Association Rule Mining

Generating assoc. rules from frequent itemsets

- Assume that we have discovered the frequent itemsets and their support
- How do we generate association rules?
- Frequent itemsets:

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
<tr>
<td>{1,3}</td>
<td>2</td>
</tr>
<tr>
<td>{2,3}</td>
<td>2</td>
</tr>
<tr>
<td>{2,5}</td>
<td>3</td>
</tr>
<tr>
<td>{3,5}</td>
<td>2</td>
</tr>
<tr>
<td>{2,3,5}</td>
<td>2</td>
</tr>
</tbody>
</table>

For each frequent itemset \( l \) find all nonempty subsets \( s \). For each \( s \) generate rule \( s \Rightarrow l-s \) if \( \sup(l)/\sup(s) \geq \min_{\text{conf}} \).

Example: for \{2,3,5\}, \( \min_{\text{conf}} = 75\% \)
- \{2,3\} \( \Rightarrow 5 \) \( \checkmark \)
- \{2,5\} \( \Rightarrow 3 \) \( \times \)
- \{3,5\} \( \Rightarrow 2 \) \( \checkmark \)
Discovering Rules

- **Naïve Algorithm**
  
  ```
  for each frequent itemset l do
    for each subset c of l do
      if (support(l) / support(l - c) >= minconf) then
        output the rule (l - c) \(\Rightarrow\) c,
        with confidence = support(l) / support(l - c)
        and support = support(l)
  ```

Discovering Rules (2)

- **Lemma.** If consequent c generates a valid rule, so do all subsets of c. (e.g. \(X \Rightarrow YZ\), then \(XY \Rightarrow Z\) and \(XZ \Rightarrow Y\))

- **Example:** Consider a frequent itemset ABCDE

  If ACDE \(\Rightarrow\) B and ABCE \(\Rightarrow\) D are the only one-consequent rules with minimum support confidence, then

  ACE \(\Rightarrow\) BD is the only other rule that needs to be tested
Is Apriori Fast Enough? — Performance Bottlenecks

- The core of the Apriori algorithm:
  - Use frequent \((k - 1)\)-itemsets to generate candidate frequent \(k\)-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets

- The bottleneck of Apriori: candidate generation
  - Huge candidate sets:
    - \(10^4\) frequent 1-itemset will generate \(10^7\) candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., \(\{a_1, a_2, ..., a_{100}\}\), one needs to generate \(2^{100} \approx 10^{30}\) candidates.
  - Multiple scans of database:
    - Needs \((n + 1)\) scans, \(n\) is the length of the longest pattern

FP-growth: Mining Frequent Patterns Without Candidate Generation

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - highly condensed, but complete for frequent pattern mining
  - avoid costly database scans

- Develop an efficient, FP-tree-based frequent pattern mining method
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Avoid candidate generation: sub-database test only!
FP-tree Construction from a Transactional DB

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
<th>(ordered) frequent items</th>
<th>min_support = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{f, a, c, d, g, i, m, p}</td>
<td>{f, c, a, m, p}</td>
<td>f: 4</td>
</tr>
<tr>
<td>200</td>
<td>{a, b, c, f, l, m, o}</td>
<td>{f, c, a, b, m}</td>
<td>c: 4</td>
</tr>
<tr>
<td>300</td>
<td>{b, f, h, j, o}</td>
<td>{f, b}</td>
<td>a: 3</td>
</tr>
<tr>
<td>400</td>
<td>{b, c, k, s, p}</td>
<td>{c, b, p}</td>
<td>b: 3</td>
</tr>
<tr>
<td>500</td>
<td>{a, f, c, e, l, p, m, n}</td>
<td>{f, c, a, m, p}</td>
<td>m: 3</td>
</tr>
</tbody>
</table>

Steps:

1. Scan DB once, find frequent 1-itemsets (single item patterns)
2. Order frequent items in descending order of their frequency
3. Scan DB again, construct FP-tree

FP-tree Construction

<table>
<thead>
<tr>
<th>TID</th>
<th>freq. Items bought</th>
<th>min_support = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{f, c, a, m, p}</td>
<td>f: 4</td>
</tr>
<tr>
<td>200</td>
<td>{f, c, a, b, m}</td>
<td>c: 4</td>
</tr>
<tr>
<td>300</td>
<td>{f, b}</td>
<td>a: 3</td>
</tr>
<tr>
<td>400</td>
<td>{c, p, b}</td>
<td>b: 3</td>
</tr>
<tr>
<td>500</td>
<td>{f, c, a, m, p}</td>
<td>m: 3</td>
</tr>
</tbody>
</table>

Item frequency:

<table>
<thead>
<tr>
<th>Item</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>m</td>
<td>3</td>
</tr>
<tr>
<td>p</td>
<td>3</td>
</tr>
</tbody>
</table>
FP-tree Construction

TID  freq. Items bought
100  {f, c, a, m, p}
200  {f, c, a, b, m}
300  {f, b}
400  {c, p, b}
500  {f, c, a, m, p}

min_support = 3
Item frequency
f  4
c  4
a  3
b  3
m  3
p  3

FP-tree Construction

TID  freq. Items bought
100  {f, c, a, m, p}
200  {f, c, a, b, m}
300  {f, b}
400  {c, p, b}
500  {f, c, a, m, p}

min_support = 3
Item frequency
f  4
c  4
a  3
b  3
m  3
p  3
FP-tree Construction

