An Improved Apriori Algorithm on the Frequent Itemsets

Xiang Fang

Department of Department of Computer Science, Shandong Institute of Business and Technology, Shandong, 264005, China
email:fangxiang782@sina.com

Keywords: Association Rule; Apriori Algorithm; Frequent Itemsets

Abstract. According to the problem that the traditional Apriori algorithm needs to scan database frequently, an improved strategy and corresponding algorithm is put forward in this paper. This method, only when $L_1$-candidates are produced, scans the transaction database $D$. The rest frequent itemsets are produced through the scan of preceding result in place of the transaction database. The improved algorithm reduces I/O load and has higher efficiency.

Introduction

A subfield of computer science, Data mining (the analysis step of the "Knowledge Discovery in Databases" process, or KDD [1]), is the computational process of extracting information and transforming it into an understandable patterns from a large data sets involving combining methods from artificial intelligence, machine learning, statistics, database systems and so on [2]. And the association rule, a well researched method for discovering interesting relations between the items in large databases, is popular and effective for decision-making in financial mathematics, medical, intrusion detection and Web analysis in recent years [3][4]. Apriori algorithm is the most classic algorithm for association rule on transactional databases. It proceeds by identifying the frequent items in the database and extending them to an itemset as long as those itemsets appear sufficiently often in the database.

Since the Apriori algorithm had been brought by R.Agrawal in 1993 [5], scholars have put forward lots of algorithmic improvements, such as AprioriTid, AprioriHybrid [6], DIC [7], DHP [8]. The improved algorithms can be divided into two kinds based on whether produces the candidate itemset: One is quite direct-viewing and easy to realize, but they have to produce the candidate itemset and scan the transactional database repetitiously, the classic Apriori algorithm is the most famous one of them [9]; the other is very efficient, as it does not need to produce candidate itemset, but they is very complex, because the association rules are found by the pattern tree. The Han’s fp-growth algorithm is the most influential one [10].

Taking into account deficient of the first kind, many scholars proposed algorithmic development, such as a row optimization based on disperses, business-compresses, business-division, optimization based on the sampling, optimization based on the dynamic itemset and so on[11]. Unfortunately, those improved algorithms have to produce the massive candidate itemsets. This paper unifies the two thoughts and puts forward the improved strategy and corresponding algorithm to the traditional Apriori algorithm: Find the set $L_k$ (frequent itemsets) from $L_{k-1}$ not $C_{k+1}$ (candidate itemsets) and simultaneously remove the candidate items which have infrequent patterns.

Traditional Apriori Algorithm and Defect of It

Let $I=\{i_1, i_2, \ldots, i_m\}$ denote the set of items that are displayed in a store. Moreover, let $D$ represent a set of transactions, where each transaction $T$ is a set of items such that $T \subseteq I$. A unique identifier, namely TID, is associated with each transaction. Let $A$ is a set of item in $I$, $A$ transaction $T$ is said to contains $A$ if $A \subseteq T$. An association rules is an implication of the form $A \Rightarrow B$, where $A \subseteq I$, $B \subseteq I$ and $A \cap B \neq \emptyset$. Support of rule, denoted by $s$, is the percentage of transactions in $D$ that contain $A$ also contain $B$, it can be computed by $|AB|/|D|$; Confidence of rule, denoted by $c$, is the ratio
between the number of transactions containing both A and B and the number of transactions containing A, it can be computed as \(|AB|/|A|\). In database D, a set of items is called itemset, and an itemset that contains k items is called k-itemset. Support count is number of transactions containing an itemset, if an itemset whose support is greater than or equal to a minimum support threshold (called minsup), we name it the frequent itemset.

Apriori algorithm finds frequent itemsets, and generates successively longer candidate itemsets from shorter ones that are known. Each produce of candidate itemset requires scanning the dataset to determine whether its frequency exceeds the minsup. Apriori algorithm can find rules in the transaction database with the known minsup and minimum confidence through the following two sub-questions:[12]:

1. Find the frequent itemset \(L_k\) from the candidate itemset \(C_{k+1}\), and scan the database to remove infrequent patterns according to a predetermined minimum support count.

2. Generate the interesting association rule from the frequent itemset.

The overall performance of association rules is determined by the first question, the following is the description of the algorithm:

**Input:** transaction database \(D\), minsup

**Output:** the set of Frequent \(L\) in the database \(D\)

**Method:**

1. \(L_1 = \{\text{large1-itemsets}\}\);
2. for \((k=2; L_{k-1} \neq \emptyset; k++)\) do
3. \(\{C_k = \text{Apriori}\_\text{gen}(L_{k-1}, \text{min-sup})\}; // \text{Produce } L_k \text{ by } L_{k-1}
4. for each transaction \(t \in D\) do
5. \(\{C_t = \text{subse}(C_k, t)\}; // \text{Find candidate}
6. for each candidate \(c \in C_t\) do
7. \(c\.\text{count}++;\)
8. \}
9. \(L_k = \{c \in C|c\.\text{count} \geq \text{min-sup}\}
10. \}
11. return \(L = \cup_k L_k\);

There is a transaction database \(D\), which has nine transactions, as shown in Table 1.

<table>
<thead>
<tr>
<th>TID</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A,B,E</td>
</tr>
<tr>
<td>2</td>
<td>B,D</td>
</tr>
<tr>
<td>3</td>
<td>B,C</td>
</tr>
<tr>
<td>4</td>
<td>A,B,D</td>
</tr>
<tr>
<td>5</td>
<td>A,C</td>
</tr>
<tr>
<td>6</td>
<td>B,C</td>
</tr>
<tr>
<td>7</td>
<td>A,C</td>
</tr>
<tr>
<td>8</td>
<td>A,B,C,E</td>
</tr>
<tr>
<td>9</td>
<td>A,B,C</td>
</tr>
</tbody>
</table>

By setting the minimum support as 20%, namely the support is 2. Find frequent itemset for D by means of the traditional Apriori algorithm. The process is shown in Figure1.
based on architecture of the traditional algorithm, the efficiency has not been much improved. The key of the improved algorithm that we propose in his paper is how to reduce the scan through the results of previous scan.

