Machine Learning: Symbolische Ansätze

Unsupervised Learning

- Clustering
- Association Rules
Different Learning Scenarios

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

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

**Reinforcement Learning**
- The teacher only provides feedback but not example values

**Semi-supervised Learning**
- Only a subset of the training examples are labeled
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-occurrences in sets
  - an *itemset* is a subset \( A \) of all possible items \( I \)

- Example Problems:
  - Which products are frequently bought together by customers? *(Market 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:
  \[ A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m \]

- Interpretation:
  - When items \( A_i \) appear, items \( B_j \) also appear with a certain probability

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

- Recommender Systems

Your Amazon.com | Today's Deals | Gifts & Wish Lists | Gift Cards

Shop All Departments | Search | Health & Personal Care | GO

Health & Personal Care
Browse Products | Bestsellers | Health Care | Personal Care | Shaving & Hair Removal | Nutrition & Fitness | Sexual Wellness

Adult Reusable Cotton/Poly Snap Diaper - Large - Fits 32" - 46" - Each
by Comfort Concepts

Price: $15.05

In stock.
Processing takes an additional 2 to 3 days for orders from this seller. Ships from and sold by KCK Medical.

Ordering for Christmas? Based on the shipping schedule of KCK Medical, choose Standard at checkout for delivery by December 24. See KCK Medical shipping details.

Frequently Bought Together
Customers buy this item with Call of Duty 4: Modern Warfare Game of the Year Edition by Activision

Price For Both: $40.11
Add both to Cart | Add both to Wish List

These items are shipped from and sold by different sellers. Show details.
Basic Quality Measures

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

- **Confidence**
  \[
  \text{confidence}(A \rightarrow B) = \frac{\text{support}(A \cup B)}{\text{support}(A)} = \frac{n(A \cup B)}{n(A)}
  \]
  - 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/frequency 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 frequent
  - all supersets of an infrequent itemset are also infrequent
Support/Confidence Filtering

- filter rules that
  - cover not enough positive examples \((p < s_{\text{min}})\)
  - are not precise enough \((h_{\text{prec}} < c_{\text{min}})\)

- effects:
  - all but a region around \((0,P)\) is filtered

**Note:**

\(P \triangleq \text{examples for which head is true}\)
\(N \triangleq \text{examples for which head is false}\)
APRIORI Step 1: 
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 \( C_k \) \hspace{1cm} ← 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 \( C_i \) 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:
    \[ C_{k+1} = \{ \langle X_1, \ldots, X_{k-1}, X_k, X_{k+1} \rangle \mid \langle X_1, \ldots, X_{k-1}, X_k \rangle \in S_k, \langle X_1, \ldots, X_{k-1}, X_{k+1} \rangle \in S_k, X_k < X_{k+1} \} \]
    - 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 a candidate set if not all $k$-item subsets are frequent (i.e., in $S_k$)
Example

- Find all itemsets with $s_{\text{min}} = 0.25$
  - $C_1 = \{ \{\text{beer}\}, \{\text{chips}\}, \{\text{pizza}\}, \{\text{wine}\} \}$
    - $S_1 = \{ \{\text{beer}\}, \{\text{chips}\}, \{\text{pizza}\}, \{\text{wine}\} \}$
  - $C_2 = \{ \{\text{beer, chips}\}, \{\text{beer, pizza}\}, \{\text{beer, wine}\}, \{\text{chips, pizza}\}, \{\text{chips, wine}\}, \{\text{pizza, wine}\} \}$
    - $S_2 = \{ \{\text{beer, chips}\}, \{\text{beer, wine}\}, \{\text{chips, pizza}\}, \{\text{chips, wine}\}, \{\text{pizza, wine}\} \}$
  - $C_3 = \{ \{\text{beer, chips, wine}\}, \{\text{chips, pizza, wine}\} \}$
    - $S_3 = \{ \{\text{beer, chips, wine}\} \}$
  - $C_4 = \emptyset$
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

\[
\begin{align*}
&\text{frequent itemset} \\
&\text{positive border} \\
&\text{negative border} \\
&\text{frequent} \\
&\text{infrequent}
\end{align*}
\]
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 $2^{\mid X\mid}$ possible association rules of the form $Y \rightarrow Z$, with $Y \cup Z = X$ and $Y \cap Z = \emptyset$

- 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:
    
    \[
    \text{confidence}(A \rightarrow B, C) \leq \text{confidence}(A, B \rightarrow C)
    \]
  - 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 $\rightarrow$ need not be searched
Example

