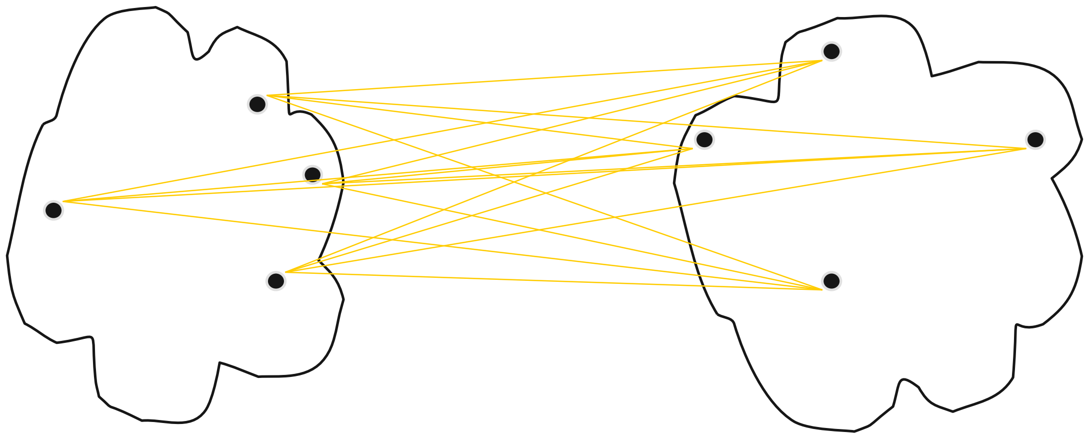








Cluster Similarity: Group Average



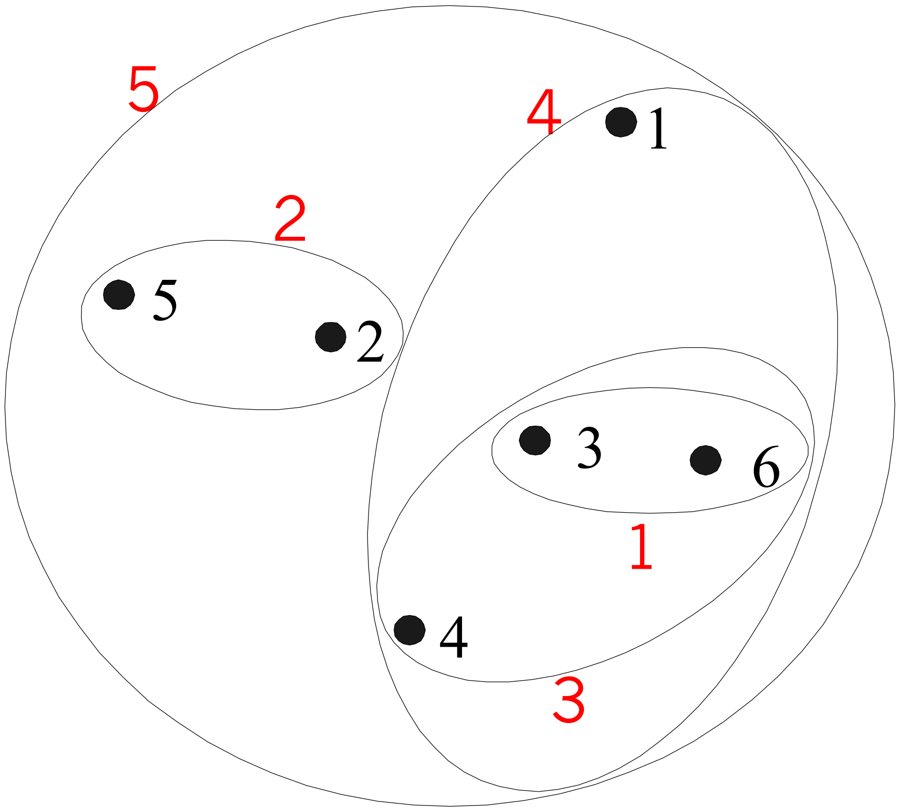
– Proximity of two clusters is the average of pair-wise proximity between all points in the two clusters.