**Improved Apriori Algorithm**

Let $TID-set(A)$ denote the set of transaction $TID$ which contain item $A$ in $D$, so the amount of transaction which contain $A$ in $D$ is the amount of item of $TID-set(A)$, more exactly, $sup-count(A)$ can be computed by $|TID-set(A)|$. The transaction set in $D$, which have $A$ and $B$, is the intersection of $TID-set(A)$ and $TID-set(B)$, and $sup-count(A \rightarrow B)$ can be computed as $|TID-set(A) \cap TID-set(B)|$. The join and the pruning of Apriori algorithm is improved correspondingly: all non-empty subset of frequent itemset must be frequent; to produces $L_k$-candidates, we only join the set whose fore $k-2$ item is same in $L_{k-1}$. So we propose the following improvement to the traditional Apriori algorithm:

1. Compute minimum sup-count by $min-sup*|D|$.
2. Scan the database once to produce $L_1$-candidates, simultaneously construct $TID-set(X_1)$ for each item. After scanning, compute sup-count for each item and find the set of frequent items $L_1$.
3. Produces $L_2$-candidates from $L_1*L_1$. Scanning $L_1$, we can find $TID-set(X_2)$ and sup-count of each itemset, deletes the patterns whose frequencies don’t satisfy the min support count, and find $L_2$.
4. In order to produce $L_k(k \geq 3)$, join itemsets which satisfy the join rule. Scanning $L_{k-1}$, we can find $TID-set(X_k)$ and compute sup-count of each itemset, then deletes the patterns whose frequencies do not satisfy the minimum support count, and find $L_k$.

The pseudo code of Algorithm is as follows:

**Input:** transaction database $D$; min-sup.

**Output:** the set of Frequent $L$ in the database $D$

**Method:**

1. $min-sup-count=min-sup*|D|$
2. $L_1$-candidates=find all one itemsets($D$)  //Scan $D$ and produce $L_1$-candidates
3. $L_1=\{<X_1,TID-set(X_1)> | \exists L_i \in L_1-candidates$ | $sup-count \geq min-sup-count\}$
4. for ($k=2$; $L_{k-1} \neq \emptyset$; $k++$) do {
5. for each $k$-itemset ($x_i,TID-set(x_i)$) $L_i \in L_{k-1}$ do
6. for each $k$-itemset ($x_j,TID-set(x_j)$) $L_j \in L_{k-1}$ do
7. if ($x_i[1]=x_j[1]$) $\land$ ($x_i[2]=x_j[2]$) $\land$ $\cdots$ ($x_i[k-2]=x_j[k-2]$) then
8. $L_k$-candidates.$X_k=x_i \ast x_j$;
9. $L_k$-candidates.$TID-set(X_k) = TID-set(X_i) \cap TID-set(X_j)$
10. }
11. for each $k$-itemset$<X_k,TID-set(X_k)> \in L_k$-candidates do
12. $sup-count=|TID-set|$
13. $L_k=\{<X_k,TID-set(X_k)> | L_k$-candidates$sup-count \geq min-sup-count\}$
14. }
15. set-count$=L_k$.itemcount  //update set-count
16. return $L=\cup_1L_k$;

We generate $L_k$ through the Line 5 to 14, thereinto line 5 to 10 generate $L_k$-candidate, line 7 judge whether the fore $k-2$ item is same of $L_{k-1}$, if the same, we join the two $L_{k-1}$, and add it to $L_k$-candidate. Line 11 to 13 compute the support for each items and deletes the patterns whose frequencies do not satisfy the minimum support count. Line 15-16 compute item-count and input the result.

In order to specify the discussion above, we give an application. According to the algorithm, first we produce $L_1$-candidates and find the set of frequent items $L_1$. Sele-join $L_1$ to produce $L_2$-candidates, scan $L_1$, and achieve $TID-set(X_1)$ and $TID-set$ set, deletes the patterns whose frequencies do not satisfy the minimum support count, and find $L_2$. By joining the itemset which satisfied rule in $L_2$, we can obtain \{\{A,B,C\}, \{A,B,E\}, \{B,C,D\}, \{B,C,E\}, \{A,C,E\}, \{B,D,E\}\}. However \{A,D\}, \{C,D\}, \{C,E\}, and \{D,E\} is not the frequent, and \{B,C,D\}, \{B,C,E\}, \{A,C,E\}, \{B,D,E\} don’t belong to $L_2$-candidates. Compute $TID-set(X_3)$ and the support count separately, $L_3$ is composed by itemsets which have the minimum support count. The itemset which satisfies the condition in $L_3$ sele-join, and obtains \{\{A,B,C,E\}\}. \{B,C,E\} is not the frequent itemset, so $L_4$-candidates$=\emptyset$, the algorithm finishes. The process is as shown in Figure2.
Fig2. The operation process of the improved Apriori algorithm

Conclusion

The biggest drawback of traditional Apriori algorithm is that each item in the candidates of frequent item sets must scan database one time, and the efficiency of execution is very low. We proposed an improvement, this algorithm scans the transaction database D only when \( L_1 \)-candidates is produced, the rest scan the preceding result instead of the transaction database, greatly reducing I/O load and increase efficiency. The interesting association rule can be accessed more effectively by the improved Apriori algorithm.

References