UNIVERSITÄT MANNHEIM

Data Mining I
Cluster Analysis

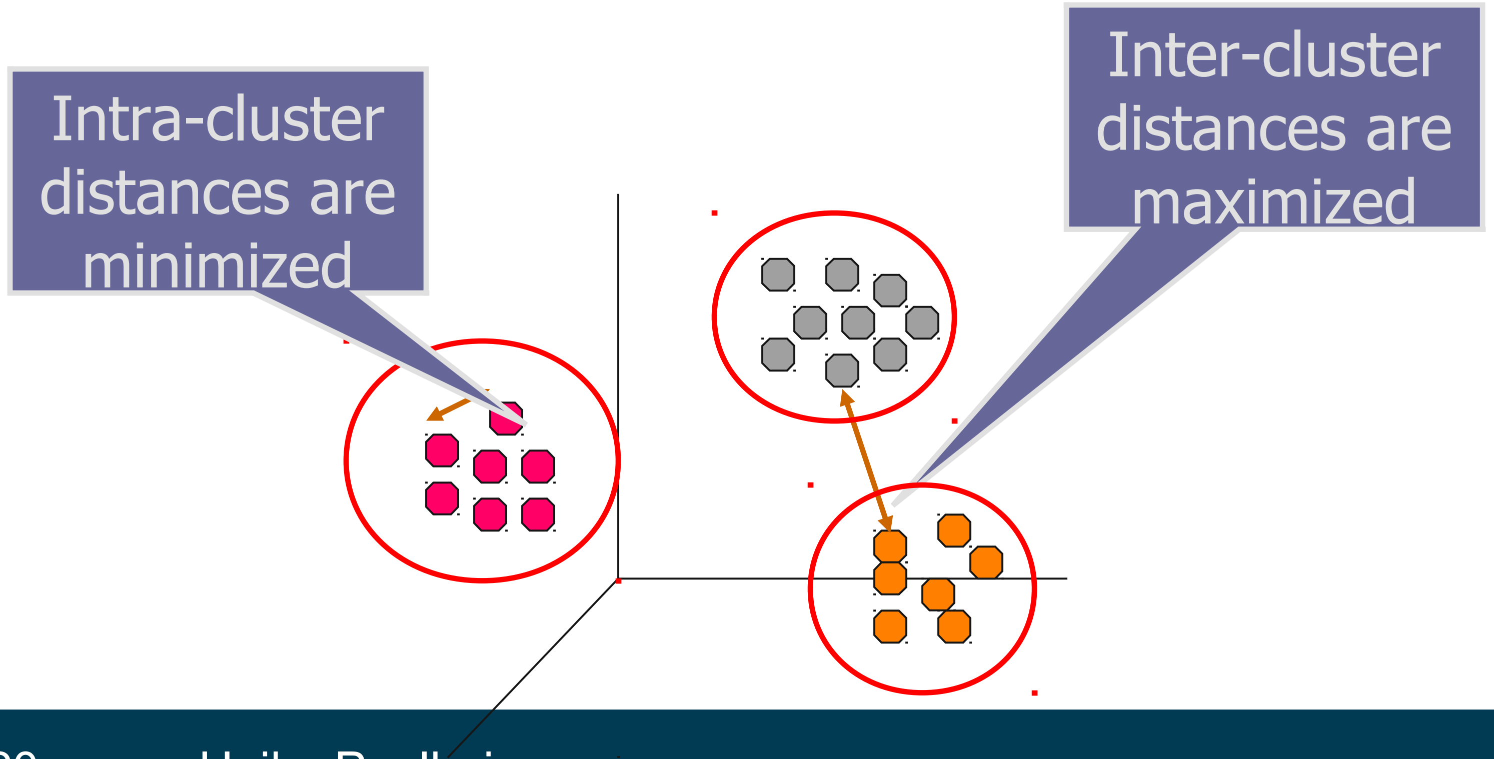
Heiko Paulheim

Outline

1. What is Cluster Analysis?
2. Applications for Clustering
3. k-Means Clustering
4. Hierarchical Clustering
5. Density-based Clustering
6. Proximity Measures

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.



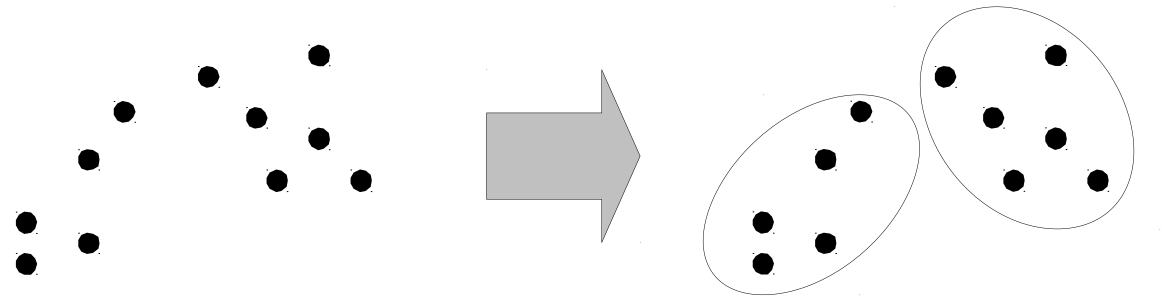
Cluster Analysis as Unsupervised Learning

- Supervised learning: Discover patterns in the data that relate data attributes with a target (class) attribute
  - The set of classes is known before
  - Class attributes are usually provided by human annotators
  - Patterns are used for prediction of the target attribute for new data

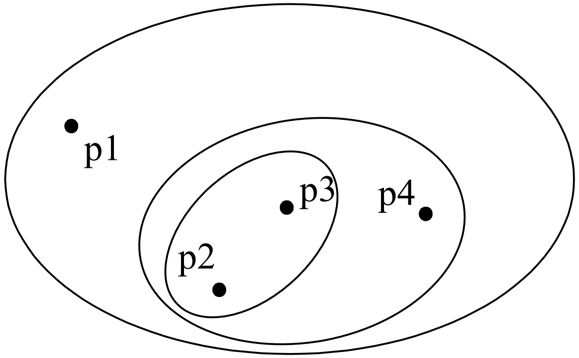
- Unsupervised learning: The data has no target attribute
  - We want to explore the data to find some intrinsic structures in it
  - The set of classes/clusters is not known before
  - Cluster Analysis and Association Rule Mining are unsupervised learning tasks

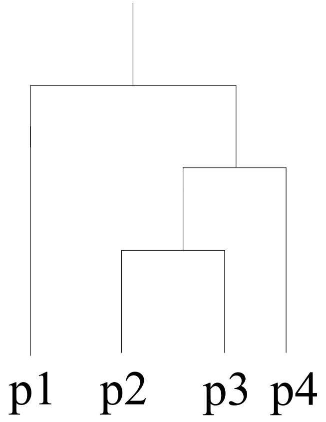
Types of Clusterings

- Partitional Clustering
  - A division 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



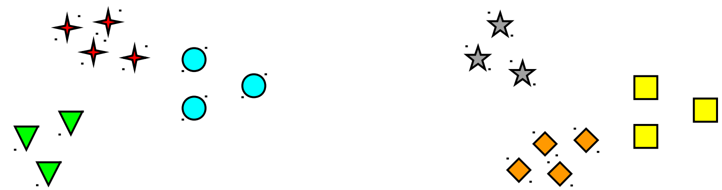


Aspects of Cluster Analysis

- Clustering algorithm
  - Partitional Algorithms
  - Hierarchical Algorithms
  - Density-based Algorithms
  - …
- Proximity (similarity, or dissimilarity) measure
  - Euclidean Distance
  - Cosine Similarity
  - Domain-specific Similarity Measures
  - …
- Clustering Quality
  - Intra-clusters distance $\Rightarrow$ minimized
  - Inter-clusters distance $\Rightarrow$ maximized

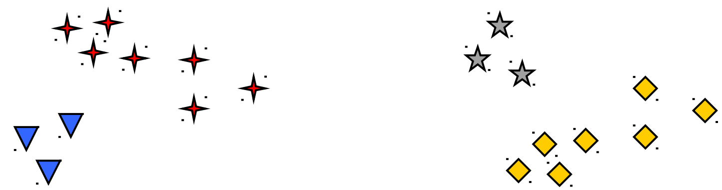
Notion of a Cluster can be Ambiguous

How many clusters?



Six Clusters

Two Clusters



Four Clusters

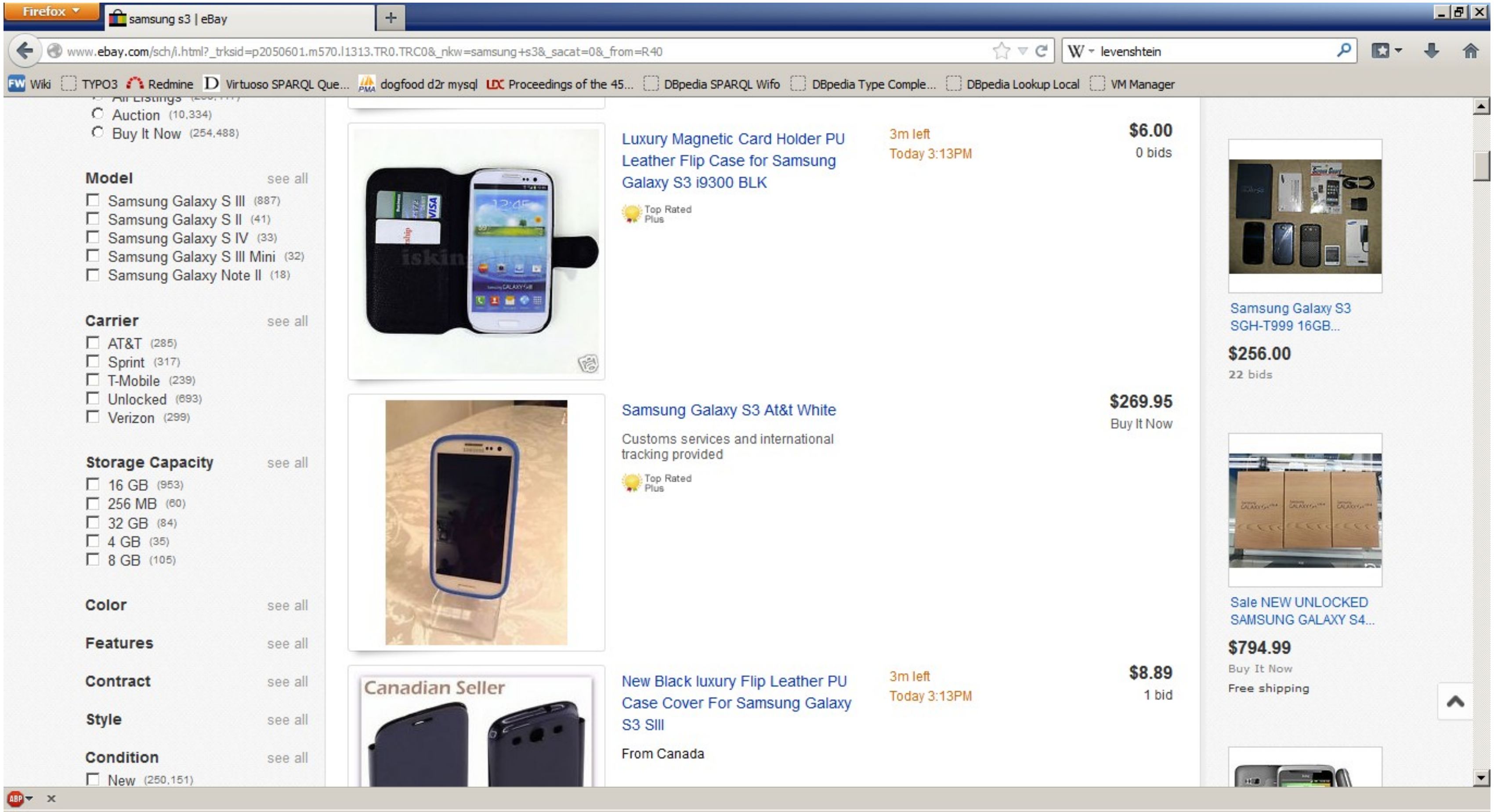
The usefulness of a clustering depends
on the goals of the analysis!