$$\textbf{proximity}(\textbf{Cluster}_i, \textbf{Cluster}_j) = \frac{\sum\limits_{\substack{p_i \in \textbf{Cluster}_i \\ p_j \in \textbf{Cluster}_j}} \textbf{proximity}(p_i, p_j)}{|\textbf{Cluster}_i| * |\textbf{Cluster}_j|}$$

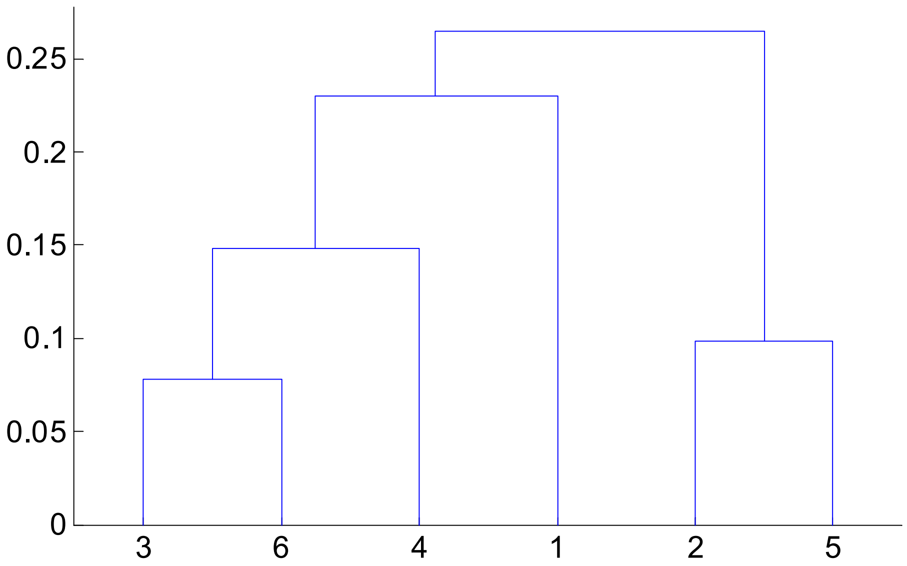
– Compromise between single and complete link

- Strength: Less sensitive to noise and outliers than single link
- Limitation: Biased towards globular clusters

