

Data Analysis and Knowledge Discovery

Finding Patterns

Jukka Heikkonen

University of Turku
Department of Information Technology

Jukka.Heikkonen@utu.fi

Outline

- 1 Clustering
- 2 Rule mining and deviation analysis

Finding patterns

- Patterns describe or summarize the data set or parts of it.
- Finding patterns is an exploratory data analysis task. There is no specific target attribute whose values should be predicted as in supervised learning.

Methods that have already been discussed in the context of data understanding:

- Visualisation techniques like scatter plots or MDS are methods for finding patterns by visual inspection.
- Correlation analysis can find patterns in the form of dependencies of pairs of variables.

Methods for finding patterns

Methods that will be discussed:

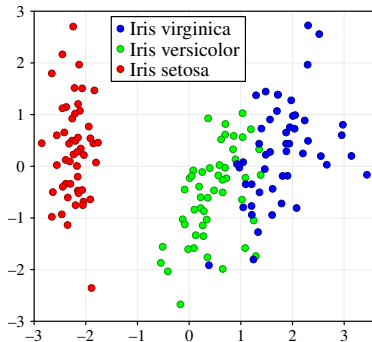
Cluster analysis. Identifying groups of “similar” data objects.

Association rules. Finding associations between attributes or typical combinations of values like

If *Demand=high* and *Supply=low* then *Price=high*.

Deviation analysis. Finding groups that deviate from the rest of the data.

Hierarchical clustering



In the two-dimensional MDS (Sammon mapping) representation of the Iris data set, two clusters can be identified. (The colours, indicating the species of the flowers, are ignored here.)

Hierarchical clustering

- **Hierarchical clustering** builds clusters step by step.
- Usually a bottom up strategy is applied by first considering each data object as a separate cluster and then step by step joining clusters together that are close to each other.
 - This approach is called **agglomerative hierarchical clustering**.
- In contrast to agglomerative hierarchical clustering, **divisive hierarchical clustering** starts with the whole data set as a single cluster and then divides clusters step by step into smaller clusters.

Hierarchical clustering

- In order to decide which data objects should belong to the same cluster, a (dis-)similarity measure is needed.
- All that is needed for hierarchical clustering is an $n \times n$ -matrix $[d_{i,j}]$, where $d_{i,j}$ is the dissimilarity of data objects i and j . (n is the number of data objects.)

Hierarchical clustering: Dissimilarity matrix

The dissimilarity matrix $[d_{i,j}]$ should at least satisfy the following conditions.

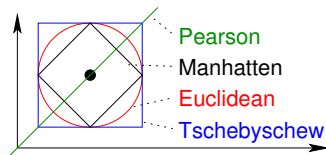
- $d_{i,j} \geq 0$, i.e. dissimilarity cannot be negative.
- $d_{i,i} = 0$, i.e. each data object is completely similar to itself.
- $d_{i,j} = d_{j,i}$, i.e. data object i is (dis-)similar to data object j to the same degree as data object j is (dis-)similar to data object i .

It is often useful if the dissimilarity is a (pseudo-)metric, satisfying also the

- **triangle inequality** $d_{i,k} \leq d_{i,j} + d_{j,k}$.

Notion of (dis-)similarity: Numerical attributes

Euclidean	L_2	$d_E(x, y) = \sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2}$
Manhattan	L_1	$d_M(x, y) = x_1 - y_1 + \dots + x_n - y_n $
Tschebyschew	L_∞	$d_\infty(x, y) = \max\{ x_1 - y_1 , \dots, x_n - y_n \}$
Minkowski	L_p	$d_p(x, y) = \left(\sqrt[p]{\sum_{i=1}^n x_i - y_i ^p} \right)^{\frac{1}{p}}$
Cosine		$d_C(x, y) = 1 - \frac{x^\top y}{\ x\ \ y\ }$
Tanimoto		$d_T(x, y) = \frac{x^\top y}{\ x\ ^2 + \ y\ ^2 - x^\top y}$
Pearson		Euclidean of z-score transformed \mathbf{x}, \mathbf{y}

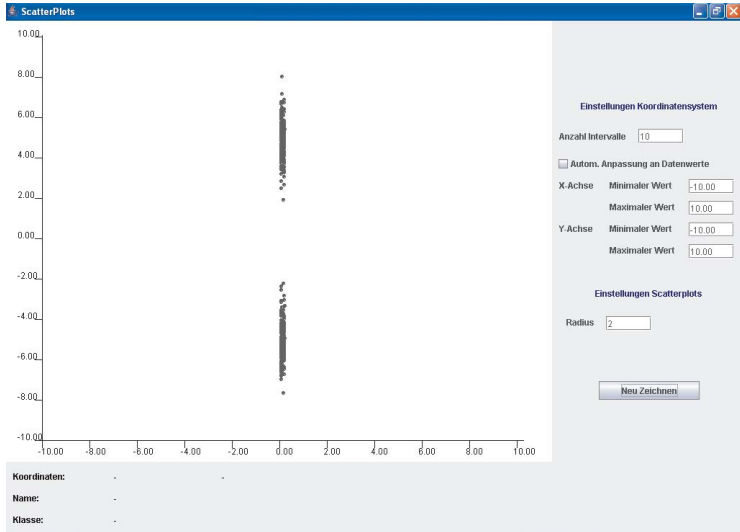


All 2-D points that have a distance of 1.0 from the small centre
bullet

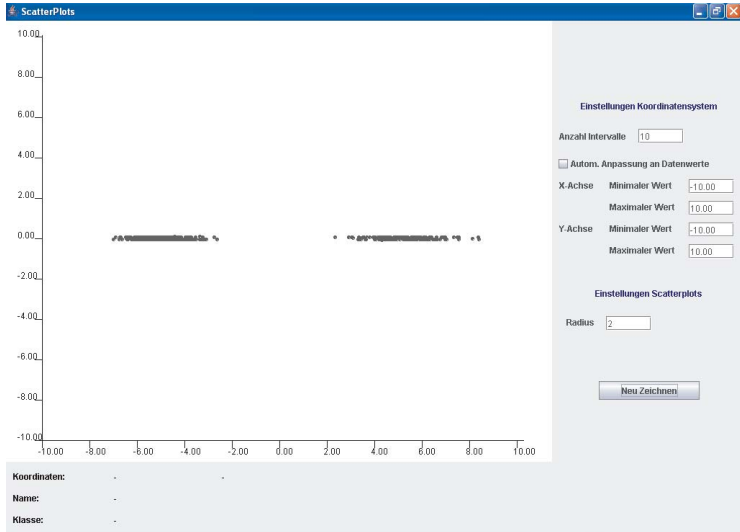
Clustering example: original data



Clustering example: x unit change



Clustering example: y unit change



Scaling

The previous three slides show the same data set.

- In the second slide, the unit on the x -axis was changed to centi-units.
- In the third slide, the unit on the y -axis was changed to centi-units.

Clusters should not depend on the measurement unit!

Therefore, some kind of normalisation (see lectures on data preparation) should be carried out before clustering or use appropriate nonisotropic distance measure.

Nonisotropic distances

- **Isotropic** means that the distance grows in all directions equally fast (e.g. Euclidean distance).
- **In nonisotropic** distances tries to capture inherent variations in variable values.
- One of the most common nonisotropic distances is Mahalanobis distance defined as

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})}$$

considers the variances across all variables. Here Σ is covariance matrix.

