SMART

JOURNAL OF BUSINESS MANAGEMENT STUDIES

(An International Serial of Scientific Management and Advanced Research Trust)

Vol.5 **No. 2** **July - December 2009**

ISSN 0973 - 1598

Chief Editor
M. SELVAM, M.Com., Ph.D.,
Bharathidasan University,
India

SMART Journal is indexed and abstracted by Ulrich's Periodicals Directory, Intute Catalogue (University of Manchester) and CABELL'S Directory

SCIENTIFIC MANAGEMENT AND ADVANCED RESEARCH TRUST
(SMART)
TIRUCHIRAPPALLI (INDIA)
http://www.geocities.com/smartbard

PERFORMANCE ANALYSIS ON ASSOCIATION RULE IN DATA MINING

T. Muthukumar
Research Scholar, Sathyabama University, Chennai, Tamil Nadu, India
(Assistant Director-Board of Studies – The Institute of Chartered Accountants of India – New Delhi)
M. Ramasamy,
Dean, PG Studies, Madha Engineering College, Chennai, Tamil Nadu, India

Abstract

One of the most important problems in data mining is to find association rules. The association rule mining can be classified into two main categories: the level-wise algorithms and the tree based algorithms. The level-wise algorithm like Apriori, scan the entire database multiple times and also generate a huge number of candidate sets. It also needs to repeatedly scan the database and check a large set of candidates by pattern matching. Tree based algorithms, like FP-tree, scan the database only twice. One scan may be needed for FP-tree construction and another scan for adding new items into the tree. But it takes more time when new data are added to an existing database. Another tree based algorithm, P-Tree is constructed by a single scan of a database and it updates the P-tree by one scan of new data. The above said three algorithms are implemented by using C++/java/.net and their performances are evaluated by using synthetic dataset with respect to number of scanning of dataset. The performance study shows that in majority of cases, Pattern Tree achieves better performance and efficiency than Apriori and FP algorithms.

***Keywords**: Data mining, association rules, level wise algorithms, Pattern Tree.*

1. Introduction

Several algorithms have been proposed in the literature to address the problem of mining association rules [2]. Existing association rule mining algorithms suffer from many problems while mining massive transactional datasets. Some of these major problems are: (1) repetitive I/O disk scans, (2) the huge computation involved during the candidacy generation, and (3) the high memory dependency [2]. One of the key algorithms, which seems to be the most popular in many applications for enumerating frequent item sets, is the Apriori algorithm [2]. It uses a monotone property stating that for a k-item set to be frequent, all its (k-l) item sets have to be frequent. The use of this fundamental property reduces the computational cost of candidate frequent item set generation. However, in the case of extremely large input sets with big frequent 1-items set, the Apriori algorithm still suffers from two main problems of repeated I/O scanning and high computational cost.

Tree based algorithms, like FP-Tree, creates a compact tree-structure, representing frequent patterns that avoid costive candidate generation and repeated occurrence frequency checking against the support threshold. It therefore achieves better performance and efficiency than Apriori algorithms. However, the database still needs to be scanned twice to get the FP-tree. One scan may be needed for FP-tree construction and another scan for adding new items into the tree. This can be very time-consuming when new data are added to an existing database because two scans are needed for not only the new data but also the existing data. But another tree based algorithm, P-Tree, is constructed by a single scan of a database. The pattern tree is used to construct the corresponding FP-Tree with a specified support threshold from the P-Tree. Updating the P-Tree with new data, needs only one scan of the dataset. The existing data do not need to be rescanned.

Our performance study shows, for synthetic dataset, Pattern Tree achieves better performance and efficiency than Apriori and FP algorithms.

1.1. Objectives

The objective of the study is to analyze the impact of information and knowledge gained, on applications ranging from business management, production control and market analysis, to engineering design and science exploration.

(i) The major reason that data mining has attracted a great deal of attention in the information industry in recent years is due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge.

(ii) Association rule mining finds interesting association or correlation relationship among a large set of data items.

(iii) The discovery of interesting association relationship among huge amounts of business transaction records can help in many business decision making processes, such as catalog design, cross marketing, and loss-leader analysis.

The rest of the paper is organized as follows: In section 2 we review the Apriori algorithm . In section 3 and 4, construction of Pattern Tree, the generation of frequent tree from Pattern Tree and also the updating of p-tree are discussed. Performance study is mentioned in section 5 and we conclude our study in section 6. Section 7 contains figure & Tables and Section 8, Reference.