Example: Group Average



Nested Clusters

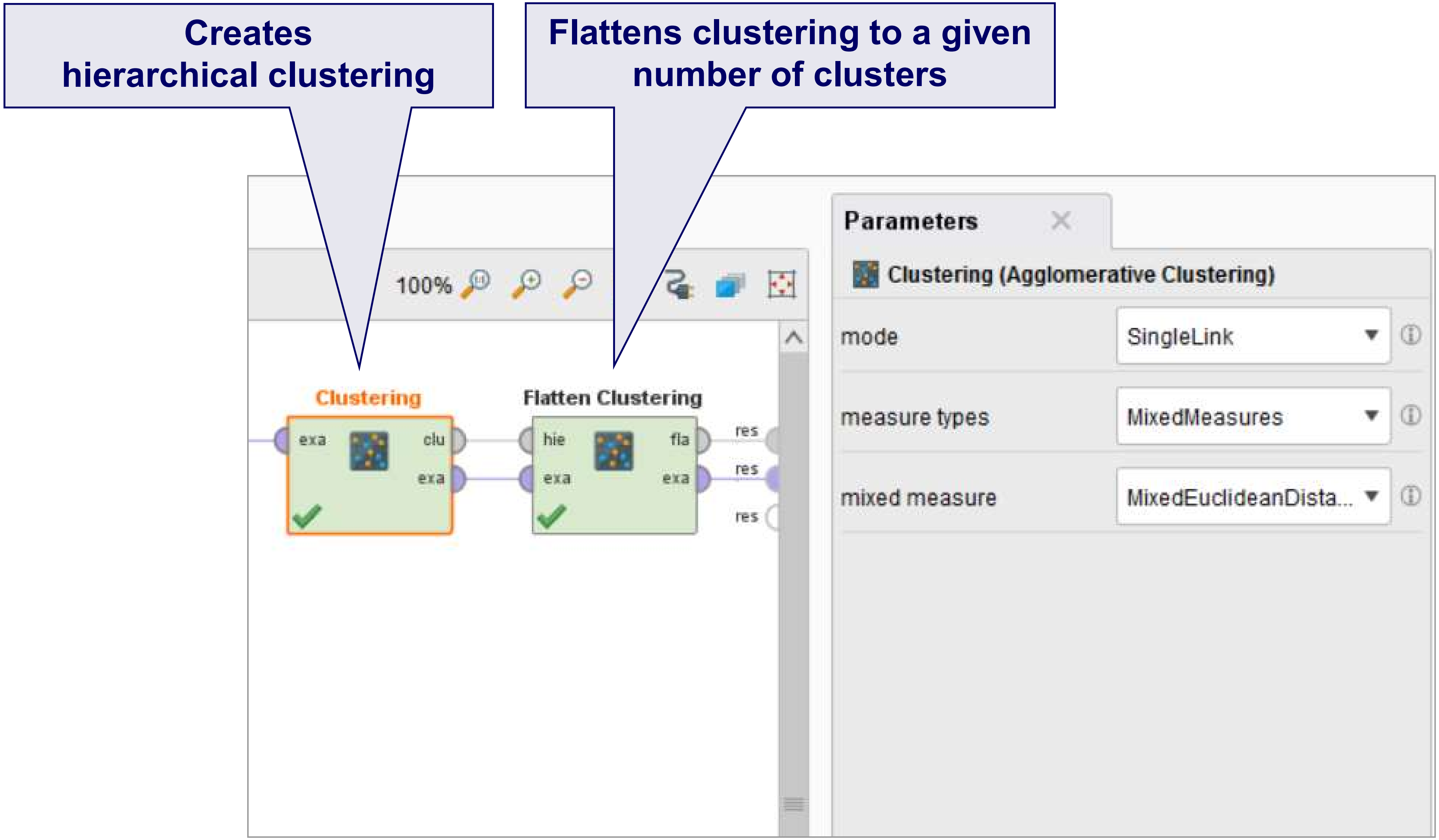


Dendrogram

Hierarchical Clustering: Problems and Limitations

- Different schemes have problems with one or more of the following:
  1. sensitivity to noise and outliers
  2. difficulty handling non-elliptic shapes
  3. breaking large clusters
- High space and time complexity
  - $O(N^2)$ space since it uses the proximity matrix
    - N is the number of points
  - $O(N^3)$ time in many cases
    - there are N steps and at each step the size N^2 proximity matrix must be searched and updated
    - complexity can be reduced to $O(N^2 \log(N))$ time in some cases
  - Workaround: Apply hierarchical clustering to a random sample of the original data (<10,000 examples)

Agglomerative Hierarchical Clustering in RapidMiner



Agglomerative Hierarchical Clustering in Python

Slide Type

```
# import linkage and dendrogram from scipy
from scipy.cluster.hierarchy import dendrogram, linkage

# create the clustering
Z = linkage(dataset[['Item1', 'Item2']], 'complete')

# plot the dendrogram
dendrogram(Z, labels=dataset['ID'].values)

# setup the labels
plt.xlabel('IDs')
plt.ylabel('distance')

# show the plot
plt.show()
```

Choose inter-cluster similarity metric, e.g. ‘single’, ‘complete’, ‘average’, ‘centroid’

5. Proximity Measures

- So far, we have seen different clustering algorithms all of which rely on proximity (distance, similarity, ...) measures
- Now, we discuss proximity measures in more detail
- A wide range of different measures is used depending on the requirements of the application
- Similarity
  - Numerical measure of how alike two data objects are
  - Often falls in the range [0,1]
- Dissimilarity / Distance
  - Numerical measure of how different are two data objects
  - Minimum dissimilarity is often 0, upper limit varies
- We distinguish proximity measures for single attributes and measures for multidimensional data points (records)

