Data Mining
Part 3. Associations Rules

3.2 Efficient Frequent Itemset Mining Methods

Fall 2009

Instructor: Dr. Masoud Yaghini
Outline

- Apriori Algorithm
- Generating Association Rules from Frequent Itemsets
- FP-growth Algorithm
- References
Apriori Algorithm
Apriori Algorithm

- Complete search approach:
  - List all possible itemsets $M = 2^d - 1$
  - Count the support of each itemset by scanning the database
  - Eliminate itemsets that fail the $min\_sup$

$\Rightarrow$ Computationally prohibitive!

- Reduce the number of candidates ($M$)
  - Use pruning techniques to reduce $M$
  - e.g. Apriori algorithm
Apriori Algorithm

- Finding frequent itemsets using candidate generation
- Proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for **Boolean association rules**.
- The name of the algorithm is based on the fact that the algorithm uses *prior knowledge* of frequent itemset properties.
- Apriori employs an iterative approach, where k-itemsets are used to explore (k+1)-itemsets.
Apriori Algorithm

- Let \( k = 1 \)
- **Scan** DB once to get frequent \( k \)-itemset
- **Repeat** until no new frequent or candidate itemsets are identified
  - **Generate** length \((k+1)\) candidate itemsets from length \( k \) frequent itemsets
  - **Prune** candidate itemsets containing subsets of length \( k \) that are infrequent
  - **Scan** DB to **count** the support of each candidate
  - **Eliminate** candidates that are infrequent, leaving only those that are frequent
Apriori Algorithm

- **Apriori property:**
  - All nonempty subsets of a frequent itemset must also be frequent.

- **Apriori pruning principle:**
  - If there is any itemset which is infrequent, its superset is not frequent either and it should not be generated as a candidate.
Apriori Principle

Efficient Frequent Itemset Mining Methods
Apriori Principle

Efficient Frequent Itemset Mining Methods
Example: Apriori Algorithm

- Transactional data for an *AllElectronics*
- the minimum support count required is 2, that is, \( \text{min}_\text{sup} = 2 \).

<table>
<thead>
<tr>
<th>TID</th>
<th>List of item IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>1, 12, 15</td>
</tr>
<tr>
<td>T200</td>
<td>12, 14</td>
</tr>
<tr>
<td>T300</td>
<td>12, 13</td>
</tr>
<tr>
<td>T400</td>
<td>11, 12, 14</td>
</tr>
<tr>
<td>T500</td>
<td>11, 13</td>
</tr>
<tr>
<td>T600</td>
<td>12, 13</td>
</tr>
<tr>
<td>T700</td>
<td>11, 13</td>
</tr>
<tr>
<td>T800</td>
<td>11, 12, 13, 15</td>
</tr>
<tr>
<td>T900</td>
<td>11, 12, 13</td>
</tr>
</tbody>
</table>

Efficient Frequent Itemset Mining Methods
Example: Apriori Algorithm

- **Step 1:**
  - generate a candidate set of 1-itemsets, C1.
  - scans all of the transactions in order to count the number of occurrences of each item.

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{I1}</td>
<td>6</td>
</tr>
<tr>
<td>{I2}</td>
<td>7</td>
</tr>
<tr>
<td>{I3}</td>
<td>6</td>
</tr>
<tr>
<td>{I4}</td>
<td>2</td>
</tr>
<tr>
<td>{I5}</td>
<td>2</td>
</tr>
</tbody>
</table>
Example: Apriori Algorithm

Step 2:

- Determine the set of frequent 1-itemsets, L1.
- All of the candidates in C1 satisfy minimum support.

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{I1}</td>
<td>6</td>
</tr>
<tr>
<td>{I2}</td>
<td>7</td>
</tr>
<tr>
<td>{I3}</td>
<td>6</td>
</tr>
<tr>
<td>{I4}</td>
<td>2</td>
</tr>
<tr>
<td>{I5}</td>
<td>2</td>
</tr>
</tbody>
</table>
Example: Apriori Algorithm

- Step 3:
  - generate a candidate set of 2-itemsets, $C_2$
Example: Apriori Algorithm

- Step 4:
  - scans all of the transactions in order to count the number of occurrences of each item in $C_2$. 