- Mahalanobis distance is scale-invariant.

Notion of (dis-)similarity: Binary attributes

The two values (e.g. 0 and 1) of a binary attribute can be interpreted as some property being absent (0) or present (1).

In this sense, a vector of binary attribute can be interpreted as a set of properties that the corresponding object has.

Example.

- The binary vector $(0, 1, 1, 0, 1)$ corresponds to the set of properties $\{a_2, a_3, a_5\}$.
- The binary vector $(0, 0, 0, 0, 0)$ corresponds to the empty set.
- The binary vector $(1, 1, 1, 1, 1)$ corresponds to the set $\{a_1, a_2, a_3, a_4, a_5\}$.

Notion of (dis-)similarity: Binary attributes

Dissimilarity measures for two vectors of binary attributes.

	binary attributes	sets of properties
simple match	$d_S = 1 - \frac{b+n}{b+n+x}$	
Russel & Rao	$d_R = 1 - \frac{b}{b+n+x}$	$1 - \frac{ X \cap Y }{ \Omega }$
Jaccard	$d_J = 1 - \frac{b}{b+x}$	$1 - \frac{ X \cap Y }{ X \cup Y }$
Dice	$d_D = 1 - \frac{2b}{2b+x}$	$1 - \frac{2 X \cap Y }{ X + Y }$

no. of predicates that...

$b =$...hold in both records

$n =$...do not hold in both records

$x =$...hold in only one of both records

x	y	set X	set Y	b	n	x	d_M	d_R	d_J	d_D
101000	111000	$\{a_1, a_3\}$	$\{a_1, a_2, a_3\}$	2	3	1	0.1 $\bar{6}$	0.6 $\bar{6}$	0.3 $\bar{3}$	0.20

Notion of (dis-)similarity: Nominal attributes

- Nominal attributes may be transformed into a set of binary attributes, each of them indicating one particular feature of the attribute.
- Only one of the introduced binary attributes may be active at a time.

Example. . Attribute *Manufacturer* with the values *BMW*, *Chrysler*, *Dacia*, *Ford*, *Volkswagen*.

manufacturer	...		binary vector
Volkswagen	...	→	00001
Dacia	...		01000
Ford	...		00100

Then one of the dissimilarity measures for binary attribute can be applied.

Agglomerative hierarchical clustering: Algorithm

Input: $n \times n$ dissimilarity matrix $[d_{i,j}]$.

- 1 Start with n clusters, each data objects forms a single cluster.
- 2 Reduce the number of clusters by joining those two clusters that are most similar (least dissimilar).
- 3 Repeat step 3 until there is only one cluster left containing all data objects.

Measuring dissimilarity between clusters

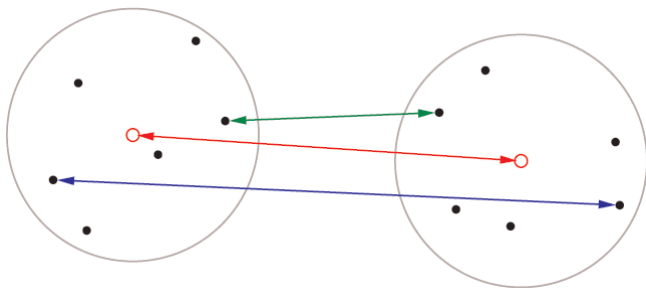
- The dissimilarity between two clusters containing only one data objects each is simply the dissimilarity of the two data objects specified in the dissimilarity matrix $[d_{i,j}]$.
- How do we compute the dissimilarity between clusters that contain more than one data object?

Measuring dissimilarity between clusters

- **Centroid** (red)
Distance between the centroids (mean value vectors) of the two clusters.
- **Average Linkage**
Average dissimilarity between two points of the two clusters.
- **Single Linkage** (green)
Dissimilarity between the two most similar data objects of the two clusters.
- **Complete Linkage** (blue)
Dissimilarity between the two most dissimilar data objects of the two clusters.

Measuring dissimilarity between clusters

- Red: Centroid
- Green: Single Linkage
- Blue: Complete Linkage

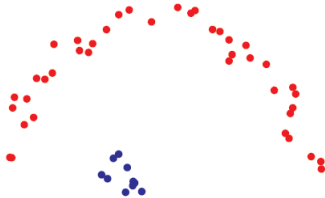


Measuring dissimilarity between clusters

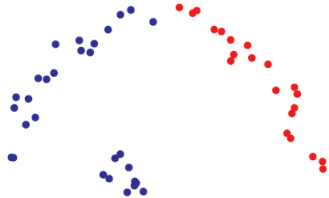
Some typical observations about dissimilarity measures:

- Single linkage can “follow chains” in the data (may be desirable in certain applications).
- Complete linkage leads to very compact clusters.
- Average linkage also tends clearly towards compact clusters.

Measuring dissimilarity between clusters



Single linkage



Complete linkage

Measuring dissimilarity between clusters

Ward's method is another strategy for merging clusters.

In contrast to single, complete or average linkage, it takes the number of data objects in each cluster into account.

Measuring dissimilarity between clusters

The updated dissimilarity between the newly formed cluster $\{\mathcal{C} \cup \mathcal{C}'\}$ and the cluster \mathcal{C}'' is computed in the following way.

Here $d'(\mathcal{C}, \mathcal{C}') = \min\{d(x, y) | x \in \mathcal{C}, y \in \mathcal{C}'\}$, i.e., the distance to a cluster is equivalent to the distance of the nearest point of the cluster.

$$d'(\{\mathcal{C} \cup \mathcal{C}'\}, \mathcal{C}'') = \dots$$

$$\text{single linkage} = \min\{d'(\mathcal{C}, \mathcal{C}''), d'(\mathcal{C}', \mathcal{C}'')\}$$

$$\text{complete linkage} = \max\{d'(\mathcal{C}, \mathcal{C}''), d'(\mathcal{C}', \mathcal{C}'')\}$$

$$\text{average linkage} = \frac{|\mathcal{C}|d'(\mathcal{C}, \mathcal{C}'') + |\mathcal{C}'|d'(\mathcal{C}', \mathcal{C}'')}{|\mathcal{C}| + |\mathcal{C}'|}$$

$$\text{Ward} = \frac{(|\mathcal{C}| + |\mathcal{C}''|)d'(\mathcal{C}, \mathcal{C}'') + (|\mathcal{C}'| + |\mathcal{C}''|)d'(\mathcal{C}', \mathcal{C}'') - |\mathcal{C}''|d'(\mathcal{C}, \mathcal{C}')}{|\mathcal{C}| + |\mathcal{C}'| + |\mathcal{C}''|}$$

$$\text{centroid (metric)} = \frac{1}{|\mathcal{C} \cup \mathcal{C}'| |\mathcal{C}''|} \sum_{x \in \mathcal{C} \cup \mathcal{C}'} \sum_{y \in \mathcal{C}''} d(\mathbf{x}, \mathbf{y})$$

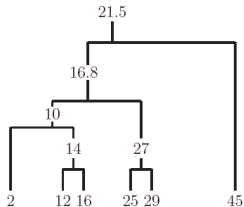
Dendrograms

- The cluster merging process arranges the data points in a binary tree.
- Draw the data tuples at the bottom or on the left (equally spaced if they are multi-dimensional).
- Draw a connection between clusters that are merged, with the distance to the data points representing the distance between the clusters.

