Different Learning Scenarios

Supervised Learning

- A teacher provides the value for the target function for all training examples (labeled examples)
- concept learning, classification, regression

Reinforcement Learning

- The teacher only provides feedback but not example values

Semi-supervised Learning

- Only a subset of the training examples are labeled

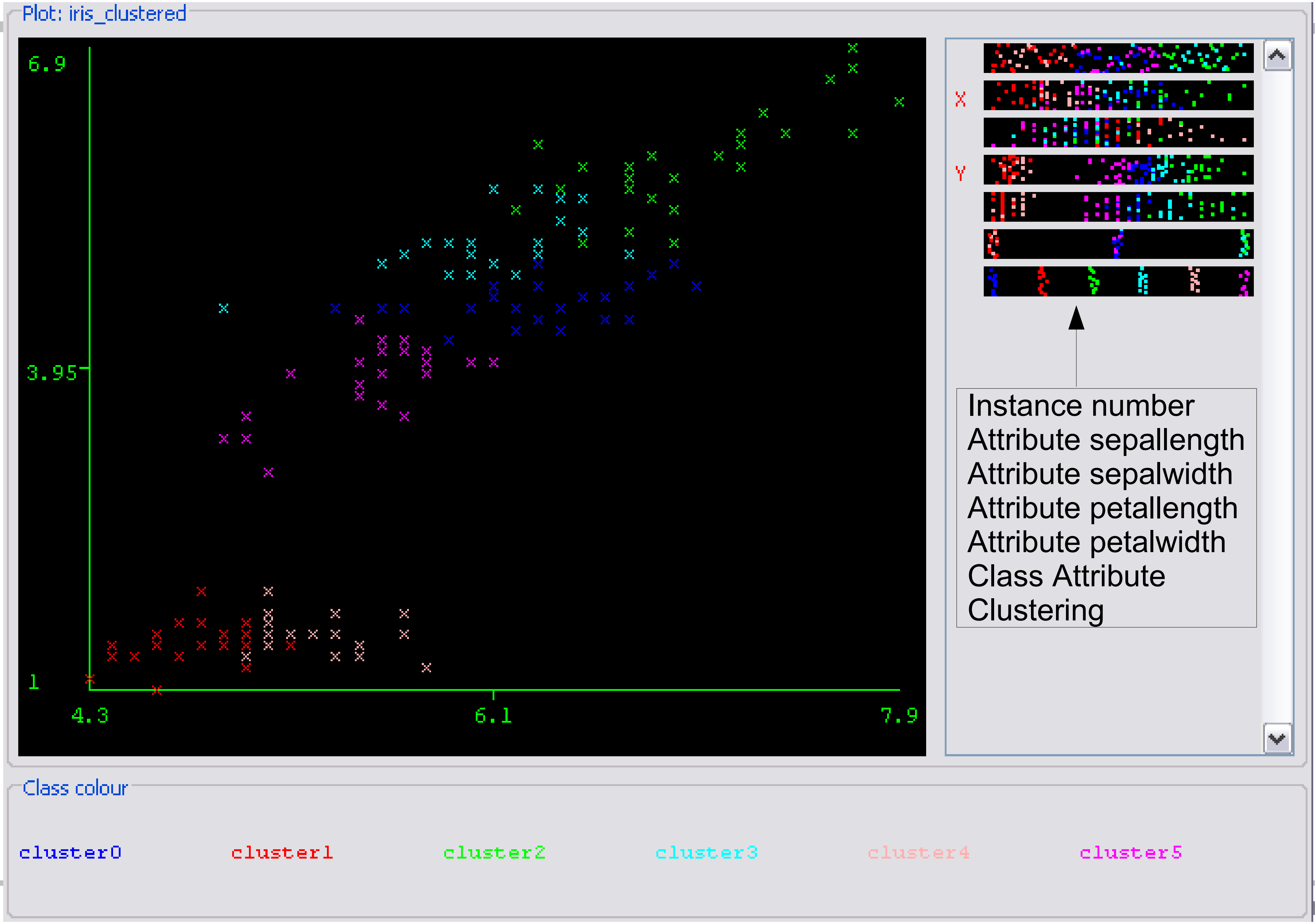
Unsupervised Learning

- There is no information except the training examples
- clustering, subgroup discovery, association rule discovery

