CHAPTER 2

DATA MINING ALGORITHMS

2.1 OVERVIEW

Data mining has recently attracted considerable attention from database practitioners and researchers because it has been applied to many fields such as market strategy, financial forecasts and decision support [12].

Data need to be analyzed for sales forecast, business planning and marketing trend etc., for any business application. To make the decision making process effective in a Business application, which is distributed over different geographical locations, data mining is very essential. As one of several essential data mining tasks, mining frequent patterns has been studied extensively in literature [37]. Research in the area of data mining during the last eight years has led to the development of a variety of algorithms for finding frequent itemsets in very large transactional databases. Frequent itemsets play an essential role in many data mining tasks that try to find interesting patterns from databases [5].

Mining frequent itemsets is a fundamental and essential problem in many data mining applications such as the discovery of association rules, strong rules, correlations, multidimensional patterns, and many other important discovery tasks. The problem is formulated as follows: Given a large database of set of items transactions, find all frequent itemsets, where a
frequent itemset is one that occurs in at least an user-specified percentage of the database [49].

From these applications, fast implementations are needed for mining frequent itemsets. Data mining uses algorithms to sift through huge volumes of information for the purpose of detecting patterns hidden in the data. Association rules describe how often various items are purchased together. Such rules can be useful for decisions concerning product pricing, promotions, store layout and many others. Since their introduction in 1993 by Argawal et al [1], the frequent itemset and association rule mining problems have received a great deal of attention. Within the past decade, hundreds of research papers have been published presenting new algorithms or improvements on existing algorithms to solve these mining problems more efficiently.

Association rule mining, one of the most important and well researched techniques of data mining, was first introduced in reference [1]. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories. Association rule mining involves finding association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database; those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence.

Since the frequent itemset mining problem was first addressed, a large number of algorithms have been proposed. There is a pressing need to completely characterize and understand the algorithmic performance space of
frequent itemset mining problem so that we can choose and integrate the best strategies to achieve good performance in general cases [56]. Most of the well studied frequent pattern mining algorithms, including Apriori and FP-growth mine the complete set of frequent itemsets [96].

2.2 APRIORI ALGORITHM

This algorithm computes frequent itemsets from a transaction database over multiple iterations. Each iteration involves (i) Candidate generation and (ii) candidate counting and selection. Utilizing the knowledge about infrequent itemsets, obtained from previous iterations, the algorithm prunes a-priori those candidate itemsets that cannot become frequent [87]. After discarding every candidate itemset that has an infrequent subset, the algorithm enters the candidate counting step.

Apriori Algorithm employs an iterative approach known as a level-wise search, where K-itemsets are used to explore (K+1) itemsets. To improve the efficiency of the level-wise search [44], generation of frequent itemsets is an important property called the Apriori property.

- **Frequent Itemset Property:**
  
  *Any subset of a frequent itemset is frequent* [26].

- **Contrapositive:**
  
  *If an itemset is not frequent, none of its supersets are frequent.*

Apriori algorithm uses large itemset property and it is easy to implement. However in this algorithm transaction database is memory resident and it requires many database scans. The pictorial representation of Apriori algorithm is shown in Figure 2.1.
Algorithm

- BFS of all candidates
  - At level $k$, generate all $k$-itemsets $C_k$ (of length $k$), using frequent item sets $L_{k-1}$ generated at previous level.
  - Prune the search tree using the anti-monotone property of the support:
    \[ X_2 \supseteq X_1 \rightarrow \text{support}(X_2) \leq \text{support}(X_1). \]

- Efficient computation of the support
  - Candidate itemsets are stored in a hash tree [29].

**Figure 2.1 Apriori Algorithm**

The Apriori Algorithm: Pseudo code

**Join Step:** $C_k$ is generated by joining $L_{k-1}$ with itself

**Prune Step:** Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent $k$-itemset [4]
**Pseudo-code:**

$C_k$: Candidate itemset of size $k$

$L_k$: frequent itemset of size $k$

$L_1 = \{\text{frequent items}\}$;

**for** ($k = 1; \; L_{k-1} \neq \Phi; \; k++$) **do begin**

$C_{k+1}$ = candidates generated from $L_k$;

**for** each transaction $t$ in database **do**

  increments the counts of all candidates in $C_{k+1}$ that are contained in $t$

$L_k = L_{k+1}$ with $\text{min\_support}$

**end**

**return** $U_k L_k$;

18
Figure 2.2 Illustration of Apriori Example
2.3 FREQUENT PATTERN GROWTH (FP-GROWTH) ALGORITHM

One of the fastest and most popular algorithms for frequent itemset mining is the FP-growth algorithm [35]. It is a projection-based pattern growth method, which may explore some compressed data structure such as FP-tree, which can save considerable amount of memory for storing the transactions [14] [37].

This algorithm adopts a divide-and-conquer strategy. This algorithm compresses the database representing frequent items into a frequent pattern tree called FP tree, but retains the itemset association information and then divides such a compressed database into a set of conditional databases, each associated with one frequent item and mine each such database separately.