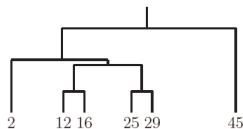
Hierarchical clustering

- Example: Clustering of the 1-dimensional data set $\{2, 12, 16, 25, 29, 45\}$.
- All three approaches to measure the distance between clusters lead to different dendrograms.

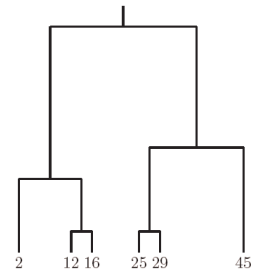
Hierarchical clustering



Centroid

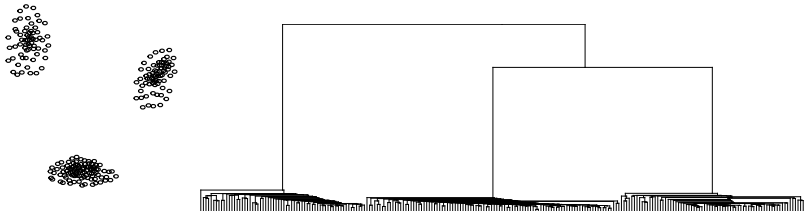


Single linkage

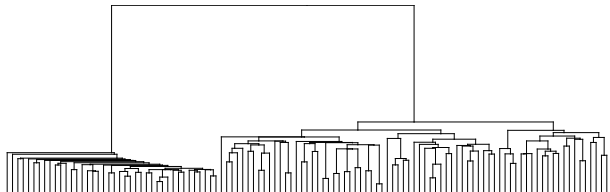
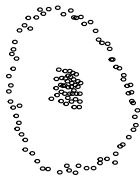


Complete linkage

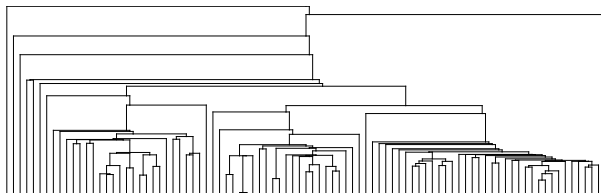
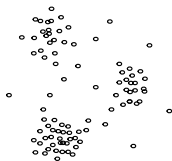
Dendrograms



Dendrograms



Dendrograms



Choosing the clusters

- **Simplest Approach:**

- Specify a minimum desired distance between clusters.
- Stop merging clusters if the closest two clusters are farther apart than this distance.

- **Visual Approach:**

- Merge clusters until all data points are combined into one cluster.
- Draw the dendrogram and find a good cut level.
- Advantage: Cut need not be strictly horizontal.

Choosing the clusters

- **More Sophisticated Approaches:**

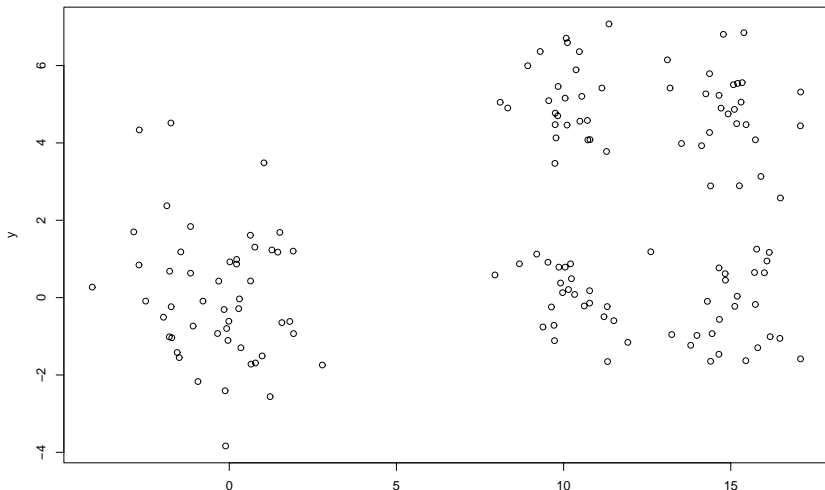
- Analyze the sequence of distances in the merging process.
- Try to find a step in which the distance between the two clusters merged is considerably larger than the distance of the previous step.
- Several heuristic criteria exist for this step selection.

Heatmaps

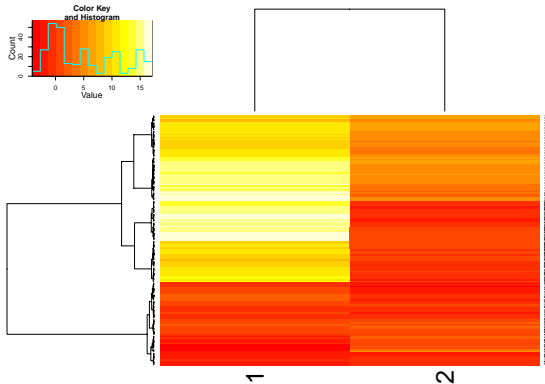
A **heatmap** combines

- a dendrogram resulting from clustering the data,
- a dendrogram resulting from clustering the attributes and
- colours to indicate the values of the attributes.

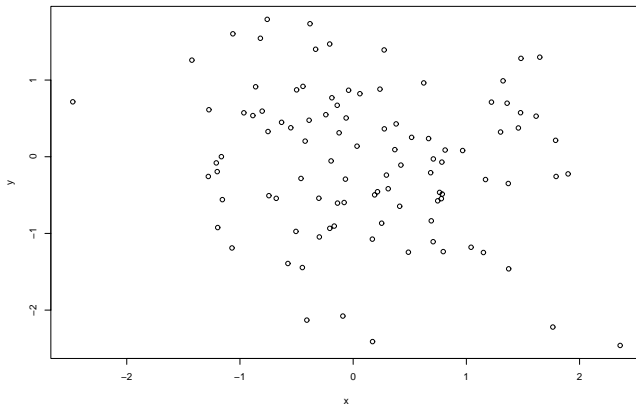
Hierarchical clustering



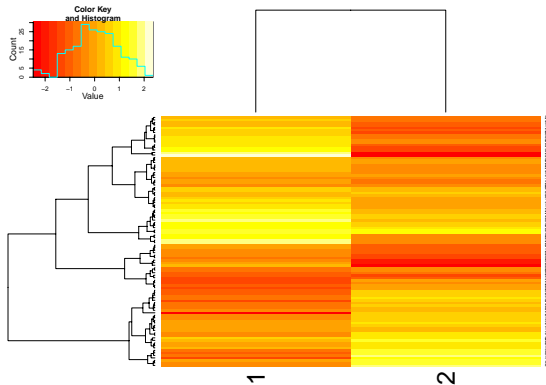
Heatmap and dendrogram



Hierarchical clustering



Heatmap and dendrogram

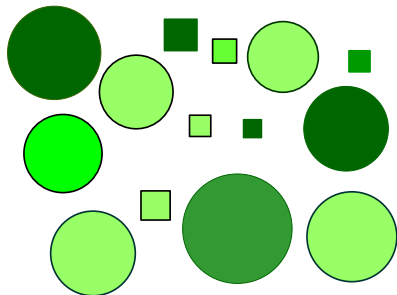


Divisive hierarchical clustering

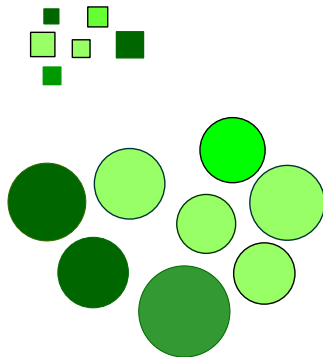
The top-down approach of divisive hierarchical clustering is seldom used.