Applications: Market Research

- Identify different groups of customers

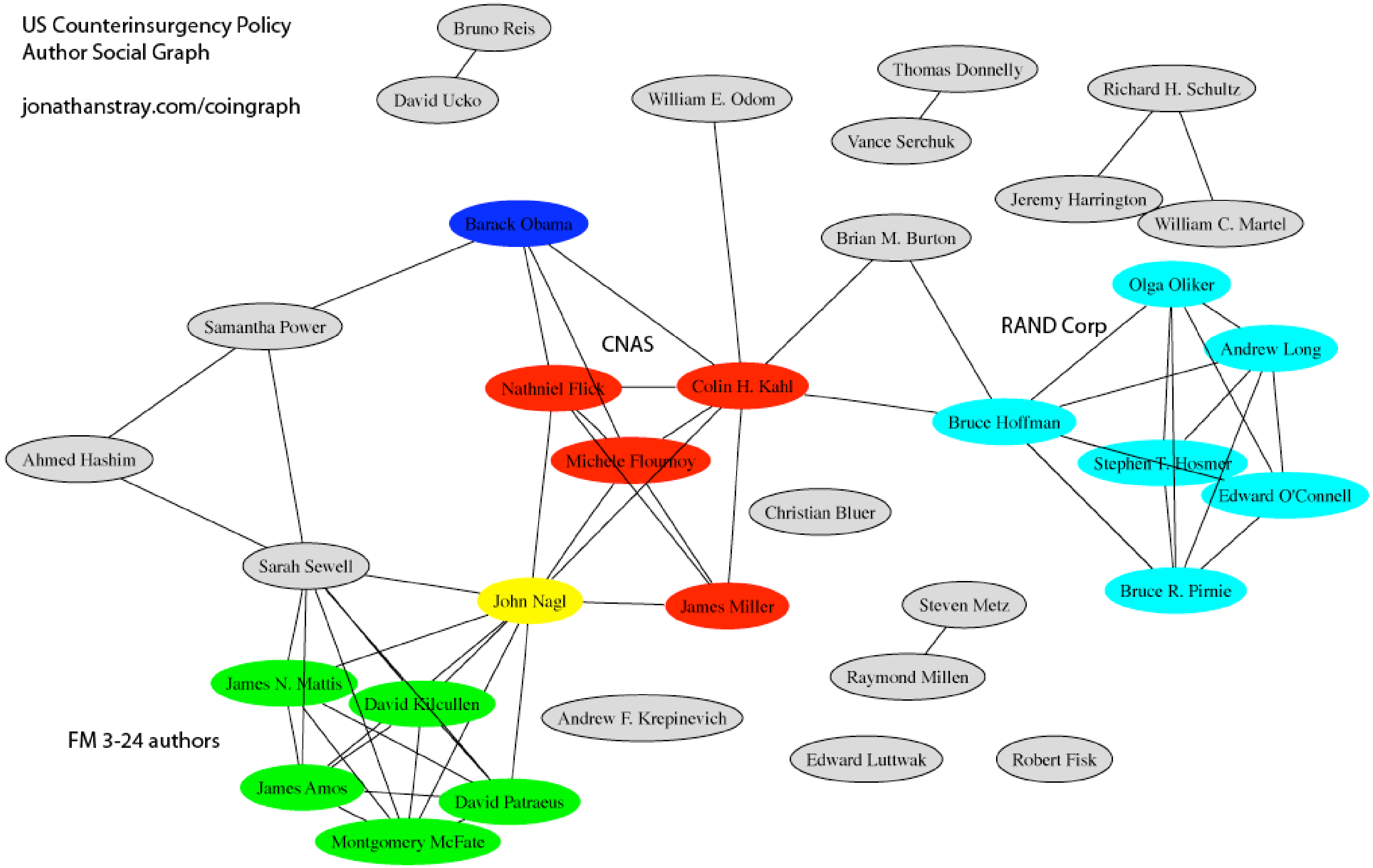
Application: Product Grouping

- Identify offers of same (or similar) products, e.g., on ebay



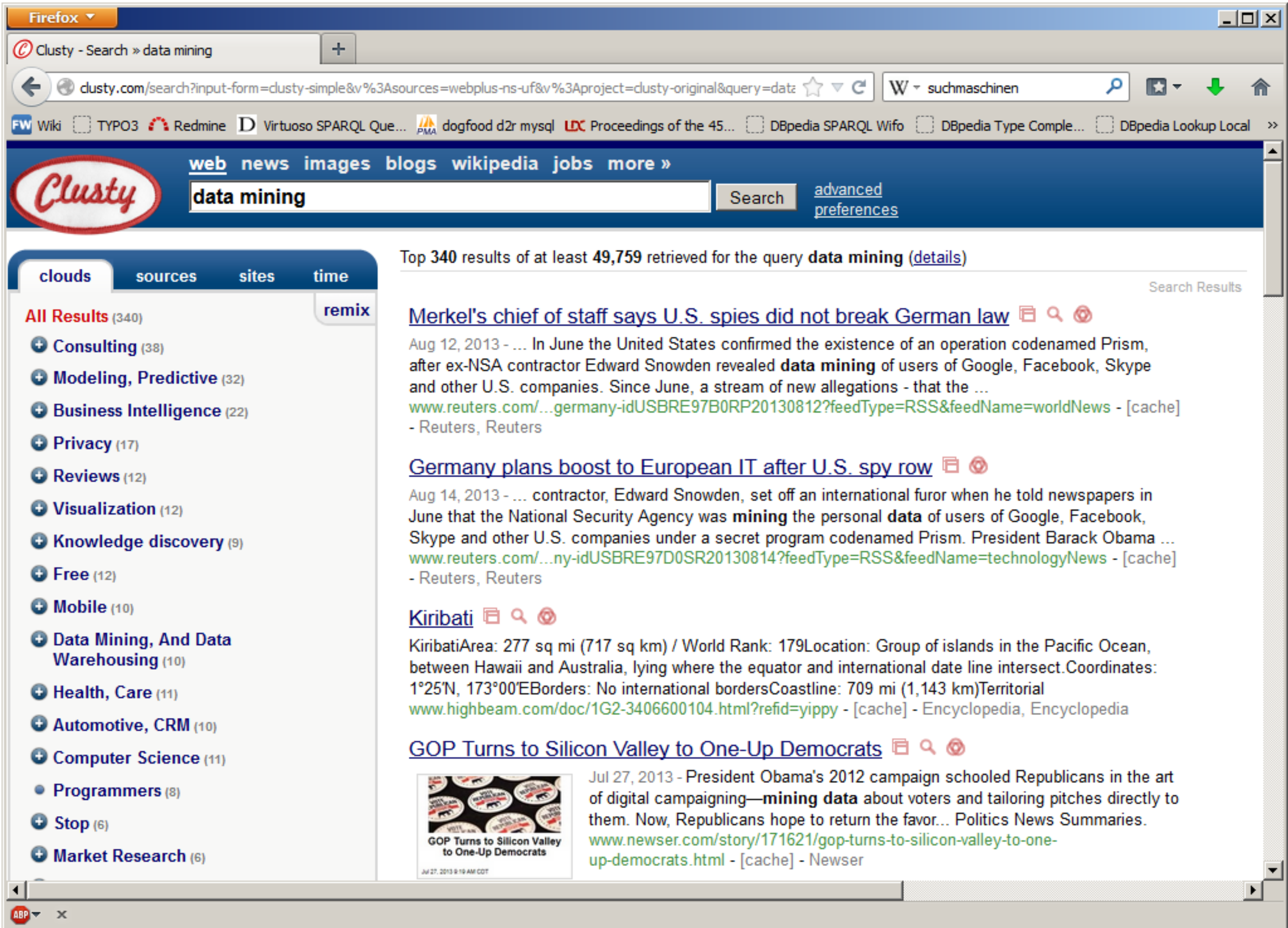
Applications: Social Network Analysis

- Identifying communities of people, e.g., with similar interests



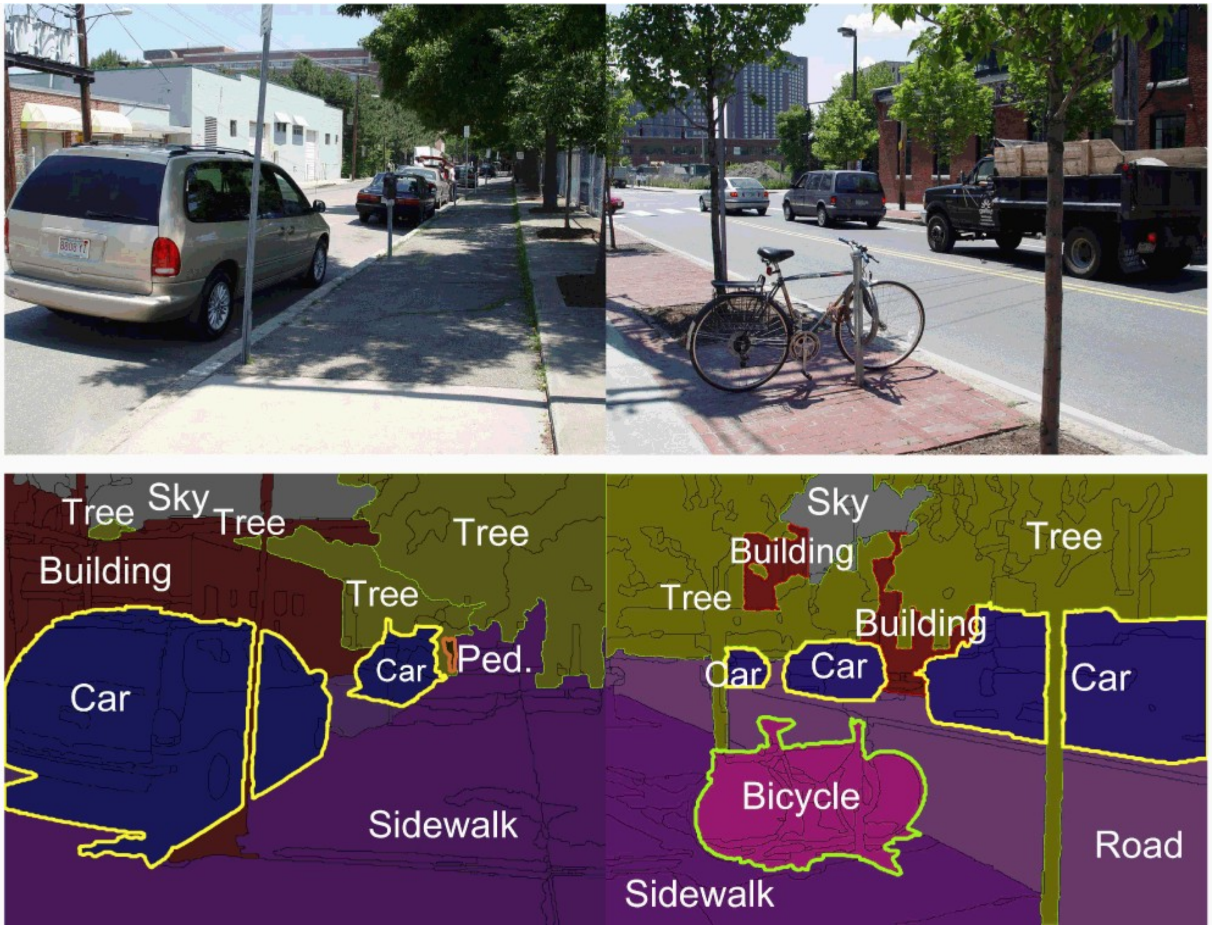
Applications: Grouping Search Engine Results

- Automatically find groups of related pages in the result set



Applications: Image Recognition

- Identify portions of an image that belong to the same object



K-Means Clustering

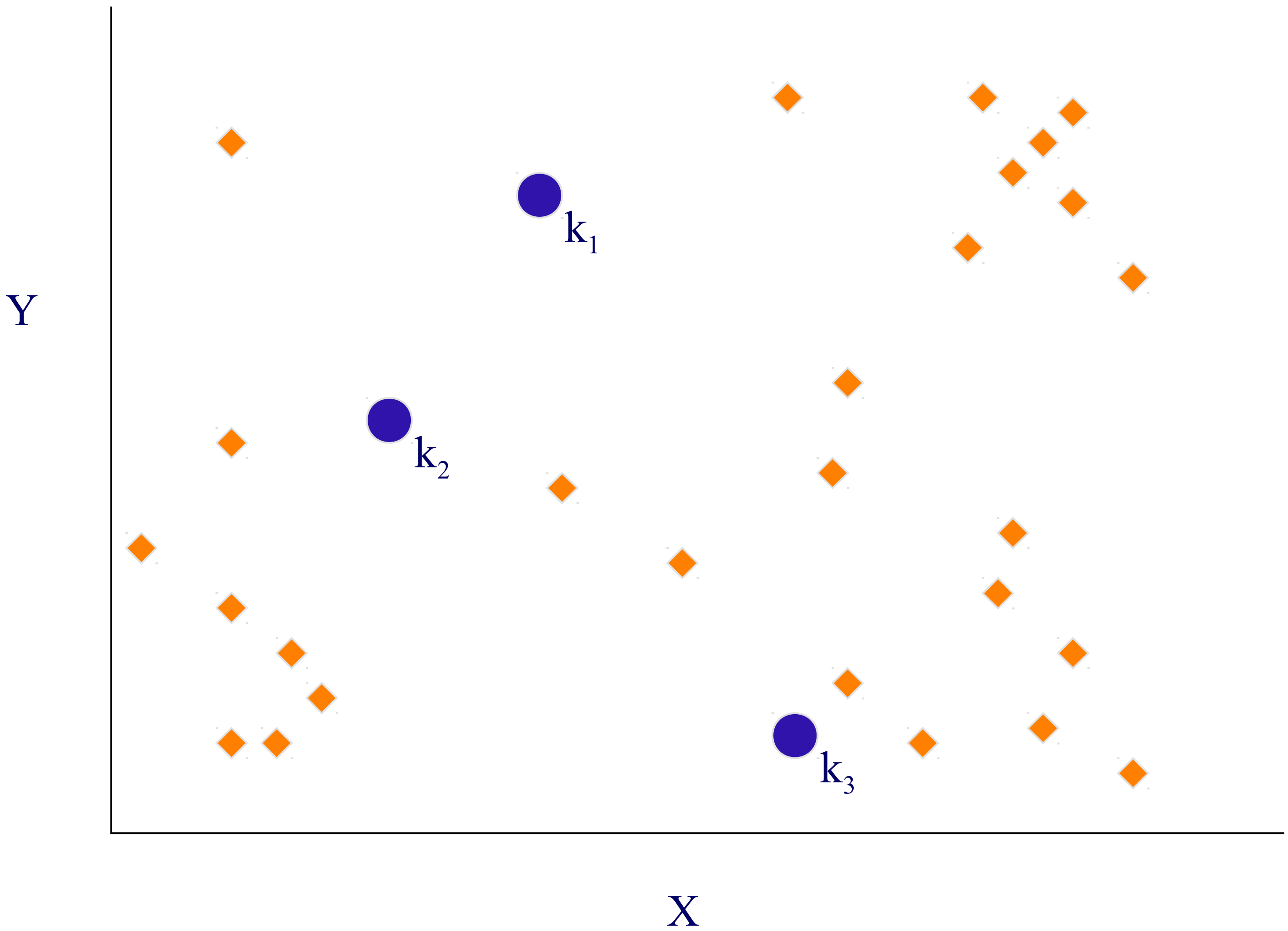
- Partitional clustering approach
- Each cluster is associated with a centroid (center point)
- Each point is assigned to the cluster with the closest centroid
- Number of clusters, K, must be specified manually

K-Means Clustering

- Basic Algorithm:

1: Select K points as the initial centroids.
2: **repeat**
3: Form K clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: **until** The centroids don't change

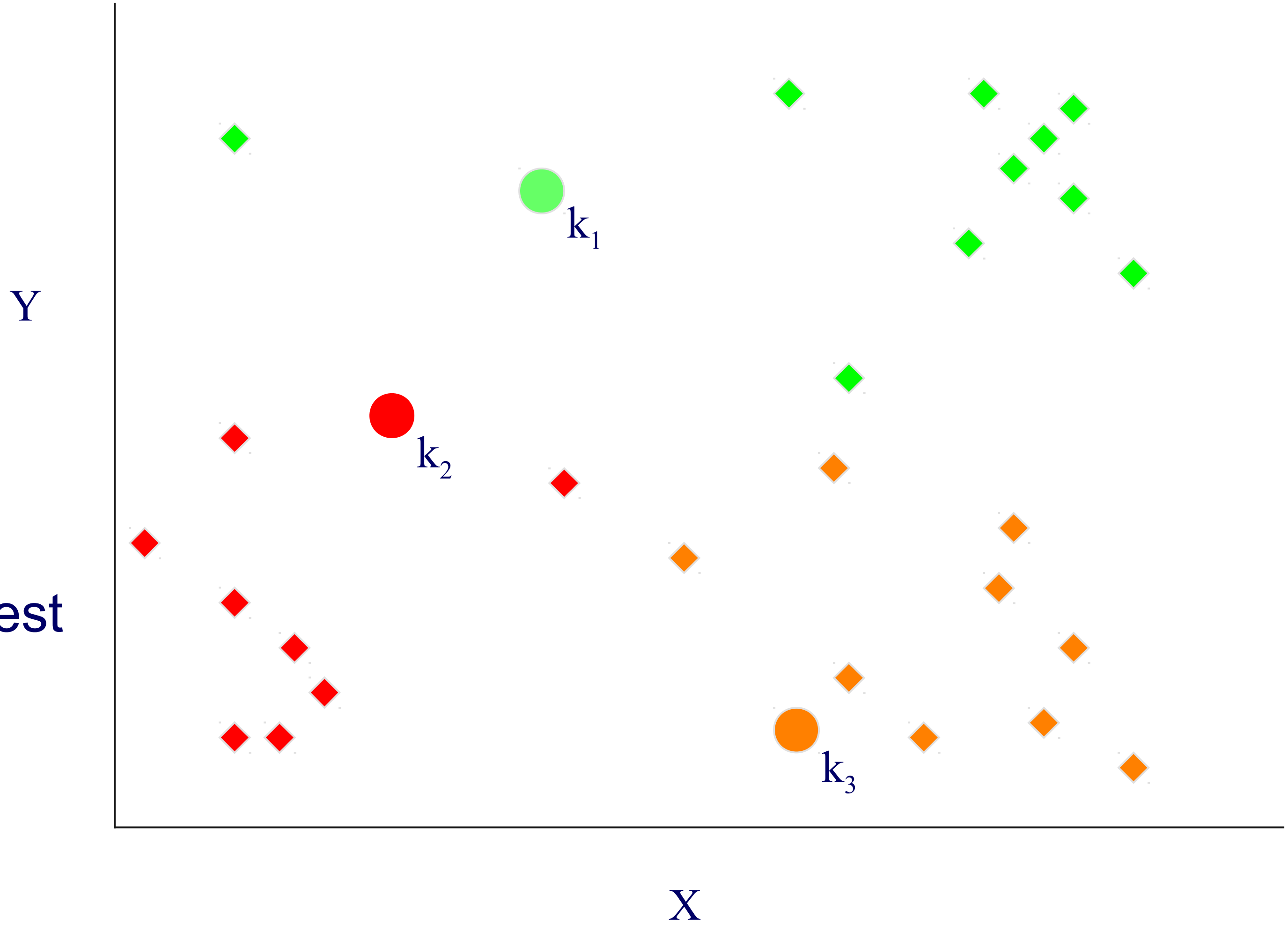
K-Means Example, Step 1



Pick 3 initial cluster centers (randomly)

