A Comparison of Association Rule Mining Methods for XML Data

Liu Yijun, He Sheng, Ye Feiyue

School of Computer Engineering, Jiangsu Teachers University of Technology, Changzhou
213001, P.R. China, E-mail: yijunliu@vip.sina.com

Abstract

With the increasing use of XML technology for data storage and data exchange, mining XML documents has become a researchable subject. This study makes a comparison of three association rule mining methods, the Apriori, FP-growth and AprioriSP, for mining XML data directly with using TinyXML, which is a XML parser implemented in the C++ language. The AprioriSP is proposed to find only short pattern frequent itemsets instead of all frequent ones in order to improve mining efficiency. Comparison experiments of performances of these three methods are performed on XML documents using different datasets and support levels.

Keywords: Association Rule Mining, XML Data, Apriori, FP-Growth

1. Introduction

With the continuous development of XML, the more and more information on Internet is represented using XML. There is an urgent need to extract potentially valuable information and knowledge from these large number of XML data in some effective way. However, XML data is very complex, and there is no specific model to describe these semi-structured or unstructured data. Therefore, the XML-oriented data mining technologies are different from those applied on the traditional relation databases.

As an effective way to discovery knowledge in a limited data set, association rule mining is an important issue in the field of data mining. In the last decade, it is widely applied in the commercial and financial domain, management, scientific research, intelligence analysis and even military fields. At present, much work has been done to mine association rules from XML data.

The goals of some works are to mine the frequent tree patterns from XML data. Termier et al. in [1] implement a tool TreeFinder with using an Inductive Logic Programming approach. However, the TreeFinder may miss many frequent subtrees and cannot produce complete results. Zaki et al. in [2] propose a vertical representation for fast subtree support counting and implement a tool TreeMiner which can produce complete results. Paik et al. in [3] propose to mine tree-based association rules in which two trees are disjunct. Mirjana et al. in [4] describe an approach to extract tree-based association rules, in which the left side is an induced subtree of the right side. These rules provide approximate, intensional information on both the structure and the content of XML documents, and can be stored in XML format to be queried later on. Other works focus on mining association rules of flat values over XML data. Wan et al. in [5] first propose the idea of mining XML data with XQuery, and use the Apriori algorithm with XQuery to mine XML documents. But they don’t apply the algorithm to large itemsets. Görkem GÜREL in [6] applies the Apriori algorithm and other two apriori-like algorithms of AprioriTid and HEA for large itemsets with XQuery and present their comparison results on XML documents. Wang Shufeng in [7] compares the performances of the Apriori and FP-growth algorithms implemented in Java and the Apriori algorithm implemented with XQuery. In this paper, we focus on the topic of applying association rule mining algorithms directly to the XML data with using TinyXML and their performance comparison.
2 Methodology and Implementation

2.1 Related concepts

Focusing on correlation relationships of different items in the same incident, association rule mining finds association rules from large amounts of data. Let \( T = \{ t_1, t_2, t_3, \ldots, t_m \} \) be a set of transactions, \( I = \{ i_1, i_2, i_3, \ldots, i_n \} \) a set of all items in \( T \). Each transaction \( t_j (1 \leq j \leq m) \) contains a subset of \( I \). The association rule is an implication relationship \( X \Rightarrow Y \), where \( X \subseteq I \), \( Y \subseteq I \) and \( X \cap Y = \emptyset \). The support and confidence are two factors related to this rule. In databases, if there are \( s \% \) transactions contain the item set \( X \cup Y \), \( s \) is called the support of the association rule \( X \Rightarrow Y \). If \( c \% \) transactions containing \( X \) also contains \( Y \), \( c \) is called the confidence of the association rule.

The support illustrates the universal degree of appearance of \( X \Rightarrow Y \) in transaction data. The confidence illustrates the necessary degree that \( X \Rightarrow Y \) holds. When the thresholds of support and confidence are satisfied, \( X \Rightarrow Y \) is taken as a meaningful association rule.