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{I1, I2}</td>
<td>4</td>
</tr>
<tr>
<td>{I1, I3}</td>
<td>4</td>
</tr>
<tr>
<td>{I1, I4}</td>
<td>1</td>
</tr>
<tr>
<td>{I1, I5}</td>
<td>2</td>
</tr>
<tr>
<td>{I2, I3}</td>
<td>4</td>
</tr>
<tr>
<td>{I2, I4}</td>
<td>2</td>
</tr>
<tr>
<td>{I2, I5}</td>
<td>2</td>
</tr>
<tr>
<td>{I3, I4}</td>
<td>0</td>
</tr>
<tr>
<td>{I3, I5}</td>
<td>1</td>
</tr>
<tr>
<td>{I4, I5}</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: Apriori Algorithm

- Step 5:
  - Determine the set of frequent 2-itemsets, $L_2$.

\[
\begin{array}{|c|c|}
\hline
\text{Itemset} & \text{Sup. count} \\
\hline
\{I_1, I_2\} & 4 \\
\{I_1, I_3\} & 4 \\
\{I_1, I_5\} & 2 \\
\{I_2, I_3\} & 4 \\
\{I_2, I_4\} & 2 \\
\{I_2, I_5\} & 2 \\
\hline
\end{array}
\]
Example: Apriori Algorithm

- Step 6: generate a candidate set of 3-itemsets, \( C_3 \)

(a) Join: \( C_3 = L_2 \times L_2 = \{\{11, 12\}, \{11, 13\}, \{11, 15\}, \{12, 13\}, \{12, 14\}, \{12, 15\}\} \times \{\{11, 12\}, \{11, 13\}, \{11, 15\}, \{12, 13\}, \{12, 14\}, \{12, 15\}\} \\
= \{\{11, 12, 13\}, \{11, 12, 15\}, \{11, 13, 15\}, \{12, 13, 14\}, \{12, 13, 15\}, \{12, 14, 15\}\}.

(b) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?

- The 2-item subsets of \{11, 12, 13\} are \{11, 12\}, \{11, 13\}, and \{12, 13\}. All 2-item subsets of \{11, 12, 13\} are members of \( L_2 \). Therefore, keep \{11, 12, 13\} in \( C_3 \).

- The 2-item subsets of \{11, 12, 15\} are \{11, 12\}, \{11, 15\}, and \{12, 15\}. All 2-item subsets of \{11, 12, 15\} are members of \( L_2 \). Therefore, \{11, 12, 15\} in \( C_3 \).

- The 2-item subsets of \{11, 13, 15\} are \{11, 13\}, \{11, 15\}, and \{13, 15\}. \{13, 15\} is not a member of \( L_2 \), and so it is not frequent. Therefore, remove \{11, 13, 15\} from \( C_3 \).

- The 2-item subsets of \{12, 13, 14\} are \{12, 13\}, \{12, 14\}, and \{13, 14\}. \{13, 14\} is not a member of \( L_2 \), and so it is not frequent. Therefore, remove \{12, 13, 14\} from \( C_3 \).

- The 2-item subsets of \{12, 13, 15\} are \{12, 13\}, \{12, 15\}, and \{13, 15\}. \{13, 15\} is not a member of \( L_2 \), and so it is not frequent. Therefore, remove \{12, 13, 15\} from \( C_3 \).

- The 2-item subsets of \{12, 14, 15\} are \{12, 14\}, \{12, 15\}, and \{14, 15\}. \{14, 15\} is not a member of \( L_2 \), and so it is not frequent. Therefore, remove \{12, 14, 15\} from \( C_3 \).

