Efficient Mining of Top-K Sequential Rules

Philippe Fournier-Viger\textsuperscript{1}
Vincent Shin-Mu Tseng\textsuperscript{2}

\textsuperscript{1}University of Moncton, Canada
\textsuperscript{2}National Cheng Kung University, Taiwan

18\textsuperscript{th} December 2011 – ADMA 2011 - Beijing
Sequence Database

- A set of sequences.
- Each **sequence** is an ordered list of transactions.
- A **transaction** is an unordered set of items (symbols), considered to occur simultaneously.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
</tr>
</tbody>
</table>

- **Ex.**: click-stream data, market basket analysis, stock market data, bioinformatics, e-learning...
Sequential Pattern Mining

- **SPM** finds subsequences that are common to more than \( \text{minsup} \) sequences (Spade, PrefixSpan, GSP...).

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
</tr>
</tbody>
</table>

- Ex.: \{b\}, \{f\}, \{e\} is a seq. pattern with support = 50%.
- However, this pattern is misleading because \{b\}, \{f\} also appear 50% of the time without \{e\} following.
- **To solve this problem**, we would need to consider the **confidence** of sequential patterns.
Sequential Rule Mining

• A sequential rule typically has the form $X \rightarrow Y$ and has a **confidence** and a **support**.

• **Algorithms for mining seq. rules** in a (1) single sequence, (2) across sequences or (3) **common to multiple sequences**.

• **Several applications**: stock market analysis (Das et al., 1998; Hsieh et al., 2006), weather observation (Hamilton & Karimi, 2005), drought management (Harms et al. 2002), alarm analysis, etc.
Mining sequential rule common to several sequences

• RuleGen proposed by Zaki (2001).
• Rules of the form $X \rightarrow Y$ where $X$ and $Y$ are sequential patterns.
• Ex.: $\{a\},\{b\},\{c\} \rightarrow \{d, e\}$
• **Problem**: rules can be too specific.
  – unlikely to match with a new sequence,
  – can have a low support value because they are very specific, despite that they may describe a common situation.
  – ...
The definition used in this paper

- A **sequential rule** $X \Rightarrow Y$ is a relationship between two disjoint and unordered itemsets $X, Y$.
- A sequential rule $X \Rightarrow Y$ has **two properties**:
  - **Support**: the number of sequences where $X$ occurs before $Y$, divided by the number of sequences.
  - **Confidence** the number of sequences where $X$ occurs before $Y$, divided by the number of sequences where $X$ occurs.
- **The task**: finding all rules with a support and confidence not less than user-defined thresholds $minSup$ and $minConf$. 
Example

\[$\text{minSup} = 0.5 \text{ and } \text{minConf} = 0.5:\$]

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
</tr>
</tbody>
</table>

\[\rightarrow\]

<table>
<thead>
<tr>
<th>ID</th>
<th>Rule</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>{a, b, c} $\Rightarrow$ {e}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>r2</td>
<td>{a} $\Rightarrow$ {c, e, f}</td>
<td>0.5</td>
<td>0.66</td>
</tr>
<tr>
<td>r3</td>
<td>{a, b} $\Rightarrow$ {e, f}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>r4</td>
<td>{b} $\Rightarrow$ {e, f}</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>r5</td>
<td>{a} $\Rightarrow$ {e, f}</td>
<td>0.75</td>
<td>1.0</td>
</tr>
<tr>
<td>r6</td>
<td>{c} $\Rightarrow$ {f}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>r7</td>
<td>{a} $\Rightarrow$ {b}</td>
<td>0.5</td>
<td>0.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A sequence database

Some rules found
Current Algorithms

- **CMRules**: An association rule mining based algorithm for the discovery of sequential rules.
- **CMDeo**: An Apriori based algorithm for the discovery of sequential rules.
- **RuleGrowth**: A pattern-growth based algorithm. *(current best)*
Find larger rules by recursively scanning the database for adding a single item at a time to the left or right part of each rule (these processes are called *left* and *right expansions*).
The problem of setting $\textit{minsup}$