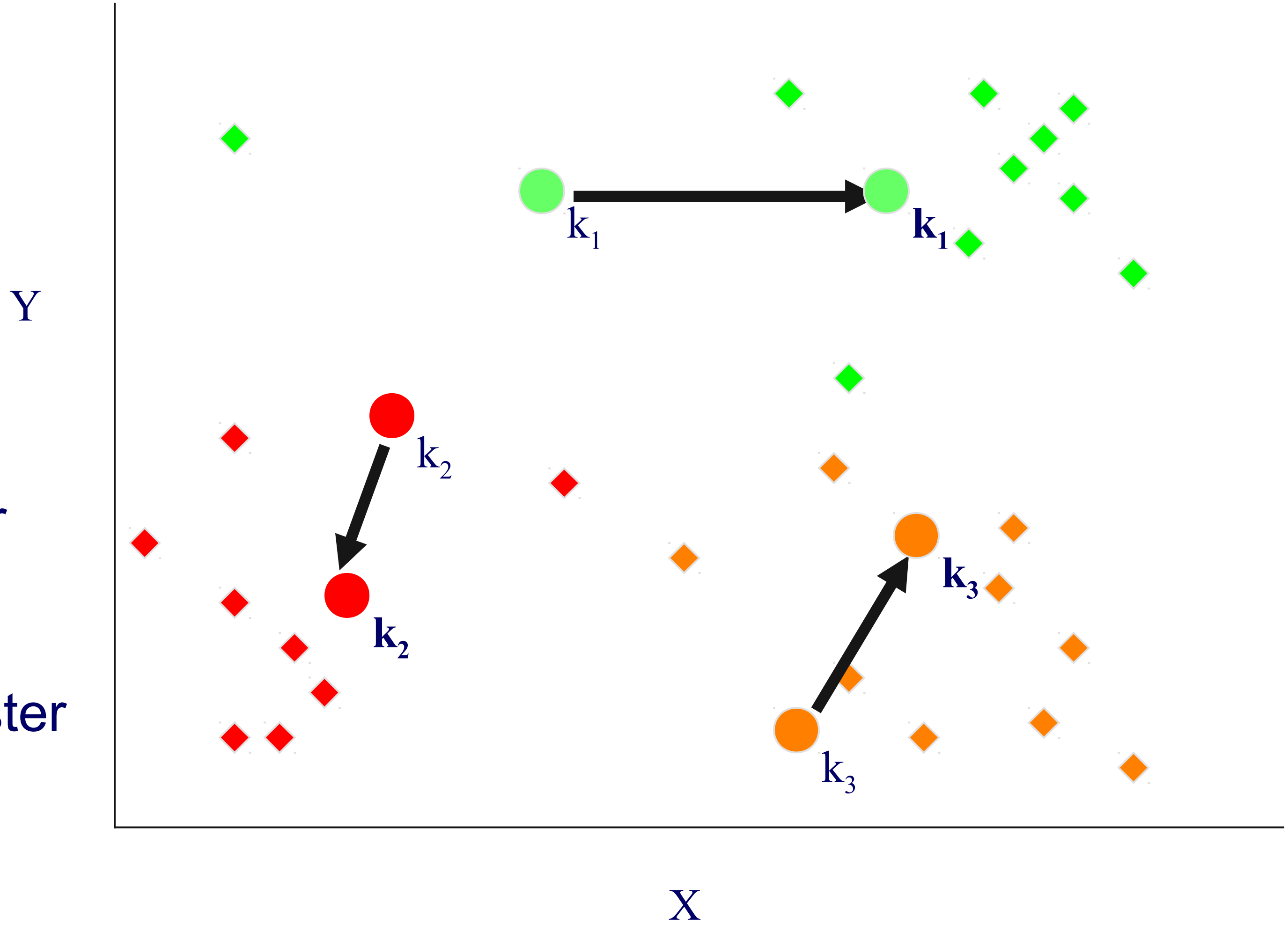
K-Means Example, Step 2

Assign each point to the closest cluster center



K-Means Example, Step 3

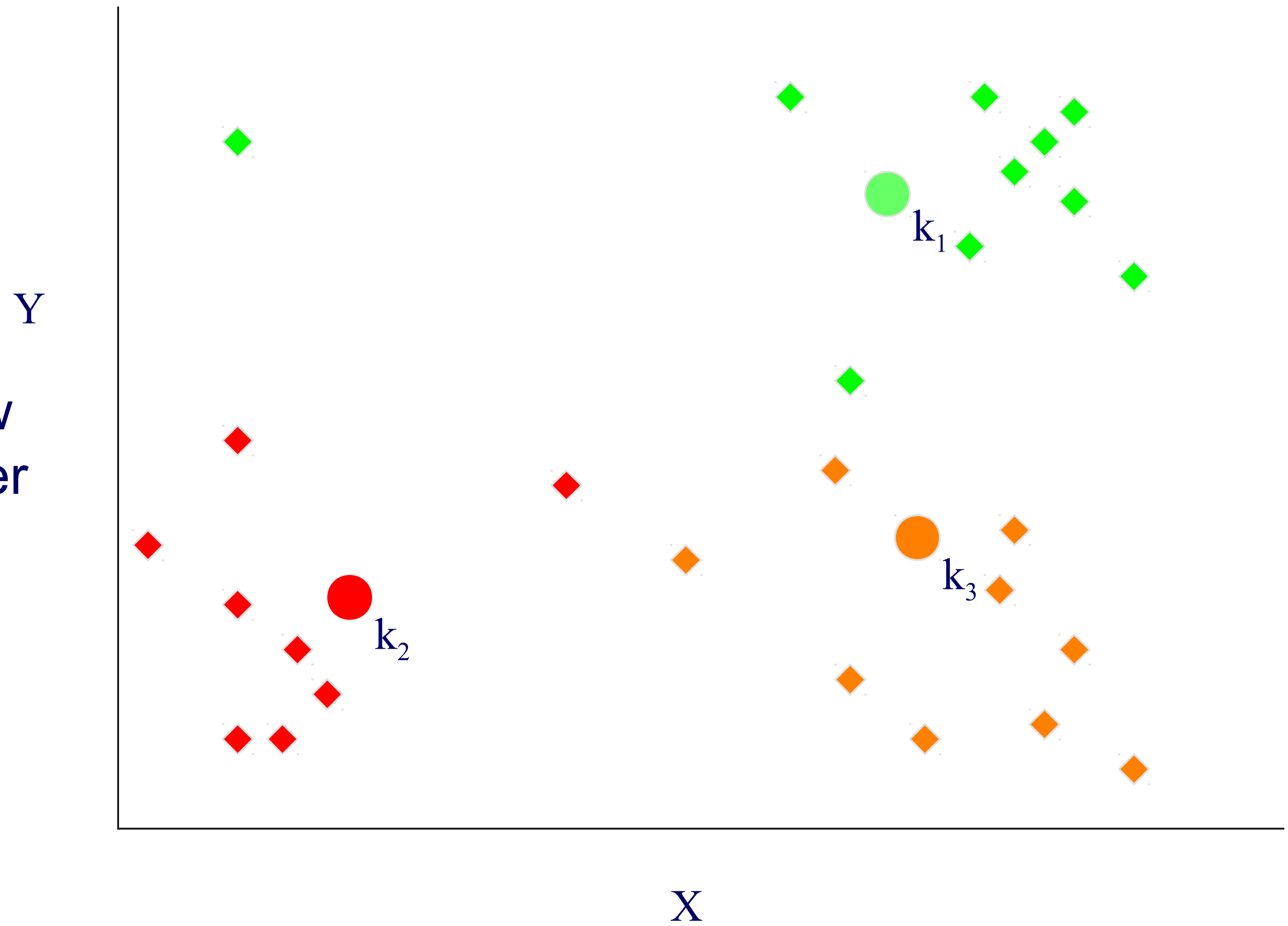
Move
each cluster
center
to the mean
of each cluster



K-Means Example, Step 4 …

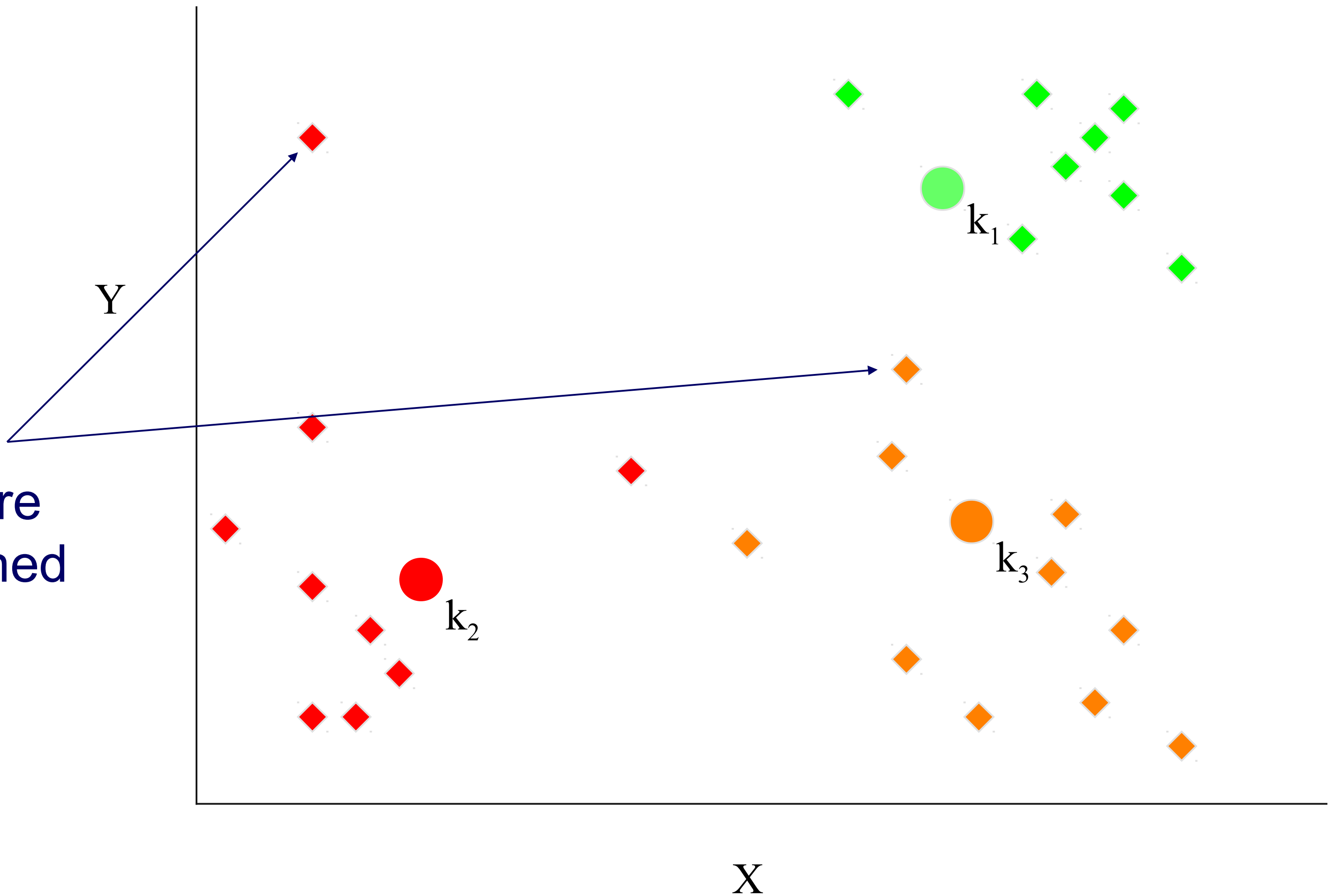
Reassign points closest to a different new cluster center

Q: Which points are reassigned?



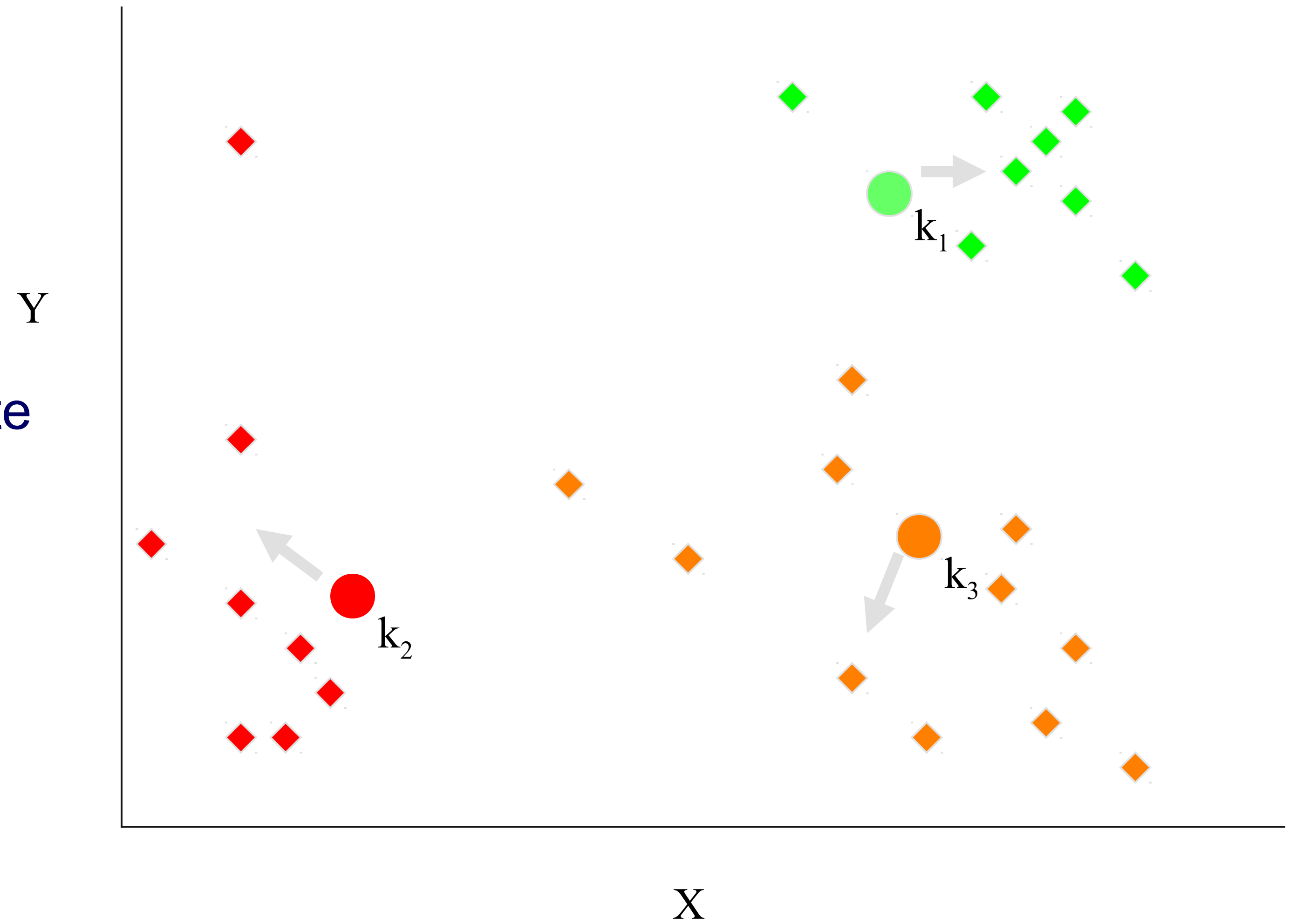
K-Means Example, Step 4

A: Two points are reassigned

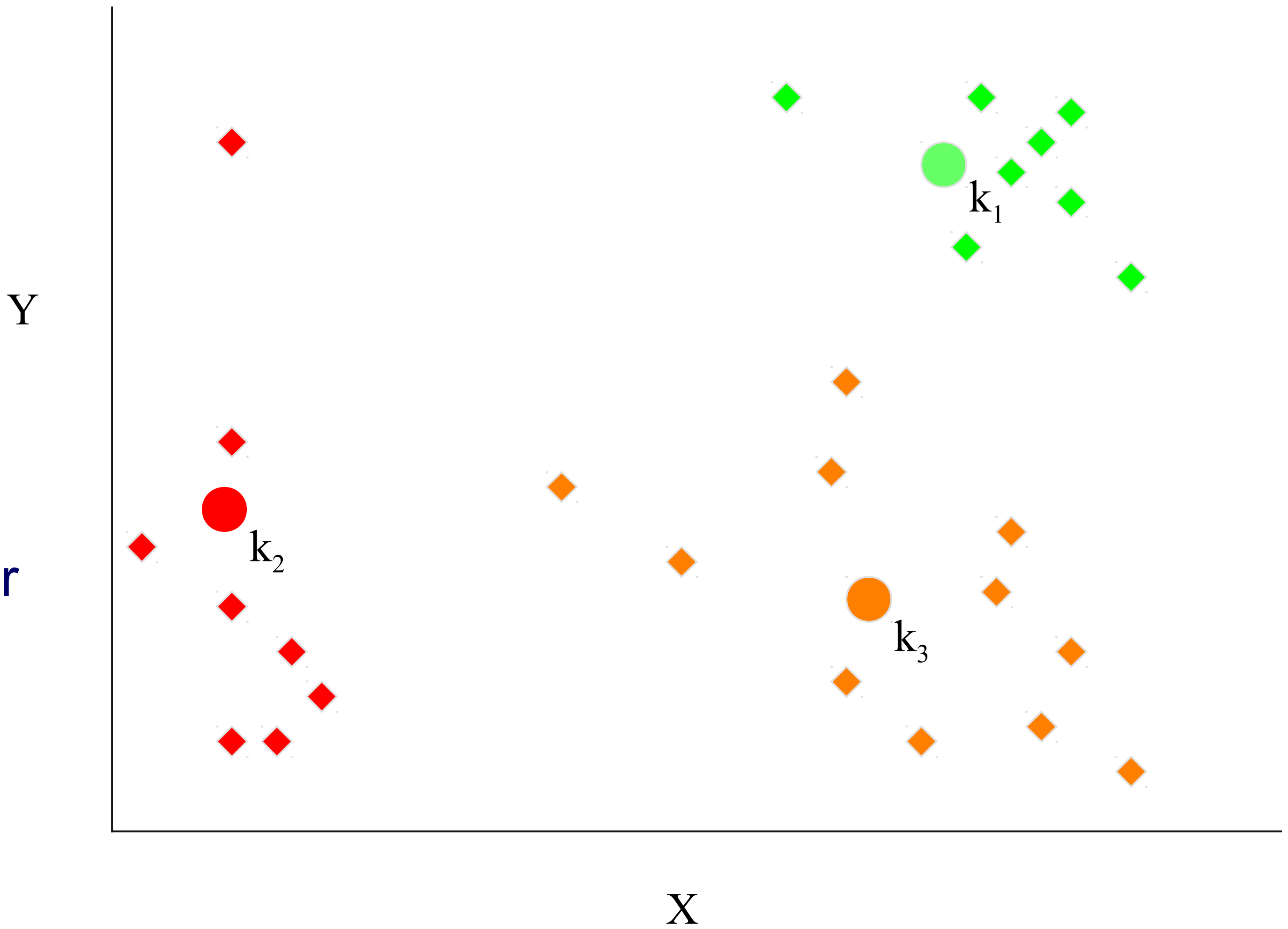


K-Means Example, Step 5

Re-compute cluster means



K-Means Example, Step 6



Move cluster centers to cluster means

Alternative Convergence Criteria

- no (or minimum) re-assignments of data points to different clusters
- no (or minimum) change of centroids, or
- minimum decrease in the sum of squared errors (SSE)
  - see next slide
- Stop after X iterations

Evaluating K-Means Clusterings

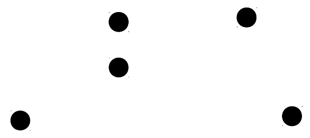
- The most common cohesion measure is the Sum of Squared Errors (SSE)
  - For each point, the error is the distance to the nearest centroid
  - To get SSE, we square these errors and sum them.

