BINARY DECISION TREE FOR ASSOCIATION RULES MINING IN INCREMENTAL DATABASES

Amaranatha Reddy P, Pradeep G and Sravani M

Department of Computer Science & Engineering, SoET, SPMVV, Tirupati

ABSTRACT

This research paper proposes an algorithm to find association rules for incremental databases. Most of the transaction databases are often dynamic. Suppose consider super market customers daily purchase transactions. Day to day customer's behaviour to purchase items may change and new products replace old products. In this scenario static data mining algorithms doesn't make good significance. If an algorithm continuously learns day to day, then we can get most updated knowledge. This is very much helpful in present fast updating world. Famous and benchmarked algorithms for Association rules mining are Apriory and FP- Growth. However, the major drawback in Appriory and FP-Growth is, they must be rebuilt all over again once the original database is changed. Therefore, in this paper we introduce an efficient algorithm called Binary Decision Tree (BDT) to process incremental data. To process continuously data we need so much of processing and storage resources. In this algorithm we scan data base only once by which we construct dynamic growing binary tree to find association rules with better performance and optimum storage. We can apply for static data also, but our main intension is to give optimum solution for incremental data.

KEYWORDS

Data Mining, Association Rules, Frequent Patterns, Incremental Database, Binary Decision Tree (BDT).

1. INTRODUCTION

Data Mining, also referred as Knowledge Discovery in Databases (KDD), is a process of finding new, interesting, previously unknown, potentially useful, and ultimately understandable patterns from very large volumes of data [1]. The regularities or exceptions discovered from databases through data mining have enabled human decision makers to better make decisions in many different areas [1]. One important topic in data mining research is concerned with the discovery of interesting association rules.

It aims to find interesting association or correlation relationships among a large set of data items. The discovery of interesting association relationships among huge amounts of business transaction records can help in many business decision making process. It can be used to improve decision making in a wide variety of applications such as: market basket analysis, medical diagnosis, bio-medical literature, protein sequences, census data, logistic regression, fraud detection in web and CRM of credit card business etc.

A typical and widely-used example of association rule mining is Market Basket Analysis. For example, data are collected using bar-code scanners in supermarkets. Such market basket databases consist of a large number of transaction records. Each record lists all items bought by a customer on a single purchase transaction. Managers would be interested to know if certain groups of items are consistently purchased together. They could use this data for adjusting store layouts (placing items optimally with respect to each other), for cross-selling, for promotions, for catalogue design and to identify customer segments based on buying patterns [5].

Association rules provide information of this type in the form of “if-then” statements. These rules are computed from the data and, unlike the if-then rules of logic, association rules are probabilistic in nature. In addition to the antecedent (the “if” part) and the consequent (the “then” part), an association rule has two numbers that express the degree of uncertainty about the rule. In association analysis the antecedent and consequent are sets of items (called item sets) that are disjoint (do not have any items in common).

The first number is called the support for the rule. The support is simply the number of transactions that include all items in the antecedent and consequent parts of the rule (the support is sometimes expressed as a percentage of the total number of records in the database). The other number is known as the confidence of the rule. Confidence is the ratio of the number of transactions that include all items in the consequent as well as the antecedent (namely, the support) to the number of transactions that include all items in the antecedent.

2. PROBLEM STATEMENT/ RELATED WORK

Many algorithms are proposed to find association rule. Bottlenecks in most of the algorithms are

1. Multiple scans of data base.
2. Intermediate candidate generation.
3. Not suitable for incremental data.

We went through few of the fundamental algorithms like Apriori and FP tree.
Apriori is a great improvement in the history of association rule mining, Apriori algorithm was proposed by Agrawal in [Agrawal and Srikant 1994] [3].

It follows two step process consisting of join and prune actions. In the join step candidate item sets are generated, then in the prune step database is scanned to check the actual support count of the corresponding item sets and remove the item sets whose support is below the threshold.

Bottlenecks of the Apriori algorithm are complex candidate generation process that takes more time, space, memory and multiple scans of the database. Changing of support may lead to repetition of the whole mining process. Another limitation is it is not suitable for incremental mining. Since as time goes on databases keep on changing, new datasets may be inserted into the database, those insertions may also lead to a repetition of the whole process if we employ Apriory algorithm.

Pattern mining, is another milestone in the development of association rule mining. The frequent item sets are generated with only two passes over the database and without any candidate generation process. FP-Tree was introduced by Han et al in [Han et al. 2000] [4].

The first scan of the database is the same as Apriori, which derives the set of frequent items (1-item sets) and their support count. The set of frequent items is sorted in the order of descending support count. An FP-Tress is then constructed as follows.

First, create the root of the tree labelled with “null”. Scan the database second time. The items in each transaction are processed in order (i.e., sorted according to descending support count) and a branch is created for each transaction. 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 links.

