Association Rule Mining: FP-Growth

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
We have already learned the Apriori algorithm for association rule mining. In this lecture, we will discuss a faster algorithm called FP-growth.
Recall:

Let \( U \) be a set of items, referred to as the \textit{universal set}. An \textit{itemset}, denoted as \( I \), is a subset of \( U \).

The input to association rule mining is a set \( S \) of itemsets, called \textit{transactions}. The \textit{support} of an itemset \( I \) is the number of transactions in \( S \) that contain \( I \), namely:

\[
support(I) = |\{T \in S \mid I \subseteq T\}|
\]
Recall:

An association rule $R$ has the form

$$l_1 \rightarrow l_2$$

where both $l_1$ and $l_2$ are non-empty itemsets satisfying $l_1 \cap l_2 = \emptyset$.

The support of $R$, denoted as $sup(R)$, equals the support of the itemset $l_1 \cup l_2$.

The confidence of $R$ equals

$$conf(R) = \frac{support(l_1 \cup l_2)}{support(l_1)}.$$
Recall:

**Association Rule Mining**: Given (i) a set $S$ of transactions, and (ii) two constants $\text{minsupt}$ and $\text{minconf}$, we want to find all the association rules $R$ such that

\[
\text{sup}(R) \geq \text{minsupt} \quad \text{and} \quad \text{conf}(R) \geq \text{minconf}.
\]
We will use the following example in the subsequent discussion.

<table>
<thead>
<tr>
<th>tran. id</th>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, b</td>
</tr>
<tr>
<td>2</td>
<td>b, c, d</td>
</tr>
<tr>
<td>3</td>
<td>a, c, d, e</td>
</tr>
<tr>
<td>4</td>
<td>a, d, e</td>
</tr>
<tr>
<td>5</td>
<td>a, b, c</td>
</tr>
<tr>
<td>6</td>
<td>a, b, c, d</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td>8</td>
<td>a, b, c</td>
</tr>
<tr>
<td>9</td>
<td>a, b, d</td>
</tr>
<tr>
<td>10</td>
<td>b, c, e</td>
</tr>
</tbody>
</table>

**Remark:** This example was taken from the book “Introduction to Data Mining” (by Tan, Steinbach, and Kumar).
Recall that we found association rules with the Apriori algorithm in two steps:

1. Find all the frequent itemsets, i.e., itemsets $I$ with $sup(I) \geq \text{minsup}$. 
2. From each itemset $I$, find the association rules $R$ with $conf(R) \geq \text{minconf}$.

The new algorithm — FP-growth — we will learn today concerns only the first step (which is the performance bottleneck). The second step is the same as in the previous lecture.
Let us first observe the following recursive strategy to find frequent itemsets from $S$ (recall that $S$ is a set of transactions).

**find-frequent-itemsets ($S$)**
1. shrink $S$ by removing from every transaction of $S$ all items that appear less than $minsup$ times overall
2. shrink $S$ by removing empty transactions
3. **if** $S = \emptyset$, **return** $\emptyset$
4. **if** all transactions are the same, and contain a single item $z$, **return** $\{\{z\}\}$
5. else
6. $z =$ an arbitrary item in $S$
7. $S_1 =$ the set of transactions in $S$ that contain $z$
8. remove $z$ from every transaction of $S_1$
9. $F_1 =$ **find-frequent-itemsets** ($S_1$)
10. add $z$ to every itemset in $F_1$, and then add $\{z\}$ to $F_1$
11. $S_2 = S$
12. remove $z$ from every transaction of $S_2$
13. $F_2 =$ **find-frequent-itemsets** ($S_2$)
14. **return** $F_1 \cup F_2$
The FP-growth algorithm adopts the above strategy, and further improves efficiency by representing the input transactions in a compressed form (i.e., the FP-tree).

Next, we elaborate on the details.
First, sort the items in **descending** order of frequency. Any item that appears less than \textit{minsup} can be removed from all the transactions. These items form frequent size-1 itemsets.

\begin{center}
\begin{tabular}{|c|c|}
\hline
\text{tran. id} & \text{itemset} \\
\hline
1 & a, b \\
2 & b, c, d \\
3 & a, c, d, e \\
4 & a, d, e \\
5 & a, b, c \\
6 & a, b, c, d \\
7 & a \\
8 & a, b, c \\
9 & a, b, d \\
10 & b, c, e \\
\hline
\end{tabular}
\end{center}

We will assume \textit{minsup} = 2 in our example. Items \textit{a, b, c, d, e} appear 7, 6, 5, 4, 3 times, respectively. Hence, no items are removed.
From now on, we will regard each transaction as a string, where the items are sorted in descending order of their global frequencies.

<table>
<thead>
<tr>
<th>tran. id</th>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, b</td>
</tr>
<tr>
<td>2</td>
<td>b, c, d</td>
</tr>
<tr>
<td>3</td>
<td>a, c, d, e</td>
</tr>
<tr>
<td>4</td>
<td>a, d, e</td>
</tr>
<tr>
<td>5</td>
<td>a, b, c</td>
</tr>
<tr>
<td>6</td>
<td>a, b, c, d</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td>8</td>
<td>a, b, c</td>
</tr>
<tr>
<td>9</td>
<td>a, b, d</td>
</tr>
<tr>
<td>10</td>
<td>b, c, e</td>
</tr>
</tbody>
</table>

For example, transaction 1 is string \(ab\), transaction 2 is string \(bcd\), and so on. This allows us to define prefixes of transactions. For example, transaction 6 has 4 prefixes: \(a, ab, abc, abcd\).
Build a trie all the strings (a.k.a. transactions). This is a set of trees satisfying:

- Every prefix of each transaction corresponds to exactly one path starting from some root.
- For every node $u$, the path from the root (of the tree containing $u$) to $u$ is a prefix of some transaction. Furthermore, the node carries a counter, which equals the number of transactions that have this prefix.
The trie for our example dataset is shown on the right hand side.