- **Scenario:** A user wants to discover the top 1000 rules from a database and do not want to find more than 2000 rules.
- For $\textit{BMSWebView1}$, the range of $\textit{minsup}$ values that will satisfy the user is 0.0011 to 0.0009.
- A user having no a priori knowledge of the database has only a 0.02 % chance of selecting a $\textit{minsup}$ value that will make him satisfied.
- Too high, not enough rules.
- Too low, performance deteriorates.
TopSeqRules

Main idea

• set $minsup = 0$.
• use the RuleGrowth strategy for rule generation.
• keep a set $L$ that contains the current top-$k$ rules found until now.
• when $k$ rules are found, raise $minsup$ to the lowest support of the rules in $L$.
• after that, for each rules added to $L$, raise the $minsup$ threshold.
TopSeqRules (2)

• The resulting algorithm has poor execution time because the search space is too large.

\[ 3^d - 2^d + 1 \]

• **Observation:** if we can find rules with higher support first, we can raise \textit{minsup} more quickly and prune the search space.

• **How to define what is the most promising?** Our experiment show that the support is a good choice.

• We added a set \( R \) containing the \( k \) rules having the highest support.
TopSeqRules (3) - optimizations

• We found that the choice of data structures for implementing L and R is also very important:
  – $L$: fibonacci heap: $O(1)$ amortized time for insertion and minimum, and $O(\log(n))$ for deletion.
  – $R$: red-black tree: $O(\log(n))$ worst case time complexity for insertion, deletion, min and max.

• Merging database scans
Experimental evaluation

- Three real-life datasets:

| Datasets       | |S|   | |I|   | Avg. item count / sequence | Type of data       |
|----------------|----------------|----------------|---------------------------------|-------------------|
| BMSWebView1    | 59601          | 497            | 2.5 ($\sigma = 4.85$)           | click-stream from |
| Sign           | 730            | 310            | 93.39 ($\sigma = 4.59$)         | language utter     |
| Snake          | 163            | 20             | 60.615 ($\sigma = 0.89$)        | protein sequence   |
Results – influence of $k$

- Execution time and maximum memory usage grow linearly with $k$. 

![Graphs showing execution time and memory usage](image)
Results – influence of \textit{minconf}

• Execution time and memory usage increase exponentially when \textit{minconf} increases because more rules have to be generated.
Results – influence of database size

• Execution time and memory increases slowly if the number of rules stay more or less the same.
Performance comparison

Fig. 9. Performance comparison for optimal parameters selection
When \textit{minsup} is chosen optimally, RuleGrowth has slightly better performance.

However, setting \textit{minsup} is very difficult.

If \textit{minsup} is set too low, RuleGrowth will not find any rule.

If \textit{minsup} is set too high, too many rules will be found and the performance deteriorates.

\begin{table}[h]
\centering
\begin{tabular}{|l|c|c|c|}
\hline
\textbf{Datasets} & \textit{minsup} for k=1000 & \textit{minsup} for k=2000 & \textbf{interval size} \\
\hline
BMSWebView1 & 0.0011 & 0.0009 & 0.0002 \\
Sign & 0.420 & 0.384 & 0.036 \\
Snake & 0.960 & 0.944 & 0.016 \\
\hline
\end{tabular}
\caption{Interval of \textit{minsup} values to find the top 1000 to 2000 rules for each dataset}
\end{table}
Conclusion

• We proposed an algorithm that let the user set $k$, the number of rules to be found.
• Excellent scalability: execution time linearly increases with $k$.
• the algorithm has no problem running in reasonable time and memory limits for $k$ values of up to 5000 for all datasets.
Thank you. Questions?

Open source Java data mining software, 43 algorithms
http://www.phillippe-fournier-viger.com/spmf/

Thanks to the FQRNT funding programs.

Thanks to the organizers of ADMA 2011!