The mining of the FP-tree proceeds as follows. Start from each frequent length 1 pattern construct its conditional pattern base then construct its FP-tree, and perform mining recursively on such tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

By avoiding the candidate generation process and less passes over the database, FP-Tree is an order of magnitude faster than the Apriori algorithm. Every algorithm has its limitations, for FP-Tree it is difficult to be used in an interactive mining system. During the interactive mining process, users may change the threshold of support according to the rules. However for FP-Tree the changing of support may lead to repetition of the whole mining process. Another limitation is it is also not suitable for incremental mining. Since as time goes on databases keep changing, new datasets may be inserted into the database, those insertions lead to a repetition of the whole process because in first scan we assign individual items in descending order based on support count, but in incremental data while we add new data it changes the support count and order of the items. Hence FP- Tree algorithm is not suitable for incremental databases [7].

3. BINARY DECISION TREE (BDT)

Many algorithms exist like this but not able to overcome completely these drawbacks. We tried to give optimum solution for these problems by the Binary Decision Tree(BDT) Algorithm.

We consider transactions of a supermarket as a case study to describe this algorithm. We call each transaction as a record and we convert reach record into corresponding bitpattern. For example consider I1, I3, I4, I6 is one transaction then the corresponding bit pattern is 101101(i.e. if the item is present then corresponding bit is ‘1’ otherwise ‘0’.

Binary Decision Tree(BDT) is a binary tree (i.e each node will have at most 2 childs.) in which each node contains 3 fields namely check_bits, decision_bit and support_count.

check_bits	decision_bit	supportcount

Figure 1: node of a Binary Decision Tree

Here check_bits field of a node used to represent each record. It is used for identification purpose. decision_bit field is used for taking decision for upcoming records. If the corresponding decision_bit is ‘0’ then traverse towards left child otherwise traverse towards rightchild of node. support_count field of a node describes number of transactions ending with the node(i.e no need to traverse farther.)

This algorithm scans database only once record by record and updates tree correspondingly. Item support_count may increase or decrease in future in incremental databases. So we are not considering minimum support count.

3.1. Binary Decision Tree (BDT) construction

Binary decision tree construction algorithm shown bellow.

```
Step 1: scan the database.
Step 2: for each record do the following steps.
Step 3: if it is the first record then place items in check_items, decision_item NULL and with
        support_count value 1.
step 4: else
            update_DBT(present_record_items, DBT_root_node)
step 5: end;
```

update_DBT(record_items, DBT_node)

```
step 1: compare the same index items present in record with DBT_node check_items
        sequentially
step 2: if all items are same
step 2.1: compare with decision_item
step 2.1.1: if it is also same
step 2.1.1.1: if it is the last item in the present record increment support_count by 1.
step 2.1.1.2: else update_DBT(present record items, DBT_node_right-child)
step 2.1.2: else update_DBT(present record items, DBT_node_left-child)
step 3: else create_DBT_new_node(record_items, DBT_node, varied_item_index)
step 4: return;
```

create_DBT_new_node(record_items, DBT_node, item_index)

```
step 1: create new node with item_index as decision_item, items between decision_items of
        parent and present node as check_items and support_count with 0.
step 2: if the decision_item present in record_items
step 2.1: if it is last item increment support count of new notde by 1
step 2.2: else create new node as right-child for decision node with remaining items in record
          after decision item as check_items, decision item 0, support_count 1. make the
          DBT_node as left-child with after decision item as check_items.
step 3:  else create new node as left-child for decision node with remaining items in record after
        decision item as check_items, decision item 0, support_count 1. make the DBT_node as
        right-child with after decision item as check_items.

Step 4: return;
```

Following example describes construction of binary decision tree.

Table 1: Super market transactions data

TID	List of item_ids
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

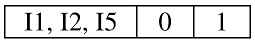

Figure 2: After insertion of 1st record

First record T100 having items I1, I2, I5. These items have to place in check_bits field, decision_bit not required so 0 and support_count is 1.

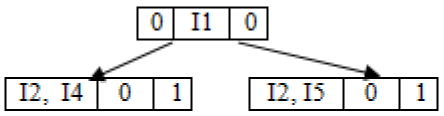

Figure 3: After insertion of 2nd record

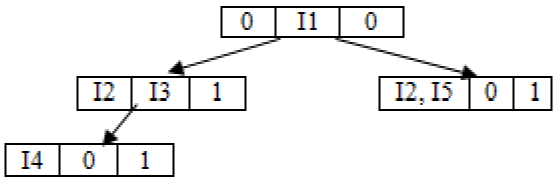

Figure 4: After insertion of 3rd record

Second record T200 having items I2, I4 to insert these items compare the items with the check_bits in root node. In T100 I1 bit pattern is 1 and in root node check_bits I1 pattern is 0. So make I1 as a decision_bit. In T100 decision_bit I1 is present that is it's bit pattern 1. So traverse towards right child and place T100 items except decision_bit as check_bits and support_count is 1. In T200 decision_bit is 0. So traverse towards left child and place T200 items as check_bits and it's support_count as 1.