5.1 Proximity of Single Attributes

Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$	$s = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$
Ordinal	$d = \frac{\|p-q\|}{n-1}$ (values mapped to integers 0 to $n-1$, where n is the number of values)	$s = 1 - \frac{\|p-q\|}{n-1}$
Interval or Ratio	$d = \|p - q\|$	$s = -d$, $s = \frac{1}{1+d}$ or $s = 1 - \frac{d - min\_d}{max\_d - min\_d}$

Similarity and dissimilarity for simple attributes

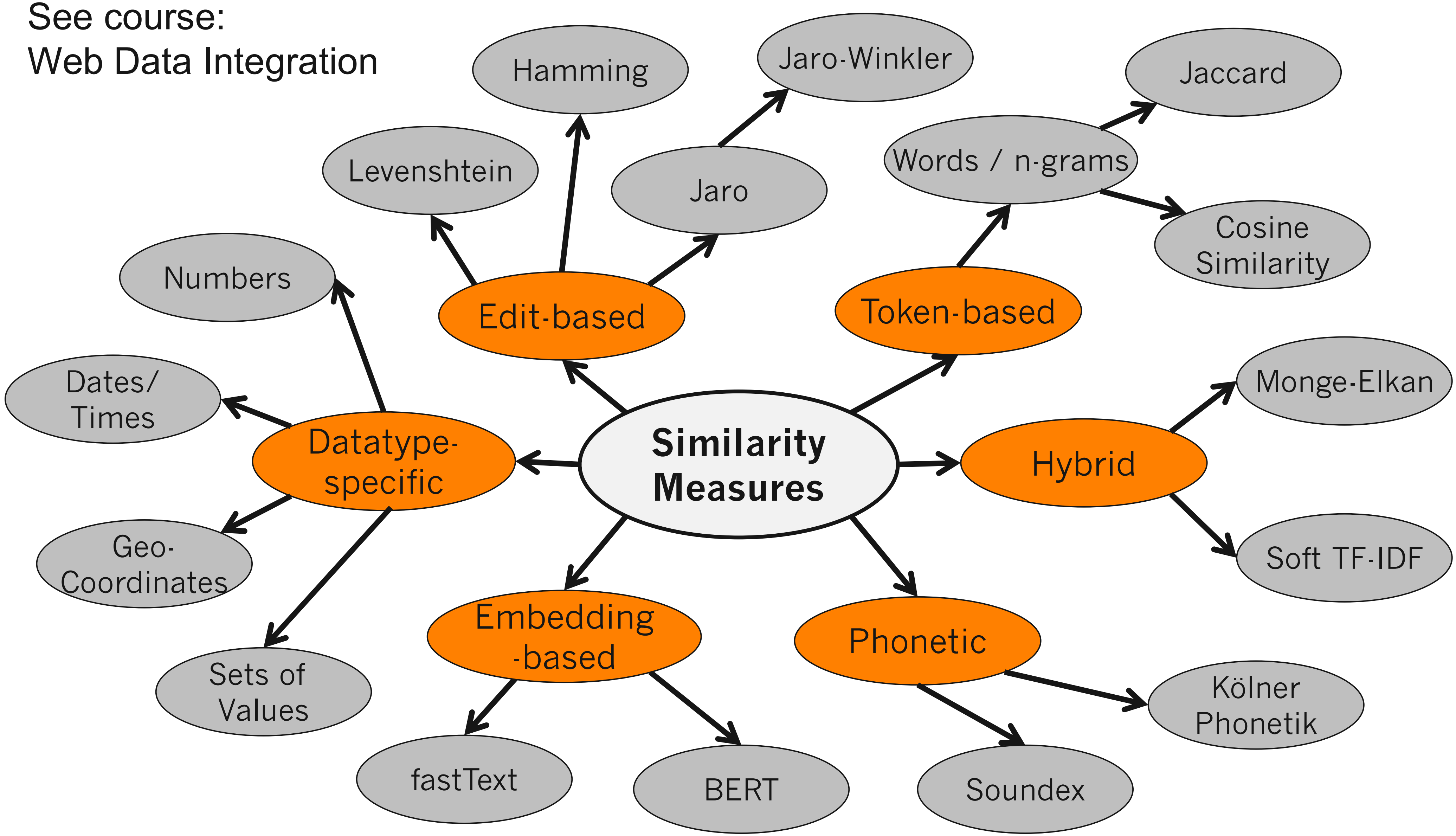
p and q are attribute values for two data objects