2. Apriori algorithm : Let I= {il,i2,i3,...,im} be a set of items. Let D, the task-relevant data, is a set of database transactions where each transaction T is a set of items such that T is a subset of I.

Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if A is the subset of T. An association rule is an implication A ℘ B, where A and B are set of items in the item set, and $A \cup B = \phi$. The rule A℘B holds in the transaction set D with supports, where s is the percentage of transaction in D that contains $A \cup B$. It becomes $P(A \cup B)$. The rule A℘B has confidence c in the transaction set D if c is the percentage of transactions in D containing A that also contains B. IT becomes conditional probability (A/B), ie. support (A℘B) =P $(A \cup B)$ and Confidence (A℘B) =P (B/A). Association rule mining is a two-step process [5]:

1. Find all frequent item sets. Each of these item sets will occur at least as frequently as predetermined minimum support count.
2. Generate strong association rules from the frequent item sets and these rules must satisfy minimum support and minimum confidence.

2.1. Illustrative Example: Let us look at the concrete example of Apriori, transaction dataset, D, of **Figure - 1**. There are nine transactions in the dataset, that is, |D|=9. **Figure - 3** illustrates the Apriori algorithm for finding frequent item sets in D [5].

3. Frequent Pattern Tree

3.1. Design and Construction : Let I = {al, a2, ..., am} be a set of items and a transaction database DS = {Tl, T2, ...Tn}, where Ti (Iε [l::n]) is a transaction which contains a set of items in I. The support of a pattern A, which is a set of items, is the number of transactions containing A in DB. A is a frequent pattern if A’s support is no less than a predefined minimum support threshold ε. Given a transaction database OB and a minimum support threshold, ε, the problem of finding the complete set of frequent patterns is called the frequent pattern mining problem.

3.2: Frequent Pattern Tree: To design a compact data structure for efficient frequent pattern mining, let's first examine an example. Let the transaction dataset DS, be (the first two columns of) **Figure - 3** and the minimum support threshold ε = 3[2].

A compact data structure can be designed based on the following observations.

I. Since only the frequent items will play a role in the frequent pattern mining, it is necessary to perform one scan of DS to identify the set of frequent items.

II. If we store the set of frequent items of each transaction in some compact structure, it may avoid repeatedly scanning of DS.

III. If multiple transactions share an identical frequent item set, they can be merged into one with the number of occurrences registered as count.

If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the count is registered properly. With these observations, one may construct a frequent pattern tree as follows. First, a scan of DB derives a list of frequent items, < (f:4), (c:4), (a:3), (b:3), (m:3), (p:3)>, (the number after ":" indicates the support), in which items are ordered in frequency descending order. This ordering is important since each path of a tree will follow this order. The frequent items in each transaction are listed in this ordering in the rightmost column of Figure - 3.

Second, one may create the root of a tree, labeled with "null". Scan the DS the second time. The scan of the first transaction leads to the construction of the first branch of the tree: <(f:l), (c:l), (a:l), (m:l) (p:l)> . For the second transaction, since its (ordered) frequent item list <f, c, a, b, m> shares a common prefix (f; c; a} with the existing path hf; c; a; m; pi, the count of each node along the prefix is incremented by 1, and one new node (b: 1) is created and linked as a child of (a:2) and another new node (m:l) is created and linked as the child of (b:l). For the third transaction, since its frequent item list {f; b} shares only the node {f} with the f-prefix sub tree, f's count is incremented by 1, and a new node (b:l) is created and linked as a child of (f:3). The scan of the fourth transaction leads to the construction of the second branch of the tree, <h(c:l), (b: 1), (p: 1)>. For the last transaction, since its frequent item list <f, c, a, *m*, p> is identical to the first one, the path is shared with the count of each node along the path incremented by 1.

After scanning all the transactions, the tree with the associated node-links is shown in **Figure - 4.**

Definition 1 (FP-tree) A frequent pattern tree is a tree structure defined below.

1. It consists of one root labeled as "null", a set of item prefix sub trees as the children of the root, and a frequent-item header table.
2. Each node in the item prefix sub tree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.
3. Each entry in the frequent-item header table consists of two fields, (1) item-name and (2) head of node-link, which points to the first node in the FP-tree carrying the item-name.

3.3. Analysis: From the FP-tree construction process, we can see that one needs exactly two scans of the transaction dataset, DS: the first collects the set of frequent items, and the second constructs the FP-tree. We will show that the FP-tree contains the complete information for frequent pattern mining.

3.4. Mining Frequent Patterns using FP-Tree: Construction of a compact FP-Tree ensures that subsequent mining can be performed with a rather compact data structure. In this section, we will study how to explore the compact information stored in a FP-Tree and develop an efficient mining method for mining the complete set of frequent patterns.