Third record T300 having items I2, I3 to insert these items compare the T300 items with root node. Root node not having check_bits then compare with decision_bit. Corresponding bit I1 is 0 in T300. So traverse towards left child. Left child having I2 as check_bits so compare with corresponding bit in T300. T300 also having I2 then compare with decision_bit I3. I3 is 0 in T300. So traverse towards left child and place T300 item remaining as check_bits and with support_count 1.These process will continue for all the transactions as shown in figures.

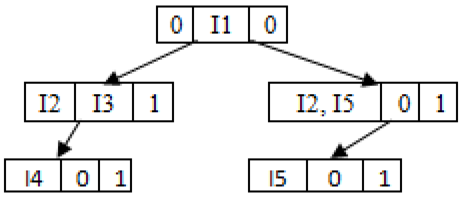

Figure 5: After insertion of 4th record

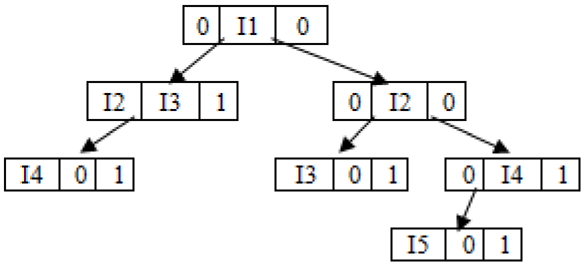

Figure 6: After insertion of 5th record

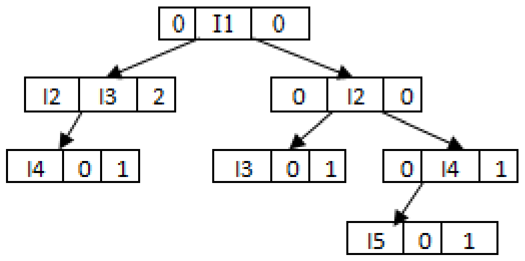

Figure 7: After insertion of 6th record

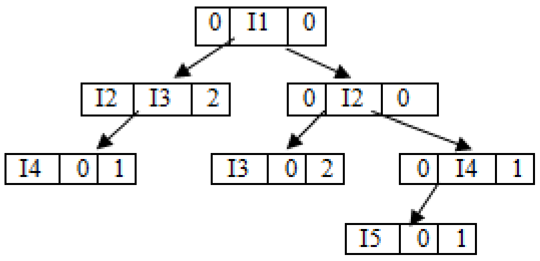

Figure 8: After insertion of 7th record

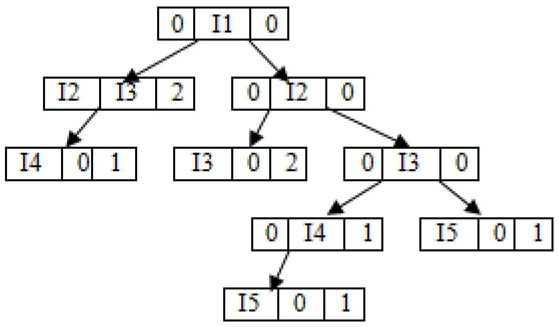

Figure 9: After insertion of 8th record

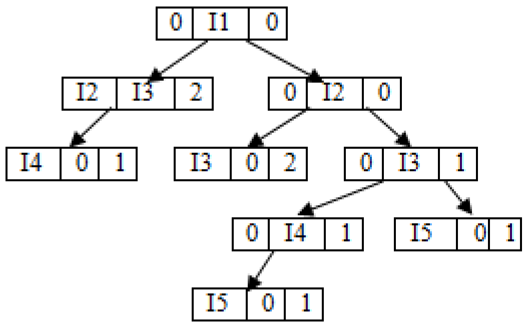

Figure 10: After insertion of 9th record

3.2. Binary Decision Tree (BDT) mining process

The mining of the DBT proceeds as follows.

