MTAT.03.183 Data Mining

Frequent Itemset Mining

Sven Laur
University of Tartu
Quick outline

▷ Background
▷ Frequent Itemset Mining
▷ Association Rules
▷ Theoretical Limitations
▷ More Efficient Support Counting
▷ Efficiency through Loopholes
▷ Additional Materials and Resources
Descriptive versus predictive data analysis

Descriptive data analysis

▷ Aims to summarise the main qualitative traits of data.
▷ Used mainly for discovering processes and relations in data.
▷ Facts are presented together with understandable explanations.
▷ Used in medical, physical and social sciences.

Predictive data analysis

▷ Tries to predict unknown features from other features.
▷ Used mainly for predicting the behaviour of individuals.
▷ Predictions are usually presented without comprehensive reasoning.
▷ Used in software, business and engineering.
Frequent Itemset Mining
Motivating example

Supermarkets often run price campaigns to increase the customer base. In such campaigns, one has to address the following questions.

▷ Which prices have to be lowered?
▷ Which prices have to be raised to compensate price reductions?
▷ What is the expected net effect on the volume of sales and profit?

An old trick. People seldomly buy a single product. Price reductions can be often compensated by increasing margins on other products.

A new twist. Instead of relying on previous experiences and conventional wisdom, we can use data mining to guide this process.
## Items and transactions

<table>
<thead>
<tr>
<th>ID</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transactions
- \{b, \, d, \, e\}
- \{a, \, b, \, c\}
- \{a, \, d, \, e\}
- \{b, \, d, \, e\}
- \{a, \, b, \, c, \, d, \, e\}
- \{b, \, d\}

Market basket data can be represented as a table, where

- rows correspond to transactions,
- the first column is the transaction identifier,
- the remaining rows correspond to items.
Cover and support

<table>
<thead>
<tr>
<th>ID</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transactions

- \{b, d, e\}
- \{a, b, c\}
- \{a, d, e\}
- \{b, d, e\}
- \{a, b, c, d, e\}
- \{b, d\}

An itemset is a set of items. The cover of an itemset \(A\) is a set of rows identifiers that contain all items of \(A\). The absolute support of an itemset \(A\) is the number of rows that contain all items of \(A\). Relative support is measured as a percentage of all transactions. In our example

- \(\text{cover}(\{a, b, c\}) = \{2, 5\}\)  \(\text{supp}(\{a, b, c\}) = 2.\)
Frequent itemsets

<table>
<thead>
<tr>
<th>ID</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transactions

- \{b, d, e\}
- \{a, b, c\}
- \{a, d, e\}
- \{b, d, e\}
- \{a, b, c, d, e\}
- \{b, d\}

For a fixed threshold \(\tau\), frequent itemsets \(\mathcal{F}(\tau)\) are all itemsets such that their support is greater or equal than the threshold \(\tau\):

\[
\mathcal{F}(\tau) = \{A : \text{supp}(A) \geq \tau\}.
\]

If we increase the threshold then some sets can become infrequent:

\[
\tau \leq \delta \quad \Rightarrow \quad \mathcal{F}(\delta) \subseteq \mathcal{F}(\tau).
\]
Illustrative example

<table>
<thead>
<tr>
<th>ID</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transactions

\{b, d, e\}
\{a, b, c\}
\{a, d, e\}
\{b, d, e\}
\{a, b, c, d, e\}
\{b, d\}

Frequent itemsets for the threshold $\tau = 5, 4, 3, \ldots$

$\mathcal{F}(5) = \{\{b\}, \{d\}\}$

$\mathcal{F}(4) = \{\{b\}, \{d\}, \{e\}, \{b, d\}, \{d, e\}\}$

$\mathcal{F}(3) = \{\{a\}, \{b\}, \{d\}, \{e\}, \{b, d\}, \{b, e\}, \{d, e\}, \{b, d, e\}\}$

$\mathcal{F}(2) = \{\ldots\}$
Apriori principle

Apriori principle. Any subset of a frequent itemset must be also frequent.

Important conclusions

▷ A frequent itemset must consist of frequent items.

▷ If we know all frequent itemsets of length \( k \) then we can weed out many itemsets of length \( k + 1 \) without computing the supports.