FP-Growth Method: An Example

- Consider the same previous example of a database D, consisting of 9 transactions.
- Suppose min. support count required is 2 (i.e. min_sup = 2/9 = 22 %)
- The first scan of database is same as Apriori, which derives the set of 1-itemsets & their support counts.
- The set of frequent items is sorted in the order of descending support count.
- The resulting set is denoted as L = {I2:7, I1:6, I3:6, I4:2, I5:2}.
<table>
<thead>
<tr>
<th>Trans.ID</th>
<th>Itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>1,2,5</td>
</tr>
<tr>
<td>I2</td>
<td>2,4</td>
</tr>
<tr>
<td>I3</td>
<td>2,3</td>
</tr>
<tr>
<td>I4</td>
<td>1,2,4</td>
</tr>
<tr>
<td>I5</td>
<td>1,3</td>
</tr>
<tr>
<td>I6</td>
<td>2,3</td>
</tr>
<tr>
<td>I7</td>
<td>1,3</td>
</tr>
<tr>
<td>I8</td>
<td>1,2,3,5</td>
</tr>
<tr>
<td>I9</td>
<td>1,2,3</td>
</tr>
</tbody>
</table>

Table 2.1 Transaction Table- Apriori algorithm

**FP-Growth Method: Construction of FP-Tree**

The construction of FP-Tree is illustrated in Figure 2.3:

- First, create the root of the tree, labeled with “null”.
- Scan the database D a second time. (First time we scanned it to create 1-itemset and then L).
- The items in each transaction are processed in L order (i.e. sorted order).
- A branch is created for each transaction with items having their support count separated by colon.
- Whenever the same node is encountered in another transaction, increment the support count of the common node or Prefix.
- 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.
- Now, the problem of mining frequent patterns in database is transformed to that of mining the FP-Tree.
Figure 2.3 Illustration of FP-growth Algorithm

Mining the FP-Tree by Creating Conditional (sub) pattern bases Steps:

1. Start from each frequent length-1 pattern (as an initial suffix pattern).
2. Construct its conditional pattern base, which consists of the set of prefix paths in the FP-Tree co-occurring with suffix pattern.
3. Then, Construct its conditional FP-Tree & perform mining on such a tree.
4. The pattern growth is achieved by concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-Tree.

5. The union of all frequent patterns (generated by step 4) gives the required frequent itemset.

<table>
<thead>
<tr>
<th>Item</th>
<th>Conditional pattern base</th>
<th>Conditional FP-Tree</th>
<th>Frequent Pattern Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>I5</td>
<td>{(I2 I1:1),(I2 I1 I3:1)}</td>
<td>&lt;I2:2, I1:2&gt;</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><a href="">I2:2</a></td>
<td>I2 I4:2</td>
</tr>
<tr>
<td>I3</td>
<td>{(I2 I1:1),(I2:2),(I1:2)}</td>
<td><a href="">I2:4,I1:2</a>,</td>
<td>I2 I3:4, I1 I3:2, I2 I1 I3:2</td>
</tr>
<tr>
<td></td>
<td></td>
<td><a href="">I1:2</a></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>{(I2:4)}</td>
<td><a href="">I2:4</a></td>
<td>I2 I1:4</td>
</tr>
</tbody>
</table>

Table 2.2 Mining the FP-Tree by creating conditional (sub) pattern bases

Now, following the above mentioned steps:

- Let's start from I5. The I5 is involved in 2 branches namely {I2 I1 I5: 1} and {I2 I1 I3 I5: 1}.
- Therefore considering I5 as suffix, its 2 corresponding prefix paths would be {I2 I1: 1} and {I2 I1 I3: 1}, which forms its conditional pattern base.
- Out of these, Only I1 & I2 are selected in the conditional FP-Tree because I3 is not satisfying the minimum support count.
- For I1, support count in conditional pattern base = 1 + 1 = 2
- For I2, support count in conditional pattern base = 1 + 1 = 2
For I3, support count in conditional pattern base = 1

Thus support count for I3 is less than required min_sup which is 2 here.

Now, create conditional FP-Tree.

All frequent patterns corresponding to suffix I5 are generated by considering all possible combinations of I5 and conditional FP-Tree.

The same procedure is applied to suffixes I4, I3 and I1.

Note: I2 is not taken into consideration for suffix because it doesn’t have any prefix at all.

FP-growth, the depth first algorithm [101] is an order of magnitude faster than Apriori, and is also faster than tree-projection. Because in this algorithm there is no need of candidate generation and candidate test. This algorithm uses the compact data structure and eliminate repeated database scan.

2.4 CONCLUSION

In this chapter, the two common techniques, Apriori Algorithm and the more-recent FP (Frequent Pattern)-Growth technique, which are applied to sequence databases for frequent itemset mining are discussed. In Apriori algorithm, multiple scanning of the transactional database is needed to find frequent itemsets, which is time consuming. Although FP-growth is fast, a major drawback is that in each recursive call, a new FP-tree has to be built. The FP-Growth algorithm has another drawback. In each outer iteration, a call of a procedure is performed. The execution of this procedure traverses the whole database. In order to reduce the number of scanning and to remove the tree construction, a new algorithm is proposed, which is simple and easy to understand as described in the following chapter.