Step 1: Create a bit pattern for required item set.
Step 2: Start traversing from root
Step 3: Pickup one by one bit pattern sequentially compare with the corresponding check_bit items at each node if check_bit is NULL or all check_bits are 1.
Step 3.1: If stills bits are present in required item set then compare with corresponding decision bit. If the bit pattern is 1 traverse towards right-child. If bit pattern is 0 traverse towards toward both child left-child and right-child.
Step 3.2: else end traversing.
Step 4: after the completion of bit pattern to find support count of required item set add support_count s of end node and all its right-childs.
Step 5: end;

For example to find support count of I2, I3 item set the DBT mining process as follows.
Bit pattern for I2, I3 is 011. Start traversing from root. Root is having chek_bit NULL. So compare with decision_bit I1. I1 bit is 0 in our item set. So traverse towards both left-child and right-child. In the left-child check bit is I2; I2 bit pattern is 1 in item set. There is no further check_bits so compare with decision_bit I3, corresponding bit pattern is 1. There are no forther items to traverse in the item set so stap traversing. In the right-child check_bit is NULL. So compare with decision_bit I2, I2 bit pattern is 1 so again traverse towards right-child is having check_bit NULL. So compare with decision_bit I3 corresponding bit pattern is 1. No further bits in itemset so stap traversing[6].

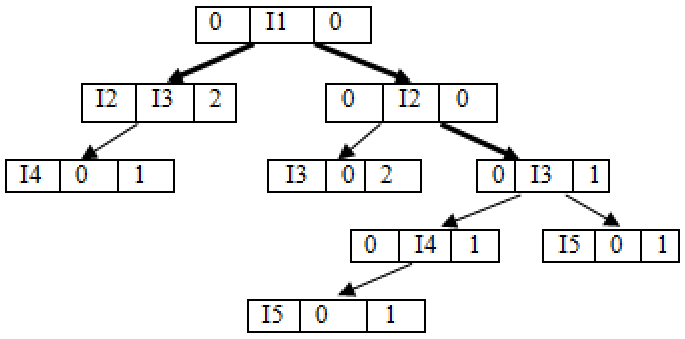

Figure 3: After insertion of 8th record

We are having 2 end nodes. First end node is not having right-child and current end node support count is 2. Other end node support count is 1 and it is having one right-child with support count 1. So total support count is 4 for item set I2, I3.

3. CONCLUSIONS

In this paper we proposed an algorithm called Decision Binary Tree (DBT) by which we can find out support count of any item set. The significance of this algorithm is it scans database only once and it accepts incremental databases without repeating the processing steps. Our algorithm is not generating candidates like Apriori. Hence we are not having problem of redundancy. In this algorithm we are not filtering item sets below the minimum support count. So based on user requirement we can check support count of any item set.

REFERENCES

[1] Cai.Y, Cercone. N, Han.j (1991) “Attribute-Oriented Induction in Relational Databases” in Knowledge Discovery in Databases AAAI/MIT Press 1991, pp. 213–228.
[2] Fayyad.U, Piatetsky-shapiro.G, Smyth.P (1996) “Knowledge Discovery and Data Mining: Towards a Unifying Framework” in KDD-96 Conference Proceedings, AAAI Press.
[3] R. Agrawal, R. Srikant(1994), Fast algorithms for mining association rules, in: Proc. of International Conference on Very Large DataBases, Santiago, Chile.
[4] Han J.,J.Pei,and Y.Yin(May 2000), Mining frequent patterns without candidate generation.inn Proc.2000 ACM-SIGMOD Int.Conf.Management of Data(SIGMOD’00), Dalas,TX.
[5] Savi Gupta and Roopal Mamtora (2014), “A Survey on Association Rule Mining in Market Basket Analysis” in International Journal of Information and Computation Technology. ISSN 0974-2239 Volume 4, Number 4 (2014), pp. 409-414.
[6] Med El Hadi Benelhadj, Khedija Arour, Mahmoud Boufaida and Yahya Slimani(2011) “A binary based approach for generating association rules” in International Conference on Data Mining DMIN'11 pp. 140-146.
[7] P. Amranatha Reddy, K. Brahma Naidu, Sreeram Keerthy (2012) “Mutually independent candidate algorithm for association rules mining”in International Conference on Computer Science and Engineering (ICCSE 2012) pp. 98-103.
[8] Pratiyush Guleria and Manu Sood (2014) “Data Mining in Education: A Review on Knowledge Discovery Perspective”.in International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol.4, No.5.
[9] [Online] Available: http://www.en.wikipedia.org/wiki/Association_rule_learning/
[10] [Online] Available: http://www.http://kdd.ics.uci.edu/
[11] [Online] Available: http://www.http://www.kdnuggets.com/datasets/

AUTHORS

Amaranatha Reddy.P received B.Tech degree in CSE from JNTU, Anantapur in 2010 and the M. Tech degree in Software Engineering from JNTU, Hyderabad in 2012. He is currently working as a Academic Consultant in School of Eng & Tech, SPMVV, Tirupati, Andhrapradesh from 2013.

Pradeep.G received B.Tech degree in CSE from JNTU, Anantapur in 2011 and Currently pursuing M.Tech degree in CSE from Swetha Institute of Tech & Science.

Sravani.M pursuing B.Tech degree in CSE from School of Engineering & Technology, Sri padmavati mahila visvavidyalayam, Tirupti, Andhrapradesh. She is very much interested in java programming.