Combinatorial Data Mining Algorithms

Handling Missing Values in Data

Markko Merzin
Missing values: example

<table>
<thead>
<tr>
<th>t</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_1$</td>
<td>$b_2$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>2</td>
<td>$a_1$</td>
<td>$b_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>3</td>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>4</td>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$c_1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_1$</td>
<td>$b_2$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
<td>$b_2$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>3</td>
<td>$a_2$</td>
<td>$b_2$</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>$a_1$</td>
<td>?</td>
<td>$c_1$</td>
</tr>
</tbody>
</table>

Figure 1: Identical databases, without and with missing values
**Missing values:** what if we ignore them?

### Without missing values
- \((A=a_1) \Rightarrow (B=b_2)\)
  - supp 50%, conf 66%
- \((A=a_1) \Rightarrow (C=c_1)\)
  - supp 50%, conf 66%

### With missing values
- \((A=a_1) \Rightarrow (B=b_2)\)
  - supp 25%, conf 50%
- \((A=a_1) \Rightarrow (C=c_1)\)
  - supp 50%, conf 100%

<table>
<thead>
<tr>
<th>t</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(a_1)</td>
<td>(b_2)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>2</td>
<td>(a_1)</td>
<td>(b_2)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>3</td>
<td>(a_2)</td>
<td>(b_2)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>4</td>
<td>(a_1)</td>
<td>(b_1)</td>
<td>(c_1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(a_1)</td>
<td>(b_2)</td>
<td>(c_1)</td>
</tr>
<tr>
<td>2</td>
<td>(?)</td>
<td>(b_2)</td>
<td>(c_2)</td>
</tr>
<tr>
<td>3</td>
<td>(a_2)</td>
<td>(b_2)</td>
<td>(?)</td>
</tr>
<tr>
<td>4</td>
<td>(a_1)</td>
<td>(?)</td>
<td>(c_1)</td>
</tr>
</tbody>
</table>
Missing values: what if we ignore them?

In general:

Support: ↓
Confidence: ↑ or ↓
Number of rules: ↓

Lowering threshold for support and confidence introduces artificial rules
Missing values: usable approach

We have to redefine some notions:

\[ \text{supp} (X) := \frac{\text{count}(X)}{\text{count}(X^*)}, \]

where \( X^* \) is itemsets that have no missing values of attributes of \( X \).

\[ \text{conf} (X \rightarrow Y) := \frac{\text{count}(X \cup Y)}{\text{count}(X \cup Y^*)} \]

Note that support with that way is no longer monotone.
Representatitivity: we need to restrict the influence of itemsets that are not observed a lot.

\[ rep(X) := \frac{\text{count}(X^*)}{|D|}. \]
Extensibility: An itemset $X$ is called extensible, if it has a frequent and representative superset, i.e.

$$\exists Y: \{ \begin{align*} \text{supp}(X \cup Y) & \geq \text{minsupp} \\ \text{rep}(X \cup Y) & \geq \text{minrep} \end{align*}$$
Missing values: usable approach

The redefined support measure is no longer monotone, but since extensibility subsumes frequency (i.e. inextensible sets are infrequent or unrepresentative), we will mine for these sets instead. For this we need a numerical test to check the extensibility of an itemset.

Theorem 1. For an itemset $X$, define the set $S_X$ as $S_X := \{Z \supseteq X | Z$ is representative and frequent$\}$, i.e. all frequent and representative supersets of $X$. Let among their local maxima, $m(X)$ be the lowest representativity, $m(X) := \min\{\text{rep}(Z) | Z \in S_X\}$, or $m(X) := +\infty$ if $S_X = \emptyset$. Then

$$X \text{ is extensible} \iff \frac{\text{count}(X)}{m(X)} \geq \text{minsup}$$

Proof. If $X$ is extensible, $S_X \neq \emptyset$, so $\text{count}(X)/m(X) \geq \text{supp}(X) \geq \text{minsup}$, since $\text{minrep} \leq m(X) \leq \text{rep}(X)$. If $X$ is inextensible, $S_X = \emptyset$, so $\text{count}(X)/m(X) = 0 < \text{minsup}$. □
Missing values: usable valid approach

Unfortunately, this does not completely solve the problem, since the aforementioned property is rather paradoxical. In order to prune the supersets of a set, we must first compute all of them to obtain the value $m(X)$. Hence, the theorem cannot be directly applied in an algorithm. However we really only need one direction of the equivalence, so we can use a lower bound for $m(X)$, as described in this corollary.