(c) Therefore, \( C_3 = \{\{11, 12, 13\}, \{11, 12, 15\}\} \) after pruning.
Example: Apriori Algorithm

- Step 6: (cont.)
  - The resulting pruned version of C3

\[
\begin{array}{|c|c|}
\hline
\text{Generate } C_3 \text{ candidates from } L_2 \\
\{I1, I2, I3\} \\
\{I1, I2, I5\} \\
\hline
\end{array}
\]
The Apriori Algorithm

- **Step 7:**
  - scans all of the transactions in order to count the number of occurrences of each item in $C_3$.

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>${I_1, I_2, I_3}$</td>
<td>2</td>
</tr>
<tr>
<td>${I_1, I_2, I_5}$</td>
<td>2</td>
</tr>
</tbody>
</table>

Efficient Frequent Itemset Mining Methods
The Apriori Algorithm

- **Step 8:**
  - Determine the set of frequent 3-itemsets, $L_3$.

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup. count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{I_1, I_2, I_3}</td>
<td>2</td>
</tr>
<tr>
<td>{I_1, I_2, I_5}</td>
<td>2</td>
</tr>
</tbody>
</table>

Efficient Frequent Itemset Mining Methods
The Apriori Algorithm

- Step 9:
  - Generate a candidate set of 4-itemsets, C4.
  - Although the join results in \{\{I1, I2, I3, I5\}\}, this itemset is pruned because its subset \{\{I2, I3, I5\}\} is not frequent.
  - Thus, C4 = \emptyset, and the algorithm terminates, having found all of the frequent itemsets.
The Apriori Algorithm

- **Pseudo-code**:
  
  \( C_k \): Candidate itemset of size k  
  \( L_k \): frequent itemset of size k

  \( L_1 = \{ \text{frequent items} \} \);
  
  \textbf{for} (\( k = 1; \ L_k \neq \emptyset; \ k++ \)) \textbf{do begin}
  
  \( C_{k+1} = \text{candidates generated from} \ L_k; \)
  
  \textbf{for each} transaction \( t \) in database \textbf{do}
  
  increment the count of all candidates in \( C_{k+1} \)
  
  \( \text{that are contained in} \ t \)
  
  \( L_{k+1} = \text{candidates in} \ C_{k+1} \text{ with min\_support} \)
  
  \textbf{end}
  
  return \( \bigcup_k L_k \);
Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining $L_k$
  - Step 2: pruning

- Example of Candidate-generation
  - $L_3$={$abc, abd, acd, ace, bcd$}
  - Self-joining: $L_3 \times L_3$
    - $abcd$ from $abc$ and $abd$
    - $acde$ from $acd$ and $ace$
  - Pruning:
    - $acde$ is removed because $ade$ is not in $L_3$
  - $C_4$={$abcd$}
Generating Association Rules from Frequent Itemsets
Generating Association Rules

- Once the frequent itemsets from transactions in a database D have been found, it is straightforward to generate strong association rules from them.
- Where strong association rules satisfy both minimum support and minimum confidence.
- Confidence:

\[
\text{confidence}(A \Rightarrow B) = P(B|A) = \frac{\text{support\_count}(A \cup B)}{\text{support\_count}(A)}
\]
Generating Association Rules

- Association rules can be generated as follows:
  - For each frequent itemset \( I \), generate all nonempty subsets of \( I \).
  - For every nonempty subset \( S \) of \( I \), output the rule
    \[ S \Rightarrow (I - S) \]
    if
    \[ \frac{\text{support}_\text{count}(I)}{\text{support}_\text{count}(S)} \geq \text{min}\_\text{conf}, \]
    where \( \text{min}\_\text{conf} \) is the minimum confidence threshold.

