Research on Mining Maximum Frequent Itemsets Based on JFP-Growth Algorithm

Wang Zerua, Wang Hongmeib
School of Computer Science and Engineering, Changchun University of Technology, Changchun, China
a wangzeru1993@163.com, bwanghm/ccut.edu.cn

Keywords: frequent itemsets, FP-Growth algorithm, frequent 2-item set, pruning, two-dimensional count table

Abstract: The FP-Growth algorithm is the most representative frequent item set mining algorithm. An improved algorithm JFP-Growth algorithm is proposed for the shortcomings of FP-Growth algorithm. When mining the maximum frequent itemsets, the JFP-Growth algorithm traverses the support of the first 1-item set and the 2-item set of the first-time data set statistics, and uses the frequent 2-item set as the pruning condition for full pruning and The merged nodes make there is no non-potential candidate 3-item set in the JFP-tree, and the conditional pattern base is generated without traversing the project header table during the mining process. Finally, the FP-Growth algorithm and DMFIA algorithm are compared in the mining results, the number of nodes in FP-tree and JFP-tree, mining efficiency, etc., which verifies the correctness and efficiency of the JFP-Growth algorithm proposed in this paper.

1. Introduction

At present, data mining is to mine useful knowledge under a large amount of data. It is especially important to find the relationship between knowledge and knowledge, and the core step is to mine frequent itemsets. Many scholars have proposed many representative algorithms on how to mine frequent itemsets, such as Apriori algorithm, FP-Growth algorithm, PARTITION algorithm and so on. The FP-Growth algorithm is the most representative because it has an order of magnitude improvement based on the Apriori algorithm. The FP-Growth algorithm is mainly composed of two parts: (1) generating FP-tree, compressing the data set into a FP-tree; (2) generating frequent itemsets, and recursively mining frequent itemsets by generating FP-trees. The FP-Growth algorithm scans the data set of two passes to construct the FP-tree, finds the conditional pattern base of each suffix pattern in the FP-tree according to the suffix pattern and the item header table, and constructs the conditional FP-tree, and then recursively mines each condition. FP-tree generates all frequent itemsets.

In recent years, many scholars have carried out a lot of research on the FP-Growth algorithm. The literature [3] uses the Web Usage data to analyze the FP-Growth algorithm and the Apriori algorithm, which shows the efficiency of the FP-Growth algorithm; the literature [4] will The three classic algorithms FP-growth, COFI-tree and CT-PRO based on FP tree are compared and analyzed to illustrate the shortcomings of FP-Growth algorithm. The literature [5] constructs FP-tree projection and improves the FP tree in parallel. Memory overflow problem; literature [6] designed computer cluster parallel FP-Growth algorithm to improve mining efficiency and solve memory overflow problem, the literature [7] proposed load balancing parallel FP-Growth algorithm to improve mining efficiency; literature [8] found frequent 2- The key pruning effect of the item set and the last pruning theorem are proposed. When calculating the support degree, all the sibling nodes with the common ancestor path are avoided. The literature [9] solves the data set based on the storage disk and the incremental FP-Growth algorithm. The problem of re-running the FP-Growth algorithm is changed; the support of the 1-item set and the 2-item set is recorded using the two-dimensional vector, and the first traversal of the conditional pattern base when the conditional FP-tree is generated is omitted, but this Methods will lose frequent prefixes. In [11], the
pre-feedback and inclusion relationship are pruned on the composite FP-tree, which solves the problem that the FP-Growth algorithm scans the database twice and the memory consumption is large. In recent years, with the development of parallelization processing, algorithms based on parallelization platform have been proposed in the literature [12], literature [13], literature [14], and literature [15], which provides a good direction for the next step.

The key to improving the performance of mining frequent itemsets is: (1) avoiding the generation of a large number of candidate sets; (2) pruning the infrequent itemsets. The FP-Growth algorithm itself has an algorithm to avoid generating a large number of candidate sets. Therefore, in this paper, the problem of pruning the infrequent itemsets is proposed in the establishment of FP-tree for pruning, in order to improve efficiency and generate FP-trees. The conditional pattern base does not have to read the project header table in turn. Finally, the correctness and efficiency of the algorithm are verified by comparing with the existing maximum frequent item set mining algorithm DFP-tree and DMFIA in the data set Mushroom.

Related theory

Definition 1 data set, transaction, item set, frequent item set

As shown in Table 1, a transaction data set T, where ti represents each thing, and the elements contained in each thing are called item sets. The number of times the item set is included in the data set T is called support, and is denoted as \( \text{sup}(X) \). Let the minimum support be \( \text{min}_\text{sup} \). If \( \text{sup}(X) \geq \text{min}_\text{sup} \), then the item set \( X \) is a frequent item set.

