COMPARATIVE STUDY OF DATA MINING ALGORITHMS

Gabriel Toma-Tumbar

University of CraiovaFaculty of Automation, Computers and Electronics
Computer and Communications Engineering Department

Abstract: Mining frequent patterns in transaction databases, time- series databases, and many other kinds of databases has been studied popularly in data mining research. Most of the previous studies adopt an Apriori-like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there exist prolific patterns and/or long patterns. We have compared two of the most common data mining algorithms: Apriori and FPGrowth

Keywords: Data mining, association rules, Apriori, FPTree, Boolean association rules.

1. OVERVIEW

The progress in the data collecting technology as barcode readers, industrial sensors, a.s.o. are all generating a huge amount of data. This explosive growth in the database dimensions has ‚generated' the need to develop new techniques and new instruments that should permit automatic intelligent transformation of this data into useful information and knowledge. Data mining is offering a series of such techniques.

Data mining, also known as Knowledge Discovery in Database (KDD) is the process of discovering new and hidden knowledge and potentially useful relations (association rules, trends, etc.) from very large databases.

2. DATA MINING TASKS

In practice, at the highest level, the main goals of the data mining systems may be classified into two categories:

- Prediction – infers the values of the current data from the databases with the goal to predict unknown or future values
- Description – realizes a data characterization that is easily interpretable by humans

These objectives are carried out by the following basic data mining tasks:

- classification – the task of determining a function that classifies the data in one or more predefined classes
- regression – the task of determining a function that permits the evaluation of real data
- clustering – the task that groups data with similar characteristics into classes or clusters. The grouping is based on similarity metrics.
- Rule generation – the task of determining or generating rules from data. The association rules are relations between the attributes of a transactional database.
- Summarizing or condensation – the task that determines a compact description for a set of data.
- Sequence analysis – this task determines sequential patterns from data.

3. BOOLEAN ASSOCIATION RULES

An important task in data mining is the process of discovering association rules. An association rules describes interesting relations between different attributes and/or objects. A classic example of using association rules is the market basket analysis, used to determine potential relations between the products purchased by the customers. These discovered associations may help producers to elaborate marketing strategies keeping into account the products that are bought more frequent together. An example of such an association rule is the following: 86 % of the customers that purchased bread also purchased butter.

3.1 Formal definition

Let $I = \{i_1, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \in I$. Associated with each transaction is a unique identifier, called its TID. We say that a transaction T contains X, a set of some items in I, if $X \subseteq T$

Definition 1.
A subset $X\{i_1, ..., i_k\} \subseteq I$ is called an itemset. An itemset that contains k articles is called a k-itemset.

3.2 Apriori Algorithm

The first algorithm used to determine the frequent item sets and to generate the Boolean association rules was the AIS algorithm introduced by A. Agrawal. The Apriori algorithm, introduced by the same author adds a major improvement to the history of determining the association rules. The Apriori algorithm tries to reduce the high number of database scans in order to determine the support, by significantly reducing the number of candidate item sets. The basis for this reduction is the following property (the Apriori property).

Apriori property. If X is frequent in DB, then any item set $Y \subseteq X$ is frequent in DB.

Corollary.
If an itemset X contains a subitemset that is not frequent, then the X itemset is not frequent.

Corrolary 2
If a k-itemset contains a (k-1)-itemset unfrequent, then the k-itemset is also unfrequent.
Apriori algorithm contains two important steps
1. the union step: at this step, in order to determine the frequent k-itemsets, L_k, there is generated a set C_k of candidate k-itemsets, superset of L_k by making the union between L_{k-1} and L_{k-1}.

As a convention, we assume that the items contained in the itemsets are lexicographically oredered.

$$C_k = L_{k-1} \times L_{k-1} = \{\{x_1, ..., x_{k-1}, y_{k-1}\} \mid X \in L_{k-1}, Y \in L_{k-1}, X = \{x_1, ..., x_{k-2}, x_{k-1}\}, Y = \{y_1, ..., y_{k-2}, y_{k-1}\}, x_1 = y1 \wedge ... \wedge x_{k-2} = y_{k-2} \wedge x_{k-1} < y_{k-1}\}$$

2. Reduction step: At this step, from the previously generated set C_k are eliminated, based on the Corollary 2, the itemsets that contain (k-1) subitemsets that do not belong to L_{k-1}. This test can be quickly carried out by keeping a hashtree containing all frequent itemsets.

Example 1
Let's consider the DB database from Table 1 with all transactions from a store. We notice that there are 9 transactions in the database, $|DB| = 9$. The Figure 1 presents the steps of applying the Apriori algorithm for determining the frequent itemsets of DB.

The first iteration of the algorithm considers every itemset from I, a candidate 1-itemset $C_1 = \{I_1, I_2, ..., I_9\}$. The algorithm scans the entire database and determines the count for each item.