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

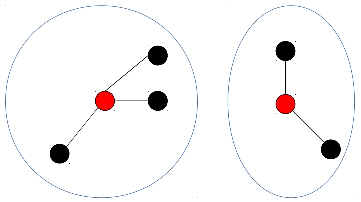
  - C_j is the j-th cluster
  - m_j is the centroid of cluster C_j (the mean vector of all the data points in C_j)
  - $dist(x, m_j)$ is the distance between data point x and centroid m_j
- Given several clusterings (and a fixed k),
  we should prefer the one with the smallest SSE

Illustration: Sum of Squared Error

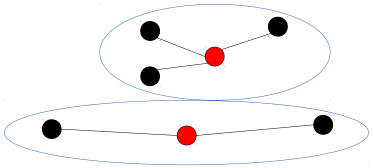
- Clustering problem given:



- Good solution:
  - i.e., small distances to centroid

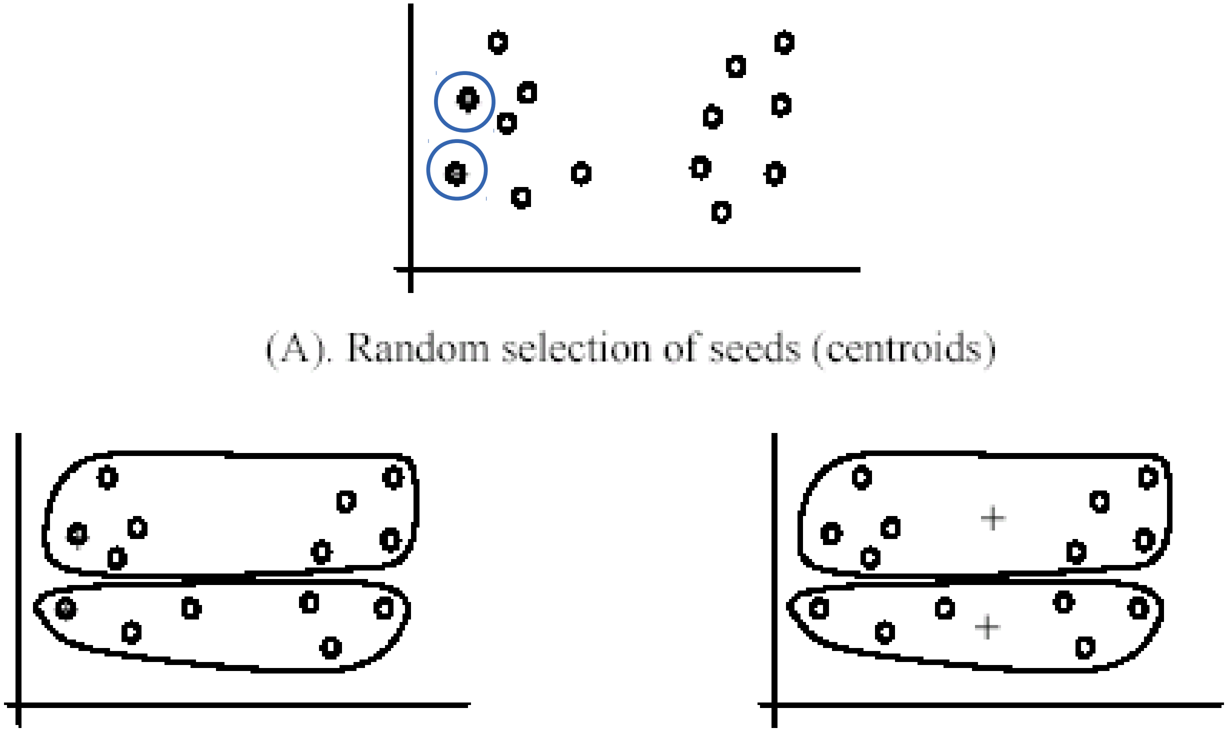


- Not so good solution:
  - i.e., larger distances to centroid



Weaknesses of K-Means: Initial Seeds

- Results can vary significantly depending on initial choice of seeds (number and position)



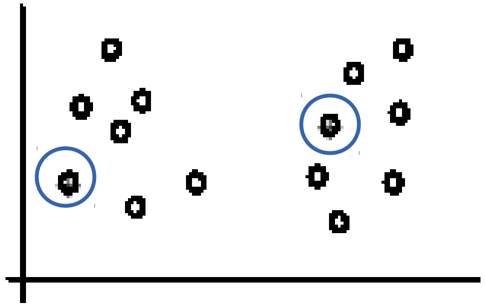
(A). Random selection of seeds (centroids)

(B). Iteration 1

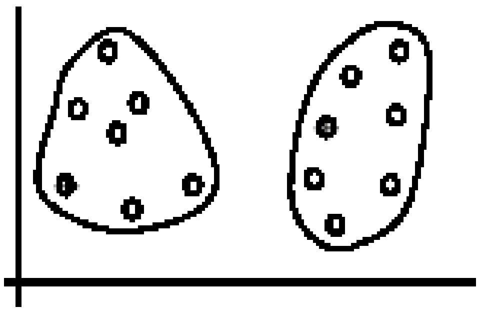
(C). Iteration 2

Weaknesses of K-Means: Initial Seeds

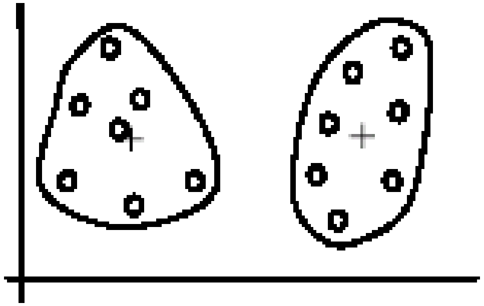
- If we use different seeds, we get good results.



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



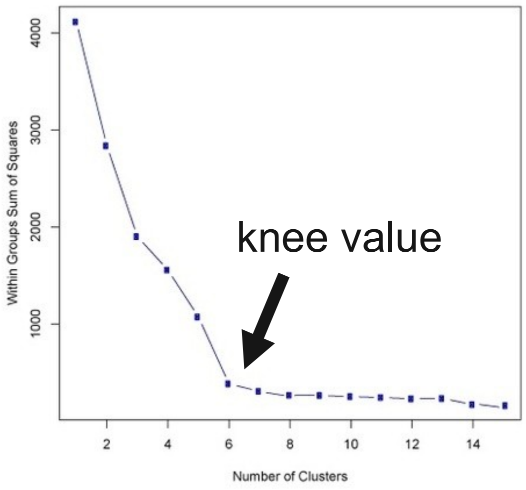
(B). Iteration 1



(C). Iteration 2

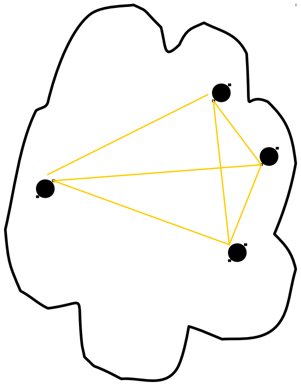
Improving the Clustering Results

- Restart a number of times with different random seeds (but fixed k)
  - chose the resulting clustering with the smallest sum of squared error (SSE)
- 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
    - Choose k where SSE improvement decreases (knee value of k)
  - Employ X-Means
    - variation of K-Means algorithm that automatically determines k
    - starts with small k, then splits large clusters until improvement decreases

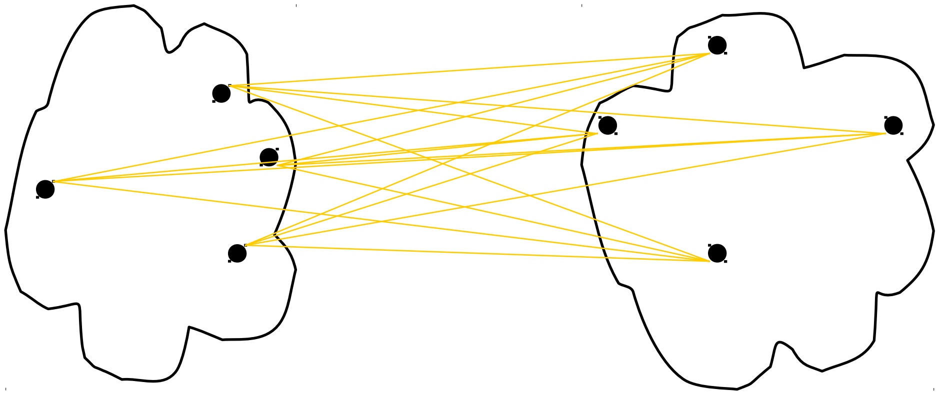


Choosing k – Cluster Evaluation

- Recap: we want to maximize
  - Cohesion: measures how closely related are objects in a cluster
  - Separation: measure how distinct or well-separated a cluster is from other clusters



cohesion



separation

Silhouette Coefficient

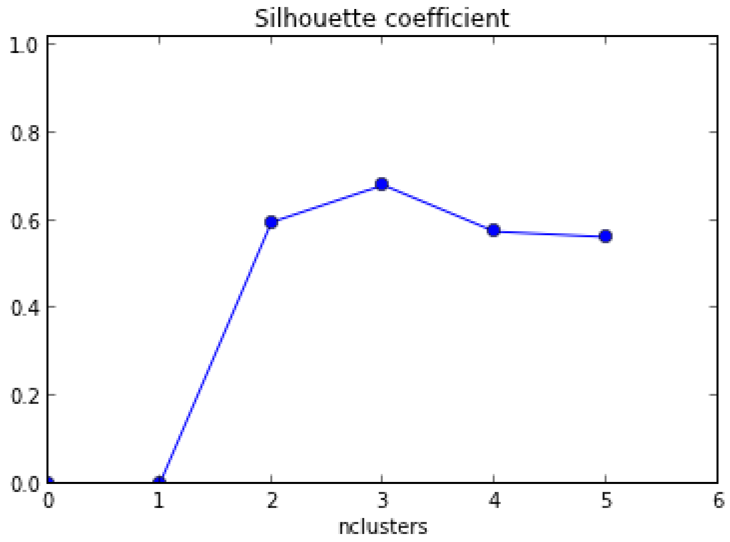
- *Cohesion a*(x): average distance of x to all other vectors in the same cluster.
- *Separation b*(x): average distance of x to the vectors in other clusters. Find the minimum among the clusters.
- *Silhouette s*(x):

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}$$

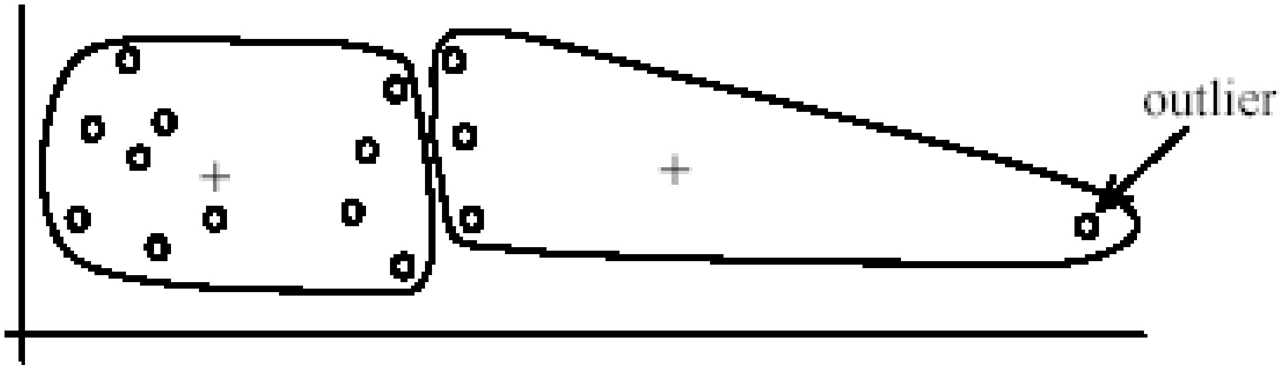
- $s(x)$ = [-1, +1]: -1=bad, 0=indifferent, 1=good
- Silhouette coefficient (SC): $$SC = \frac{1}{N}\sum_{i=1}^{N} s(x)$$

Selecting k Using the Silhouette Coefficient

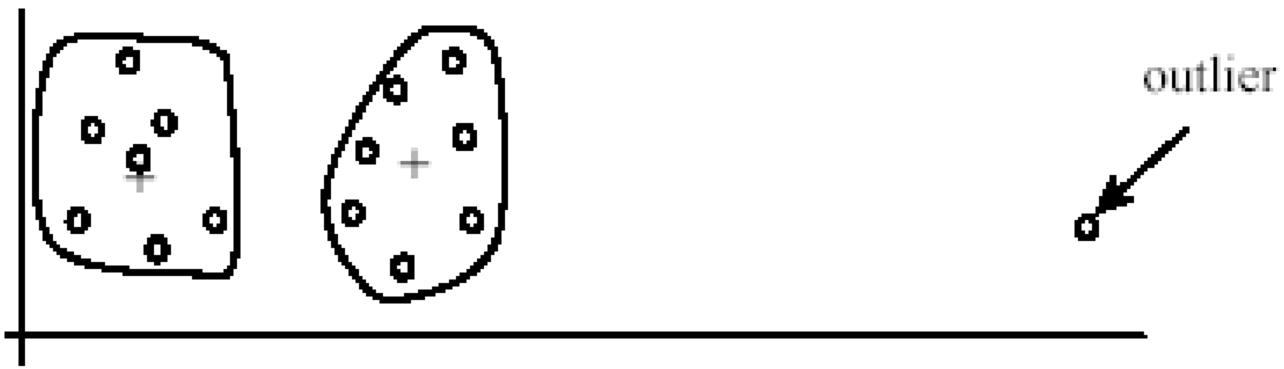
- Approach
  - Run k-means with different k values
  - Plot the Silhouette Coefficient
  - Pick the best (i.e., highest silhouette coefficient



Weaknesses of K-Means: Outlier Handling



(A): Undesirable clusters



(B): Ideal clusters

Weaknesses of K-Means: Outlier Handling

- Possible remedy:
  - remove data points far away from centroids
  - to be safe: monitor these possible outliers over a few iterations and then decide to remove them
- Other remedy: random sampling
  - choose a small subset of the data points
  - the chance of selecting an outlier is very small if the data set is large enough
  - after determining the centroids based on samples, assign the rest of the data points
  - also a method for improving runtime performance!