- In agglomerative clustering the minimum of the pairwise dissimilarities has to be determined, leading to a quadratic complexity in each step (quadratic in the number of clusters still present in the corresponding step).
- In divisive clustering for each cluster all possible splits would have to be considered.
- In the first step, there are $2^{n-1} - 1$ possible splits, where n is the number of data objects.

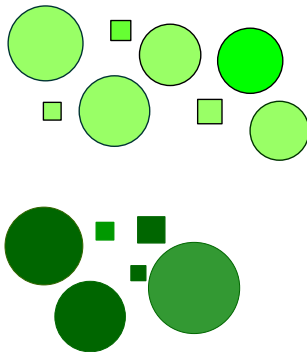
How to cluster these objects?



How to cluster these objects?



How to cluster these objects?



k -Means clustering

- Choose a number k of clusters to be found (user input).
- Initialize the cluster centres randomly
(for instance, by randomly selecting k data points).
- **Data point assignment:**
Assign each data point to the cluster centre that is closest to it (i.e. closer than any other cluster centre).
- **Cluster centre update:**
Compute new cluster centres as the mean vectors of the assigned data points. (Intuitively: centre of gravity if each data point has unit weight.)

k -Means clustering

- Repeat these two steps (data point assignment and cluster centre update) until the clusters centres do not change anymore.
- It can be shown that this scheme must converge, i.e., the update of the cluster centres cannot go on forever.

k -Means clustering

Aim: Minimize the objective function

$$f = \sum_{i=1}^k \sum_{j=1}^n u_{ij} d_{ij}$$

under the constraints $u_{ij} \in \{0, 1\}$ and

$$\sum_{i=1}^k u_{ij} = 1 \quad \text{for all } j = 1, \dots, n$$

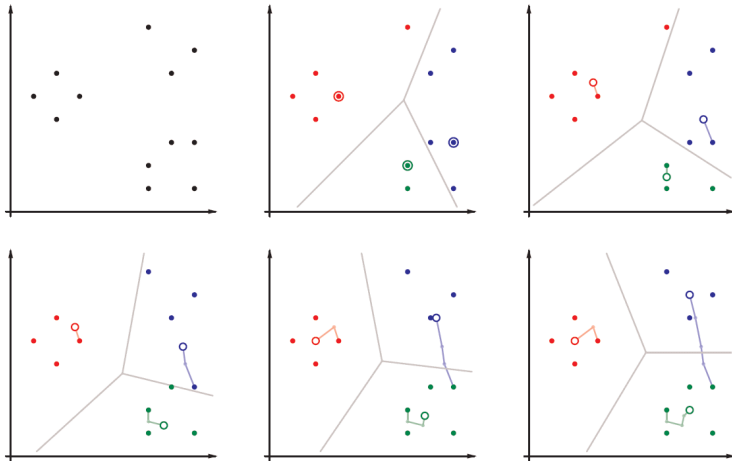
meaning that each data object can only belong to a one cluster.

Alternating optimization

- Assuming the cluster centres to be fixed, $u_{ij} = 1$ should be chosen for the cluster i to which data object x_j has the smallest distance in order to minimize the objective function.
- Assuming the assignments to the clusters to be fixed, each cluster centre should be chosen as the mean vector of the data objects assigned to the cluster in order to minimize the objective function.

This is a greedy algorithm.

k-Means clustering: Example

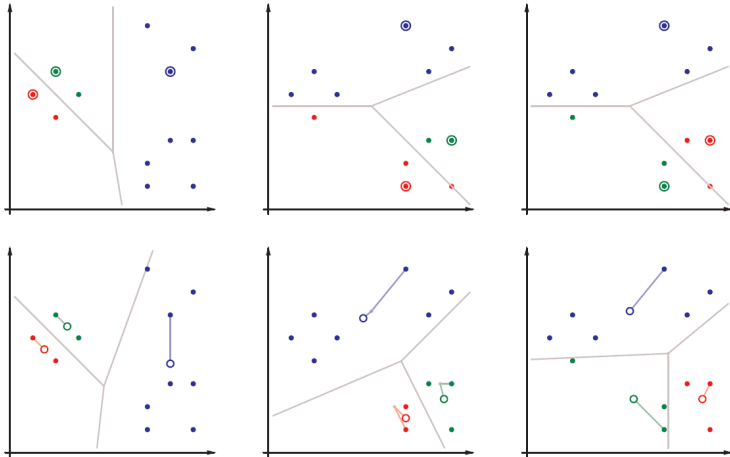


k -Means clustering: Local minima and k

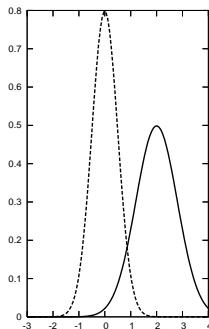
- Clustering is successful in this example:
The clusters found are those that would have been formed intuitively.
- Convergence is achieved after only 5 steps.
(This is typical: convergence is usually very fast.)
- However: The clustering result is fairly **sensitive to the initial positions** of the cluster centres.
- With a bad initialisation clustering may fail
(the alternating update process gets stuck in a local minimum).
- Optimal k can be estimated by methods shown in the book, but many times is selected manually by checking the meanings of clusters.

k -Means clustering: Local minima

Three examples of bad initialization

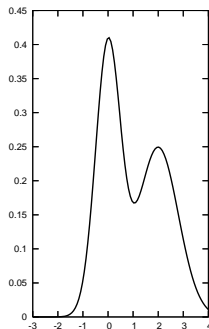


Gaussian mixture models



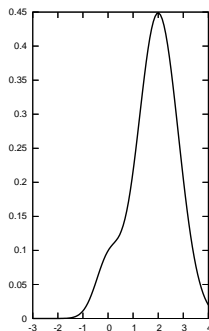
Two normal distributions

Gaussian mixture models



Mixture model (both normal distributions contribute 50%)

Gaussian mixture models



Mixture model (one normal distributions contributes 10%, the other 90%)

Gaussian mixture models – EM clustering

- **Assumption:** Data were generated by sampling a set of normal distributions.
(The probability density is a mixture of normal distributions.)
- **Aim:** Find the parameters for the normal distributions and how much each normal distribution contributes to the data.
- **Algorithm:** EM clustering (expectation maximisation).
Alternating scheme in which the parameters of the normal distributions and the likelihoods of the data points to be generated by the corresponding normal distributions are estimated.

GMM and EM

GMM is based on the overall density (mixture distribution) given by

$$p(x) = \sum_{i=1}^c p(x|i)P(i) ,$$

where $P(i)$ is aprior probability that x comes distribution $p(x|i)$.

If we assume that

$$p(x|i) = \frac{1}{(2\pi\sigma_i^2)^{d/2}} e^{-\frac{\|x-\mu_i\|^2}{2\sigma_i^2}}$$

we can derive an iterative Maximum Likelihood solution for estimating μ_i, σ_i , and $P(i)$.