Table 1 Data set T

<table>
<thead>
<tr>
<th>Affairs</th>
<th>Items</th>
<th>Affairs</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>A, B, C, D, E, F</td>
<td>t6</td>
<td>B, D, E, G</td>
</tr>
<tr>
<td>t2</td>
<td>A, B, D</td>
<td>t7</td>
<td>A, C, E, G</td>
</tr>
<tr>
<td>t3</td>
<td>B, D, F</td>
<td>t8</td>
<td>A, F</td>
</tr>
<tr>
<td>t4</td>
<td>C, D</td>
<td>t9</td>
<td>A, C, D</td>
</tr>
<tr>
<td>t5</td>
<td>A, B, D, E</td>
<td>t10</td>
<td>A, D, E</td>
</tr>
</tbody>
</table>

Pruning theorem 1 if the item set \( X = \{x_1, \ldots, x_k\} \) is frequent, \( \forall \alpha \in X \), then \( \alpha \) must be frequent. On the contrary, if \( \alpha \) is infrequent, the item set \( X \) must be infrequent.

Pruning theorem 2 If the itemset \( X = \{x_1, \ldots, x_k\} \) is frequent, \( \forall \alpha \in X \land \beta \in X \land \alpha \neq \beta \). The 2-item set \( \{\alpha, \beta\} \) must be frequent. on the contrary, \( \exists \alpha \in X \land \beta \in X \land \alpha \neq \beta \). If the 2-item set \( \{\alpha, \beta\} \) is infrequent, the item set \( X \) must be infrequent.

Corollary 1 sets the itemset \( \{\alpha, \beta\} \) and \( \{\gamma\} \) to be frequent. If \( \{\alpha, \gamma\} \) is infrequent or \( \{\beta, \gamma\} \) is infrequent, the itemset \( \{\alpha, \beta, \gamma\} \) must be infrequent.

Corollary 2 If the itemsets \( \{\alpha, \beta, \gamma\} \) are infrequent, all itemsets prefixed by \( \{\alpha, \beta, \gamma\} \) must be infrequent.

Definition 2 A tree structure that satisfies the following characteristics is called a Frequent-pattern growth tree (FP-tree):

1) The root node of the tree is "null" and its identifier is ['null': support];
2) Except that the root node assumes that other nodes are frequent 1-item sets, the identifier is [1-item set: support];
3) The project header table is sorted by the frequent 1-item set in descending order according to the support count, and its identifier is [1-item set: support: link pointer]

2. Improve the focus

Establishing FP-trees

In the traditional FP-Growth algorithm, the process of establishing FP-tree does not involve branching, so that the number of each element is counted in the mining process. When the established tree is relatively large, it will affect the mining efficiency. The pseudo-code of the branching process algorithm is described as follows:
1) The function Gen(T) converts the data set T into a two-dimensional count table.

Algorithm: Gen(T)
Input: data set T
Output: two-dimensional count table D, the complete set of pruned items I'

for each \( t \in T \)
  for(i = 0; i < |t|; i++)
    for(j = i; j < |t|; j++)
      \( D_{ti}[tj]++ \);
  for(i = 0; i < |t|; i++)
    if (\( D_{ti}[ti] \geq \text{min}_\text{sup} \))
      \{ add(\( ti \), 1-item); if (\( \text{sign}[ti] = 1 \land \neg \exists j \land D_{ti}[tj] < \text{min}_\text{sup} \) insert(\( ti, I' \)); \} 

2) The function CreateTree(T, min_sup, I') generates a pruned JFP-tree according to the rule that proposes to establish an FP-tree.

Algorithm: CreateTree(T, min_sup, I')
Input: two-dimensional count table T, minimum support min_sup, complete set of pruned items I'
Output: FP-tree after pruning

for each \( t \in T \)
  \( t' = \text{sort}(t, I') \);
  for(i = 0; i < |t'|; i++)
    for(j = i + 1; j < |t'| - 1; j++)
      if (\( D_{ti}[tj] \geq \text{min}_\text{sup} \))
        for (k = j + 1; j < |t'|; k++)
          if (\( D_{ti}[tk] \geq \text{min}_\text{sup} \land D_{tj}[tk] \geq \text{min}_\text{sup} \))
            while (\( D_{tk}[t_k] < \text{min}_\text{sup} \land D_{tj}[tk] \geq \text{min}_\text{sup} \))
            \( t_{k-1} = t_{k-1}.parent \);
          \( t_k = t_k.parent \);
          \( t_k.count++ \);

For example, taking the data set T of Table 1 as an example, a two-dimensional count table and an FP-tree are established, and then the branching comparison is performed by the branching algorithm.

1) The first pass scan data set of the FP-Growth algorithm only calculates the support of the 1-item set. Considering the time cost of scanning the data set, the FP-Growth algorithm scans the data set to count all 1-item sets and 2 - The support of the item set. For the data set T shown in Table 1, the support for scanning the data set calculation 1-item set and 2-item set is shown in Table 2.

<table>
<thead>
<tr>
<th>item</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

2) The FP-Growth algorithm deletes all infrequent patterns according to the pruning theorem 1. The pruning FP-Growth algorithm deletes frequent patterns that do not necessarily produce frequent 2-item sets according to the pruning theorem 2. For the data set T shown in Table 1, according to
the statistical result of Table 2, let min_sup=3, delete the item G (infrequent mode) according to the pruning theorem 1, and delete the item F according to the pruning theorem 2 (all contain F The 2-item set is infrequent), and the remaining frequent patterns are arranged in non-ascending order of support, resulting in \( I' = \{D, A, B, E, C\} \).