Association rule mining is divided into two steps. The first step is to find all frequent itemsets, whose support is not less than \( \min\_sup \), a minimum support threshold. The second step is to use frequent itemsets to generate association rules. For each frequent itemset \( F \), we find all its non-empty subset \( X \), and then calculate \( \text{support}(F) / \text{support}(X) \). The association rule \( X \Rightarrow F - X \) is generated if \( \text{support}(F) / \text{support}(X) \) is not less than \( \min\_conf \), a minimum confidence threshold. Because the first problem is more important and has more computing time, there are various endeavors to improve the efficiency of finding frequent itemsets.

2.2 The AprioriSP method for short pattern mining

Since the problem of association rule mining is proposed in [8], it has become a hot research topic in the field of data mining and many algorithms have been developed. In [9] R. Agrawal et al. propose the support-confidence framework and the Apriori algorithm to find frequent itemsets using candidate generation. Han et al. in [10] proposes an interesting method FP-growth to generate frequent itemsets using frequent pattern tree FP-tree. For more details about the Apriori and FP-growth, the two classical association rule mining algorithms, please refer to [9,10,11].

It takes a great deal of time to mine all association rules from a large set of data items comprising tens of thousands or even millions of records. However, in some practical use case scenarios the extraction of all rules is not necessary. In the case of on-line recommendation in electronic commerce, the significant patterns and rules underlying the databases are mostly the habits of purchasing and other consumers’ behavior and can support one-to-one on-line marketing via recommending products accordingly. In a rule, \( X \) is the set of items purchased by shoppers before, and \( Y \) is the set of ones to be recommended. Shoppers may feel confused when they are recommended too many products. So short patterns are possibly more helpful and give a more targeted recommendation. Here we consider mining short pattern association rules. Let pattern length be the number of items in \( X \cup Y \). Short pattern mining is to find only the rules whose pattern length is no longer than the pre-designated value instead of all rules in database.

Extending the Apriori algorithm, this paper proposes an AprioriSP approach to find short pattern frequent itemsets. Besides the inputs of \( D \) and \( \min\_sup \) in Apriori, another input \( \max\_len \) of max pattern length is needed in AprioriSP. The \( k \) is used to record the pattern length in each iteration. The mining process must stop when \( \max\_len \) is reached.
Algorithm: AprioriSP

Input: Database D of transactions; minimum support threshold min_sup; max pattern length of rules max_len;
Output: short pattern frequent itemsets L.

Process:
1: C1 = find_1-itemsets(D);
2: k = 1;
3: while (Ck ≠ Φ and k ≤ max_len)
   4: { 
      5:     Lk = gen_frequent_level(min_sup, Ck);
      6:     /*The while loop is terminated if Lk is empty.*/
      7:     if Lk == Φ {k--; break;}
      8:     Ck+1 = apriori_gen(Lk, min_sup);
      9:     k++;
   10: }
11: return L = ∪ kLk;

2.3 Implementation of algorithms for XML data

Various association rule mining algorithms were proposed only to handle relational data, and they
can not be applied directly to complex type of data, such as XML. To mine rules from XML data with
the Apriori, FP-growth and AprioriSP proposed in section 2.2, we implement them in C/C++ languages
with Visual C++ 6.0 development environment. A function GetTransaction(int tran_i) is realized to
collect the items of the transaction tran_i in an XML document. XML documents are accessed with the
API provided by an XML parser TinyXML. As an open source XML parser based on DOM model,
TinyXML is small in size and easy to use. Using the C++ language to implement a XML parsing
library, TinyXML can be compiled on Windows or Linux. Due to the corresponding DOM model of an
XML file generated in memory, the XML tree can be easily traversed. The parsing library mainly
comprises the DOM model class and the operating class, which are defined according to the various
elements of XML. The former includes TiXmlBase, TiXmlNode, TiXmlAttribute, TiXmlComment,
TiXmlDeclaration, TiXmlDocument, TiXmlElement, TiXmlText and TiXmlUnknown. The latter
includes TiXmlHandler. TinyXML comprises two head files, i.e. .h file, and four .cpp files, i.e. .cpp
file.