GMM and EM

The following ML estimates can be derived

$$\mu_i = \frac{\sum_{j=1}^n p(i|x_j) x_j}{\sum_{j=1}^n p(i|x_j)}$$

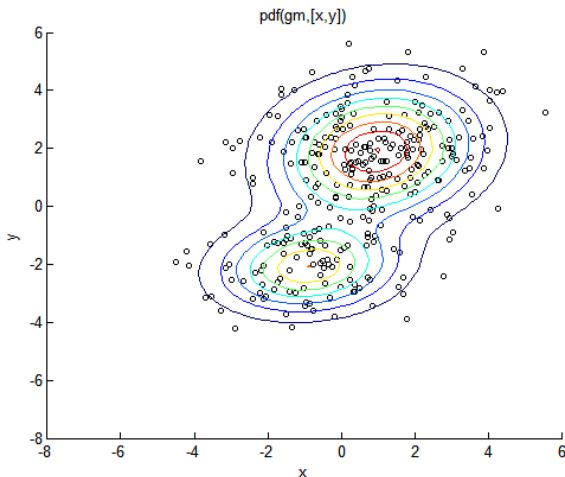
$$\sigma_i = \sqrt{\frac{1}{d} \frac{\sum_{j=1}^n p(i|x_j) \|x_j - \mu_i\|^2}{\sum_{j=1}^n p(i|x_j)}}$$

$$P(i) = \frac{1}{n} \sum_{j=1}^n p(i|x_j) .$$

Iterate until convergence.

This is also known as Expectation Maximization (EM) algorithm and can be easily extended to full covariance case.

Gaussian mixture models



Density-based clustering

For numerical data, [density-based clustering algorithm](#) can be also considered.

Principle: A connected region with high data density corresponds to one cluster.

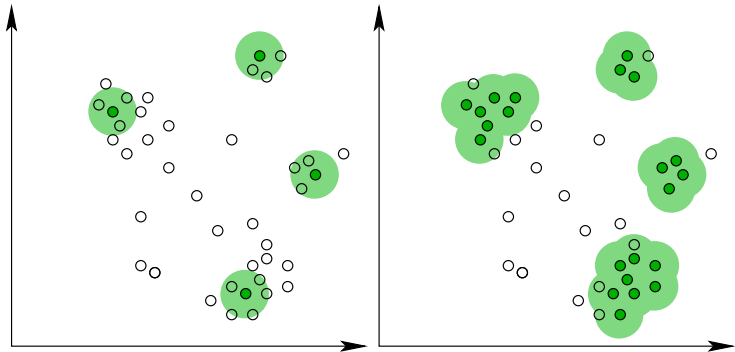
[DBScan](#) is one of the density-based clustering algorithms.

Density-based clustering: DBScan


Principle idea of DBScan:

- 1 Find a data point where the data density is high, i.e. in whose ε -neighbourhood are at least ℓ other points. (ε and ℓ are parameters of the algorithm to be chosen by the user.)
- 2 All the points in the ε -neighbourhood are considered to belong to one cluster.
- 3 Expand this ε -neighbourhood (the cluster) as long as the high density criterion is satisfied.
- 4 Remove the cluster (all data points assigned to the cluster) from the data set and continue with 1. as long as data points with a high data density around them can be found.

Density-based clustering: DBScan



 neighbourhood cell
with at least 3 hits

 neighbourhood cell
with at least 3 hits

Association rule mining

- Association rule induction: Originally designed for **market basket analysis**.
- Aims at finding patterns in the shopping behaviour of customers of supermarkets, mail-order companies, on-line shops etc.
- More specifically:
Find sets of products that are frequently bought together.
- Example of an association rule:
*If a customer buys bread and wine,
then she/he will probably also buy cheese.*

Association rule mining

- Possible applications of found association rules:
 - Improve arrangement of products in shelves, on a catalog's pages.
 - Support of cross-selling (suggestion of other products), product bundling.
 - Fraud detection, technical dependence analysis.
 - Finding business rules and detection of data quality problems.
 - ...

Association rules

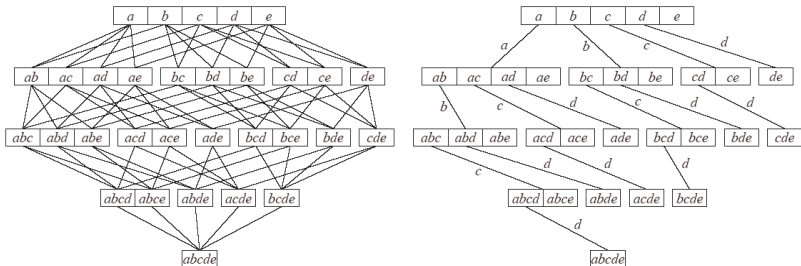
- Assessing the quality of association rules:
 - **Support of an item set:**
Proportion of transactions (shopping baskets/carts) that contain the item set.
 - **Support of an association rule $X \rightarrow Y$:**
Either: Support of $X \cup Y$:
 (more common: rule is correct)
 Or: Support of X
 (more plausible: rule is applicable)
 - **Confidence of an association rule $X \rightarrow Y$:**
Support of $X \cup Y$ divided by support of X (estimate of $P(Y | X)$).

Association rules

- Two step implementation of the search for association rules:
 - Find the **frequent item sets** (also called *large item sets*), i.e., the item sets that have at least a user-defined **minimum support**.
 - Form rules using the frequent item sets found and select those that have at least a user-defined **minimum confidence**.

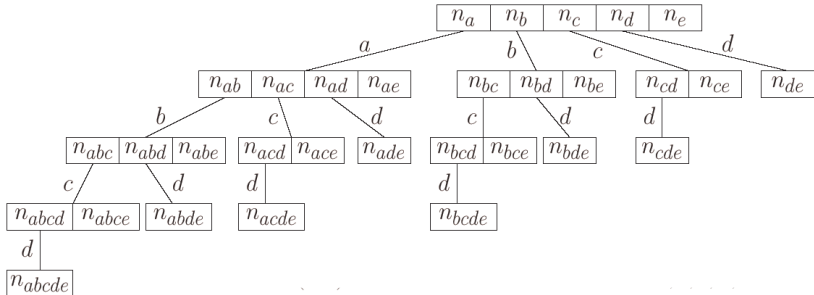
Finding frequent item sets

Subset lattice and a prefix tree for five items:



- It is not possible to determine the support of all possible item sets, because their number grows exponentially with the number of items.
- Efficient methods to search the subset lattice are needed.

Item set trees



A (full) item set tree for the five items a , b , c , d , and e .

- Based on a global order of the items.
- The item sets counted in a node consist of
 - all items labeling the edges to the node (common prefix) and
 - one item following the last edge label.

Item set tree pruning

In applications item set trees tend to get very large, so pruning is needed.

- **Structural Pruning:**

- Make sure that there is only one counter for each possible item set.
- Explains the unbalanced structure of the full item set tree.

- **Size Based Pruning:**

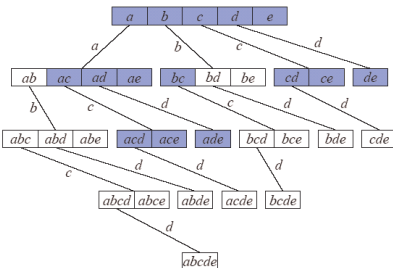
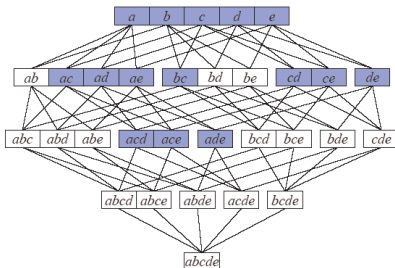
- Prune the tree if a certain depth (a certain size of the item sets) is reached.
- Idea: Rules with too many items are difficult to interpret.