3) The FP-Growth algorithm does not consider the pruning effect of the 2-item set on FP-Tree. Because the pruning is not sufficient, the scale cannot be controlled when the FP-Tree is established, and the subsequent traversal FP-Tree and the construction condition FP are added. The cost of -Tree. Since the pruning FP-Growth algorithm obtains all the frequent 2-item sets after the first scan of the data set, only the frequent k-item set \((k \geq 2)\) needs to be mined for the second scan of the data set.

4) According to the inference 1 of the pruning theorem 2, when constructing the pruning FP-tree, all the items corresponding to the non-potential frequent 3-item set are subtracted. For example, when pruning FP-tree is constructed for things \( t1 = \{D, A, B, E, C\} \), for item C, since the 2-item set \{B, C\} is infrequent, the item \{C\} is directly subtracted. As shown in Figure 1:

![Figure 1: t1 before and after the branch](image)

5) Combine all the things in Table 2 according to the branch reduction strategy, and finally get the FP-tree comparison before and after the branching as shown in Figure 2.

![Figure 2(a): FP-tree constructed by FP-Growth](image)

![Figure 2(b): JFP-tree after branching](image)

Figure 2 FP-tree comparison before and after branching

36
It can be seen from the above process that the spatial complexity of the improved algorithm will increase \( L(n^2) \) on the basis of the original, but the pruning time complexity of the FP-Tree after the two-dimensional table is established is \( O(1) \). Comparing FP-tree and FP-tree after branching, it can be clearly seen that the F term does not appear in the node when the FP-tree is established, because the F element and the elements in the tree cannot form a frequent 2-item set because According to the subtraction theorem 1, the branch can be directly reduced. Then, when other items of things are inserted into the FP-tree, and then subtracted according to the subtraction theorem 2, the FP-tree merge item can be reduced, thereby reducing the size of the tree and generating unnecessary frequent itemsets.

3. Analysis of experimental results

In order to verify the correctness and effectiveness of the algorithm, in the ubuntu16.04.3 operating system, clocked at 2.5GHz, memory 4G, the following three experiments were carried out on the data set using Python language.

Experiment 1: Verify the correctness of the algorithm under the data set mushroom. The experimental results are shown in Table 3.

It can be seen from Table 3 that the maximum frequent itemsets mined by the branching algorithm in the data set mushroom are completely consistent with the DFMIA algorithm and the DFP algorithm, which fully demonstrates the correctness of the algorithm.

<table>
<thead>
<tr>
<th>Min_sup</th>
<th>Maximum number of frequent itemsets</th>
<th>Total number of frequent itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JFP</td>
<td>DFMIA</td>
</tr>
<tr>
<td>0.02</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>0.04</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>0.08</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>0.1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>0.2</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

Experiment 2: In the dataset mushroom, the comparison of the number of nodes comparing FP-tree and JFP-tree under different support thresholds is shown in Table 4:

<table>
<thead>
<tr>
<th>Min_sup</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP-tree</td>
</tr>
<tr>
<td>0.08</td>
<td>391291</td>
</tr>
<tr>
<td>0.1</td>
<td>351898</td>
</tr>
<tr>
<td>0.15</td>
<td>69564</td>
</tr>
<tr>
<td>0.2</td>
<td>51754</td>
</tr>
<tr>
<td>0.25</td>
<td>4821</td>
</tr>
</tbody>
</table>

It can be seen from Table 4 that the number of nodes of the JFP-tree is smaller than the number of FP-trees, because the process of establishing the JFP-tree is accompanied by the reduction of the redundant items, and the branching effect is obvious.

Experiment 3: In the dataset T10I4D100K and Mushroom, the effect of data size on the algorithm efficiency under the same support threshold is investigated. The experimental results are shown in Figure 3 and Figure 4.

It can be seen from Fig. 3 that the JFP algorithm has a significant improvement in mining efficiency as the support degree decreases. This is because the pruning strategy and the bottom-up of the project header are not traversed as the support degree decreases. The greater the impact on the mining algorithm, the higher the efficiency of the algorithm.
4. Summary

This paper analyzes the advantages and disadvantages of the FP-Growth algorithm, and proposes an improved algorithm while inheriting its advantages. A two-dimensional count table is generated when traversing the first pass database, thereby generating a 1-item set and a 2-item set, and performing partial subtraction while generating. When the tree is built, the frequent 2-item set is used as the pruning basis, and the process of building the tree is accompanied by the process of branching, thereby improving the operating efficiency of the algorithm. And the problem of repeating the same path is repeated when the conditional pattern is generated, and the recursive backtracking method is proposed to solve the problem of repeated traversal. Although the space consumption is increased when JFP-tree is established, in the subsequent mining process, the efficiency improvement is very obvious, and the space spent is worthwhile. Finally, through the mining experiments on T10I4D100K and Mushroom data, comparing the classic DFMIA, DFP algorithm and improved algorithm proves that the improved algorithm has advantages when it is strict for large data sets or mining conditions.

References