- Because the rules are generated from frequent itemsets, each one automatically satisfies minimum support.
Example: Generating Association Rules

- Suppose the data contain the frequent itemset \( I = \{I_1, I_2, I_5\} \).
- What are the association rules that can be generated from \( I \)?
- The nonempty subsets of \( I \) are
  - \( \{I_1, I_2\}, \{I_1, I_5\}, \{I_2, I_5\}, \{I_1\}, \{I_2\}, \text{ and } \{I_5\} \).
Example: Generating Association Rules

- The resulting association rules are as shown below, each listed with its confidence:

  \[ I_1 \land I_2 \Rightarrow I_5, \quad \text{confidence} = \frac{2}{4} = 50\% \]
  \[ I_1 \land I_5 \Rightarrow I_2, \quad \text{confidence} = \frac{2}{2} = 100\% \]
  \[ I_2 \land I_5 \Rightarrow I_1, \quad \text{confidence} = \frac{2}{2} = 100\% \]
  \[ I_1 \Rightarrow I_2 \land I_5, \quad \text{confidence} = \frac{2}{6} = 33\% \]
  \[ I_2 \Rightarrow I_1 \land I_5, \quad \text{confidence} = \frac{2}{7} = 29\% \]
  \[ I_5 \Rightarrow I_1 \land I_2, \quad \text{confidence} = \frac{2}{2} = 100\% \]

- If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output,
Factors Affecting Complexity

● Choice of minimum support threshold
  – lowering support threshold results in more frequent itemsets
  – this may increase number of candidates and max length of frequent itemsets

● Dimensionality (number of items) of the data set
  – more space is needed to store support count of each item
  – if number of frequent items also increases, both computation and I/O costs may also increase

● Size of database
  – since Apriori makes multiple passes, run time of algorithm may increase with number of transactions

● Average transaction width
  – transaction width increases with denser data sets
  – This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)
FP-growth Algorithm
Bottleneck of Apriori Algorithm

- Apriori has two shortcoming:
  - Generating a huge number of candidate sets
  - Scanning lots of candidates
- To find frequent itemset $i_1i_2\ldots i_{100}$
  - # of scans: 100
  - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{0} = 2^{100}-1 = 1.27*10^{30}$!
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?
  - An interesting method in this attempt is called frequent-pattern growth
FP-growth

- **FP-growth**
  - finding frequent itemsets without candidate generation
  - adopts a divide-and-conquer strategy.

- **First**, FP-growth compresses the database representing frequent items into a frequent-pattern tree, or **FP-tree**,
  - which retains the itemset association information.

- **Then**, It divides the compressed database into a set of conditional databases (a special kind of projected database),
  - each associated with one frequent item or “pattern fragment,” and mines each such database separately.
Example: FP-growth

- Transactional data for an *AllElectronics*
- the minimum support count required is 2, that is, \( \text{min\_sup} = 2 \).

<table>
<thead>
<tr>
<th>TID</th>
<th>List of item IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>11, 12, 15</td>
</tr>
<tr>
<td>T200</td>
<td>12, 14</td>
</tr>
<tr>
<td>T300</td>
<td>12, 13</td>
</tr>
<tr>
<td>T400</td>
<td>11, 12, 14</td>
</tr>
<tr>
<td>T500</td>
<td>11, 13</td>
</tr>
<tr>
<td>T600</td>
<td>12, 13</td>
</tr>
<tr>
<td>T700</td>
<td>11, 13</td>
</tr>
<tr>
<td>T800</td>
<td>11, 12, 13, 15</td>
</tr>
<tr>
<td>T900</td>
<td>11, 12, 13</td>
</tr>
</tbody>
</table>
Example: FP-growth

1. Scan DB once, find frequent 1-itemset (same as Apriori)
2. Sort frequent items in frequency descending order, denoted $L$.
3. Construct FP-tree
Constructing FP-tree

- First, create the root of the tree, labeled with “null.”
- Scan database $D$ a second time.
- The items in each transaction are processed in $L$ order, and a branch is created for each transaction.
Constructing FP-tree