1) TiXmlBase: The base classes of the entire DOM model of an XML document.
2) TiXmlNode: The nodes in the DOM structure.
3) TiXmlAttribute: The attributes of elements in XML.
4) TiXmlComment: The notation in XML.
5) TiXmlDeclaration: Declaration part, e.g. < ? version="1.0" ?>.
7) TiXmlElement: Elements in XML.
8) TiXmlText: Text part of XML.
9) TiXmlUnknown: Unknown part in XML.
10) TiXmlHandler: Operations for an XML document.

3 Experiment and results

3.1 Generation and Analysis of Synthetic Data
3.1.1 Synthetic Data Generation

There is no standard test data set used to evaluate the effects of association rule algorithms for
XML documents. Figure 1 shows the XML document in our experiment, which is in the form of [5]. A
set of transactions identified by tag <transactions> are contained and the tag <transaction> identifies
each transaction in the transaction set. The itemset in a transaction is identified by the tag <items> and
each item in the itemset is identified by the tag <item>.
The three methods are tested on 10 different synthetic data sets. These synthetic data sets are simulated using the algorithm of Synthetic_Data_Generation, which generates an XML document with \( t_{\text{num}} \) transactions. The \( \text{max}_{\text{tsize}} \) is maximum transaction size, that is, each transaction probably contains 1 to \( \text{max}_{\text{tsize}} \) items. The \( \text{i}_{\text{num}} \) is the number of items, that is, the items in the basket could be 1 to \( \text{i}_{\text{num}} \). The matrix \( \text{item}_{\text{matrix}}(1 \times \text{i}_{\text{num}}) \) is defined to hold the occurrence probability of each item.

### Algorithm: Synthetic_Data_Generation

**Input:** Maximum transaction size \( \text{max}_{\text{tsize}} \), number of items \( \text{i}_{\text{num}} \), number of transactions \( \text{t}_{\text{num}} \), matrix \( \text{item}_{\text{matrix}} \) of items' occurrence probability  

**Output:** XML document \( D \) of transactions  

**Process:**  
1: /*add \( \text{t}_{\text{num}} \) transactions to XML document*/  
2: for (\( i=1; i <= \text{t}_{\text{num}}; i++ \))  
3: {  
4:   /*the number of transactions contained in transaction \( i \) is generated*/  
5:   randomly*/  
6:   \( \text{t}_{\text{size}} = \text{rand()} \ % \ \text{max}_{\text{tsize}} +1; \)  
7:   /*generate a basket whose number of items is \( \text{t}_{\text{size}} \), and the items with*/  
8:   range of 1- \( \text{i}_{\text{num}} \) are generated according to \( \text{item}_{\text{matrix}} \)*/  
9:   basket=getbasket(\( \text{i}_{\text{num}}, \text{t}_{\text{size}}, \text{item}_{\text{matrix}} \));  
10:   /*add transaction \( i \) with items in basket to the xml document \( D \)*/  
11:   addtransaction(\( D, i, \text{basket} \));  
12: }  
13: return \( D \);

### 3.1.2 Synthetic Data Analysis

Using the algorithm of Synthetic_Data_Generation, we create 10 XML documents with 1000, 2000,……,10000 transactions respectively. The \( \text{i}_{\text{num}} \) is set to be 50 and \( \text{max}_{\text{tsize}} \) 20. Figure 2 illustrates the statistical features of the synthetic data set with 5000 transactions.
3.2 Experimental results and analysis

In this section, we perform two types of experiments to evaluate the three approaches. The first type of experiments are performed in order to monitor the time required for extracting frequent itemsets over XML files by using the AprioriSP method with different preset values of $max_{len}$. The second type is performed in order to monitor the time needed to mine XML files by using three methods of Apriori, FP-growth and AprioriSP. Furthermore, some experiments are performed to monitor the time needed to generate association rules from frequent itemsets. The algorithms are implemented in the C/C++ language with Microsoft Visual C++ 6.0. All experiments are conducted on computers with an INTEL Core 2DuoProcessorT5470 and 1G memory, running on Windows XP.