Levenshtein Distance

- Measures the dissimilarity of two strings
- Measures the minimum number of edits needed to transform one string into the other
- Allowed edit operations:
  1. insert a character into the string
  2. delete a character from the string
  3. replace one character with a different character
- Examples:
  - levensthein('table', 'cable') = 1 (1 substitution)
  - levensthein('Bizer, Chris', 'Chris Bizer') = 11 (10 substitution, 1 deletion)

Further String Similarity Measures

See course:
Web Data Integration



5.2 Proximity of Multidimensional Data Points

- All measures discussed so far cover the proximity of single attribute values
- But we usually have data points with many attributes
  - e.g., age, height, weight, sex...
- Thus, we need proximity measures for data points
  - taking multiple attributes/dimensions into account

Euclidean Distance

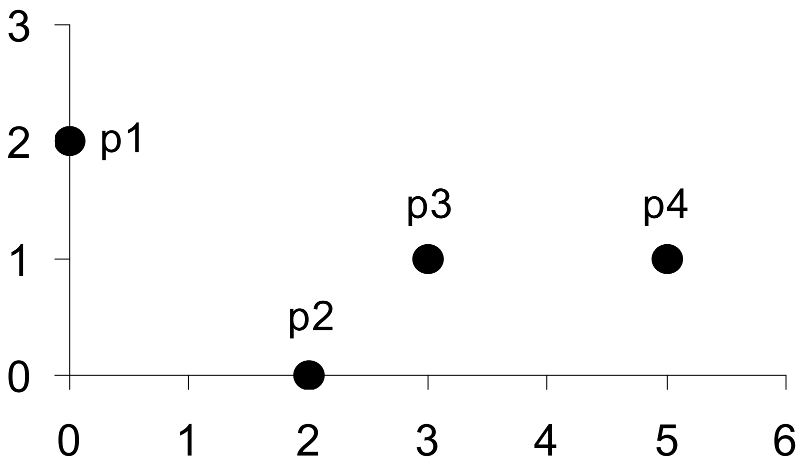
Definition

$$dist = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}$$

Where *n* is the number of dimensions (attributes) and p_k and q_k are the k^th attributes of data points *p* and *q*

- p_k - q_k is squared to increase impact of long distances
- All dimensions are weighted equality

Example: Euclidean Distance



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Distance Matrix

Normalization

– Attributes should be normalized so that all attributes can have equal impact on the computation of distances

– Consider the following pair of data points

  - x_i: (0.1, 20) and x_j: (0.9, 720).

