Frequent substructure mining - an introduction

R. Kessl

SUI, 7. May 2009
Outline

1. Introduction
2. Frequent itemset mining
3. The Apriori algorithm
4. Abstract problem formulation
5. The Eclat algorithm
6. The FPGrowth algorithm
We have a database $D$ of transactions $T$.

$T$ can be an arbitrary object.

For example: itemsets (basket market), time sequences, graphs

Mining of frequent substructures has exponential complexity (in the worst case)
The problem is to find a set of items (itemsets) that occurs in at least $p\%$ of transactions,

exponential complexity with respect to the # of items (in the worst case)
Example:

Database $D$:

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${1, 2, 3, 4}$</td>
</tr>
<tr>
<td>2</td>
<td>${3, 5}$</td>
</tr>
<tr>
<td>3</td>
<td>${1, 3, 4}$</td>
</tr>
<tr>
<td>4</td>
<td>${1, 2}$</td>
</tr>
<tr>
<td>5</td>
<td>${1, 3, 4, 5}$</td>
</tr>
<tr>
<td>6</td>
<td>${1, 2, 3, 4, 5}$</td>
</tr>
</tbody>
</table>

Are there any correlations among items in $D$?
Example:

Database $D$:

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2, 3, 4}</td>
</tr>
<tr>
<td>2</td>
<td>{3, 5}</td>
</tr>
<tr>
<td>3</td>
<td>{1, 3, 4}</td>
</tr>
<tr>
<td>4</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>5</td>
<td>{1, 3, 4, 5}</td>
</tr>
<tr>
<td>6</td>
<td>{1, 2, 3, 4, 5}</td>
</tr>
</tbody>
</table>

- are there any correlations among items in $D$?
- Itemset $\{3, 5\}$ occurs in 3 transactions $\Rightarrow$ $Supp(\{3, 5\}, D) = 3$
Example:

Database $D$:

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1,2,3,4}</td>
</tr>
<tr>
<td>2</td>
<td>{3,5}</td>
</tr>
<tr>
<td>3</td>
<td>{1,3,4}</td>
</tr>
<tr>
<td>4</td>
<td>{1,2}</td>
</tr>
<tr>
<td>5</td>
<td>{1,3,4,5}</td>
</tr>
<tr>
<td>6</td>
<td>{1,2,3,4,5}</td>
</tr>
</tbody>
</table>

- are there any correlations among items in $D$?
- Itemset $\{3, 5\}$ occurs in 3 transactions $\Rightarrow$ $\text{Supp}(\{3, 5\}, D) = 3$
- *Association rule* $\{3, 5\} \rightarrow \{2\}$ in $33.33\%$ of cases $\Rightarrow$ *confidence* $\text{Conf}(\{3, 5\}, \{2\}) = 0.33$
Example:

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 2, 3, 4}</td>
</tr>
<tr>
<td>2</td>
<td>{3, 5}</td>
</tr>
<tr>
<td>3</td>
<td>{1, 3, 4}</td>
</tr>
<tr>
<td>4</td>
<td>{1, 2}</td>
</tr>
<tr>
<td>5</td>
<td>{1, 3, 4, 5}</td>
</tr>
<tr>
<td>6</td>
<td>{1, 2, 3, 4, 5}</td>
</tr>
</tbody>
</table>

Database $D$:

- Are there any correlations among items in $D$?
- Itemset \{3, 5\} occurs in 3 transactions $\Rightarrow$ $\text{Supp}(\{3, 5\}, D) = 3$
- *Association rule* \{3, 5\} $\rightarrow$ \{2\} in 33.33% of cases $\Rightarrow$ *confidence* $\text{Conf}(\{3, 5\}, \{2\}) = 0.33$
- $\Rightarrow$ associations (correlations) among items
Lattice of frequent itemsets for \( \text{min\_support} = 3 \)

⇒ maximal frequent itemsets (MFIs)
⇒ base set of items \( B = \{1, 2, 3, 4, 5\} \)
Frequent itemset mining

Lattice of frequent itemsets for min_support = 3

⇒ maximal frequent itemsets (MFIs)
⇒ base set of items $B = \{1, 2, 3, 4, 5\}$
Frequent itemset mining

Lattice of frequent itemsets for $\text{min\_support} = 3$

⇒ maximal frequent itemsets (MFIs)
⇒ base set of items $B = \{1, 2, 3, 4, 5\}$
Input: database $D$, $min\_support$ and $min\_conf$

1. generate frequent itemsets $X$ with $Supp(X) \geq min\_support$ /* most time consuming part */

2. generate association rules with confidence $Conf(X) \geq min\_conf$
Algorithms

- The Apriori algorithm: the first and slowest algorithm for generation of frequent itemsets
- The FPGrowth algorithm: based on prefix tree (trie)
- The Eclat algorithm: uses vertical representation of the database
Lemma (Monotonicity of support)

Let $U \subseteq B$ be an itemset with support $\text{Supp}(U)$ in database $D$. For every superset $V$ of $U$ holds: $\text{Supp}(U) \geq \text{Supp}(V)$.