▷ A candidate for a frequent itemset can be obtained by adding a frequent item to a frequent itemset.
Apriori algorithm

1: Find all frequent singletons: \( \mathcal{F}_1 \leftarrow \{ \{x\} : \text{supp}(\{x\}) \geq \tau \} \).
2: Generate a list of candidates: \( \mathcal{G} \leftarrow \text{CandidateGeneration}(\mathcal{F}_1, \mathcal{F}_1) \).
3: Set \( k = 2 \).
4: while \( \mathcal{G} \neq \emptyset \) do
5: \hspace{1em} Set \( \mathcal{F}_k \leftarrow \emptyset \).
6: \hspace{1em} foreach \( A \in \mathcal{G} \) do
7: \hspace{2em} if \( \text{supp}(A) \geq \tau \) then Add \( A \) to \( \mathcal{F}_k \).
8: \hspace{1em} od
9: \hspace{1em} Generate new candidates: \( \mathcal{G} \leftarrow \text{CandidateGeneration}(\mathcal{F}_k, \mathcal{F}_1) \).
10: \hspace{1em} Set \( k \leftarrow k + 1 \).
11: od
12: Output \( \mathcal{F}_1, \mathcal{F}_2, \ldots \)
Candidate generation algorithm

Task. Given a list of frequent sets $F_k$ of size $k$ and singleton sets $F_1$, find all possible $(k+1)$-element sets that do not contradict the Apriori principle.

Naive solution

1: Set $G \leftarrow \emptyset$.
2: Generate all possible candidates
   
   $\text{foreach } A \in F_k \text{ and } B \in F_1 \text{ do}$
   
   $\text{If } |A \cup B| = k + 1 \text{ then Add } A \cup B \text{ to } G.$
   
   $\text{od}$
3: Eliminate duplicates from $G$.
4: Eliminate elements that violate the Apriori principle:
   
   $\text{foreach } A \in G \text{ do}$
   
   $\text{if some } k\text{-element subset of } A \text{ is not in } F_k \text{ then eliminate } A \text{ from } G.$
   
   $\text{od}$
Possible optimisations

**First observation.** If we add items that are strictly larger than elements of the set $A \in \mathcal{F}_k$, then we generate all candidate sets only once.

**Second observation.** If we order sets in $\mathcal{F}_k$ lexicographically and use the first observation, then all remaining $k$ subsets, that must be frequent according to Apriori principle, must be located further in the list $\mathcal{F}_k$.

**Third observation.** If we use C++ STL map or set templates for storing lists of frequent itemsets, then testing the Apriori condition is relatively straightforward and efficient enough.

**Fourth observation.** If we manage to generate candidate sets only once, then other optimisations are in practice irrelevant compared to the time that is needed to compute the supports.
Association Rules
**Motivating example**

Consider a long-time study of diseases, which contains many facts about the background: genetic markers, environment, other diseases, etc. Given this list of transactions, it is natural to ask the following questions.

- Are there any background factors that seem to cause a disease?
- How strong is the corresponding association?
- Are there enough examples to derive such a rule?
- What is the amount of variation that is covered by the rule?

Association rule mining can be viewed as a simple but still powerful tool for finding putative causal relations between events encoded as items.

- Temporal relations between events are omitted in such an analysis.
Association rules

**If** bread and wine **then** cheese

A shorthand $X \Rightarrow Y$ denotes an *association rule*, which informally reads itemset $X$ in a transaction implies that itemset $Y$ is also in the transaction.

The *support* of a rule $X \Rightarrow Y$ is defined as cumulative support

$$\text{supp}(X \Rightarrow Y) := \text{supp}(X \cup Y) .$$

The *confidence* of a rule $X \Rightarrow Y$ is defined as

$$\text{conf}(X \Rightarrow Y) := \frac{\text{supp}(X \cup Y)}{\text{supp}(X)} .$$

Confidence can be viewed as an *estimate* on conditional probability $\Pr [Y|X]$. 
Properties of association rules

- Association rules with low confidence are uninformative.
- Association rules with low support and high confidence
  - are usually artefacts of a particular data set
  - or capture too small fraction of total variance to be practical.
- The confidence does not increase if we move items from left to right
  \[ \forall x \in X : \quad \text{conf}(X \Rightarrow Y) \geq \text{conf}(X \setminus \{x\} \Rightarrow Y \cup \{x\}) . \]
- The confidence cannot decrease if we delete elements from the right
  \[ \forall y \in Y : \quad \text{conf}(X \Rightarrow Y) \leq \text{conf}(X \Rightarrow Y \setminus \{y\}) . \]
Association rule mining

**First observation.** A support of a rule $X \Rightarrow Y$ has a support over the threshold $\tau$ only if $X \cup Y$ is frequent set, i.e., we must first generate $\mathcal{F}(\tau)$.