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
<th>min_support = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>[f, c, a, m, p]</td>
<td>f: 4</td>
</tr>
<tr>
<td>200</td>
<td>[f, c, a, b, m]</td>
<td>c: 4</td>
</tr>
<tr>
<td>300</td>
<td>[f, b]</td>
<td>a: 3</td>
</tr>
<tr>
<td>400</td>
<td>[c, p, b]</td>
<td>b: 3</td>
</tr>
<tr>
<td>500</td>
<td>[f, c, a, m, p]</td>
<td>m: 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>p: 3</td>
</tr>
</tbody>
</table>

**Header Table**

<table>
<thead>
<tr>
<th>Item</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>m</td>
<td>2</td>
</tr>
<tr>
<td>p</td>
<td>2</td>
</tr>
</tbody>
</table>

**Benefits of the FP-tree Structure**

- Completeness:
  - never breaks a long pattern of any transaction
  - preserves complete information for frequent pattern mining

- Compactness
  - reduce irrelevant information—in frequent items are gone
  - frequency descending ordering: more frequent items are more likely to be shared
  - never be larger than the original database (if not count node-links and counts)
  - Example: For Connect-4 DB, compression ratio could be over 100
Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree
- Method
  - For each item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path (single path will generate all the combinations of its sub-paths, each of which is a frequent pattern)

Mining Frequent Patterns Using the FP-tree (cont’d)

- Start with last item in order (i.e., p).
- Follow node pointers and traverse only the paths containing p.
- Accumulate all of transformed prefix paths of that item to form a conditional pattern base

Conditional pattern base for p

\[ fc:am:2, \, cb:1 \]

Construct a new FP-tree based on this pattern, by merging all paths and keeping nodes that appear \( \geq \text{sup} \) times. This leads to only one branch \( c:3 \)
Thus we derive only one frequent pattern cont. p. Pattern \( cp \)
Mining Frequent Patterns Using the FP-tree (cont’d)

- Move to next least frequent item in order, i.e., \( m \)
- Follow node pointers and traverse only the paths containing \( m \).
- Accumulate all of transformed prefix paths of that item to form a **conditional pattern base**

\[
\text{m-conditional pattern base: } \quad \text{fca:2, fcab:1}
\]

\[
\begin{align*}
\{\} & \Rightarrow m, \\
f:3 & \Rightarrow \text{fm, cm, am,} \\
c:3 & \Rightarrow \text{fcm, fam, cam,} \\
a:3 & \Rightarrow \text{fcam}
\end{align*}
\]

\( m \)-conditional FP-tree (contains only path \( \text{fca:3} \))

Properties of FP-tree for Conditional Pattern Base Construction

- **Node-link property**
  - For any frequent item \( a_i \), all the possible frequent patterns that contain \( a_i \) can be obtained by following \( a_i \)'s node-links, starting from \( a_i \)'s head in the FP-tree header

- **Prefix path property**
  - To calculate the frequent patterns for a node \( a_i \) in a path \( P \), only the prefix sub-path of \( a_i \) in \( P \) need to be accumulated, and its frequency count should carry the **same count** as node \( a_i \).
Conditional Pattern-Bases for the example

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional pattern-base</th>
<th>Conditional FP-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{ (fca:2), (cb:1) }</td>
<td>{(c:3)}</td>
</tr>
<tr>
<td>m</td>
<td>{ (fca:2), (fcab:1) }</td>
<td>{(f:3, c:3, a:3)}</td>
</tr>
<tr>
<td>b</td>
<td>{ (fca:1), (f:1), (c:1) }</td>
<td>Empty</td>
</tr>
<tr>
<td>a</td>
<td>{ (fc:3) }</td>
<td>{(f:3, c:3)}</td>
</tr>
<tr>
<td>c</td>
<td>{ (f:3) }</td>
<td>{(f:3)}</td>
</tr>
<tr>
<td>f</td>
<td>Empty</td>
<td>Empty</td>
</tr>
</tbody>
</table>

Principles of Frequent Pattern Growth

- **Pattern growth property**
  - Let $\alpha$ be a frequent itemset in DB, $B$ be $\alpha$’s conditional pattern base, and $\beta$ be an itemset in $B$. Then $\alpha \cup \beta$ is a frequent itemset in DB iff $\beta$ is frequent in $B$.

- “$abcdef$” is a frequent pattern, if and only if
  - “$abcde$” is a frequent pattern, and
  - “$f$” is frequent in the set of transactions containing “$abcde$”
Why Is Frequent Pattern Growth Fast?

- Performance studies show
  - FP-growth is an order of magnitude faster than Apriori, and is also faster than tree-projection

- Reasoning
  - No candidate generation, no candidate test
  - Uses compact data structure
  - Eliminates repeated database scan
  - Basic operation is counting and FP-tree building

FP-growth vs. Apriori: Scalability With the Support Threshold

Data set T25I20D10K