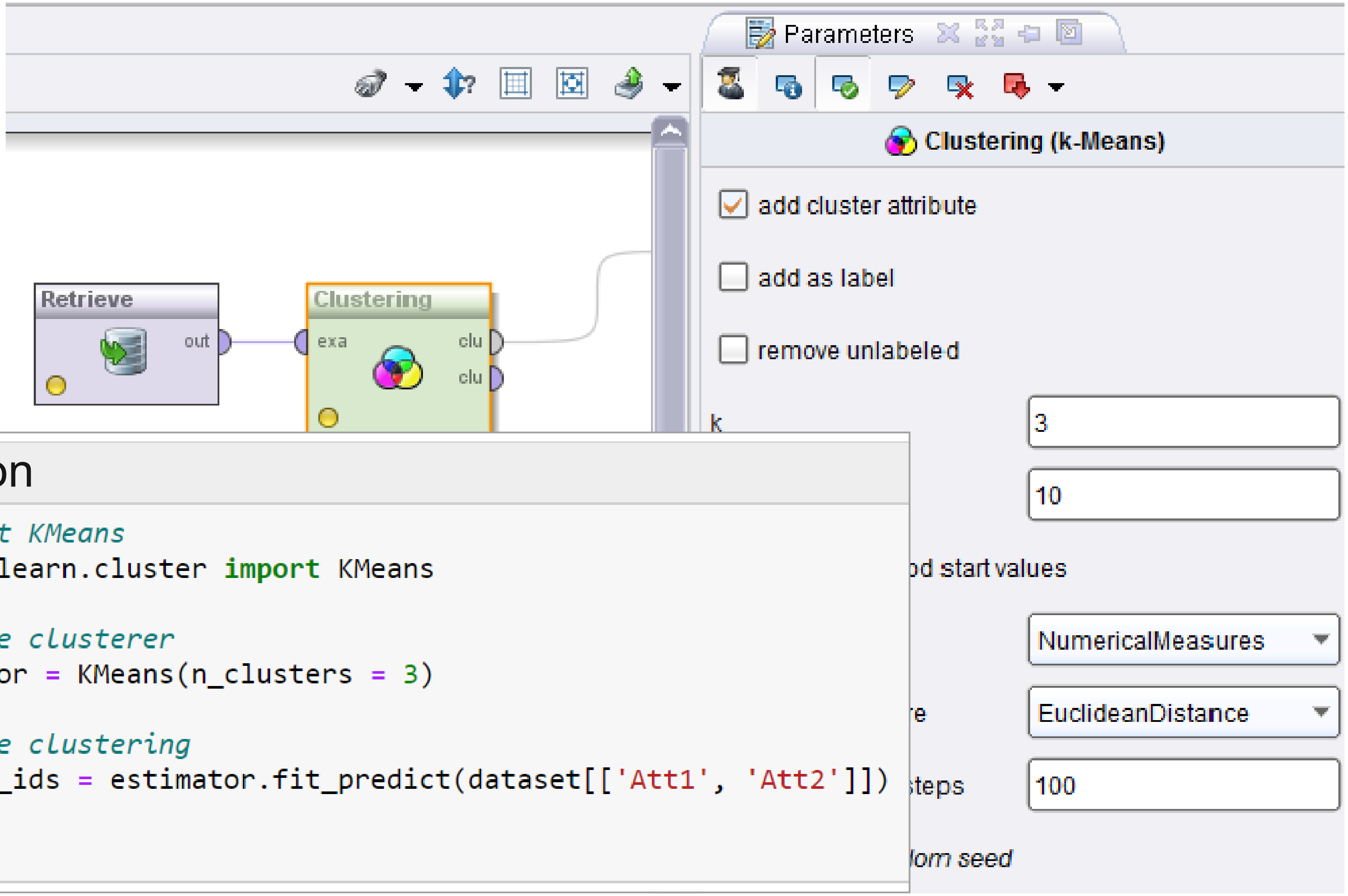
K-Medoids

- K-Medoids is a K-Means variation that uses the medians 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 not 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

K-Means Clustering Summary

- Advantages
  - Simple, understandable
  - Efficient time complexity: O(t k n)
    - n: number of data points
    - k: number of clusters
    - t: number of iterations
- Disadvantages
  - Must pick number of clusters before hand
  - All items are forced into a cluster
  - Sensitive to outliers
  - Sensitve to initial seeds

K-Means Clustering in RapidMiner & Python



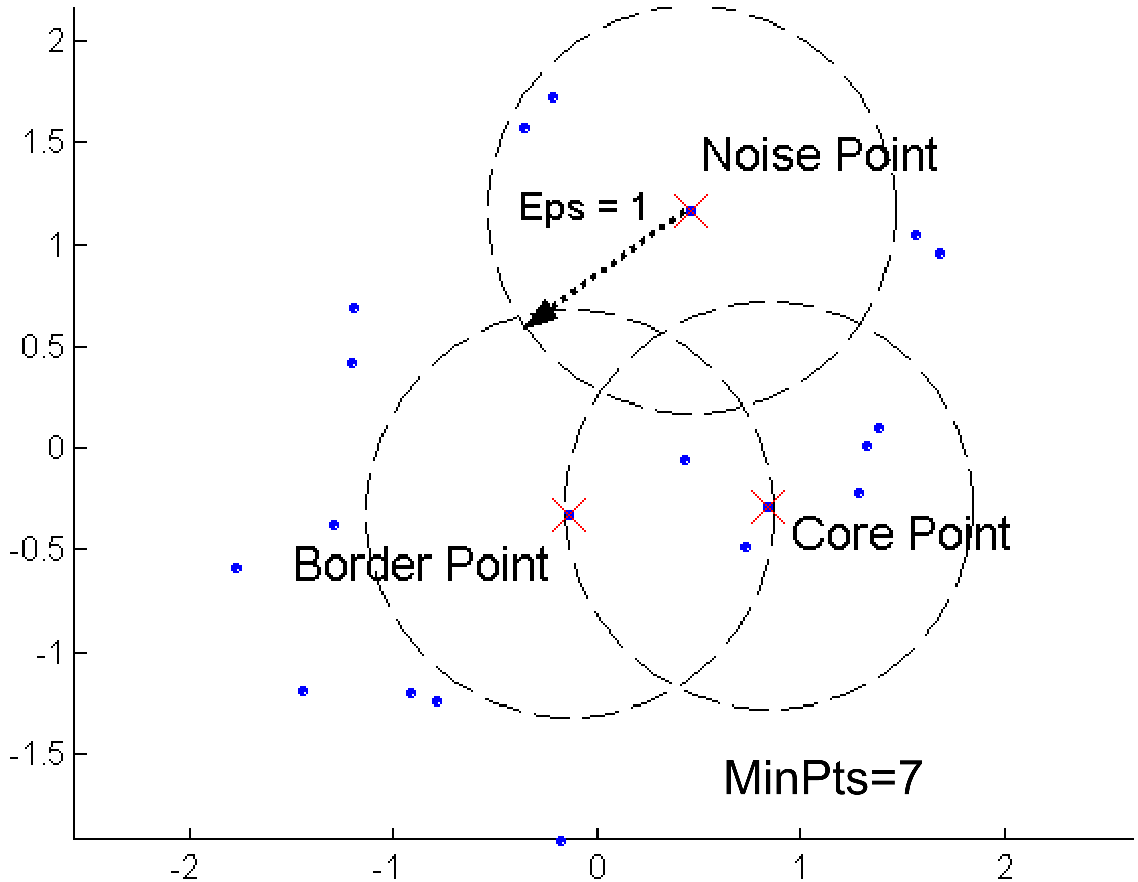
Python

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

# create clusterer
estimator = KMeans(n_clusters = 3)

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

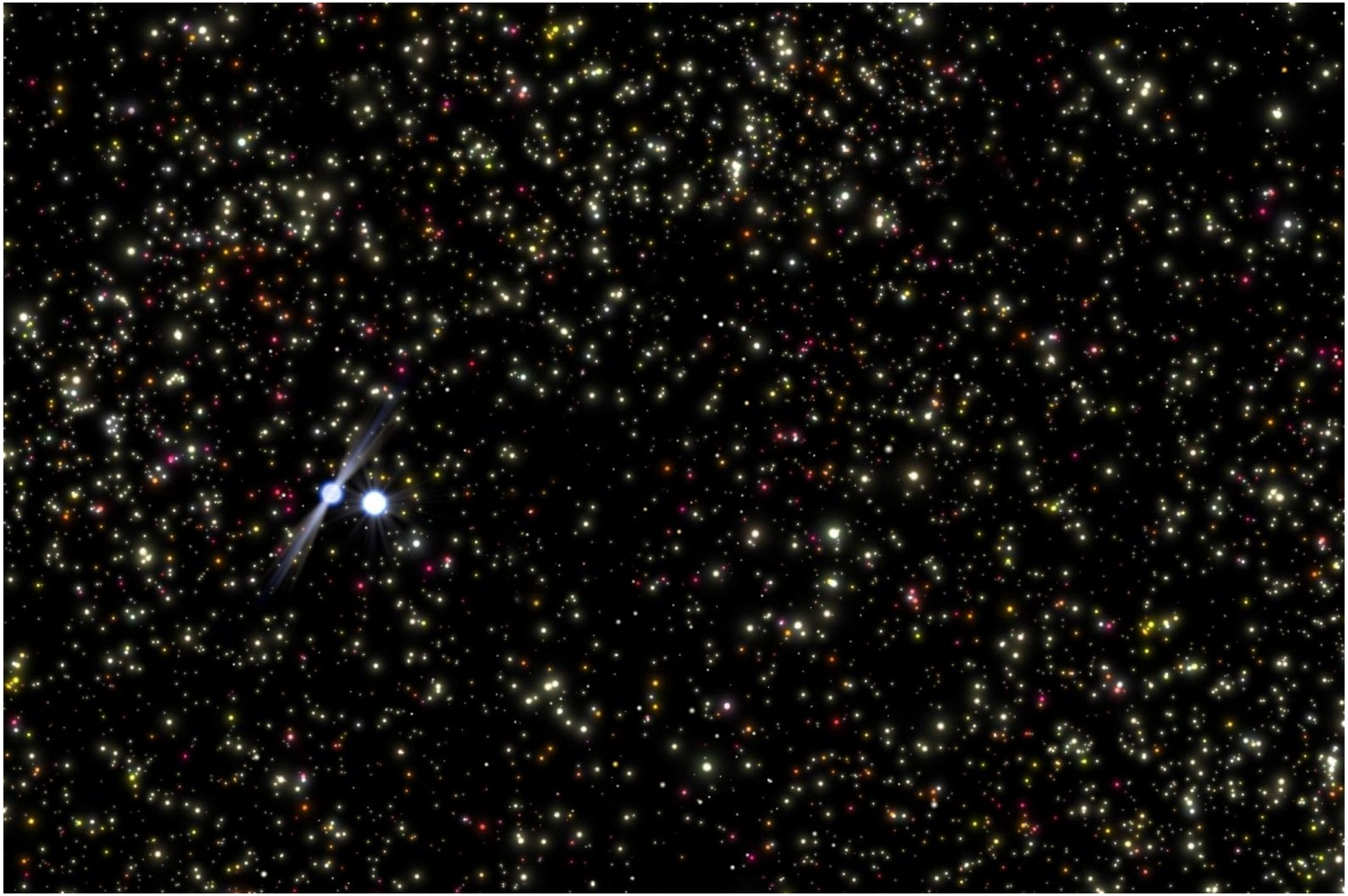
DBSCAN

- DBSCAN is a density-based algorithm
  - Density = number of points within a specified radius (Eps)
- Divides data points in three classes:
  - A point is a core point if it has more than a specified number of points (MinPts) within Eps, including the point itself
    - These are points that are at the interior of a cluster
  - A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point
  - A noise point is any point that is not a core point or a border point
    - like a cluster named “other” or “misc.”

DBSCAN: Core, Border, and Noise Points



DBSCAN: Illustrative Example



DBSCAN Algorithm

- Eliminate noise points
- Perform clustering on the remaining points

perform recursion for all points in the Eps-neighborhood of the point

$current\_cluster\_label \leftarrow 1$

for all core points **do**

  if the core point has no cluster label **then**

    $current\_cluster\_label \leftarrow current\_cluster\_label + 1$

    Label the current core point with cluster label $current\_cluster\_label$

  end if

  for all points in the Eps-neighborhood, except i^{th} the point itself **do**

    if the point does not have a cluster label **then**

      Label the point with cluster label $current\_cluster\_label$

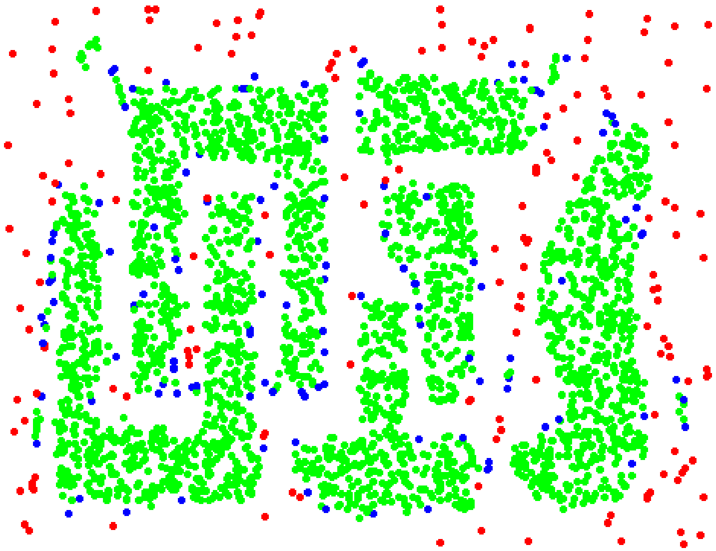
    end if

  end for

end for

DBSCAN: Core, Border and Noise Points

Original Points

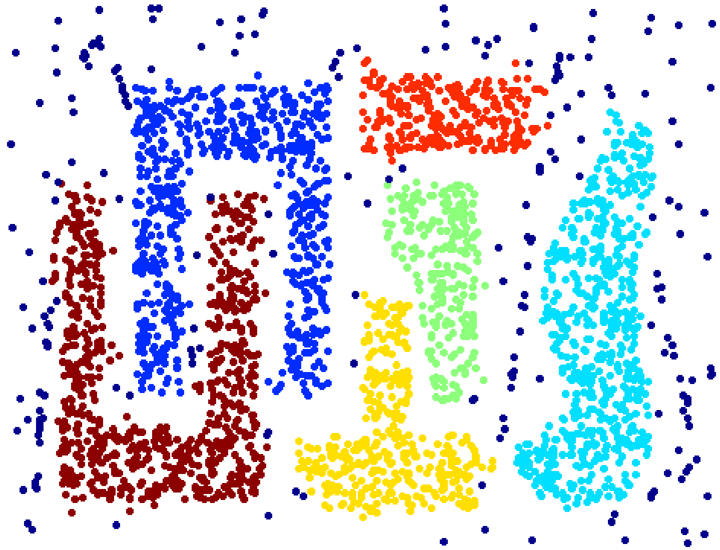


Point types: core, border and noise

Eps = 10, MinPts = 4

When DBSCAN Works Well

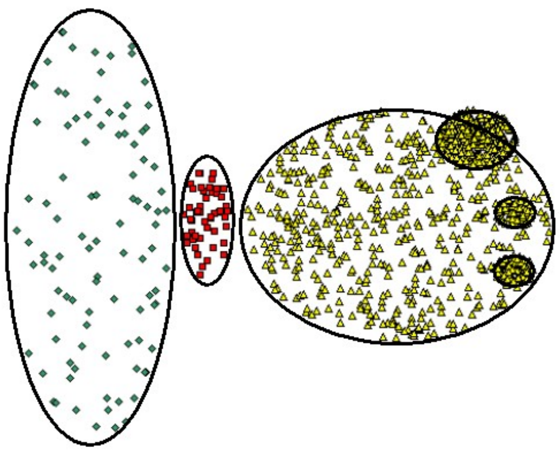
Original Points



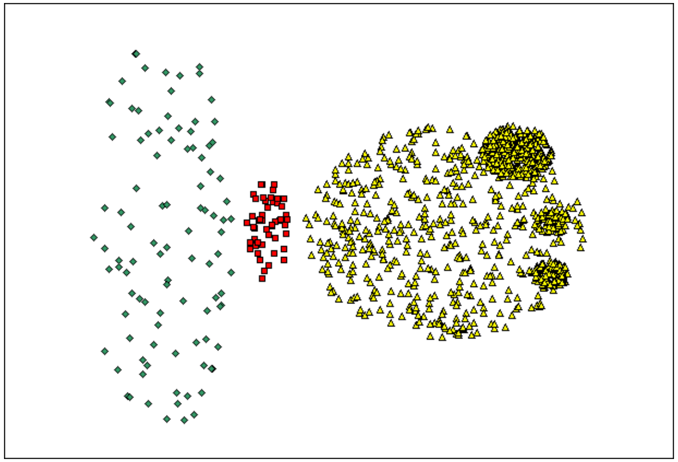
Clusters

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

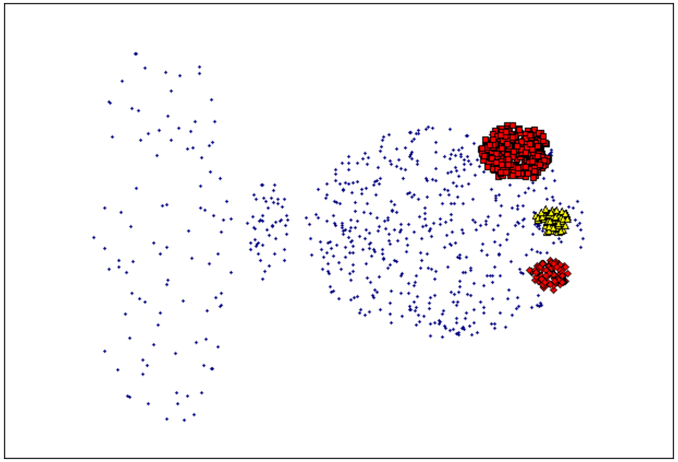
When DBSCAN Does NOT Work Well



Original Points



(MinPts=4, Eps=9.75).