- **Support Based Pruning:**

- **No superset of an infrequent item set can be frequent.**
- No counters for item sets having an infrequent subset are needed.

Searching the subset lattice

Boundary between frequent (blue) and infrequent (white) item sets:



- **Apriori**: Breadth-first search (item sets of same size).
- **Eclat**: Depth-first search (item sets with same prefix).

Apriori: Breadth first search

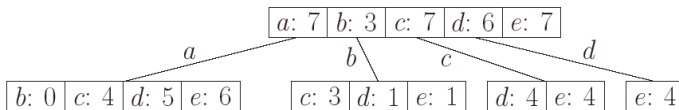
- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{c, b, e\}$
- 10: $\{a, d, e\}$

a: 7	b: 3	c: 7	d: 6	e: 7
------	------	------	------	------

- Example transaction database with 5 items and 10 transactions.
- Minimum support: 30%, i.e., at least 3 transactions must contain the item set.
- All one item sets are frequent → full second level is needed.

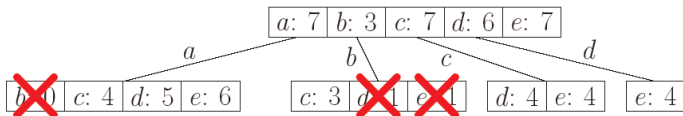
Apriori: Breadth first search

- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{c, b, e\}$
- 10: $\{a, d, e\}$



- Determining the support of item sets: For each item set traverse the database and count the transactions that contain it (highly inefficient).
- Better: Traverse the tree for each transaction and find the item sets it contains (efficient: can be implemented as a simple double recursive procedure).

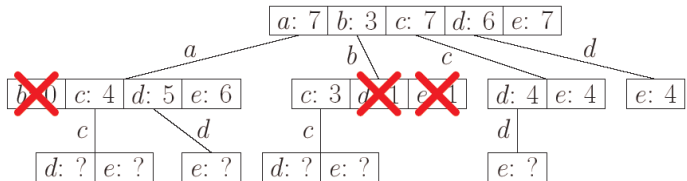
Apriori: Breadth first search

1: $\{a, d, e\}$ 2: $\{b, c, d\}$ 3: $\{a, c, e\}$ 4: $\{a, c, d, e\}$ 5: $\{a, e\}$ 6: $\{a, c, d\}$ 7: $\{b, c\}$ 8: $\{a, c, d, e\}$ 9: $\{c, b, e\}$ 10: $\{a, d, e\}$ 

- Minimum support: 30%, i.e., at least 3 transactions must contain the item set.
- Infrequent item sets: $\{a, b\}$, $\{b, d\}$, $\{b, e\}$.
- The subtrees starting at these item sets can be pruned.

Apriori: Breadth first search

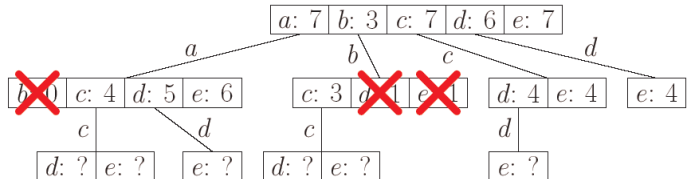
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Generate candidate item sets with 3 items (parents must be frequent).

Apriori: Breadth first search

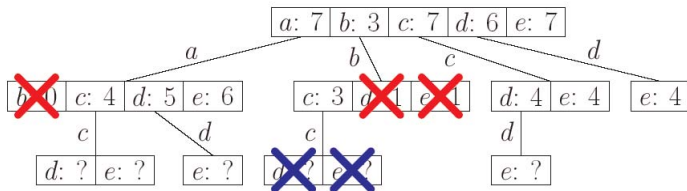
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Before counting, check whether the candidates contain an infrequent item set.
 - An item set with k items has k subsets of size $k - 1$.
 - The parent is only one of these subsets.

Apriori: Breadth first search

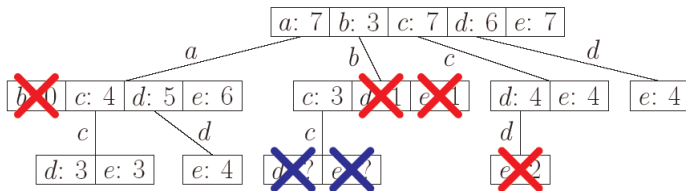
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- The item sets {b, c, d} and {b, c, e} can be pruned, because
 - {b, c, d} contains the infrequent item set {b, d} and
 - {b, c, e} contains the infrequent item set {b, e}.
- Only the remaining four item sets of size 3 are evaluated.

Apriori: Breadth first search

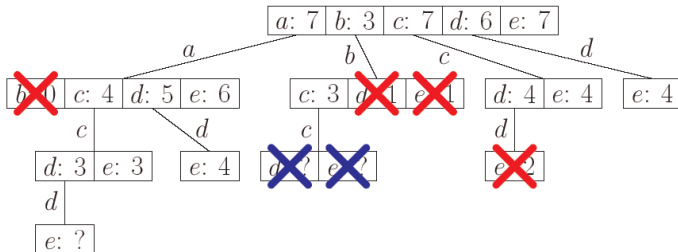
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Minimum support: 30%, i.e., at least 3 transactions must contain the item set.
- Infrequent item set: {c, d, e}.

Apriori: Breadth first search

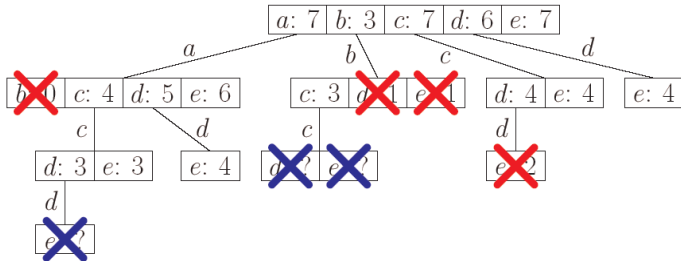
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Generate candidate item sets with 4 items (parents must be frequent).
- Before counting, check whether the candidates contain an infrequent item set.

Apriori: Breadth first search






- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- The item set {a, c, d, e} can be pruned, because it contains the infrequent item set {c, d, e}.
- Consequence: No candidate item sets with four items.
- Fourth access to the transaction database is not necessary.

Eclat: Depth first search

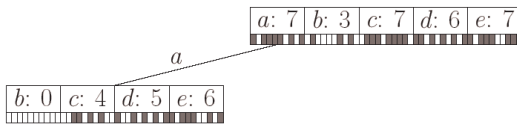
- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{c, b, e\}$
- 10: $\{a, d, e\}$

a: 7	b: 3	c: 7	d: 6	e: 7
				

- Form a transaction list for each item. Here: bit vector representation.
 - grey: item is contained in transaction
 - white: item is not contained in transaction
- Transaction database is needed only once (for the single item transaction lists).