Proof.

(By induction)

(i) It is obvious that for an item $b_i$, $|D| = \text{Supp}(\emptyset) \geq \text{Supp} \left( \{ b_i \} \right)$.

(ii) If $U$ has support $\text{Supp}(U)$, then for arbitrary $b_i \in B - U$ holds: $\text{Supp}(U) \geq \text{Supp} \left( U \cup \{ b_i \} \right)$, since not all transactions containing $U$ must contain $U \cup \{ b_i \}$. 
Based on the monotonicity property of support

*Generate* & *test* algorithm

**The Apriori algorithm can make** $k = |B|$ scans of $D$

A *candidate* itemset $U, |U| = k$:

- Support of $U$ is unknown
- all $W \subset U, |W| = k - 1$ are frequent, i.e. $\text{Supp}(W) \geq \text{min\_support}$

From frequent itemsets $F_k$ generate candidates $C_{k+1}$:

- $f_1, f_2 \in F_k$ such that $f_1, f_2$ identical in $k - 1$ items;
- $c = \{f_1[j] : j < k\} \cup \{f_1[k], f_2[k]\}$. 
The Apriori algorithm contd.

**APRIORI** *(In: Database *D*, In: Set *B*, In: Integer *min_support*, In/Out: Set *F*)

1:  \( k \leftarrow 1 \)
2:  \( C_k \leftarrow \{\{b_i\} : b_i \in B\} \)
3:  while \( C_k \) not empty do
4:    \( \text{COMPUTE-SUPPORT}(D, C_k) \)
5:    for all \( c \in C_k \) do
6:      if \( \text{Supp}(c) < \text{min_support} \) then
7:        delete \( c \) from \( C_k \)
8:      end if
9:    end for
10:  \( F_k \leftarrow C_k \)
11:  \( C_{k+1} \leftarrow \text{GENERATE-CANDIDATES}(F_k) \)
12:  \( k \leftarrow k + 1 \)
13:  end while
14:  return \( F_1 \cup F_2 \cup \ldots \cup F_{k-1} \)
A database $D$, a language $\mathcal{L}$;

sentences $\varphi, \Phi \in \mathcal{L}$;

a frequency criterion $q(\varphi) \in \{\text{true, false}\}$;

a monotone specialization/generalization relation: $\varphi \preceq \Phi$

$q(\Phi) = \text{true} \Rightarrow q(\varphi) = \text{true}$
Abstract problem formulation

Generalization of the Apriori algorithm

1: \( C_1 \leftarrow \{ \varphi \in \mathcal{L} | \text{there is no } \varphi' \text{ such that } \varphi' \prec \varphi \} \)
2: \( i \leftarrow 1 \)
3: \( \text{while } C_i \text{ not empty do} \)
4: \( F_i \leftarrow \{ \varphi \in C_i | q(\varphi) = \text{true} \} \)
5: \( C_{i+1} \leftarrow \{ \varphi \in \mathcal{L} | \forall \varphi' \prec \varphi \text{ we have } \varphi' \in \bigcup_{j \leq i} F_j \} \setminus \bigcup_{j \leq i} C_j \)
6: \( i \leftarrow i + 1 \)
7: \( \text{end while} \)
8: \( \text{return } F_1 \cup F_2 \cup \ldots \cup F_{k-1} \)
Abstract problem formulation

Subgraph mining
Subgraph mining
The Eclat algorithm is much quicker than the Apriori algorithm.

It is a depth-first algorithm ⇒ lower memory consumption than the Apriori algorithm.

Based on the fact that the powerset $\mathcal{P}(B)$ forms a lattice.

Uses so called prefix-based equivalence classes.

Uses the vertical representation of the database.
The prefix-based equivalence classes

Definition (equivalence relation)
Let $P$ be a set and $X, Y, Z \in P$ a relation $\equiv$ is called an equivalence relation if the relation is:

1. Reflexive: $X \equiv Y$.
2. Symmetric: if $X \equiv Y$ then $Y \equiv X$.
3. Transitive: if $X \equiv Y$ and $Y \equiv Z$ then $X \equiv Z$.