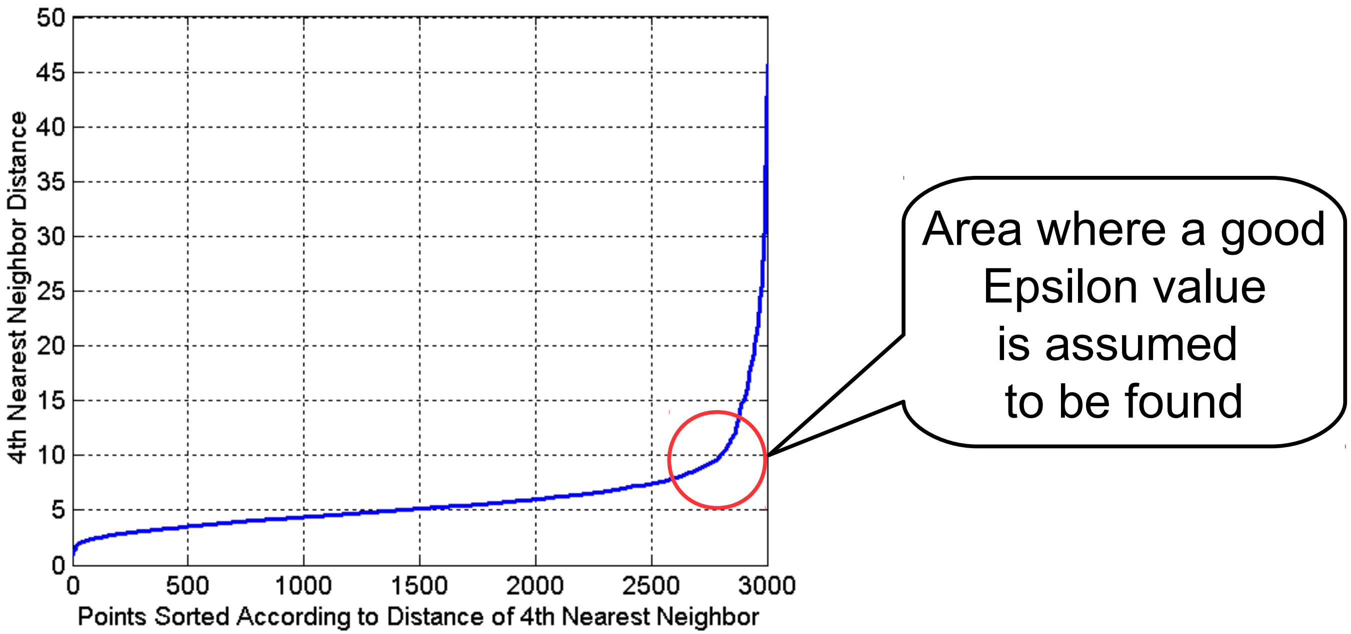


(MinPts=4, Eps=9.92)

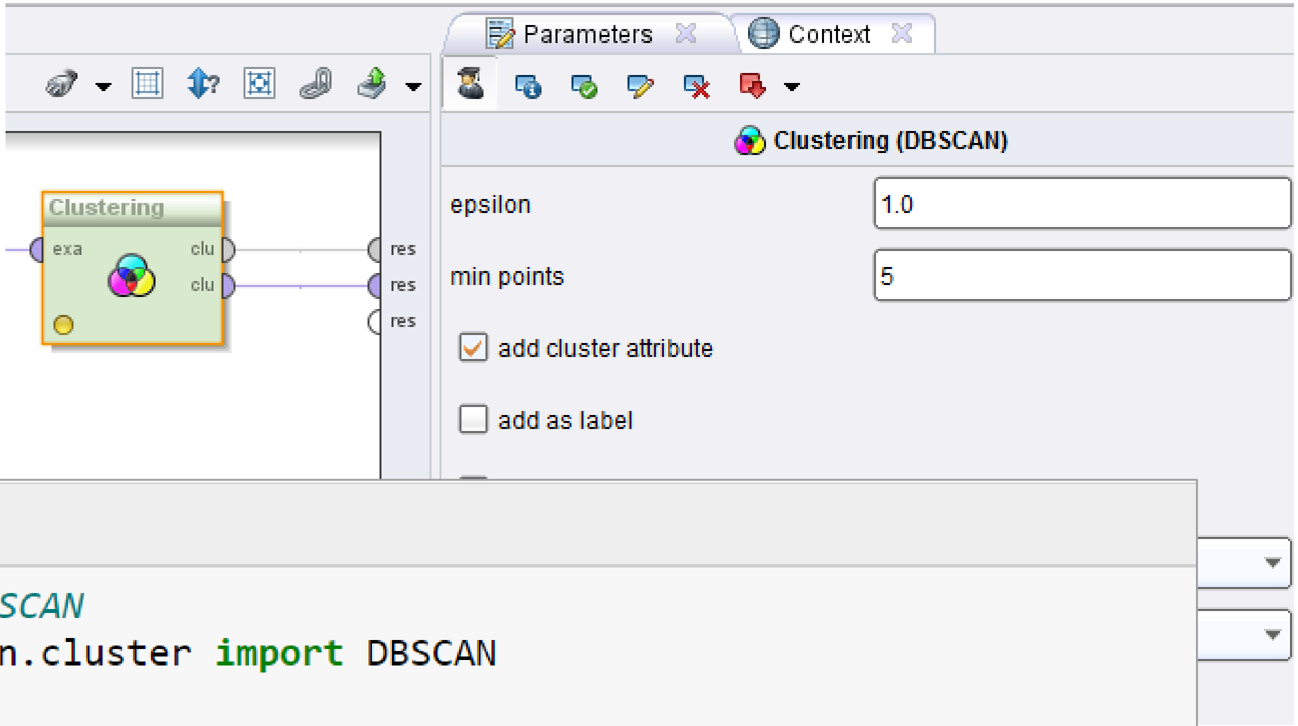
- **Varying densities**
- **High-dimensional data**

DBSCAN: Determining EPS and MinPts

- Idea: for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance
- Noise points have the k^{th} nearest neighbor at farther distance
- So, plot sorted distance of every point to its k^{th} nearest neighbor



DBScan in RapidMiner & Python

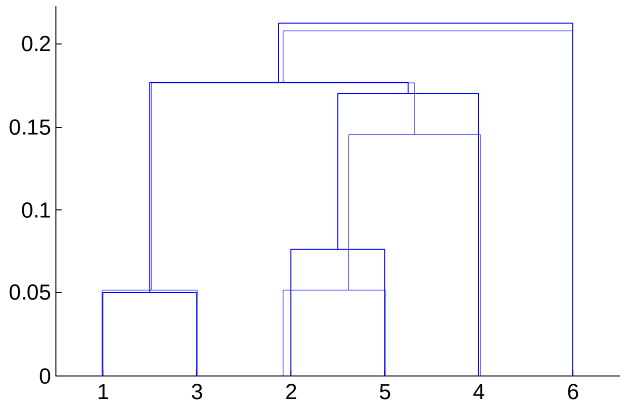


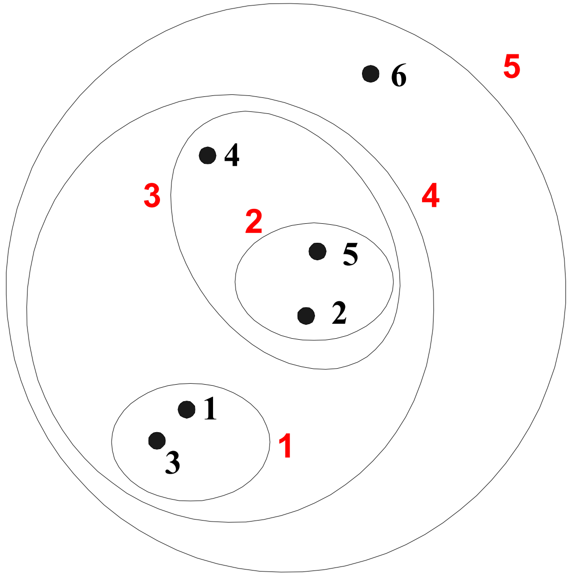
Python

```
# 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']])
```

Hierarchical Clustering

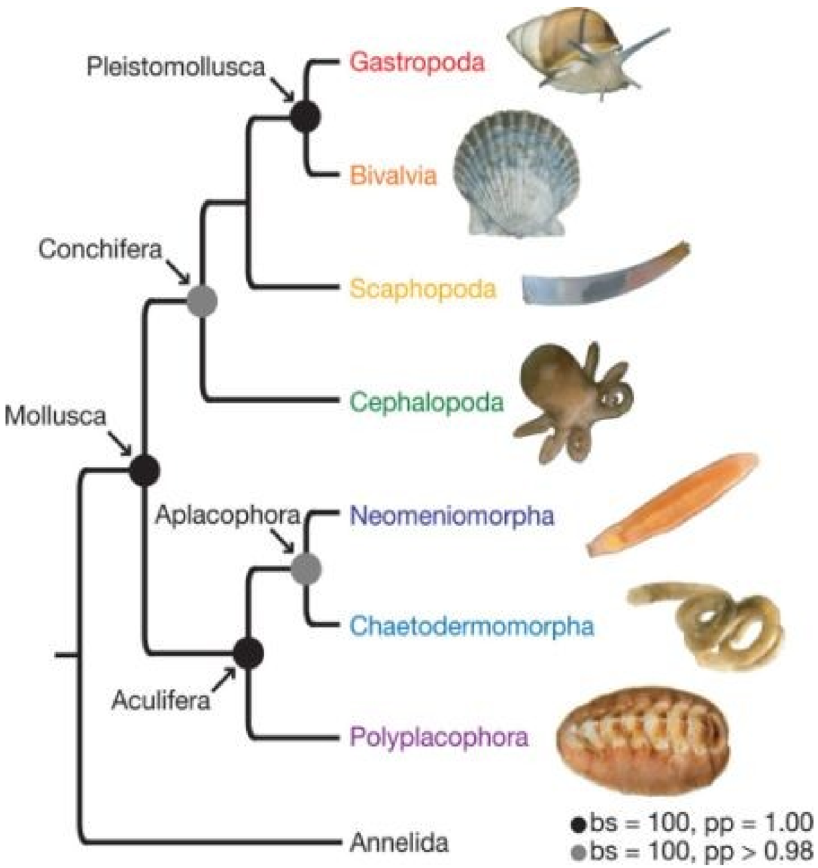
- Produces a set of nested clusters organized as a hierarchical tree.
- Can be visualized as a Dendrogram
  - A tree like diagram that records the sequences of merges or splits.
  - The y-axis displays the former distance between merged clusters.





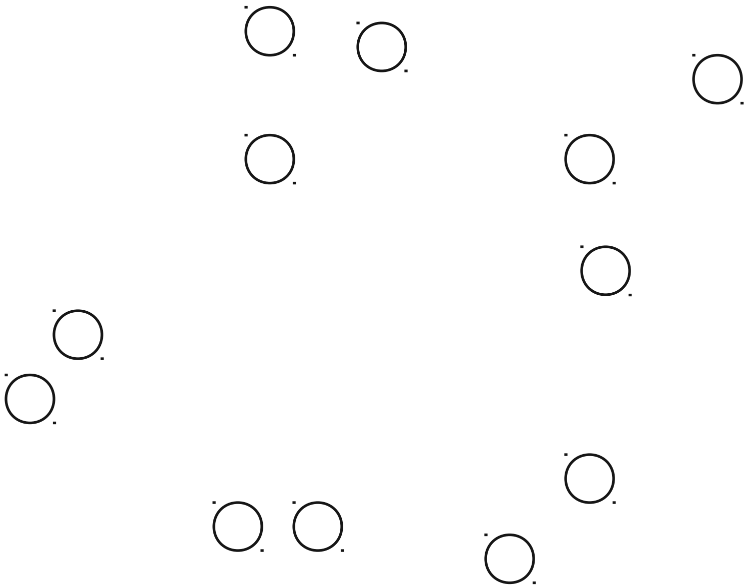
Strengths of Hierarchical Clustering

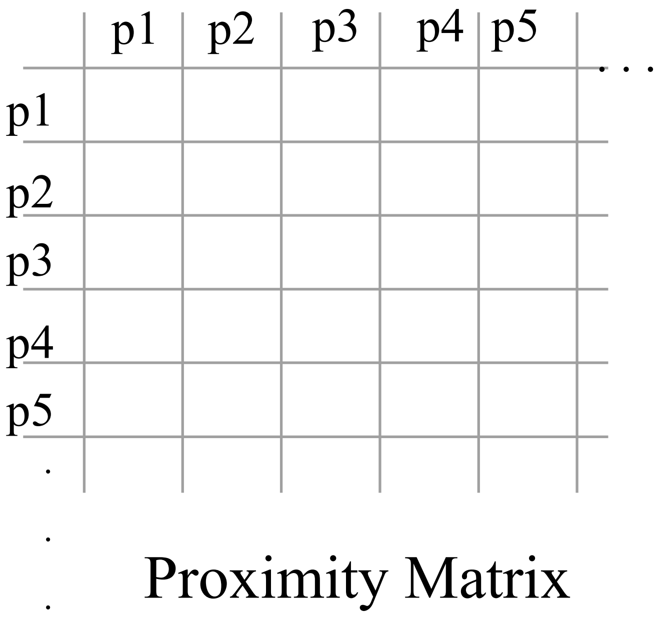
- We do not have to assume any particular number of clusters
  - Any desired number of clusters can be obtained by ‘cutting’ the dendogram at the proper level
- May be used to look for meaningful taxonomies
  - taxonomies in life sciences
  - taxonomy of customer groups