Eclat: Depth first search

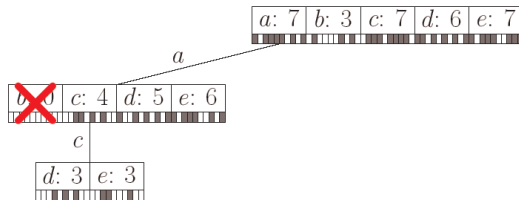
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Intersect the transaction list for item *a* with the transaction lists of all other items.
- Count the number of set bits (containing transactions).
- The item set {*a*, *b*} is infrequent and can be pruned.

Eclat: Depth first search

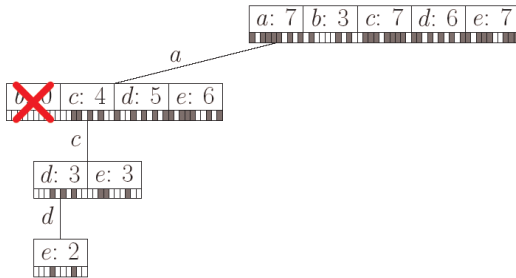
- 1: $\{a, d, e\}$
- 2: $\{b, c, d\}$
- 3: $\{a, c, e\}$
- 4: $\{a, c, d, e\}$
- 5: $\{a, e\}$
- 6: $\{a, c, d\}$
- 7: $\{b, c\}$
- 8: $\{a, c, d, e\}$
- 9: $\{c, b, e\}$
- 10: $\{a, d, e\}$



- Intersect the transaction list for $\{a, c\}$ with the transaction lists of $\{a, x\}$, $x \in \{d, e\}$.
- Result: Transaction lists for the item sets $\{a, c, d\}$ and $\{a, c, e\}$.

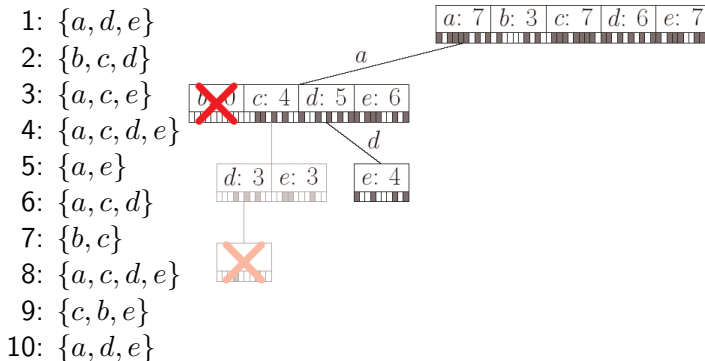
Eclat: Depth first search

- 10: $\{a, d, e\}$



- Intersect the transaction list for $\{a, c, d\}$ and $\{a, c, e\}$.
- Result: Transaction list for the item set $\{a, c, d, e\}$.
- With Apriori this item set could be pruned before counting, because it was known that $\{c, d, e\}$ is infrequent.

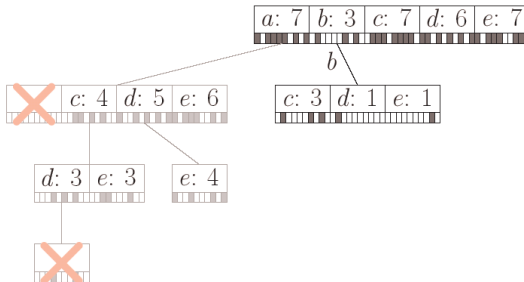
Eclat: Depth first search



- Backtrack to the second level of the search tree and intersect the transaction list for {a, d} and {a, e}.
- Result: Transaction list for {a, d, e}.

Eclat: Depth first search

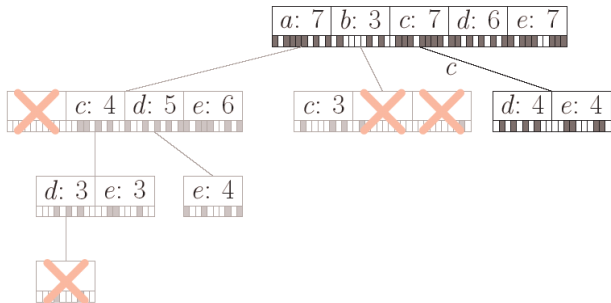
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Backtrack to the first level of the search tree and intersect the transaction list for *b* with the transaction lists for *c*, *d*, and *e*.
- Result: Transaction lists for the item sets {*b*, *c*}, {*b*, *d*}, and {*b*, *e*}.
- Only one item set with sufficient support → prune all subtrees.

Eclat: Depth first search

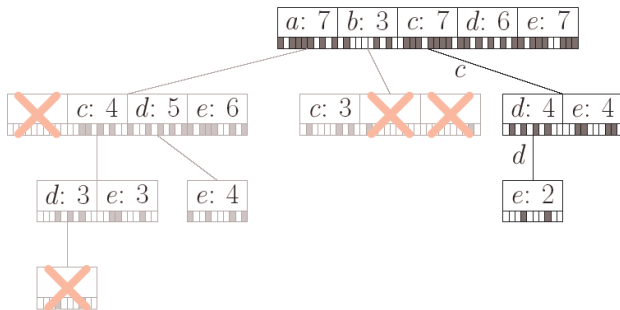
- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Backtrack to the first level of the search tree and intersect the transaction list for *c* with the transaction lists for *d* and *e*.
- Result: Transaction lists for the item sets {*c*, *d*} and {*c*, *e*}.

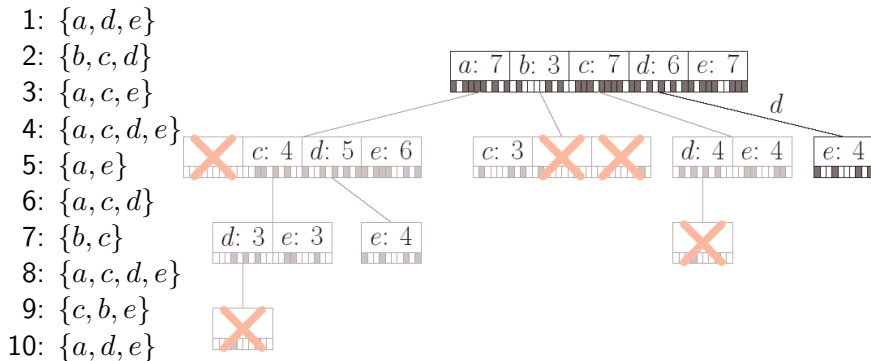
Eclat: Depth first search

- 1: {a, d, e}
- 2: {b, c, d}
- 3: {a, c, e}
- 4: {a, c, d, e}
- 5: {a, e}
- 6: {a, c, d}
- 7: {b, c}
- 8: {a, c, d, e}
- 9: {c, b, e}
- 10: {a, d, e}



- Intersect the transaction list for {c, d} and {c, e}.
- Result: Transaction list for {c, d, e}.
- Infrequent item set: {c, d, e}.

Eclat: Depth first search



- Backtrack to the first level of the search tree and intersect the transaction list for d with the transaction list for e .
- Result: Transaction list for the item set $\{d, e\}$.
- With this step the search is finished.