**Second observation.** We can move elements in the rule form the left to the right until we hit the confidence bound, i.e., we do not have to consider all partitions of frequent sets to mine rules with a prescribed confidence.

**Third observation.** Generation of all rules generates quickly information overload and thus a human guided search is needed.

- In a medical study, the list of interesting targets might be diseases.
- In diagnostic study, the list of interesting sources might be a list of symptoms or errors.
Lift

**Problem.** If $X$ and $Y$ have large supports then the rule $X \Rightarrow Y$ has large confidence as a direct consequence of supports and not causality.

**Lift.** Lift shows how many times $\Pr[X \land Y]$ is larger than $\Pr[X] \Pr[Y]$. Formally, the *lift* of association rule $X \Rightarrow Y$ is defined as a ratio

$$
lift(X \Rightarrow Y) := n \cdot \frac{\text{supp}(X \cup Y)}{\text{supp}(X) \text{supp}(Y)}$

where $n$ is the number of transactions.

- If $\text{lift}(X \Rightarrow Y) = 1$ then itemsets $X$ and $Y$ are independent.
- If $\text{lift}(X \Rightarrow Y) \gg 1$ then $X \cup Y$ is overrepresented.
- If $\text{lift}(X \Rightarrow Y) \ll 1$ then $X \cup Y$ is underrepresented.
Some Theoretical Results
Running-time of the Apriori algorithm

The running-time of the Apriori algorithm consists of two parts.

▷ **Time for candidate validation.** This is proportional to the number of candidates sets $c$ generated during the Apriori algorithm:
   ◇ all frequent itemsets.
   ◇ all sets that are not frequent but which proper subsets are frequent.

▷ **Time for candidate generation.** Usually the time needed to candidate generation is much smaller than the time needed to verify candidates. Nevertheless, we can bound the time in terms of candidate sets:
   ◇ We manage maps or sets of size up to $c$.
   ◇ Consequently, each set operation takes $O(\log c)$ time.
   ◇ For each $(k + 1)$-element we have to do $k$ reads form a set.
   ◇ By construction $k$ is smaller than the number of items $m$. 
Negative border and its size

**Negative border** $B^-\!$ consists of all itemsets that are not frequent but all their proper subsets are frequent. The size of the negative border determines the amount of unnecessary work done by the Apriori algorithm.

- Each frequent set can create up to $m$ infrequent candidate sets.
- Hence $|B^-| \leq m |\mathcal{F}(\tau)|$ and thus $c \leq (m + 1) |\mathcal{F}(\tau)|$. 

Final estimate on the running-time

As the support counting is linear in the number of transactions \( n \), the candidate validation runs in time

\[
O(nm |\mathcal{F}(\tau)|) .
\]

Similarly, the candidate generation runs in time

\[
O(m^2 |\mathcal{F}(\tau)| (\log m + \log |\mathcal{F}(\tau)|)) .
\]

**Corollary.** If the number of items is bounded in transactions, the Apriori algorithm runs in linear time w.r.t. its input and output size. Otherwise the running time is linear w.r.t. the output and a polynomial w.r.t. the input.
Interpretation of this result

To increase the efficiency, we must either increase the efficiency of the support counting or reduce the size of the output.

Improvements in support counting can give a significant speedup, as support counting can be quite slow if the database does not fit into the main memory.