Definition (prefix-based equivalence class)
Let $\pi$ be a function $\pi : \mathcal{P}(B) \times N \rightarrow \mathcal{P}(B)$ such that $\pi(X, l) = X[1 : l]$. The prefix-based equivalence $\pi_l$ on the lattice $\mathcal{L}$ is defined as follows: $\forall U, V \subseteq B, U \equiv_{\pi_l} V \iff \pi(U, l) = \pi(V, l)$. 
Lemma

Each equivalence class $[W]_\pi$ induced by the equivalence relation $\pi$ is a sub-lattice of powerset $\mathcal{P}(B)$. 
The vertical representation of a database

Database $D$:

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${1, 2, 3, 4}$</td>
</tr>
<tr>
<td>2</td>
<td>${3, 5}$</td>
</tr>
<tr>
<td>3</td>
<td>${1, 3, 4}$</td>
</tr>
<tr>
<td>4</td>
<td>${1, 2}$</td>
</tr>
<tr>
<td>5</td>
<td>${1, 3, 4, 5}$</td>
</tr>
<tr>
<td>6</td>
<td>${1, 2, 3, 4, 5}$</td>
</tr>
</tbody>
</table>

Vertical representation of the database is a set of pairs $\{(b_i, T(b_i))\}$, where $b_i \in B$ and $T(b_i)$ is a transaction list of item $b_i$ (tidlist in short).

Example:

(1, $\{1, 3, 4, 5, 6\}$)
(2, $\{1, 4, 6\}$)
(3, $\{1, 2, 3, 5, 6\}$)
(4, $\{1, 3, 5, 6\}$)
(5, $\{5, 6\}$)
Support of a set $U \subset B$ can be computed using the tidlists:

$\text{Supp}(U) = |\bigcap_{b_i \in U} T(b_i)|$

- The Eclat algorithm makes a single scan of $D$.
- The datastructures are processor-cache friendly (it is just an array of numbers).
- Can be implemented using bitmaps.
### The Eclat algorithm

<table>
<thead>
<tr>
<th>itemset</th>
<th>{1}</th>
<th>{2}</th>
<th>{3}</th>
<th>{4}</th>
<th>{5}</th>
</tr>
</thead>
<tbody>
<tr>
<td>TID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Frequent | ×     | ×     | ×     | ×     | ×     | ×     |
| itemset  | \{1, 2\} | \{1, 3\} | \{1, 4\} | \{1, 5\} | \{2, 3\} | \{2, 4\} |
| TID      | 1     | 1     | 1     | 5     | 1     | 1     |
|          | 4     | 3     | 3     | 6     | 6     | 6     |
|          | 6     | 5     | 5     |       |       |       |
|          |       | 6     | 6     |       |       |       |

| itemset  | \{2, 5\} | \{3, 4\} | \{3, 5\} | \{4, 5\} |
| TID      | 6     | 1     | 2     | 5     |
|          | 3     | 5     | 5     | 6     |
|          | 5     | 6     |       |       |
The Eclat algorithm

**ECLAT-BOTTOM-UP** (**In:** Atoms \( \mathcal{A} \), **In:** Itemset \( P \), **Out:** Set \( F \))

1. for all atom \( a_i \in \mathcal{A} \) do
2. \( \mathcal{A}_i \leftarrow \emptyset \)
3. for all atom \( a_j \in \mathcal{A}, a_i < a_j \) do
4. \( \text{if } |\mathcal{I}(P \cup \{a_j\})| \geq \text{min\_support} \text{ then} \)
5. \( \mathcal{A}_i \leftarrow \mathcal{A}_i \cup \{a_j\} \)
6. \( f \leftarrow P \cup \{a_j\} \)
7. \( F \leftarrow F \cup f \)
8. end if
9. end for
10. ECLAT-BOTTOM-UP(\( \mathcal{A}_i, P \cup \{a_i\}, F \))
11. end for
Optimalizations to the Eclat algorithm

1. The “closed itemset” optimalization
2. Dynamic items ordering
3. The diffsets
## Optimalizations – The “closed itemset” optimization

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{...}</td>
</tr>
<tr>
<td>2</td>
<td>{...}</td>
</tr>
<tr>
<td>3</td>
<td>{1,2}</td>
</tr>
<tr>
<td>4</td>
<td>{1,2}</td>
</tr>
<tr>
<td>5</td>
<td>{1,2,3,4,5}</td>
</tr>
<tr>
<td>6</td>
<td>{1,2,3,4,5}</td>
</tr>
</tbody>
</table>
Let have a prefix \( \Pi = \{\pi_1, \ldots, \pi_k\}, \pi_i \in B \), and extensions \( \Sigma = \{\sigma_1, \ldots, \sigma_l\}, \sigma_i \in B \), and
\[
\text{Supp}^*(\Pi \cup \{\sigma_1\}) \leq \text{Supp}^*(\Pi \cup \{\sigma_2\}) \leq \ldots \leq \text{Supp}^*(\Pi \cup \{\sigma_l\}).
\]
For the prefix \( \Pi \):