Frequent item sets

1 item	2 items	3 items
$\{a\}^+$: 70%	$\{a, c\}^+$: 40% $\{c, e\}^+$: 40%	$\{a, c, d\}^{+*}$: 30%
$\{b\}$: 30%	$\{a, d\}^+$: 50% $\{d, e\}$: 40%	$\{a, c, e\}^{+*}$: 30%
$\{c\}^+$: 70%	$\{a, e\}^+$: 60%	$\{a, d, e\}^{+*}$: 40%
$\{d\}^+$: 60%	$\{b, c\}^{+*}$: 30%	
$\{e\}^+$: 70%	$\{c, d\}^+$: 40%	

Types of frequent item sets

- **Free Item Set:** Any frequent item set (support is higher than the minimal support).
- **Closed Item Set** (marked with $^+$): A frequent item set is called *closed* if no superset has the same support.
- **Maximal Item Set** (marked with *): A frequent item set is called *maximal* if no superset is frequent.

Generating association rules

For each frequent item set S :

- Consider all pairs of sets $X, Y \in S$ with $X \cup Y = S$ and $X \cap Y = \emptyset$. Common restriction: $|Y| = 1$, i.e. only one item in consequent (then-part).
- Form the association rule $X \rightarrow Y$ and compute its confidence.

$$\text{conf}(X \rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)} = \frac{\text{supp}(S)}{\text{supp}(X)}$$

- Report rules with a confidence higher than the minimum confidence.

Generating association rules

Example: $S = \{a, c, e\}$, $X = \{c, e\}$, $Y = \{a\}$.

$$\text{conf}(c, e \rightarrow a) = \frac{\text{supp}(\{a, c, e\})}{\text{supp}(\{c, e\})} = \frac{30\%}{40\%} = 75\%$$

Minimum confidence: 80%

association rule	support of all items	support of antecedent	confidence
$b \rightarrow c$:	30%	30%	100%
$d \rightarrow a$:	50%	60%	83.3%
$e \rightarrow a$:	60%	70%	85.7%
$a \rightarrow e$:	60%	70%	85.7%
$d, e \rightarrow a$:	40%	40%	100%
$a, d \rightarrow e$:	40%	50%	80%

1 item	2 items		3 items
$\{a\}^+$: 70%	$\{a, c\}^+$: 40%	$\{c, e\}^+$: 40%	$\{a, c, d\}^{+*}$: 30%
$\{b\}$: 30%	$\{a, d\}^+$: 50%	$\{d, e\}$: 40%	$\{a, c, e\}^{+*}$: 30%
$\{c\}^+$: 70%	$\{a, e\}^+$: 60%		$\{a, d, e\}^{+*}$: 40%
$\{d\}^+$: 60%	$\{b, c\}^{+*}$: 30%		
$\{e\}^+$: 70%	$\{c, d\}^+$: 40%		

Summary association rules

- **Association Rule Induction is a Two Step Process**

- Find the frequent item sets (minimum support).
- Form the relevant association rules (minimum confidence).

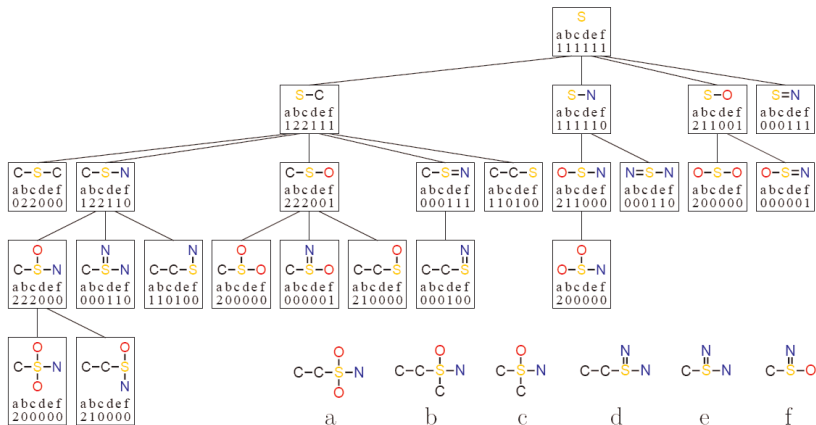
- **Finding the Frequent Item Sets**

- Top-down search in the subset lattice / item set tree.
- Apriori: Breadth first search; Eclat: Depth first search.
- Other algorithms: FP-growth, H-Mine, LCM, Mafia, Relim etc.
- Search Tree Pruning: *No superset of an infrequent item set can be frequent.*

- **Generating the Association Rules**

- Form all possible association rules from the frequent item sets.
- Filter “interesting” association rules.

Finding frequent molecule substructures



Applications

- Finding business rules and detection of data quality problems.
 - Association rules with confidence close to 100% could be business rules.
 - Exceptions might be caused by data quality problems.
- Construction of partial classifiers.
 - Search for association rules with a given conclusion part.
 - If ..., then the customer probably buys the product.

Deviation analysis and subgroup discovery

- Frequent item set and association rule mining can be seen as a discovery of subgroups of the dataset that share common properties
 - e.g., buy a certain product much more often than the average customer.
- **Deviation analysis** is another form of **subgroup discovery**.
- The goal is to find subgroups of the populations that are statistically most interesting, that is,
 - they are as large as possible and
 - deviate from the whole population with respect to the property of interest as much as possible.

Example. 10% of customers in the database have bought product A, but 30% of customers in the subgroup *female & married* have bought product A.

Deviation analysis and subgroup discovery

The ingredients of deviation analysis are

- a target measure and a verification test serving as a filter for irrelevant or incidental patterns,
- a quality measure to rank subgroups (often the same as the target measure)and
- a search method that enumerates candidates subgroups systematically.

Deviation analysis and subgroup discovery

Example.

- Assume we are interested in identifying subgroups of churners in our customer database with N customers.
- Assume that a proportion of p_0 customers in the whole database are churners.
- Consider a subgroup (for instance *male & under 30*).
- Assume that in this subgroup the proportion of churners p .
- If p highly deviates from p_0 this seems to be an indicator for a subgroup with different (churn) behaviour.
- But how much must p deviate from p_0 in order to consider the effect as significant?

Deviation analysis and subgroup discovery

- One could apply Fisher's exact test (analysis of contingency tables where sample sizes are small) and use the p-value as an indicator (the lower the better).
- Another measure is the z-score

$$z = \frac{np_0 - np}{\sqrt{np_0(1-p_0)}} = \frac{\sqrt{n}(p - p_0)}{\sqrt{p_0(1-p_0)}},$$

the difference between the expected number of churners (np_0) in the subgroup and the true number of churners (np) in the subgroup, divided by the variance $\sqrt{np_0(1-p_0)}$ (of the underlying binomial distribution, assuming that the customers in the subgroup are picked randomly with replacement).

- Overall depending on the case there are many possible statistical tests, such as Chi2-test, weighted relative accuracy, etc...

Deviation analysis and subgroup discovery

- Since subgroups are to be described by a conjunction of features (such as *variable = value*), the number of possible subgroups on the syntactical level grows exponentially with the number of attributes and the number of possible values.
- Given an attribute A with possible values $\Omega_A = \{high, medium, low, absent\}$, we may form four conditions on identity (e.g., $A = high$), four on impurity (e.g., $A \neq medium$), and as A has ordinal scale even more features regarding the order (e.g. $A \leq low$ or $A \geq medium$).
- When constructing 11 features from A alone, another 9 attributes with 4 values would allow us to define

$$11^{10} \approx 25 \cdot 10^9$$

syntactically different subgroups.