Property 1 : (Node-link) For any frequent item ai, all the possible frequent patterns that contain ai can be obtained by following ai's node-links, starting from ai's head in the FP-tree header.

Property 2 : (Prefix path) To calculate the frequent patterns for a node ai in a path P, only the prefix sub path of node ai in P need to be accumulated, and the frequency count of every node in the prefix path should carry the same count as node.

Lemma 3.1 (Fragment growth) Let α be an item set in DB, B is α's conditional pattern base, and β be an item set in B. Then the support of $\alpha \cup \beta$ in DB is equivalent to the support of β in B.

According to the definition of conditional pattern base, each (sub) transaction in B occurs under the condition of the occurrence of α in the original transaction database DB. If an item set β appears in B ψ times, it appears with α in DB ψ times as well.

Lemma 3.2: (Single FP-Tree path pattern generation) : Suppose a FP-tree T has a single path P. The complete set of the frequent patterns of T can be generated by the enumeration of all the combinations of the sub paths of P with the support being the minimum support of the items contained in the sub path.

Rationale. Let the single path P of the FP-tree be <al:sl a2:s2 ... : ak :sk>. The support frequency si of each item ai (for 1 i k) is the frequency of ai co-occurring with its prefix string. That is the combination such as <ai, ... , aj > (for 1 i, j k), is a frequent pattern, with their co-occurrence frequency being the minimum support among those items.

4. Pattern Tree Algorithm

4.1. Pattern Generation with the Pattern Tree: The FP-tree based method has to scan the database twice to get a FP-Tree. The central idea of FP-Tree is to get the list L of item frequencies in the first time and then construct the FP-Tree in the second time according to L.

1. A Pattern Tree not only contains the frequent items but also contains all the items in the transaction set. We can obtain a P-Tree through one scan of the database and get the corresponding FP-Tree from the P-Tree later. The construction of a P-Tree can be divided into two steps, i.e., the transaction database and a minimum support threshold are given (figure 5 & figure 6).
2. While retrieving transaction from a database, first we have to sort the items of each transaction in some order (alphabetic, numerical or other specific order).
3. After the first and only scan of the database, we sort L according to item supports. The restructure of the P-Tree consists of similar insertions in the first step. The only difference is that one needs to sort the path according to L before inserting it into a new tree.
4. The approach makes the best use of the occurrence of the common suffix in transactions, thereby constructing a more compact tree structure than FP-Tree.

4.1.1 Analysis : **Figure -7** shows P-Tree for the dataset specified in figure 5. The construction of P-Tree requires exactly one scan of the database and initial P-Tree. FP-Tree is constructed faster for more high frequent patterns in the database than the less frequent patterns in the database. The lower and upper bound is the runtime of one scan and two scan of the database respectively.

4.2 : Generation of FP-Tree from the P-Tree : FP-Tree is a sub-tree of the P-Tree with a specified support threshold, which contains those frequent items that meet this threshold and hereby we can exclude all infrequent items. After the generation of the P-Tree, we can easily get the frequent item list given a specific support threshold. Next, we have to remove infrequent items from the frequency list. Then, we prune the P-Tree to exclude the infrequent nodes by checking the frequency of each node along the path from the root to leaves. It is already defined that frequency of each node is not less than that of its children or its descendents. So we can delete the node and its sub trees at the same time if it is infrequent.

4.2.1: Analysis: Figure- 8 shows FP-Tree which is constructed from the P-Tree. In practice, we can compare the user-defined minimum support threshold with the occurrence recorded in the item frequency list. So the pruning could be done according to the following rules:

1. If the mining support threshold is higher than the occurrence of most items, then we can check the items along the path beginning from the root.

2. When the occurrence of most items is above the minimum support threshold, we can check the items along the path beginning from the leaves in the inverse order following the first rule.

3. Regardless of which rule is applied, the algorithm checks at most half the amount of items in a pattern tree. If the support threshold is set too high, the process may produce fewer frequent items and some important rules can not be generated.

4.3: Pattern Tree updating with new data: There are two ways to update a FP-Tree. One is to apply the construction algorithm to the new database. Next is to set a validity support threshold (watermark) which is used in [6]. The water mark goes up to exclude the originally infrequent items while their frequency goes up. But unlike FP-Tree, we are able to update P-Tree by one scan of new data without the need for two scans of the existing database and the second scan for the new data. We can first insert the new transactions into the P-Tree according to the item frequency list and update the list. Then a new P-Tree can be restructured according to the updated item frequency list. In case there comes a new item, which does not appear in the existing data base, we can assume its support is 0 and append it as a leaf node.