Using the order it follows:
Let have a prefix $\Pi = \{\pi_1, \ldots, \pi_k\}, \pi_i \in B$, and extensions $\Sigma = \{\sigma_1, \ldots, \sigma_l\}, \sigma_i \in B$, and

$$\text{Supp}^*(\Pi \cup \{\sigma_1\}) \leq \text{Supp}^*(\Pi \cup \{\sigma_2\}) \leq \ldots \leq \text{Supp}^*(\Pi \cup \{\sigma_l\}).$$

For the prefix $\Pi$:

1. consider $\sigma_1, \sigma_2, \ldots, \sigma_l$ as extensions (in that order)

Using the order it follows:

1. the smallest portion of the database gets the largest portion of the search space.
Let have a prefix $\Pi = \{\pi_1, \ldots, \pi_k\}$, $\pi_i \in B$, and extensions $\Sigma = \{\sigma_1, \ldots, \sigma_l\}$, $\sigma_i \in B$, and $\text{Supp}^*(\Pi \cup \{\sigma_1\}) \leq \text{Supp}^*(\Pi \cup \{\sigma_2\}) \leq \ldots \leq \text{Supp}^*(\Pi \cup \{\sigma_l\})$.

For the prefix $\Pi$:

1. consider $\sigma_1, \sigma_2, \ldots, \sigma_l$ as extensions (in that order)
2. consider $\sigma_l, \sigma_{l-1}, \ldots, \sigma_1$ as extensions (in that order)

Using the order it follows:

1. the smallest portion of the database gets the largest portion of the search space.
2. the largest portion of the database gets the largest portion of the search space.
Definition

Let \( U \subset B \) be an itemset and \( i \in B - U \) an item. \( T(U) \) denotes a set of transaction id’s. *Difference set* (or diffset in short) is

\[
\mathcal{D}(U \cup \{i\}) = T(U) - T(\{i\})
\]

The support can be computed as:

\[
\text{Supp}(U \cup \{i\}) = \text{Supp}(U) - |\mathcal{D}(U \cup \{i\})|
\]

Computation of diffsets:

\[
\mathcal{D}(U \cup \{i\} \cup \{j\}) = T(U \cup \{i\}) - T(U \cup \{j\})
= T(U \cup \{i\}) - T(U \cup \{j\}) + T(U) - T(U)
= (T(U) - T(U \cup \{j\})) - (T(U) - T(U \cup \{i\}))
= \mathcal{D}(U \cup \{j\}) - \mathcal{D}(U \cup \{i\})
\]
The FPGrowth algorithm

- Very similar to the Eclat algorithm
- the only difference is the used datastructure
- The datastructure (called FPTrie) is a prefix tree with linked nodes
The FPGrowth algorithm - the FP Trie

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1, 3, 4}</td>
</tr>
<tr>
<td>2</td>
<td>{5, 4, 6}</td>
</tr>
<tr>
<td>3</td>
<td>{1, 3, 5, 6}</td>
</tr>
<tr>
<td>4</td>
<td>{1, 3, 2}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
Each node is a tuple \((\text{item, support, upper-link, header-link, children})\).
### 6’s conditional tree

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{5, 4}</td>
<td>1</td>
</tr>
<tr>
<td>{5, 3, 1}</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
H \\
R = (\emptyset, 2, \text{null}, \text{null}, \{5\})
\]

- \((5, l) \downarrow \rightarrow (5, 2, u, \text{null}, \{1, 4\})\)
- \((1, l) \downarrow \rightarrow (1, 1, u, \text{null}, \{3\}) \rightarrow (4, 1, u, \text{null}, \{\})\)
- \((3, l) \downarrow \rightarrow (3, 1, u, \text{null}, \{\})\)
**The FP-Growth algorithm**

\[ \text{FP-Growth}(\text{In: FP-Tree } \text{tree}, \text{In: Itemset } I) \]

1. **if** \( \text{tree} \) contains only single path \( P \) **then**
   - for all combination \( c \) in path \( P \) **do**
     - \( s \leftarrow \min\{s | i \in c, s = \text{Supp}(i)\} \)
     - **if** \( s \geq \text{min\_support} \) **then**
       - generate pattern \( c \cup I \) with support \( s \)
     - **end if**
   - **end for**
2. **else**
   - for all \( a_i \) in the header of \( \text{tree} \) **do**
     - **if** \( \text{Supp}(a_i) \geq \text{min\_support} \) **then**
       - generate pattern \( c = \{a_i\} \cup I \) with support \( \text{Supp}(a_i) \)
     - **end if**
   - construct \( c \)'s conditional pattern base and then \( c \)'s conditional \text{FP-Tree} \( \text{tree}_c \)
   - **if** \( \text{Size}(\text{tree}_c) \neq 0 \) **then**
     - \( \text{FP-Growth}(\text{tree}_c, c) \)
   - **end if**
   **end for**
**end if**