Clustering

- Given:
  - a set of examples
  - in some description language (e.g., attribute-value)
  - no labels (-> unsupervised)
- Find:
  - a grouping of the examples into meaningful *clusters*
  - so that we have a high
    - **intra-class similarity:** similarity between objects in same cluster
    - **inter-class dissimilarity:** dissimilarity between objects in different clusters

6 clusters on Iris dataset



Clustering Algorithms

- k-means clustering
  - given a similarity metric (like k-NN algorithms)
  - initialize k cluster centers
  - iteratively assign examples to closest neighbor
  - until procedure converges
- bottom-up hierarchical clustering
  - each example is a cluster
  - iteratively merge clusters, similar to chi-merge
- Cobweb
  - incrementally build up a tree structure
  - each node/cluster can estimate a probability that an example belongs to this cluster
  - examples are sorted into the tree in a top-down way

Association Rule Discovery

- Association Rules describe frequent co-occurences in sets
  - an *itemset* is a subset A of all possible items I

- Example Problems:
  - Which products are frequently bought together by customers? (*Basket Analysis*)
    - DataTable = Receipts x Products (or Customer x Products)
    - Results could be used to change the placements of products in the market
  - Which courses tend to be attended together?
    - DataTable = Students x Courses
    - Results could be used to avoid scheduling conflicts....

Association Rules

- General Form:

$$A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m$$

- Interpretation:
  - When items A_i appear, items B_i also appear with a certain probability

- Examples:
  - **`Bread, Cheese → RedWine.`**
    Customers that buy bread and cheese, also tend to buy red wine.
  - **`MachineLearning → WebMining, MLPraktikum.`**
    Students that take 'Machine Learning' also take 'Web Mining' and the 'Machine Learning Praktikum'

Basic Quality Measures

- **Support** $support(A \to B) = support(A \cup B) = \frac{n(A \cup B)}{n}$
  - proportion of examples for which both the head and the body of the rule are true
  - How many examples does this rule cover?

$n(A \cup B)$ is the no. of customers that bought all items in item sets A and B.

If A and B are interpreted as logical conjuncts, this should be A∧B

- **Confidence** $confidence(A \to B) = \frac{support(A \cup B)}{support(A)} = \frac{n(A \cup B)}{n(A)}$
  - proportion of examples for which the head is true among those for which the body is true
  - How strong is the implication of the rule?

- Example:
  - **`Bread, Cheese => RedWine`** (S = 0.01, C = 0.8)
    80% of all customers that bought bread and cheese also bought red wine.
    1% of all customers bought all three items.

Learning Problem

Find all association rules with a given *minimum support* s_{min} and a given *minimum confidence* c_{min}