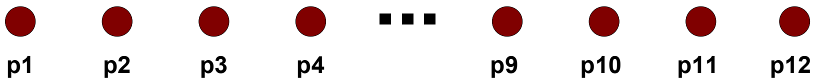
Starting Situation

- Start with clusters of individual points and a proximity matrix



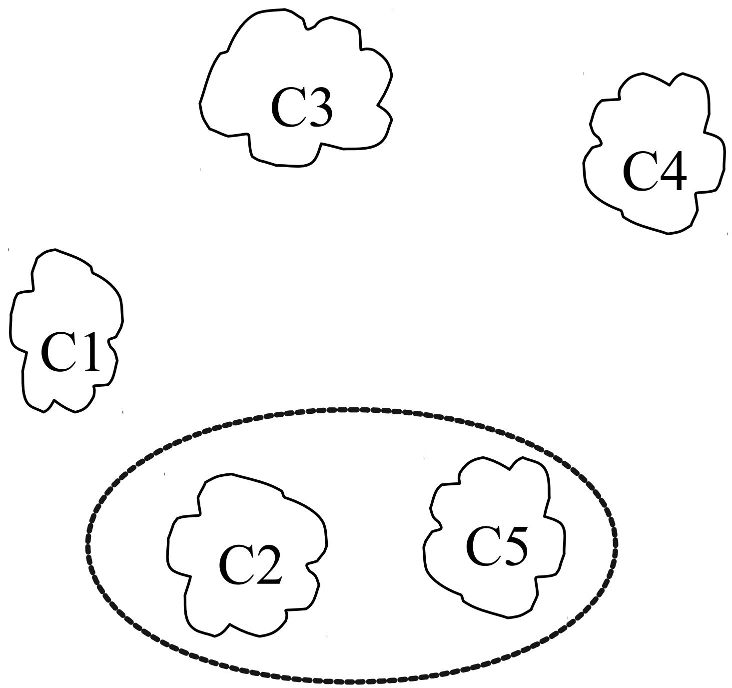


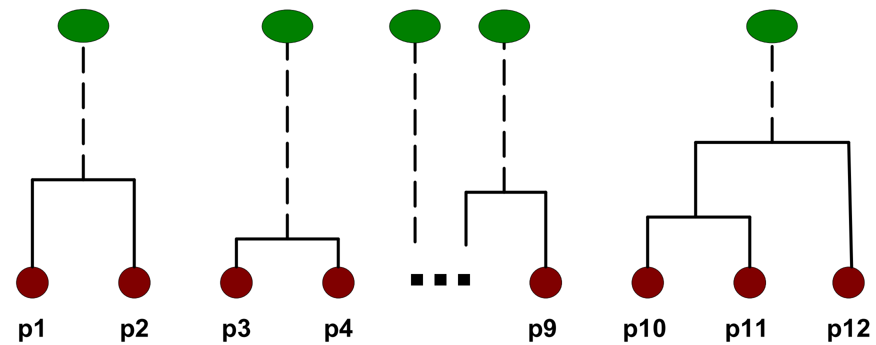
Proximity Matrix



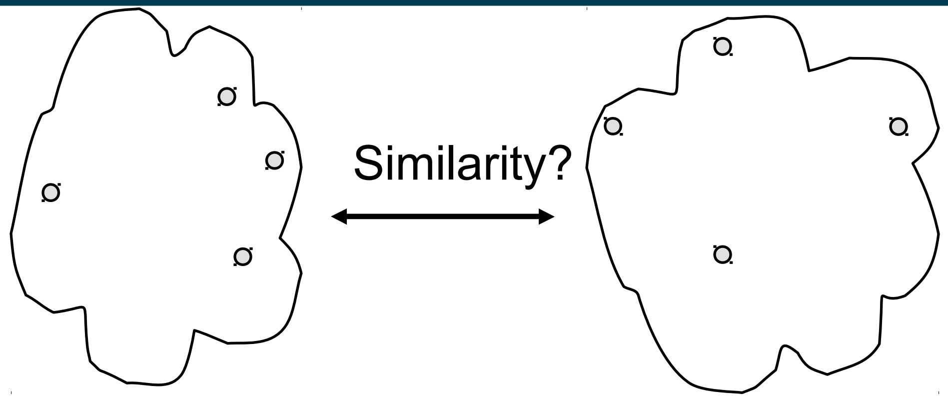
Intermediate Situation

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





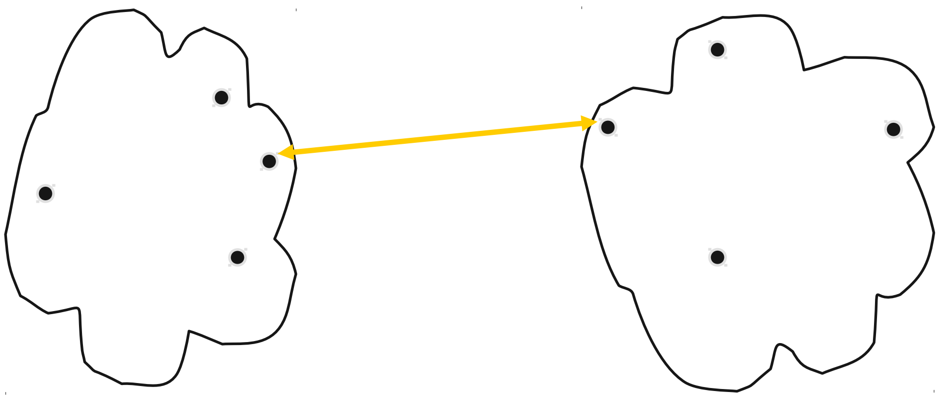
How to Define Inter-Cluster Similarity?



Possible approaches:

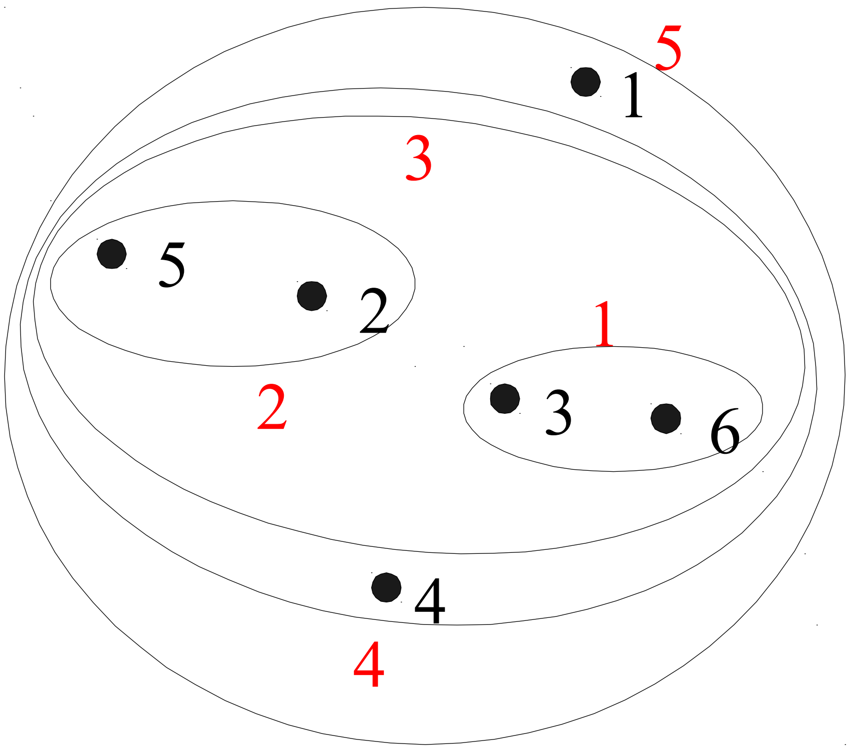
- Single Link (MIN)
- Complete Link (MAX)
- Group Average
- Distance Between Centroids

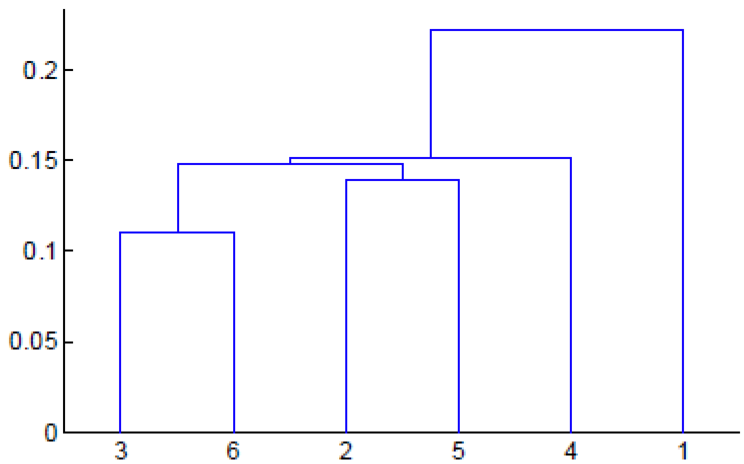
Cluster Similarity: Single Link



- 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

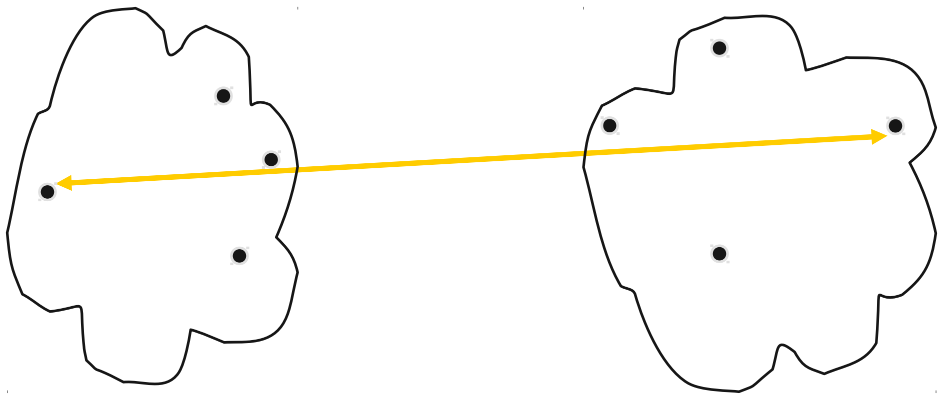




Nested Clusters

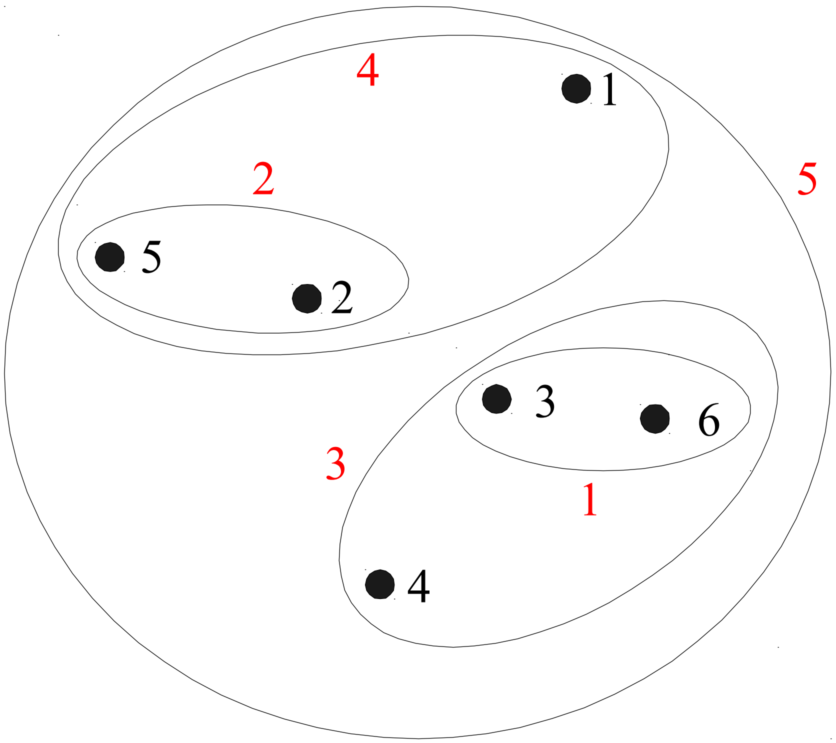
Dendrogram

Cluster Similarity: Complete Linkage

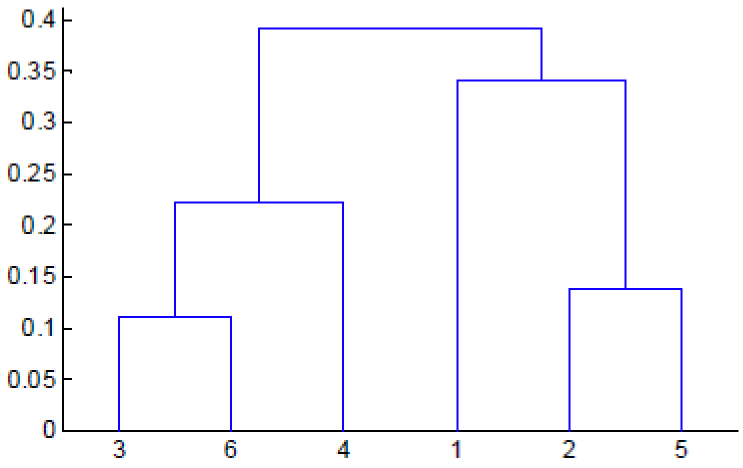


- Similarity of two clusters is based on the two least similar (most distant) points in the different clusters
- Determined by all pairs of points in the two clusters

Example: Complete Linkage



Nested Clusters



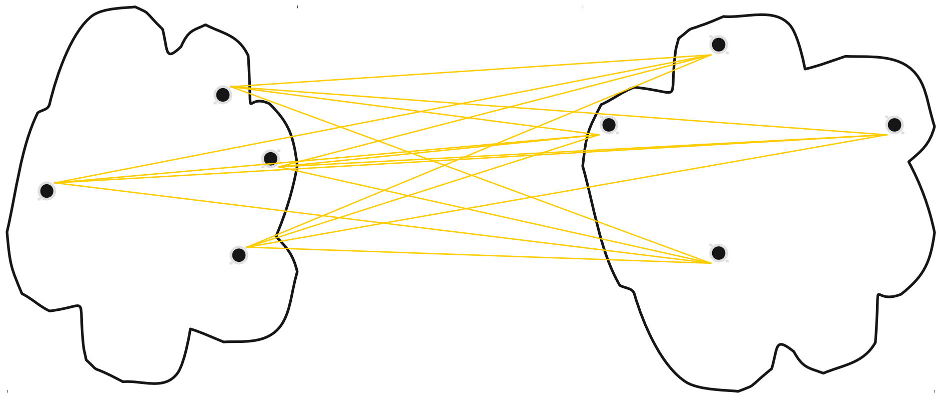
Dendrogram

Single Link vs. Complete Linkage

- Single Link:
  - Pro: Can handle non-elliptic shapes
  - Con: Sensitive to outliers

- Complete Linkage:
  - Pro: Less sensitive to noise and outliers
  - Con: biased towards globular clusters
  - Con: tends to break large clusters

Cluster Similarity: Group Average

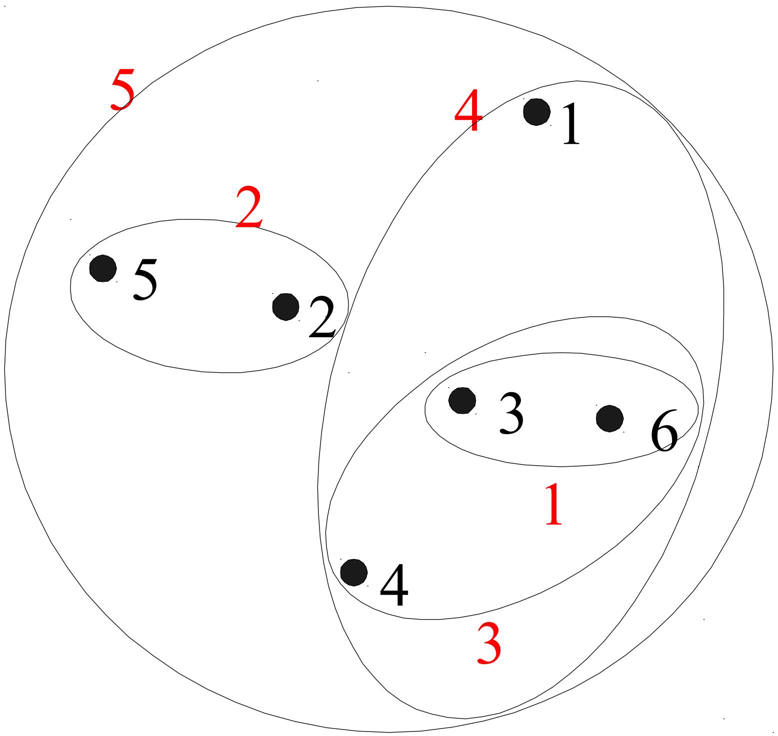


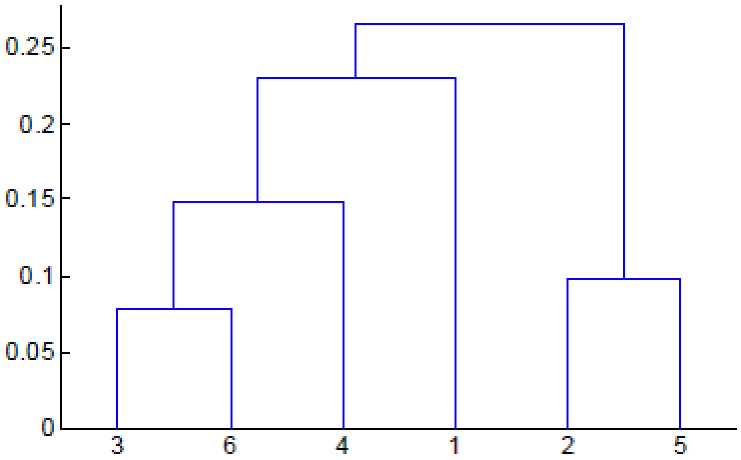
- Proximity of two clusters is the average of pair-wise proximity between points in the two clusters.