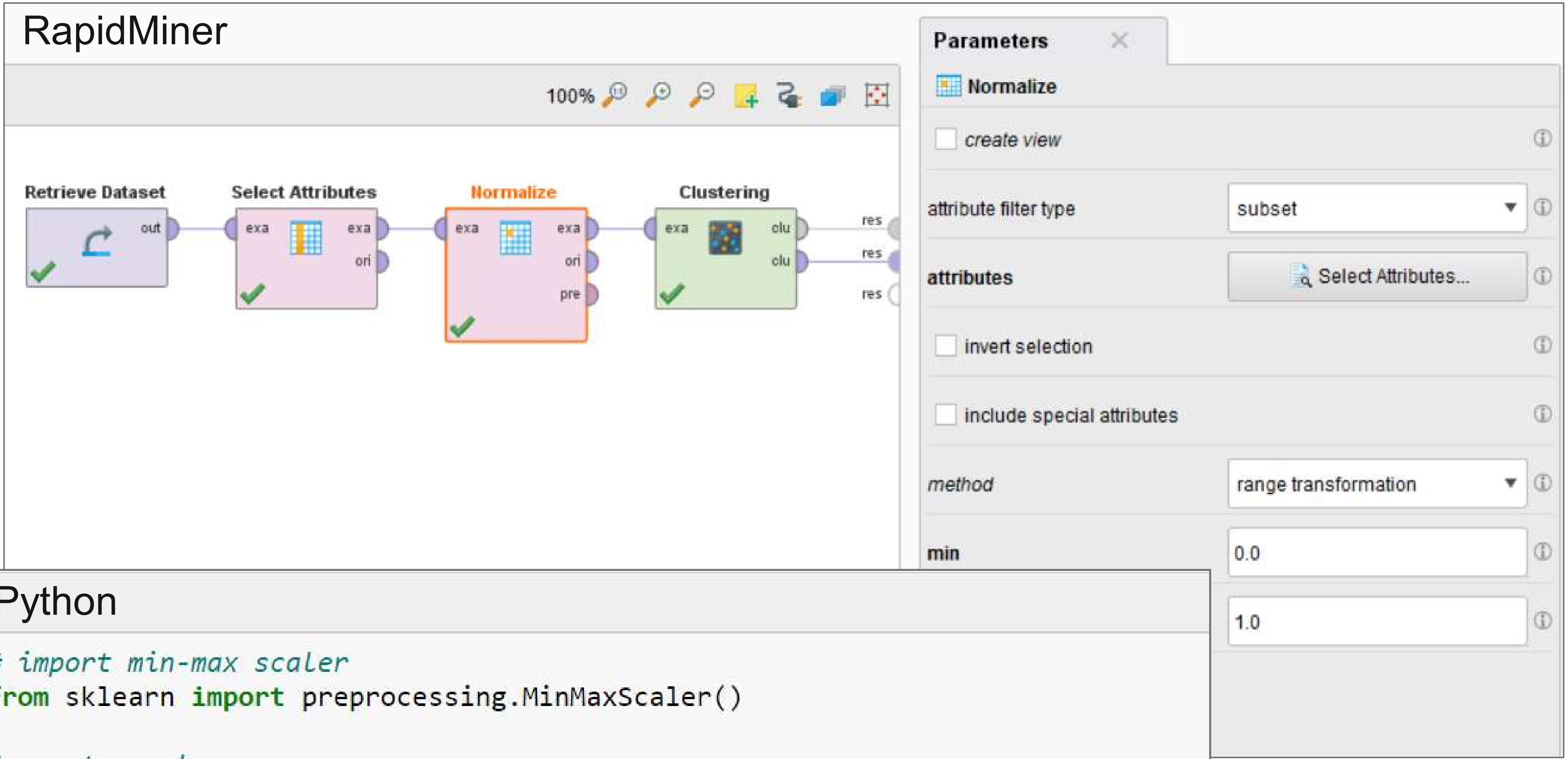
$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(0.9 - 0.1)^2 + (720 - 20)^2} = 700.000457$$

– The distance is almost completely dominated by (720-20) = 700

– Solution: Scale attributes to have a common value range, for instance [0,1]

Scaling in RapidMiner and Python

RapidMiner



Python

```python
# import min-max scaler
from sklearn import preprocessing.MinMaxScaler()

# create scaler
scaler = MinMaxScaler()

# normalize the relevant attributes
dataset[['Att1', 'Att2']] = scaler.fit_transform(dataset[['Att1', 'Att2']])
```

Similarity of Binary Attributes

– Common situation is that objects, *p* and *q*, have only binary attributes
  - products in shopping basket
  - courses attended by students

– We compute similarities using the following quantities:

M_{11} = the number of attributes where p was 1 and q was 1
M_{00} = the number of attributes where p was 0 and q was 0
M_{01} = the number of attributes where p was 0 and q was 1
M_{10} = the number of attributes where p was 1 and q was 0

Symmetric Binary Attributes

– A binary attribute is symmetric if both of its states (0 and 1) have equal importance, and carry the same weights, e.g., male and female

– Similarity measure: Simple Matching Coefficient

$$SMC(\mathbf{x}_i, \mathbf{x}_j) = \frac{M_{11} + M_{00}}{M_{01} + M_{10} + M_{11} + M_{00}}$$