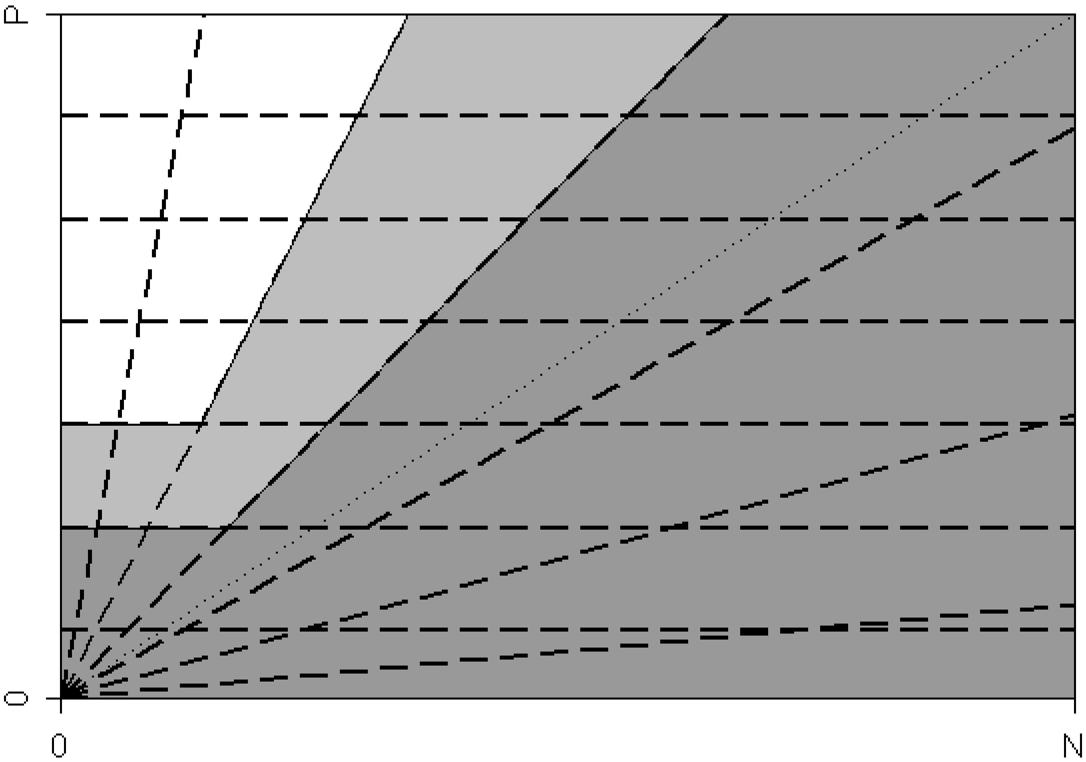
- Frequent itemsets:
  - An itemset A is *frequent* if $support(A) \geq s_{min}$
- Key Observation (*anti-monotonicity of support*):
  - Adding a condition (specializing the rule) may never increase support/freqency of a rule (or of its itemset).

$$C \subseteq D \Rightarrow support(C) \geq support(D)$$

  - Therefore:
    - an itemset can only be frequent if *all* of its subsets are freqent
    - all supersets of an infrequent itemset are also infrequent

Support/Confidence Filtering

- filter rules that
  - cover not enough positive examples ($p < s_{min}$)
  - are not precise enough ($h_{prec} < c_{min}$)
- *effects:*
  - all but a region around $(0,P)$ is filtered



Note: $P \triangleq$ examples for which head is true
$N \triangleq$ examples for which head is false

APRIORI Step1: FreqSet: Find all Frequent Itemsets

1. $k = 1$
2. $C_1 = I$ (all items)
3. while $C_k > \emptyset$
   (a) $S_k = C_k \setminus$ all infrequent itemsets in C_k ← check on data
   (b) $C_{k+1} =$ all sets with $k+1$ elements that can be formed by uniting of two itemsets in S_k
   (c) $C_{k+1} = C_{k+1} \setminus$ itemsets that do not have all subsets of size k in S_k
   (d) $S = S \cup S_k$
   (e) k++
4. return S

Candidate itemsets are stored in efficient data structures such as hash trees or tries.

Efficient Candidate Generation

- Formation of C_{k+1} (Step 3(b) of the algorithm):
  - combines two frequent k-itemsets to a candidate for a (k+1)-itemset
  - can be performed efficiently:

$$C_{k+1}=\{\langle X_1,\dots,X_{k-1},X_k,X_{k+1}\rangle \mid \langle X_1,\dots,X_{k-1},X_k\rangle\in S_k, \langle X_1,\dots,X_{k-1},X_{k+1},\rangle\in S_k, X_k<X_{k+1}\}$$

    - assumes items are ordered in some way (e.g., alphabetically)
    - will generate each itemset only once (sorted from X_1 to X_{k+1})
    - no candidate will be missed (anti-monotonicity of support)
- Pruning of C_{k+1} (Step 3(c) of the algorithm):
  - testing all k-item subsets of a $k+1$-itemset
  - generated by deleting each of the first k-1 conditions
  - delete the candidate set if not all k-item subsets are frequent (i.e., in S_k)

Example

	beer	chips	pizza	wine
customer 1	1	1	0	1
customer 2	1	1	0	0
customer 3	0	0	1	1
customer 4	0	1	1	0

- Find all itemsets with $s_{min} = 0.25$
  - C_1 = { {beer}, {chips}, {pizza}, {wine} }

    S_1 = { {beer}, {chips}, {pizza}, {wine} }
  - C_2 = { {beer, chips}, {beer, pizza}, {beer, wine}, {chips, pizza}, {chips, wine}, {pizza, wine} }