Search space for itemset \{beer, chips, wine\}

\{beer, chips, wine\} \Rightarrow \{\}

\{chips, wine\} \Rightarrow \{beer\}  \quad \{beer, wine\} \Rightarrow \{chips\}  \quad \{beer, chips\} \Rightarrow \{wine\}

\{wine\} \Rightarrow \{beer, chips\}  \quad \{chips\} \Rightarrow \{beer, wine\}  \quad \{beer\} \Rightarrow \{chips, wine\}

\{\} \Rightarrow \{beer, chips, wine\}

All rules for Confidence $\geq 0.5$

<table>
<thead>
<tr>
<th>Rule</th>
<th>Support</th>
<th>Frequency</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>{beer} $\Rightarrow$ {chips}</td>
<td>2</td>
<td>50%</td>
<td>100%</td>
</tr>
<tr>
<td>{beer} $\Rightarrow$ {wine}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{chips} $\Rightarrow$ {beer}</td>
<td>2</td>
<td>50%</td>
<td>66%</td>
</tr>
<tr>
<td>{pizza} $\Rightarrow$ {chips}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{pizza} $\Rightarrow$ {wine}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{wine} $\Rightarrow$ {beer}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{wine} $\Rightarrow$ {chips}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{wine} $\Rightarrow$ {pizza}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{beer, chips} $\Rightarrow$ {wine}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{beer, wine} $\Rightarrow$ {chips}</td>
<td>1</td>
<td>25%</td>
<td>100%</td>
</tr>
<tr>
<td>{chips, wine} $\Rightarrow$ {chips}</td>
<td>1</td>
<td>25%</td>
<td>100%</td>
</tr>
<tr>
<td>{beer} $\Rightarrow$ {beer}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
<tr>
<td>{wine} $\Rightarrow$ {beer, chips}</td>
<td>1</td>
<td>25%</td>
<td>50%</td>
</tr>
</tbody>
</table>

Source: Bart Goethals, Survey on Frequent Pattern Mining, 2002
Example 2

- Find all association rules with $s_{\text{min}} = 0.5$ and $c_{\text{min}} = 1.0$

1. Find frequent itemsets:
   - $C_1 = \{ \{\text{bread}\}, \{\text{butter}\}, \{\text{coffee}\}, \{\text{milk}\}, \{\text{sugar}\} \}$
     $S_1 = \{ \{\text{bread}\}, \{\text{coffee}\}, \{\text{milk}\}, \{\text{sugar}\} \}$
   - $C_2 = \{ \{\text{bread, coffee}\}, \{\text{bread, milk}\}, \{\text{bread, sugar}\}, \{\text{coffee, milk}\}, \{\text{coffee, sugar}\}, \{\text{milk, sugar}\} \}$
     $S_2 = \{ \{\text{bread, sugar}\}, \{\text{coffee, milk}\}, \{\text{coffee, sugar}\}, \{\text{milk, sugar}\} \}$
   - $C_3 = \{ \{\text{coffee, milk, sugar}\} \}$
     $S_3 = \{ \{\text{coffee, milk, sugar}\} \}$
   - $C_4 = 0$
Example 2 (Ctd.)

2. Find all rules with $c_{\text{min}} = 1.0$
   - 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 \( (2^{|I|} \text{ different subsets}) \)

- **Results**
  - generates far too many rules for practical purposes
  - further filtering of result sets is necessary
    - e.g., sort rules by some interestingness measure 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)

- 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* expectation for head
  \[
  \text{lift}(A \rightarrow B) = \frac{\frac{n(A \cup B)}{n}}{\frac{n(A)}{n(B)}} = \frac{\text{confidence}(A \rightarrow B)}{\text{confidence}(\rightarrow B)} = \frac{\text{support}(A \rightarrow B)}{\text{support}(A) \cdot \text{support}(B)}
  \]

- **Leverage:**
  - Difference between support and expected support if rule head and body were independent
  \[
  \text{leverage}(A \rightarrow B) = \text{support}(A \rightarrow B) - \text{support}(A) \cdot \text{support}(B)
  \]

- 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

<table>
<thead>
<tr>
<th></th>
<th>beer</th>
<th>wine</th>
<th>chips</th>
<th>pizza</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>beer</th>
<th>wine</th>
<th>chips</th>
<th>pizza</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- 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 as in APriori, only the frequent pattern mining phase is different
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


Software:
- 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.