Number of matches / number of all attributes values

Asymmetric Binary Attributes

- Asymmetric: If one of the states is more important than the other
  - by convention, state 1 represents the more important state
  - 1 is typically the rare or infrequent state
  - examples: Shopping baskets, word vectors
- Similarity measure: Jaccard Coefficient

$$J(\mathbf{x}_i, \mathbf{x}_j) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

Number of 11 matches / number of not-both-zero attributes values

SMC versus Jaccard: Example

$p =$ 1 0 0 0 0 0 0 0 0 0
$q =$ 0 0 0 0 0 0 1 0 0 1

Interpretation of the example:
Customer p bought item 1
Customer q bought item 7 and 10

$M_{11} = 0$ (the number of attributes where p was 1 and q was 1)
$M_{00} = 7$ (the number of attributes where p was 0 and q was 0)
$M_{01} = 2$ (the number of attributes where p was 0 and q was 1)
$M_{10} = 1$ (the number of attributes where p was 1 and q was 0)

$$SMC = (M_{11} + M_{00})/(M_{01} + M_{10} + M_{11} + M_{00}) = (0+7) / (2+1+0+7) = 0.7$$

$$J = (M_{11}) / (M_{01} + M_{10} + M_{11}) = 0 / (2 + 1 + 0) = 0$$

SMC versus Jaccard: Question

- Which of the two measures would you use …

- ...for a dating agency?
  - hobbies
  - favorite bands
  - favorite movies
  - …

- ...for the Wahl-O-Mat?
  - (dis-)agreement with political statements
  - recommendation for voting