    S_2 = { {beer, chips}, {beer, wine}, {chips, pizza}, {chips, wine}, {pizza, wine} }
  - C_3 = { {beer, chips, wine}, {chips, pizza, wine} }

    S_3 = { {beer, chips, wine} }
  - $C_4 = \emptyset$

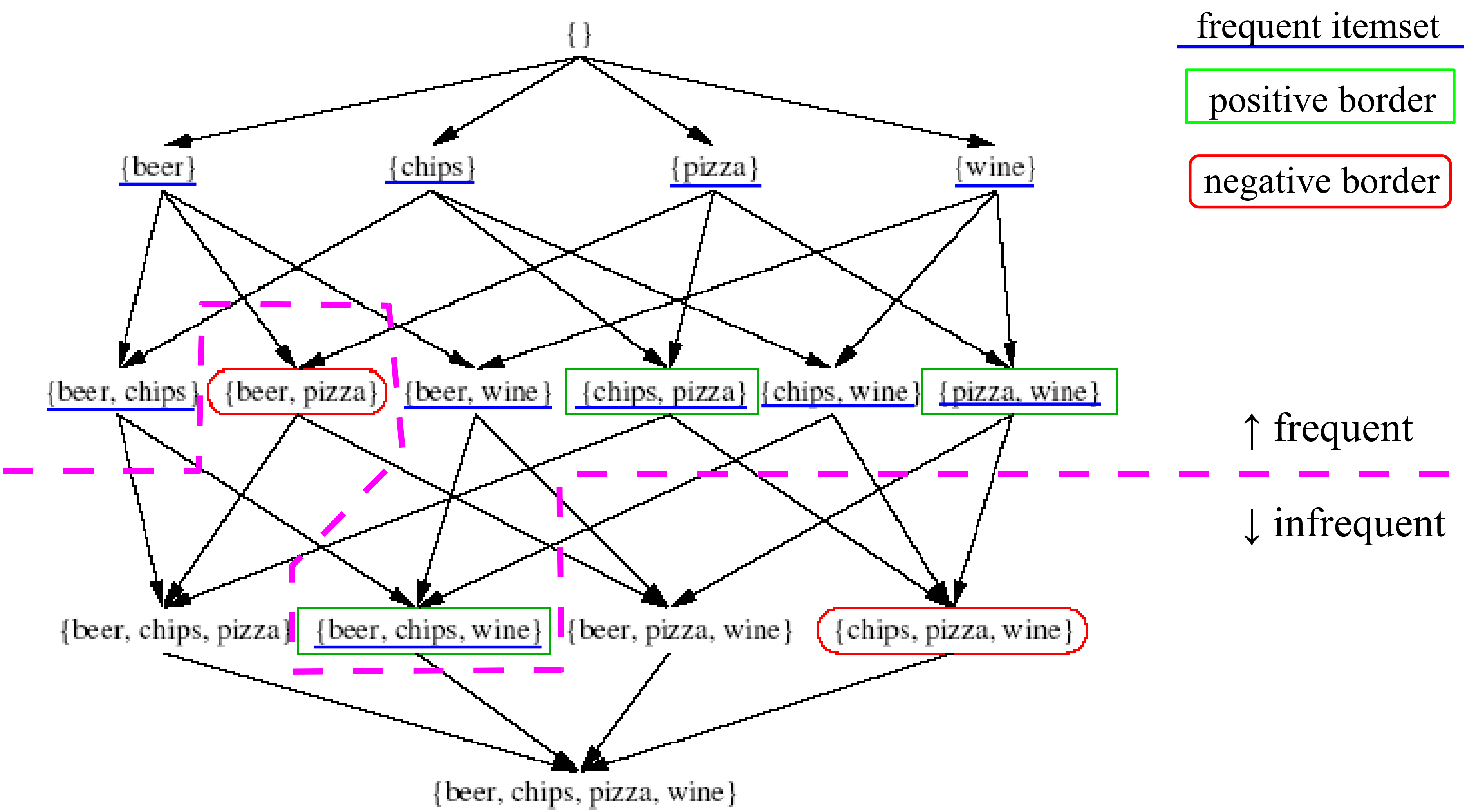
Search Space and Border

- Search Space:
  - The search space for frequent itemsets can be structured with the subset relationship