4.3.1: analysis: The major problem concerning the FP-tree is to handle updates in the database. Once some new transactions are added, a new FP-tree has to be constructed to deal with these changes. But the advantages of P-tree algorithm for updating the database are as follows.

1. There is no further need to scan the existing database.
2. We need to scan the new data only once. But a FP-Tree is obtained by two scans of the entire database, including the existing and new database.

5. Tests and Results: In this section we present a performance analysis of FP-Tree with the frequent pattern algorithm Apriori and a recently proposed method of Pattern Tree.

Pattern Tree Technique is implemented with C++. We have performed experiments with multiple FP-Tree generation and FP-Tree updating with P-Tree while new data are added. To test the efficiency of the Pattern tree, we have implemented Apriori algorithm and FP-Tree specified on [4] by using C++ and executed for various synthetic dataset in ' Pentium IV processor, 256 Mbytes RAM, l.8 GHz CPU machine with windows 2000 operating system. Our test results show that Pattern Tree method outperforms for synthetic datasets FP-Tree and Apriori algorithm.

6. Conclusion: In this paper, we have discussed about how to obtain the P-Tree by one database scan and how to update the P-Tree by one scan of new data and we also discussed how to get the corresponding FP-Tree from the P-Tree with different user specified thresholds.

Based on the study, Pattern tree method is preferable to the traditional minimum support algorithm Apriori and the Frequent Pattern Tree pattern. Apriori and FP Tree require more than one scan of database whereas P-Tree method scan the database only once and the updating of the database also requires only one scan.

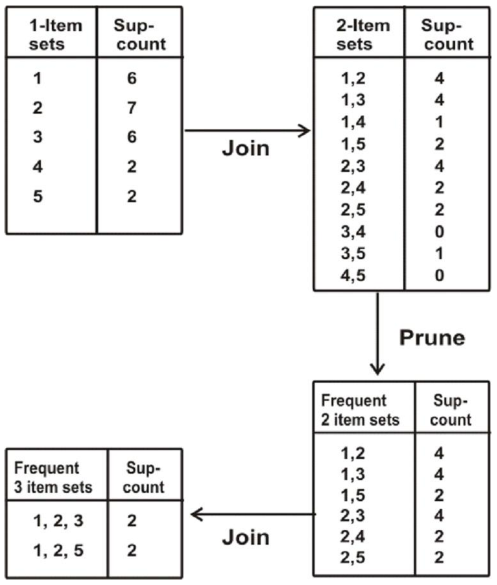
References :

1. Mohammad El-Hajj, Osmar R.Zaiane, COFI-tree mining: A New Approach to Pattern Growth with Reduced Candidacy Generation.
2. J.Han, J. Pel, J. Yin. Mining Frequent Patterns without Candidate Generation. In: Proc. of SIGMOD'00. pp. 1-12, 2000.
3. M.Y.Chen, J.Han, and P.Yu. Data Mining: An overview from a Database Perspective. An IEEE Transactions on knowledge and Data Engineering.
4. H.Huang, X. Wu and R.Retue. Association Analysis with one scan of databases. University of Vermont computer science technique report.
5. Jaiwei Han, Micheline Kamber.Data Mining concepts and Techniques. (Morgan Kaufmann publishers).
6. Wuhan University Journal of Natural Sciences, Multi-dimensional customer data analysis in online auctions Wuhan University Journals Press, Volume 12, Number 5 / September, 2007.
7. Dejiang Jin and Sotirios G. Ziavras, "A Super-Programming Approach For Mining Association Rules in Parallel on PC Clusters, " IEEE Trans. Parallel and Distributed Systems, Vol. 15, No. 9, Sept. 2004.
8. Ke Wang, Yu He, and Jiawel Pushing Support Constraints into Association rules Mining IEEE transaction on knowledge and data engineering May 2003.
9. Margaret H. Dunham Data Mining and Advanced Topics Pearson Education 2004.
10. Coenen, F., Leng, P., Goulbourne, G. Tree Structures for Mining Association Rules. Journal of Data Mining and Knowledge Discovery, Vol 15 (7), pp391- 398.
11. Dr. E. Ramaraj and N. Venkatesan, Discrete Topological Mining Association Rules-An Approach, IRIS'06 at National Engineering College, January-2006.