- Scanning the transactions:
  - The first transaction,
    - “T100: I1, I2, I5,” which contains three items (I2, I1, I5 in order), leads to the construction of the first branch of the tree with three nodes, (I2: 1), (I1:1), and (I5: 1), where I2 is linked as a child of the root, I1 is linked to I2, and I5 is linked to I1.
  - The second transaction,
    - T200, contains the items I2 and I4 in order, which would result in a branch where I2 is linked to the root and I4 is linked to I2.
    - This branch would share a common prefix, I2, with the existing path for T100.
    - We increment the count of the I2 node by 1, and create a new node, (I4: 1), which is linked as a child of (I2: 2).
Constructing FP-tree

- When considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links.
FP-tree

Efficient Frequent Itemset Mining Methods
Benefits of the FP-tree Structure

- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction

- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the count field)
Now we should mine the FP-tree.

- Start from each frequent length-1 pattern as an initial **suffix pattern**, 
- construct its **conditional pattern base** (which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern),
- then construct its **conditional FP-tree**, and perform mining recursively on such a tree.
- The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.
Example: Mining FP-tree

- First consider I5, which is the last item in \( L \)
  - I5 occurs in two branches of the FP-tree, the paths (I2, I1, I5: 1) and (I2, I1, I3, I5: 1).
  - Considering I5 as a suffix, its corresponding two prefix paths are (I2, I1: 1) and (I2, I1, I3: 1), which form its conditional pattern base
  - Its **conditional FP-tree** contains only a single path, (I2: 2, I1: 2); I3 is not included because its support count of 1 is less than the minimum support count 2.
  - The single path generates all the combinations of frequent patterns:
    - \( \{I2, I5: 2\} \), \( \{I1, I5: 2\} \), \( \{I2, I1, I5: 2\} \).
Example: Mining FP-tree

- the paths in which I5 occurs
Mining FP-tree

- For I4,
  - its two prefix paths form the conditional pattern base, 
    \[ \{ I2, I1: 1 \}, \{ I2: 1 \} \],
  - which generates a single-node conditional FP-tree, 
    \( I2: 2 \),
  - and derives one frequent pattern, \( \{ I2, I1: 2 \} \).
Mining FP-tree

- For I3

  - conditional pattern base is \{\{I2, I1: 2\}, \{I2: 2\}, \{I1: 2\}\}.
  - conditional FP-tree has two branches, (I2: 4, I1: 2) and (I1: 2), as shown in the following Figure.
  - which generates the set of patterns:
    \{\{I2, I3: 4\}, \{I1, I3: 4\}, \{I2, I1, I3: 2\}\}.

Efficient Frequent Itemset Mining Methods
Mining FP-tree

- For I1
  - conditional pattern base is \{I2: 4\},
  - whose FP-tree contains only one node, (I2: 4),
  - which generates one frequent pattern, \{I2, I1: 4\}. 

Efficient Frequent Itemset Mining Methods
## Mining FP-tree

- Summarization of the FP-tree:

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional Pattern Base</th>
<th>Conditional FP-tree</th>
<th>Frequent Patterns Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>I5</td>
<td>{{I2, I1: 1}, {I2, I1, I3: 1}}</td>
<td>⟨I2: 2, I1: 2⟩</td>
<td>{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}</td>
</tr>
<tr>
<td>I4</td>
<td>{{I2, I1: 1}, {I2: 1}}</td>
<td>⟨I2: 2⟩</td>
<td>{I2, I4: 2}</td>
</tr>
<tr>
<td>I1</td>
<td>{{I2: 4}}</td>
<td>⟨I2: 4⟩</td>
<td>{I2, I1: 4}</td>
</tr>
</tbody>
</table>
Why Is FP-Growth the Winner?

- Divide-and-conquer:
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases

- Other factors
  - no candidate generation, no candidate test
  - compressed database: FP-tree structure
  - no repeated scan of entire database
  - basic ops—counting local freq items and building sub FP-tree, no pattern search and matching
References
References

- J. Han, M. Kamber, *Data Mining: Concepts and Techniques*, Elsevier Inc. (2006). (Chapter 5)
The end