- Border:
  - The ***border*** are all itemsets for which
    - all subsets are frequent
    - no superset is frequent
  - positive border:
    - elements of the border that are frequent
  - negative border:
    - elements of the border that are infrequent

  - Frequent itemsets = subsets of border + positive border

Search Space and Border



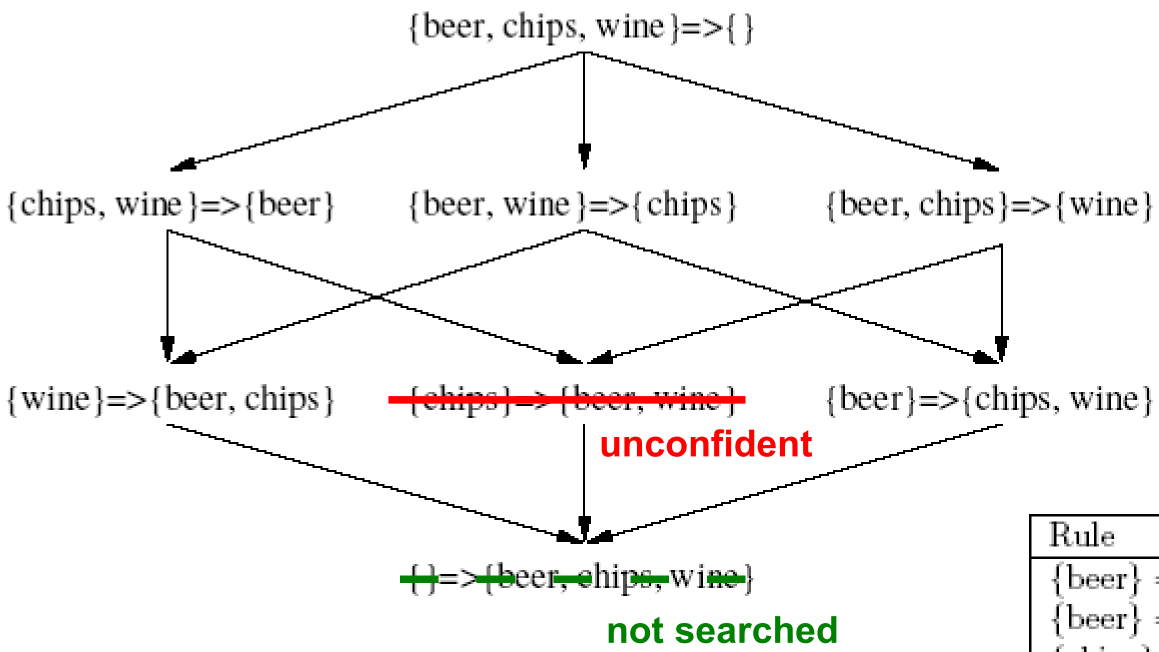
based on Bart Goethals, Survey on Frequent Pattern Mining, 2002

APRIORI Step 2: Generate Association Rules

- Association rules can be generated from frequent item sets
  - confidence of the rule can be computed efficiently from the support of Y and Z, but generating all rules may be expensive
  - for each frequent item set X there are $2^{|X|}$ possible association rules of the form $Y \rightarrow Z$, with $Y \cup Z = X$ and $Y \cap Z = \{\}$
- Efficient generation of association rules:
  - the generation of all subsets can be made much more efficient by exploiting the anti-monotonicity property in the heads of the rules
  - Confidence Anti-monotonicity:
    - $confidence(A \rightarrow B, C) \leq confidence(A, B \rightarrow C)$
    - Why?
  - Thus, rules can be generated with an algorithm similar to FreqSet (starting with heads with length 1, etc.)
    - if a rule with a head is unconfident, adding conditions from the body to the head will also result in unconfident rules → need not be searched