Figure - 1 : Transaction dataset

T-id	List of item I Ds
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Figure - 2 : Generation of candidate item sets and Frequent item sets with the minimum support = 2



1-Item sets	Sup-count
1	6
2	7
3	6
4	2
5	2

2-Item sets	Sup-count
1,2	4
1,3	4
1,4	1
1,5	2
2,3	4
2,4	2
2,5	2
3,4	0
3,5	1
4,5	0

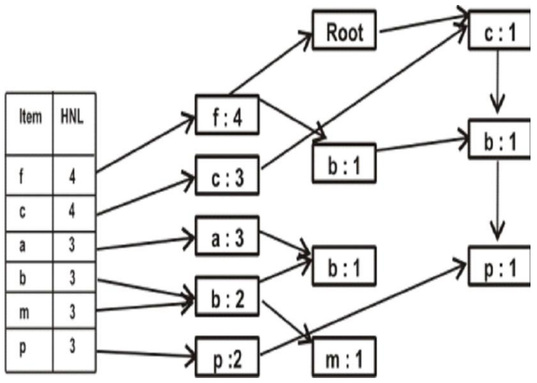
Frequent 2 item sets	Sup-count
1,2	4
1,3	4
1,5	2
2,3	4
2,4	2
2,5	2

Frequent 3 item sets	Sup-count
1, 2, 3	2
1, 2, 5	2

Figure - 3 : A Transaction Dataset DS

TID	Items	Frequent items
100	f,a,c,d,g,I,m,p	f,c a,m,p
200	a,b,c,f,l,m,a	f,c,a,b,m
300	b,f,n,j,o	f,b
400	b,f,c,k,a,p	c,b,p
500	a,f,c,e,l,p,m,n	f,c,a,m,p

Figure - 4 : The FP-tree construction



Item	HNL
f	4
c	4
a	3
b	3
m	3
p	3

Figure - 5 : A transaction database TDB

TID	Items	Ordered Items
T100	i,c,d,e,g,h	c,d,e,h,l,g
T200	m,a,p,c,e,d	c,d,e,a,p,m
T300	a,l,b,d,e,g	d,e,a,b,l,g
T400	b,a,d,h,c,n,	c,d,a,h,b,n
T500	a,e,c,	c,e,a,
T600	n,a,c,d,e,	c,d,e,a,n
T700	p,,b,c,d,e,h	c,d,e,a,h,b,p

Figure - 6 : FL-List

Item	Count
D	6
c	6
e	6
a	6
h	3
b	3
i	2
g	2
n	2

Figure - 7
The P-tree constructed from TDB

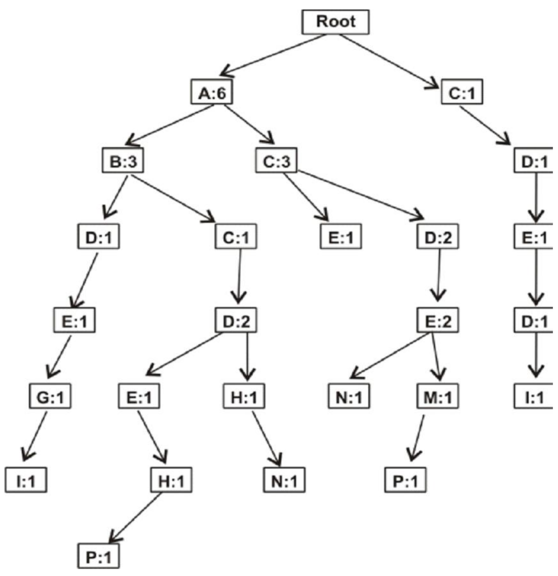
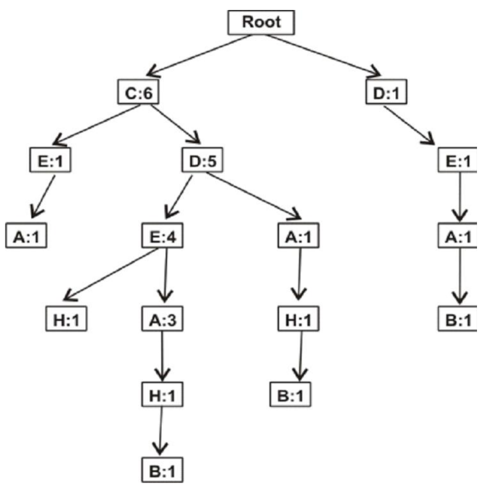


Figure - 8
FP-tree constructed from P-tree in figure 3


Root
C:6
D:1
E:1
D:5
E:1
A:1
E:4
A:1
A:1
H:1
A:3
H:1
B:1
H:1
B:1
B:1