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

- Need to use average connectivity for scalability since total proximity favors large clusters

Example: Group Average





Nested Clusters

Dendrogram

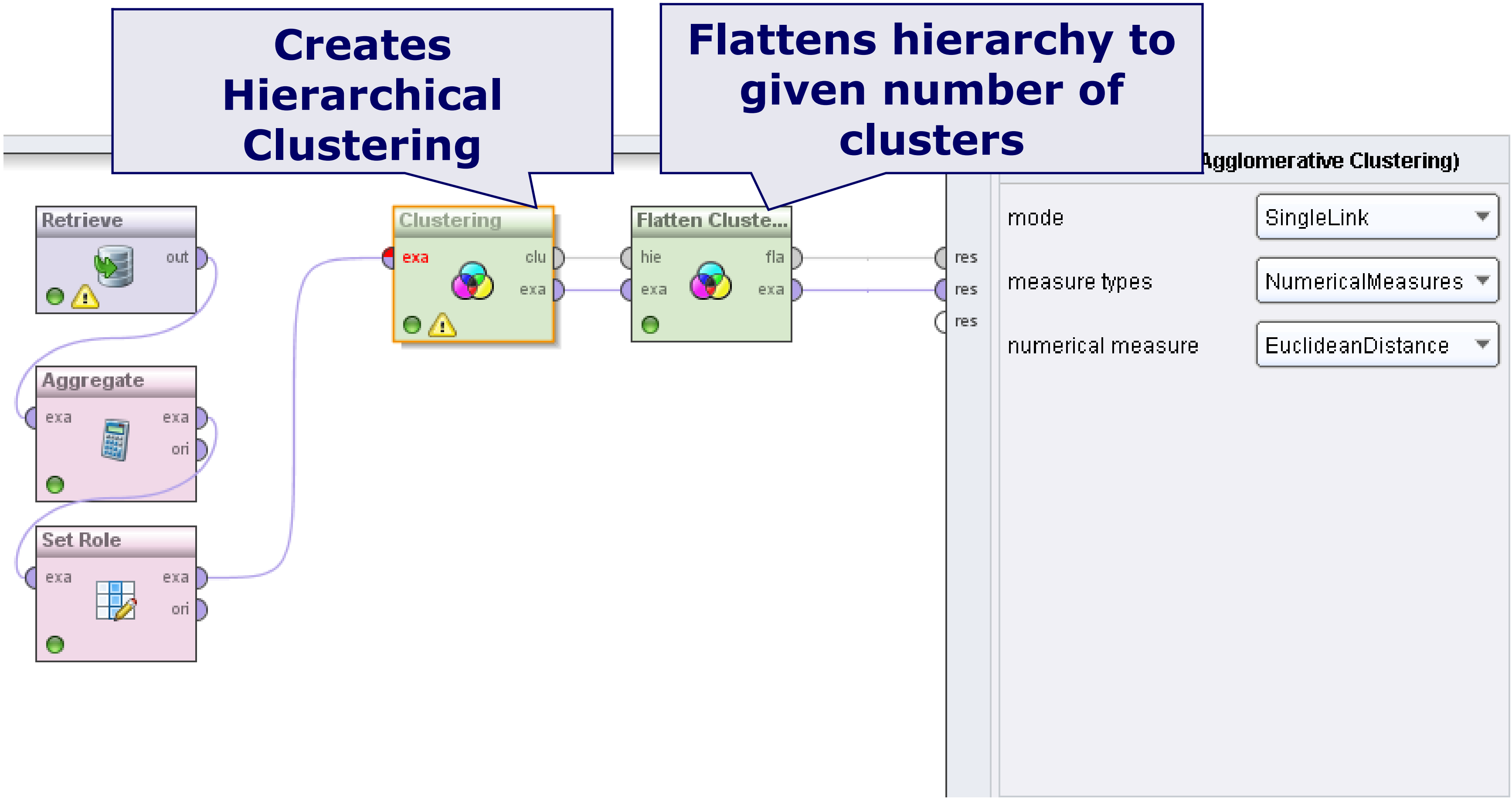
Hierarchical Clustering: Group Average

- Compromise between Single and Complete Link
- Strengths
  - Less susceptible to noise and outliers
- Limitations
  - Biased towards globular clusters

Hierarchical Clustering: Problems & Limitations

- Greedy algorithm:
  - decision taken (i.e., merge two clusters) cannot be undone
- Different variants have problems with one or more of the following
  - Sensitivity to noise and outliers
  - Difficulty handling different sized clusters and convex shapes
  - Breaking large clusters
- High Space and Time Complexity
  - $O(N^2)$ space since it uses the proximity matrix (N: number of data points)
  - $O(N^3)$ time in many cases
  - N steps procesing the similarity matrix (N^2)
    - Complexity can be reduced to $O(N \log(N))$ time for some approaches

Agglomerative Clustering in RapidMiner



Agglomerative 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’

Proximity Measures

- So far, we have seen different clustering algorithms
  - all of which rely on distance (proximity, similarity, ...) measures
- Similarity
  - Numerical measure of how alike two data objects are (higher: more alike)
  - Often falls in the range [0,1]
- Dissimilarity (or distance)
  - Numerical measure of how different are two data objects (higher: less alike)
  - Minimum dissimilarity is often 0
  - Upper limit varies
- A wide range of different measures is used depending on the requirements of the application

Proximity of Single Attributes

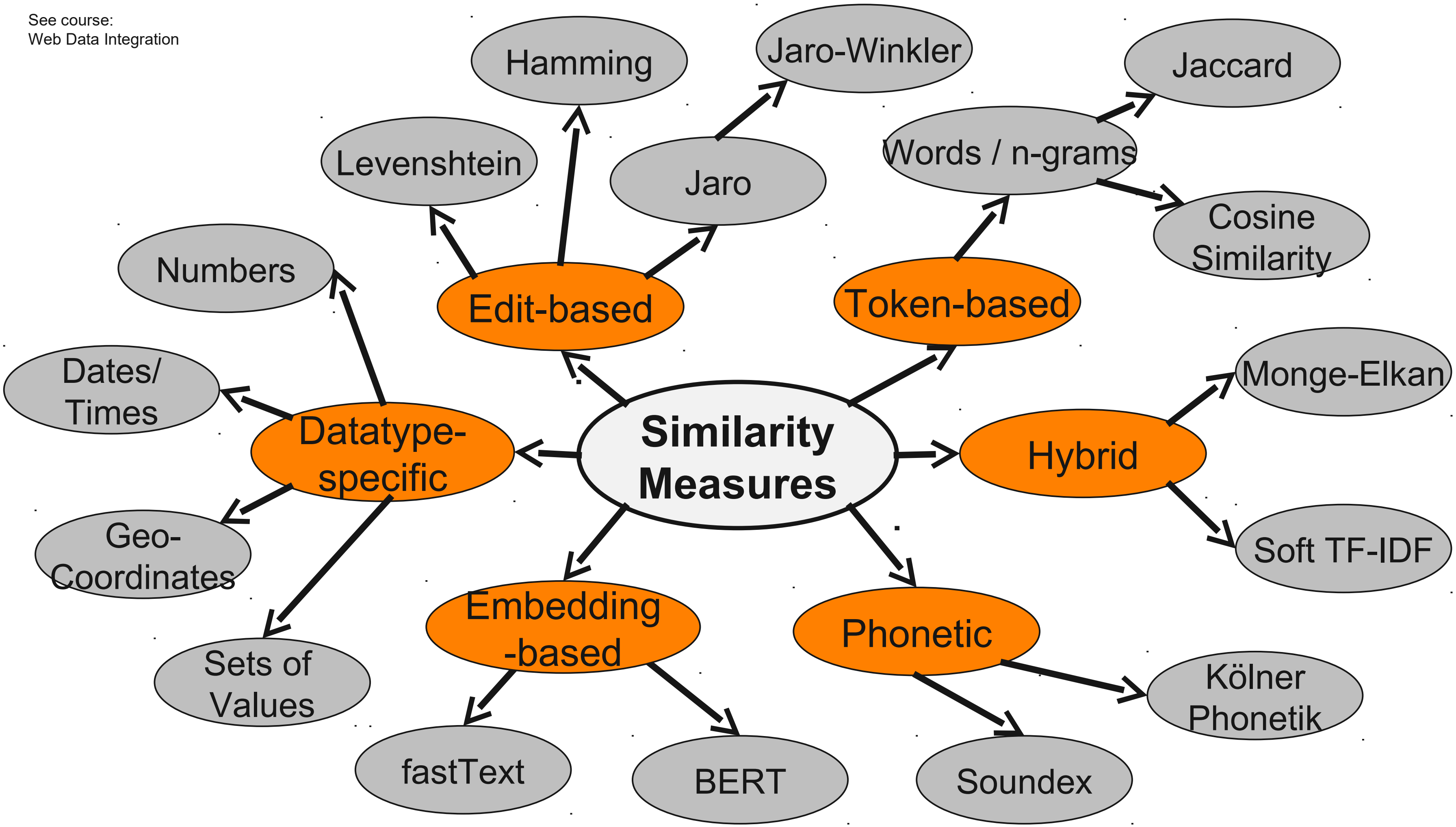
Attribute Type	Dissimilarity	Similarity
Nominal	$d = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$	$s = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$
Ordinal	$d = \frac{\lvert p-q \rvert}{n-1}$ (values mapped to integers 0 to $n-1$, where n is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

Similarity and dissimilarity for simple attributes

p and q are the attribute values for two data objects

Similarity Functions: an Overview

See course:
Web Data Integration



Proximity of Data Points

- All those measures 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

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 objects p and q.

- More generally: L_p norm:

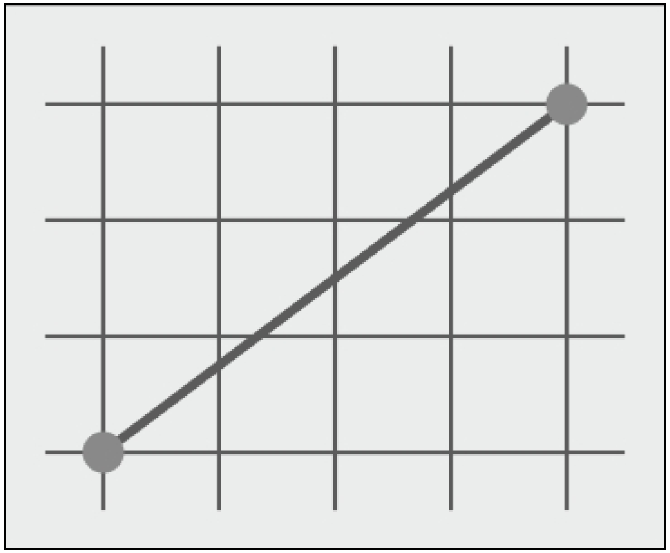
$$dist = \left(\sum_{k=1}^{n} (p_k - q_k)^p\right)^{\frac{1}{p}}$$

L_1 vs. L_2 Norm

- L_1 norm: also called Manhattan distance
  - minimum distance to go from one crossing to another
  - in a squared city (like Manhattan)

- L_2 norm: Euclidean Distance

- Example:
  - $L_1 = 7$
  - $L_2 = 5$



Caution: Pitfalls!

- Let us try to cluster the German federal states
- We have to determine the (semantic) distance, e.g., between
  - Baden-Württemberg
    - population = 10,569,111
    - area = 35,751.65 km²
  - Bavaria
    - population = 12,519,571
    - area = 70,549.44 km²
- Euclidean $= \sqrt{(10{,}569{,}111 - 12{,}591{,}571)^2 + (35{,}751.65 - 70{,}549.44)^2}$

$$= \sqrt{4.090.344.451.600 + 1.210.886.188}$$

Caution: Pitfalls!

- Let us try to cluster the German federal states
- We have to determine the distance, e.g., between
  - Baden-Württemberg
    - population = 10,569,111
    - area = 35,751,650,000 m²
  - Bavaria
    - population = 12,519,571
    - area = 70,549,440,000 m²
- Euclidean =

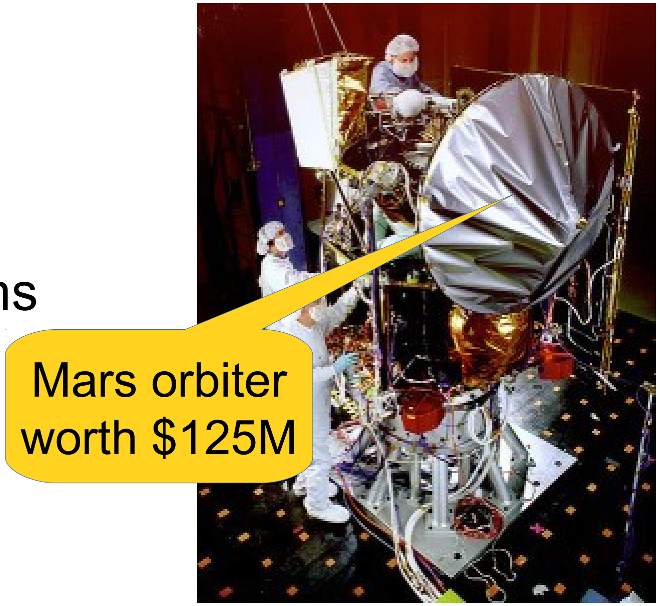
$$\sqrt{(10{,}569{,}111 - 12{,}591{,}571)^2 + (35{,}751{,}650{,}000 - 70{,}549{,}440{,}000)^2}$$

$$= \sqrt{4.090.344.451.600 + 1.210.886.188.884.100.000.000}$$

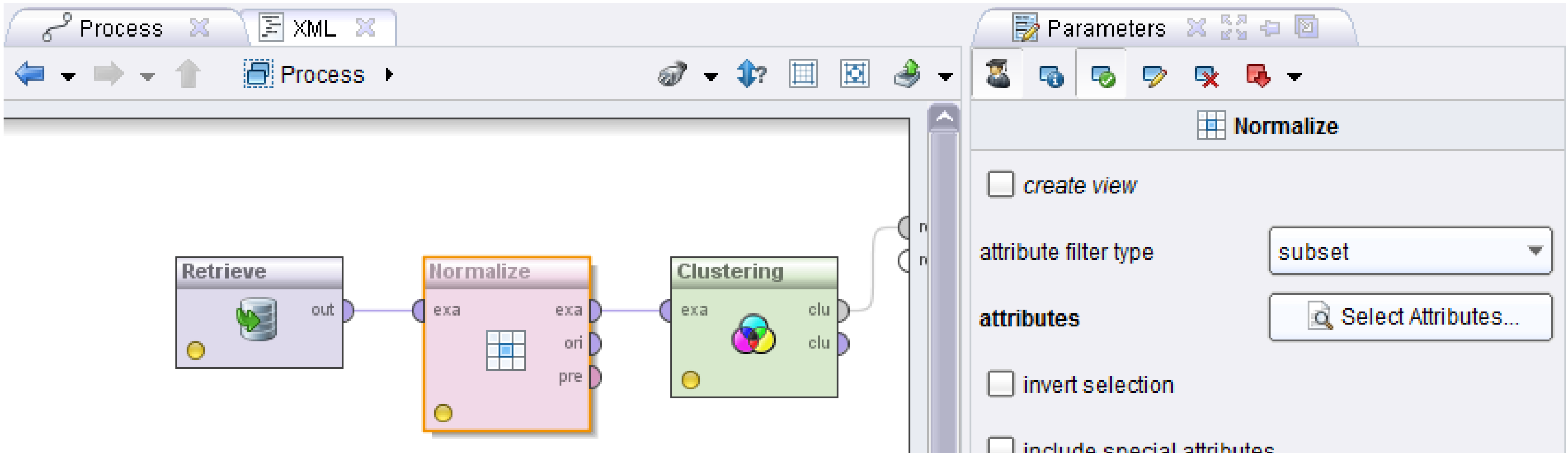
Caution: Pitfalls!

- We are easily comparing apples and oranges
  - and changing units of measurement changes the clustering result!
  - imagine: the same dataset processed in Europe (metric units) and the US (imperial units)

- Recommendation:
  - use *normalization* before clustering
  - generally: for all data mining algorithms involving *distances*



Normalization in RapidMiner & Python



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
  - e.g., customer bought an item (yes/no)

- Compute similarities using the following quantities
  - M01 = the number of attributes where p was 0 and q was 1
  - M10 = the number of attributes where p was 1 and q was 0
  - M00 = the number of attributes where p was 0 and q was 0
  - M11 = the number of attributes where p was 1 and q was 1

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 of the attribute Gender
- Similarity measure: Simple Matching Coefficient

$$SMC(x_i, x_j) = \frac{M_{11} + M_{00}}{M_{01} + M_{10} + M_{11} + M_{00}}$$

Number of matches / number of all attributes values

Asymmetric Binary Attributes

- Asymmetric: If one of the states is more important or more valuable than the other.
  - By convention, state 1 represents the more important state.
  - 1 is typically the rare or infrequent state.
  - Example: Shopping Basket, Word/Document Vector
- Similarity measure: Jaccard Coefficient

$$J(x_i, x_j) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

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

SMC versus Jaccard: Example

$p = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

$q = 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1$

example interpretation:
p bought item 1
q bought item 7 and 10

$M_{01} = 2$ (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)

$M_{00} = 7$ (the number of attributes where p was 0 and q was 0)

$M_{11} = 0$ (the number of attributes where p was 1 and q was 1)

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

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

J: same items bought → similar customers
SMC: same items *not* bought → similar customers

SMC vs. Jaccard

- 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

Take Home Messages

- Clustering groups similar objects
  - for analyzing the data at hand
- We know partitional and hierarchical clustering
- All clustering methods rely on distances
  - there are different distance functions
  - normalization is essential

Questions?