Example

Search space for itemset {beer, chips, wine}



All rules for Confidence ≥ 0.5

Rule	Support	Frequency	Confidence
{beer} ⇒ {chips}	2	50%	100%
{beer} ⇒ {wine}	1	25%	50%
{chips} ⇒ {beer}	2	50%	66%
{pizza} ⇒ {chips}	1	25%	50%
{pizza} ⇒ {wine}	1	25%	50%
{wine} ⇒ {beer}	1	25%	50%
{wine} ⇒ {chips}	1	25%	50%
{wine} ⇒ {pizza}	1	25%	50%
{beer, chips} ⇒ {wine}	1	25%	50%
{beer, wine} ⇒ {chips}	1	25%	100%
{chips, wine} ⇒ {beer}	1	25%	100%
{beer} ⇒ {chips, wine}	1	25%	50%
{wine} ⇒ {beer, chips}	1	25%	50%

Source: Bart Goethals, Survey on Frequent Pattern Mining, 2002

Example 2

	bread	butter	coffee	milk	sugar
customer 1	1	1	0	0	1
customer 2	0	0	1	1	1
customer 3	1	0	1	1	1
customer 4	0	0	1	1	0

- Find all association rules with $s_{\min} = 0.5$ and $c_{\min} = 1.0$
  1. find frequent itemsets:
     - C_1 = { {bread}, {butter}, {coffee}, {milk}, {sugar} }
       S_1 = { {bread}, {coffee}, {milk}, {sugar} }
     - C_2 = { {bread, coffee}, {bread, milk}, {bread, sugar}, {coffee, milk}, {coffee, sugar}, {milk, sugar} }
       S_2 = { {bread, sugar}, {coffee, milk}, {coffee, sugar}, {milk, sugar} }
     - C_3 = { {coffee, milk, sugar} }
       S_3 = { {coffee, milk, sugar} }
     - $C_4 = 0$

Example 2 (Ctd.)

bread	butter	coffee	milk	sugar
1	1	0	0	1
0	0	1	1	1
1	0	1	1	1
0	0	1	1	0

2. Find all rules with $c_{\min} = 1.0$
   - **`bread => sugar`** (0.5,1.0)
   - **`milk => coffee`** (0.75,1.0)
   - **`coffee => milk`** (0.75,1.0)
   - **`milk, sugar => coffee`** (0.5, 1.0)
   - **`sugar, coffee => milk`** (0.5, 1.0)

- Other rules have
  - too small frequency (filtered out by Step 1)
    - **`butter => bread, sugar`** (0.25, 1.0)
  - too small confidence (filtered out by Step 2)
    - **`milk, coffee => sugar`** (0.5, 0.67)

Properties of APRIORI

- Efficiency
  - only needs k passes through the database to find all association rules of length k
    - important if database is too big for memory
  - bottleneck:
    - large number of itemsets must be tested for each item
- Search space
  - significant reduction of search space over searching all possible rules ($2^{|I|}$ different subsets)
- Results
  - generates far too many rules for practical purposes
  - further filtering of result sets is necessary
    - e.g., sort rules by some measure of interestingness and report the best n rules

Filtering Association Rules

- assume rules R_1: $A, B \rightarrow C$ and R_2: $A \rightarrow C$
- OpusMagnum (Webb, 2000) filters rule R_1 if it is
  - **trivial:**
    - R_2 covers the same examples
  - **unproductive:**
    - R_2 has an equal or higher confidence
  - **insignificant:**
    - R_2's confidence is not significantly worse (binomial test)

Justification:
Adding Condition B does not add information about the target attribute

- Interestingness Measures:
  - sort rules by some numerical measure of interestingness
  - return the n best rules (n set by user)
    - it is hard to formalize the notion of „interestingness“

Interestingness Measures

- Basic problem:
  - support and confidence are not sufficient for capturing whether a rule is interesting or not
  - a rule may have high support and confidence, but still not be interesting of surprising

- Example:
  - **`diapers => beer`** (c=0.9)
    90% of customers that buy diapers also buy beer.
  - looks like an interesting finding
  - BUT: if we know that 90% of *all* customers buy beer, the rule is not at all interesting

Lift & Leverage

- Lift:
  - ratio of confidence over *a priori* expectaction for head

$$lift(A \rightarrow B) = \frac{\frac{n(A \cup B)}{n(A)}}{\frac{n(B)}{n}} = \frac{confidence(A \rightarrow B)}{confidence(\rightarrow B)} = \frac{support(A \rightarrow B)}{support(A)\,support(B)}$$