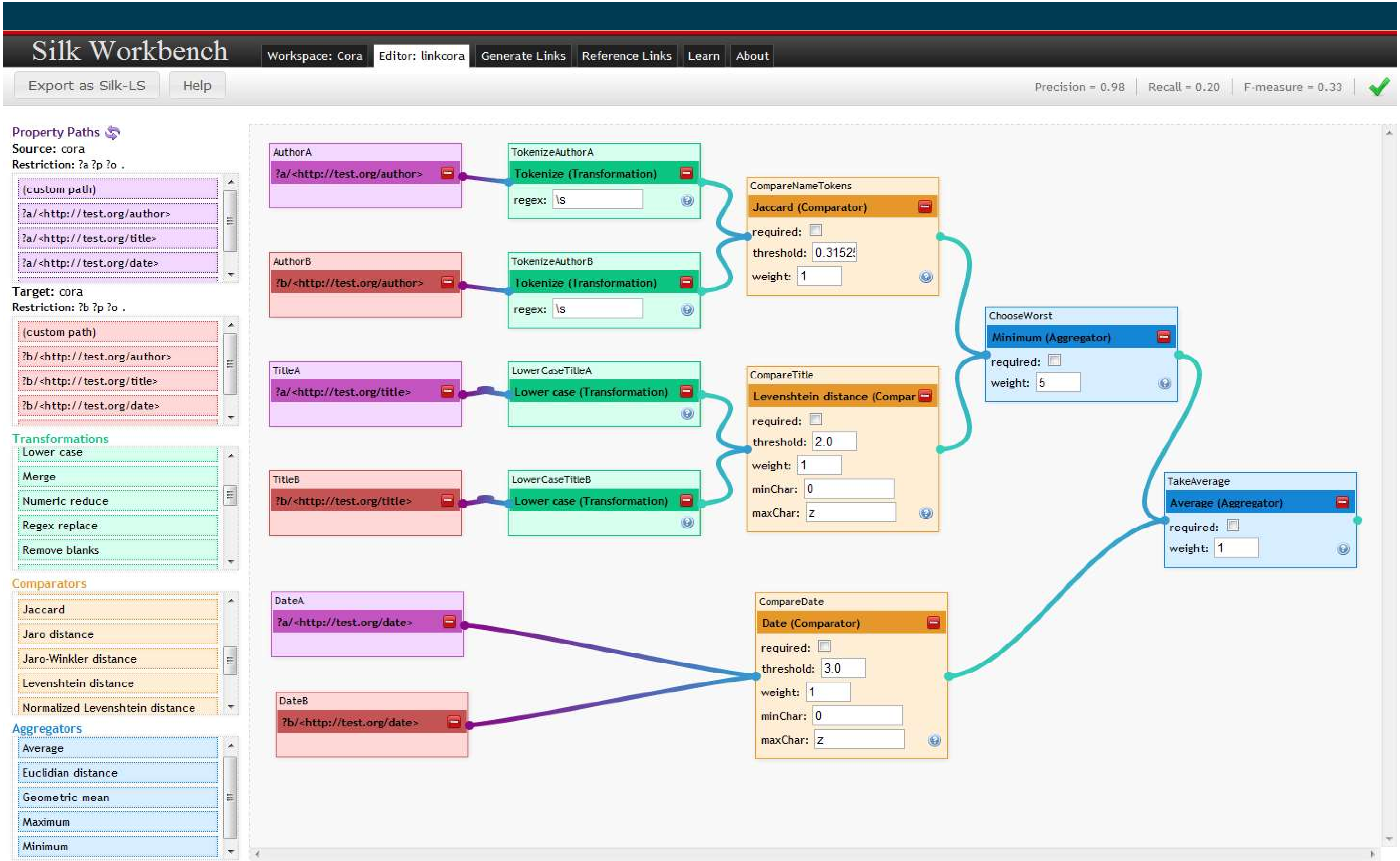
Using Weights to Combine Similarities

– You may not want to treat all attributes the same
  - use weights w_k which are between 0 and 1 and sum up to 1
  - weights are set according to the importance of the attributes

– Example: Weighted Euclidean Distance

$$dist(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{w_1(x_{i1} - x_{j1})^2 + w_2(x_{i2} - x_{j2})^2 + \ldots + w_r(x_{ir} - x_{jr})^2}$$

Combining Different Similarity Measures

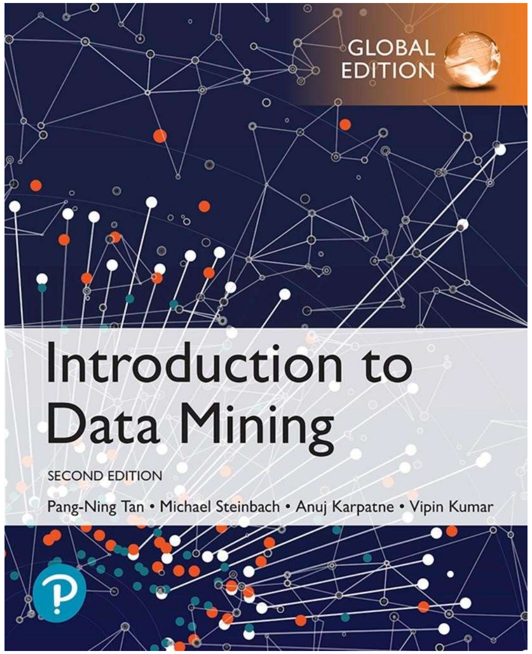


How to Choose a good Clustering Algorithm?

– “Best” algorithm depends on
  1. the analytical goals of the specific use case
  2. the distribution of the data
– Normalization, feature selection, distance measure, and parameter settings have equally high influence on results
– Due to these complexities, the common practice is to
  1. run several algorithms using different distance measures, feature subsets and parameter settings, and
  2. then visualize and interpret the results based on knowledge about the application domain as well as the goals of the analysis

Literature for this Slideset

Pang-Ning Tan, Michael Steinbach, Anuj Karpatne,
Vipin Kumar: **Introduction to Data Mining.**
2nd Edition. Pearson.



Chapter 5: Cluster Analysis

Chapter 5.2: K-Means

Chapter 5.3: Agglomerative Hierarchical Clustering

Chapter 5.4: DBSCAN

Chapter 2.4: Measures of Similarity and Dissimilarity

The respective chapters of the 1st Edition are available as PDF in ILIAS