<table>
<thead>
<tr>
<th>tran. id</th>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a, b</td>
</tr>
<tr>
<td>2</td>
<td>b, c, d</td>
</tr>
<tr>
<td>3</td>
<td>a, c, d, e</td>
</tr>
<tr>
<td>4</td>
<td>a, d, e</td>
</tr>
<tr>
<td>5</td>
<td>a, b, c</td>
</tr>
<tr>
<td>6</td>
<td>a, b, c, d</td>
</tr>
<tr>
<td>7</td>
<td>a</td>
</tr>
<tr>
<td>8</td>
<td>a, b, c</td>
</tr>
<tr>
<td>9</td>
<td>a, b, d</td>
</tr>
<tr>
<td>10</td>
<td>b, c, e</td>
</tr>
</tbody>
</table>

For example, node $b : 5$ corresponds to the prefix $ab$, which exists in 5 transactions: those with ids 1, 5, 6, 8, and 9.

Note that the trie can be used to replace the input set of transactions.
For every item, link up all its occurrences in the trie from left to right with pointers. This gives the **FP-tree**.

An FP-tree can be created in $O(n)$ time where the $n$ is the total size of all the transactions.

(Hint: first create the trie, and then perform a depth first traversal to add the red pointers).
The FP-tree allows us to identify quickly the transactions that contain the least frequent item $z$.

In our example, to find the 3 transactions that end with $e$, we identify the subtree shown on the right:

Note the counter changes as indicated. This subtree can be generated in time linear to the size of the subtree (think: how).
The FP-growth algorithm first finds all frequent itemsets that involve the least frequent item \( z \).

We will explain how to do so step by step.

First, it suffices to look at the transactions we identified.

In our example, to find all frequent itemsets involving \( e \), it suffices to focus on only 3 transactions, all of which are inferred by the subtree generated on the previous slide:

\[
\begin{align*}
\text{counters changed} & \quad \Rightarrow \\
\text{a : 2} & \quad \text{b : 1} \\
\text{c : 1} & \quad \text{d : 1} \\
\text{d : 1} & \quad \text{e : 1} \\
\text{e : 1} & \quad \text{c : 1} \\
\end{align*}
\]

\[
\begin{align*}
\frac{a, c, d, e}{a, d, e} & \quad \Rightarrow \\
\frac{b, c, e}{a, d, e} &
\end{align*}
\]
The FP-growth algorithm first finds all frequent itemsets that involve the least frequent item \( z \).

Second, remove \( z \) from all those transactions (because all of them must contain \( z \) anyway).

In our example, we remove \( e \) from the three transactions:

\[
\begin{align*}
\text{a, c, d, e} & \quad \text{a, c, d} \\
\text{a, d, e} & \quad \text{a, d} \\
\text{b, c, e} & \quad \text{b, c}
\end{align*}
\]
The FP-growth algorithm first finds all frequent itemsets that involve the least frequent item $z$.

Third, find all the frequent subsets on those transactions recursively.

Continuing our example, consider the 3 transactions obtained from the previous slide:

- $a, c, d$
- $a, d$
- $b, c$

Recall that we first sort all items by frequency and remove the items that appear less than $\text{minsup} = 2$ times:

- $a, c, d$
- $a, d$
- $c$

Note that $b$ has been removed.
The FP-growth algorithm first finds all frequent itemsets that involve the least frequent item \( z \).

Third, find all the frequent subsets on those transactions recursively.

Now build the FP-tree:

\[
\begin{align*}
  a, c, d \\
  a, d \\
  c
\end{align*}
\]

\[
\Rightarrow
\]

This is the FP-tree of the original dataset \textbf{conditioned on} \( e \).

Notice that we are facing essentially the same problem, i.e., finding frequent itemsets from an FP-tree. Our illustration, therefore, ends here. The frequent itemsets are: \( \{a\} \), \( \{c\} \), \( \{d\} \), and \( \{a, d\} \).
We have now settled:

The FP-growth algorithm first finds all frequent itemsets that involve the least frequent item $z$.

Adding $e$ to each of $\{a\}$, $\{c\}$, $\{d\}$, and $\{a,d\}$ gives $\{a,e\}$, $\{c,e\}$, $\{d,e\}$, and $\{a,d,e\}$. They, together with $\{e\}$, are all the frequent itemsets involving $e$ in the original dataset.
Now it remains to

Find all the frequent itemsets that do not contain \( z \).

For this purpose, the FP-growth algorithm simply removes the least frequent item \( z \) from all transactions, and recurs on the remaining transactions.

In our example, we remove \( e \) from the FP-tree (on the original dataset), and recurs on the FP-tree on the right.

\[
\begin{array}{c}
a : 8 \\
b : 5 \\
c : 3 \\
d : 1 \\
e : 1 \\
d : 1 \\
e : 1 \\
b : 2 \\
c : 2 \\
d : 1 \\
\end{array}
\]