**Corollary 1.** For an itemset $X$, let $k \leq m(X)$. Then $\frac{\text{count}(X)}{k} < \text{mins}up \Rightarrow X$ is not extensible.
Baseline algorithm. First, we propose a straightforward solution which will serve as a baseline for comparison. This algorithm is based on the observation that if an itemset $X$ is representative, its support is at most $\text{count}(X)/\text{minrep}$. Otherwise checking this fraction is unnecessary. Using the corollary of theorem 1, $\text{minrep}$ plays the role of lower bound. Formally, the algorithm first checks if $\text{rep}(X) \geq \text{minrep}$, and if true $\text{count}(X)$ is counted. If $\frac{\text{count}(X)}{\text{minrep}} \lt \text{minsupt}$ the algorithm concludes $X$ is inextensible. Otherwise the algorithm must continue with the supersets of $X$.

The fact that the $m(X)$ lower bound is a fixed constant for all itemsets, facilitates implementation of the baseline.
Missing values: baseline algorithm

We choose to adapt Eclat [7] for its simplicity and speed. It does a depth first traversal, for each itemset maintaining a tid list of transactions supporting that itemset. At each step itemsets with a common prefix are combined to obtain larger ones (this is where the monotonicity of extensibility is used). Our implementation also uses diffsets as an optimization [8].

The baseline is equivalent to (albeit more efficient than) finding all representative itemsets whose count is larger than \( \text{minsup} \times \text{minrep} \), and filtering out the frequent ones in a postprocessing step.
Missing values: XMiner algorithm

- Approximates $m(X)$ better than $\text{minrep}$.
- As efficient as Eclat algorithm on complete datasets (generates same amount of candidates).
Missing values: XMiner algorithm

Algorithm 1 XMiner(set of itemsets $P$)

Require: $P$ is a set of representative and possibly extensible ordered itemsets with a common (omitted) prefix

1. for all $X_i \in P$ do
2. \hspace{0.5cm} compute $m'(X) = \max(\text{count}(\cap_{j \geq i} X_j^*), \text{minrep})$
3. \hspace{0.5cm} for all values $x_i$ of $X_i$ with
\hspace{1cm} count($X_i = x_i$)/$m'(X_i) \geq \text{minsups}$ do
4. \hspace{0.5cm} for all $X_j \in P$ with $X_j > X_i$ do
5. \hspace{1cm} if count($X_{ij} = *$) $\geq \text{minrep}$ then
6. \hspace{1.5cm} for all values $x_j$ of $X_j$ do
7. \hspace{2cm} $(X_{ij} = x_{ij}).\text{tids} =$
\hspace{2cm} $(X_i = x_i).\text{tids} \cap (X_j = x_j).\text{tids}$
8. \hspace{1cm} if count($X_{ij} = x_{ij}$)/$m'(X_i) \geq \text{minsups}$ then
9. \hspace{1.5cm} $P^i = P^i \cup \{(X_{ij} = x_{ij})\}$
10. \hspace{1cm} if count($X_{ij} = x_{ij}$)/count($X_{ij} = *$) $\geq \text{minsups}$ then
11. \hspace{1.5cm} report ($X_{ij} = x_{ij}$) as frequent
12. \hspace{1.5cm} XMiner($P^i$)
Missing values: XMiner algorithm

(b) minrep=10%

Missing values: XMiner algorithm

Missing values: XMiner algorithm

Missing values: XMiner algorithm