Reducing the size of the output can be beneficial in two respects:
- it reduces the number of patterns to be interpreted;
- it can potentially reduces the running-time of the algorithm.
More Efficient Support Counting
Operations with covers

<table>
<thead>
<tr>
<th>ID</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Transactions

\[
\{b, d, e\}, \{a, b, c\}, \{a, d, e\}, \{b, d, e\}, \{a, b, c, d, e\}, \{b, d\}
\]

Recall that a cover of an itemset is a set of row identifiers such that the corresponding rows contain the itemset. Hence,

\[
supp(X) = |cover(X)|,
\]

\[
cover(X \cup Y) = cover(X) \cap cover(Y).
\]

As there are usually few elements in columns, working with covers lowers the memory footprint and reduces the running-time.
Main principles behind Eclat

**Depth-first search.** Keeping all covers in memory can be infeasible if we use breath-first search for candidate sets.

**Pattern-growth principle.** Frequent itemset can be obtained only by adding a frequent element to a frequent itemset.

- The pattern-growth principle is less powerful than the Apriori principle, since we might test elements outside the negative border.

- If we generate candidate sets only once during the search, then the number of additional validations is often tolerable.

- If we generate itemsets in lexicographic order then no itemset can appear twice in the candidate generation.
Sketchy implementation

The core of the recursive Eclat algorithm is captured by the following.

1: \textbf{GenerateFrequentItemsets}(\textit{Prefix}, \textit{S})
2: \text{Set } \mathcal{F} \leftarrow \emptyset.
3: \textbf{foreach } \textit{x} \in \textit{S} \textbf{ do}
4: \hspace{1em} \textbf{if } \text{supp}(\textit{Prefix} \cup \{\textit{x}\}) < \tau \textbf{ then continue}
5: \hspace{1em} \text{Set } \textit{S} \leftarrow \textit{S} \setminus \{\textit{x}\}.
6: \hspace{1em} \text{Set } \mathcal{F} \leftarrow \mathcal{F} \cup \textbf{GenerateFrequentItemsets}(\textit{Prefix} \cup \{\textit{x}\}, \textit{S}).
7: \textbf{od}
8: \text{Output } \mathcal{F}.

Each time we join \textit{Prefix} and \textit{x} we can compute

\[
\text{cover}(\textit{Prefix} \cup \{\textit{x}\}) = \text{cover}(\textit{Prefix}) \cap \text{cover}(\textit{x}) .
\]

As a result we must keep only at most $2m$ covers in the main memory.
Brief summary of the FP-growth algorithm

In many cases, maximal number of items in the transactions is much smaller than the number of potential items.

FP-tree is a tree structure which encodes the transaction database so that it is much smaller and facilitates efficient support counting.

▷ To save memory, the FP-tree contains complete information for frequent sets and some partial information about infrequent sets.
▷ Instead of counting the supports in the database, we have to traverse some parts of the FP-tree. For that purpose, the tree is equipped with additional links.

The FP-growth algorithm is a specially designed algorithm for enumerating frequent itemsets by first converting the database to the FP-tree and then traversing it for computing supports.
Efficiency through Loopholes
Maximal frequent itemsets

A frequent itemset is *maximal* if all its proper supersets are infrequent.

**Rationale behind maximal itemsets.**

- Each maximal $k$-element frequent itemset induces $2^k$ frequent itemsets.
- Thus one cannot easily mine maximal frequent itemsets with many elements if we output all frequent itemsets.
- On the other hand, guessing the locations of maximal frequent itemsets is a non-trivial task.

There are special algorithms for finding maximal frequent itemsets:

- MAFIA, GenMax, FPmax, ...
Closed sets

A frequent itemset is *closed* is all its proper supersets have smaller support.

**Rationale behind closed itemsets**

▷ There are usually more frequent itemsets than closed frequent itemsets.
▷ However, the number of maximal frequent itemsets is often smaller than closed frequent itemsets.
▷ Any frequent itemsets have the same support than some closed itemset. This closed itemset contains also the corresponding frequent itemset.

There are special algorithms for finding closed frequent itemsets:

◇ CHARM, CLOSE, CLOSET, ...
Additional Materials and Resources
Nice overviews and standard results


▷ Han, J.; Pei, J.; Yin, Y. & Mao, R. Mining Frequent Patterns without Candidate Generation: A Frequent- Pattern Tree Approach. Data Min. Knowl. Discov., 2004, 8, 53-87.


See also the webpage of the Combinatorial Data Mining Algorithms course: http://courses.cs.ut.ee/2009/fast-counting/
Freely available implementations

- FIMI 2004 repository
  http://fimi.cs.helsinki.fi/src/

- A C++ Frequent Itemset Mining Template Library