- Leverage:
  - Difference between support and expected support if rule head and body were independent

$$leverage(A \rightarrow B) = support(A \rightarrow B) - support(A)\,support(B)$$

  - leverage is a lower bound for support
    - high leverage implies high support
    - optimizing only leverage guarantees a certain minimum support (contrary to optimizing only confidence or only lift)

Vertical Database Layout

- horizontal database
  - each transaction lists bought items

	beer	wine	chips	pizza
100	1	1	1	0
200	1	0	1	0
300	0	1	0	1
400	0	0	1	1

- vertical database
  - each item lists the transactions that bought it

	beer	wine	chips	pizza
100	1	1	1	0
200	1	0	1	0
300	0	1	0	1
400	0	0	1	1

- if the vertical database fits into memory
  - itemsets can be joined by computing the intersection of the transactions that bought it
    - e.g., { beer } = {1,1,0,0} $\cup$ { wine } = {1,0,1,0 } $\Rightarrow$ { beer, wine } = {1,0,0,0}
  - transactions that appear in no k-item can be deleted
    - will not appear in any $(k+1)$-item
  - algorithm works only if database fits into memory!

Depth-First Search for Frequent Itemsets

- Apriori searches for itemsets in a breadth-first fashion
- There are other algorithms that find frequent item sets depth-first:
  - Eclat (Zaki, 2000)
    - recursively generates all item-sets with the same prefix
    - uses vertical database layout
      - but data can be divided into smaller subsets based on common prefixes
  - FP-Growth (Han, Pei, Yin, 2000)
    - quite similar to Eclat, but uses an elaborate data structure, a frequent pattern tree (FP-tree)
- The Association rule growing phase is the same for these algorithms

Best-First Search

- Frequent set based search (Apriori)
  - typically far too many rules
  - pruning is based on support/frequency, even if interesting measure is different
  - focus on minimizing the number of database scans
- OpusMagnum (Webb, KDD-2000)
  - assumes examples fit in main memory
  - directly searches for the n best rules in a best-first fashion
    - rule quality can be based on a variety of criteria
  - many pruning options
    - *optimistic pruning:* prune a rule if the highest possible value of its successors is too low to be of interest
  - syntactic constraints really reduce search space
    - for Apriori they only affect phase 2.

Representational Extensions

- Nominal Attributes:
  - each n-valued attribute can be transformed into n binary attributes
  - increased efficiency if the algorithm knows that only one of these n values can appear in an item set
- Abstraction Hierarchies:
  - forming groups of items (e.g., dairy products) and using them as additional items
- Sequences:
  - efficient extension of FreqSet to find frequent subsequences
- Rule Schemata:
  - the user may restrict the pattern of rules of interest (e.g., only rules with a certain set of attributes in the head)

Application: Telecommunication Alarm Sequence Analyzer (TASA)

- Goal:
  - find temporal dependencies in alarm sequences for
    - recognizing redundant alarms
    - detecting problems in the networks
    - early warning of severe problems
- Data:
  - temporal sequence of alarms in a finnish telecommunications network
  - 200-10000 alarms/day, 73679 alarms over 50 days, 287 different alarm types
- Find:
  - associations in time sequences of a certain length
  - IF alarm A and alarm B occur within 5 seconds THEN with probability 0.7, alarm C will follow within 60 seconds

References

- Bart Goethals. *Survey on Frequent Pattern Mining.* Manuscript, 2003. http://www.adrem.ua.ac.be/~goethals/publications/survey.pdf
- Ian H. Witten, Eibe Frank, Data Mining: *Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 2nd edition 2005. (sections 3.4 and 4.5)

Software:

- Geoff Webb, *Magnum Opus*, Demo Version (limited to 1000 examples). http://www.csse.monash.edu.au/~webb/software.htm
- Other Association Rule Learning software is also available by Mohammed Zaki, Bart Goethals, or Christian Borgelt, and a version of APriori is implemented in Weka.

Weiteres Programm

- 3.2. Übung
- 5.2. Fragestunde
- 10.2. Vorlesung (Pre-Processing)
- 12.2. letzte Übung