3.2.1 Performance of the AprioriSP method

![Figure 3: Performance of AprioriSP in dataset with 5000 transactions]
Figure 3 shows, when using the AprioriSP, how extraction time of frequent itemsets depends on \textit{max\_len}, whose value is set to be 3, 5 and 7 respectively. As the minimum support increases, the total number of frequent itemsets decreases, and so the extraction time required by the AprioriSP with all preset pattern length limits also decreases. At first, e.g. given the \textit{min\_sup}=5\%, it costs less time for AprioriSP with \textit{max\_len}=3 than \textit{max\_len}=5, and the AprioriSP with \textit{max\_len}=7 spends most of time. The reason is that the number of frequent itemsets to be extracted when \textit{max\_len} is set to be 3 is less than that when \textit{max\_len} is set to be 5 and 7. However, the curves of extraction time meet as the minimum support increases because of the gradual loss of long pattern frequent itemsets. Figure 4 shows performance of AprioriSP in datasets with different transaction size, given \textit{min\_sup}=20\%. With three values of \textit{max\_len}, performance of AprioriSP increases as the transaction size increases. AprioriSP with \textit{max\_len}=3 spends less time because of the small number of short pattern frequent itemsets.

3.2.2 Performance comparison of three methods

Figure 5: Performance of algorithms in dataset with 5000 transactions

Figure 6: Performance of algorithms, given \textit{min\_sup}=20\%
Here some experiments have been performed to monitor the time needed to find frequent itemsets for Apriori, FP-growth and AprioriSP with $\text{max}_\text{len}=3$. Figure 5 shows the time the three algorithms take to find frequent itemsets with different support thresholds. Figure 6 shows, for each XML document we considered, the time the three algorithms take to obtain frequent itemsets. In both figures, the FP-growth algorithm is more efficient than the other two, because it need not repeatedly scan the database and check a large set of candidates by pattern matching. The AprioriSP is faster than Apriori because of the pattern length constraint.

### 3.2.3 Time of association rule generation

![Figure 7: Time of rule generation in dataset with 5000 transactions](image)

![Figure 8: Time of rule generation, given $\text{min}_\text{sup}=5\%$](image)

Figure 7 shows, for different support levels, the time to generate association rules from frequent itemsets, given $\text{min}_\text{conf}=70\%$. Figure 8 shows, for each XML document with a certain transaction size, the time to obtain association rules, given $\text{min}_\text{conf}=70\%$ too. The association rule generation time depends on the transaction size of an XML file and the total number of frequent itemsets in the document.

### 4. Conclusions

In this work we have proposed an Apriori based method AprioriSP to extract frequent association rules with a fixed pattern length constraint. The three algorithms of Apriori, FP-growth and AprioriSP are applied to mining association rules from XML documents, and some experiments are performed to compare their performances. These algorithms proved to be effective and efficient for XML mining, and their performances are largely dependent on the number of frequent itemsets. The larger the support threshold is, the smaller the number of frequent itemsets will probably be. As the support
increases, the number of frequent itemsets decreases, and consequently there is a decrease in execution time of algorithms. Among the three algorithms, the FP-growth is the most efficient one. However, it needs large amounts of space to store the FP-tree, which is constructed by compressing the database representing frequent items. When storage space is limited, the Apriori and AprioriSP methods are preferable to the FP-growth. Compared with the Apriori algorithm, the AprioriSP can provide a considerable improvement in time performance by pruning long frequent patterns when mining very large XML datasets, and could be applied to the recommendation system in electronic commerce.

5. Acknowledgement

This work is supported by National Natural Science Foundation of China (61142007). Furthermore, we are indebted to the support and encouragements received from the staff and colleagues of the school of computer engineering.

6. References

[10] Jiawei Han, Jian Pei, Yiwen Yin and Runying Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, 2004(8):53-87.