TID	Items
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

Table 1 – Example of a database containing transactions from a store

Scan DB $\rightarrow$

Itemset	Frequency
$\{I_1\}$	6
$\{I_2\}$	7
$\{I_3\}$	6
$\{I_4\}$	2
$\{I_5\}$	2

(A) C_1

Compare the frequency With minimum support =2 $\rightarrow$

Itemset	Frequency
$\{I_1\}$	6
$\{I_2\}$	7
$\{I_3\}$	6
$\{I_4\}$	2
$\{I_5\}$	2

(B) L_1

Generate C_2 from L_1 $\rightarrow$

Itemset	Frequency
$\{I_1,I_2\}$	4
$\{I_1,I_3\}$	4
$\{I_1,I_4\}$	1
$\{I_1,I_5\}$	2
$\{I_2,I_3\}$	4
$\{I_2,I_4\}$	2
$\{I_2,I_5\}$	2
$\{I_3,I_4\}$	0
$\{I_3,I_5\}$	1
$\{I_4,I_5\}$	0

(C) C_2

Compare the frequency With minimum support=2 $\rightarrow$

Itemset	Frequency
$\{I_1,I_2\}$	4
$\{I_1,I_3\}$	4
$\{I_1,I_5\}$	2
$\{I_2,I_3\}$	4
$\{I_2,I_4\}$	2
$\{I_2,I_5\}$	2

(D) L_2

Generate C_3 from L_2 $\rightarrow$

Itemset	Frequency
$\{I_1,I_2,I_3\}$	2
$\{I_1,I_3,I_5\}$	2

(E) C_3

Compare the frequency With minumum support=2 $\rightarrow$

Itemset	Frequency
$\{I_1,I_2,I_3\}$	2
$\{I_1,I_3,I_5\}$	2

(F) L_3

Figure 1 – Using the Apriori algorithm

Further it is presented the result of using the Apriori algorithm, implemented in Java, on the database containing the transactions from Table 1.

Itemset	Support
I1	0.6666666666666666
I2	0.7777777777777778
I3	0.6666666666666666
I4	0.2222222222222222
I5	0.2222222222222222
I1, I2	0.4444444444444444
I1, I3	0.4444444444444444
I1, I5	0.2222222222222222
I2, I3	0.4444444444444444
I2, I4	0.2222222222222222
I2, I5	0.2222222222222222
I1, I2, I3	0.2222222222222222
I1, I2, I5	0.2222222222222222

Table 2 - Apriori results for the items in DB

The next table presents the results of the Apriori algorithm applied to the determined frequent itemsets, in order to discover the association rules.

Antecedent	Consequent	Support	Confidence
I4	I2	0.2222222222222222	1.0
I5	I2	0.2222222222222222	1.0
I5	I1, I2	0.2222222222222222	1.0
I5	I1	0.2222222222222222	1.0
I2, I5	I1	0.2222222222222222	1.0
I1, I5	I2	0.2222222222222222	1.0

FPGrowth

The Apriori heuristic achieves good performance gain by (possibly significantly) reducing the size of candidate sets. However, in situations with prolific frequent patterns, long patterns, or quite low minimum support thresholds, an Apriori-like algorithm may still suffer from the following two nontrivial costs:

- It is costly to handle a huge number of candidate sets. For example, if there are 104 frequent 1-itemsets, the Apriori algorithm will need to generate more than 107 length-2 candidates and accumulate and test their occurrence frequencies. Moreover, to discover a frequent pattern of size 100, such as $\{a_1, \ldots a_{100}\}$, it must generate more than $2^{100} = 10^{30}$ candidates in total. This is the inherent cost of candidate generation, no matter what implementation technique is applied.
- It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.

Definition (FP-tree) A frequent pattern tree (or FP-tree in short) is a tree structure defined below.

1. It consists of one root labeled as "null", a set of item prefix subtrees as the children of the root, and a frequent-item header table.
2. Each node in the item prefix subtree 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.

Based on this definition, we have the following FP-tree construction algorithm.

Algorithm 1 (FP-tree construction)
Input: A transaction database DB and a minimum support threshold mTh.
Output: Its frequent pattern tree, FP-tree
Method: The FP-tree is constructed in the following steps.

1. Scan the transaction database DB once. Collect the set of frequent items F and their supports. Sort F in support descending order as L, the list of frequent items.
2. Create the root of an FP-tree, T, and label it as "null". For each transaction Trans in DB do the following.

Select and sort the frequent items in Trans according to the order of L. Let the sorted frequent item list in Trans be [pjP], where p is the first element and P is the remaining list. Call insert_tree([pjP]; T).

The function insert_tree([pjP]; T) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link be linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P;N) recursively.

We have the following algorithm for mining frequent patterns using FP-tree.

Procedure FP-Growth(Tree, α)
1: if Tree contains a single path P then
2: then for each combination (denoted as β) of the nodes in the path P do
3: generate pattern $\alpha \cup \beta$ with support = minimum support of nodes in b
4: end for
5: else

6: else for each a_i in the header of Tree do
7: generate pattern $\beta = a_i \cup \alpha$ with support = a_i.support
8: construct β's conditional pattern base and then β's conditional FP-tree Tree $Tree_\beta$
9: if $Tree_\beta \neq \emptyset$ then
10: call FP – Growth($Tree_\beta$, β)
11: end if
12: end for
13: end if

Experimental results

The experimental results were obtained by running both algorithms, on the same database. The database is synthetic, that is, it is generated by an external program.

The system used for testing is detailed below:
CPU: AMD Athlon XP 2200+ (1800 MHz)
RAM: 256 Mbytes
HDD: 7200 rpm, ATA 100, 8Mb Cache
OS: Windows XP Professional SP2
JVM: Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_02-b03)

The size of the databases used for testing:

Items	Transactions	Size (kb)
50	1000	44
	10000	412
	100000	4150
100	1000	50
	10000	424
	100000	4185

Minimum support: 0,5

Items	Transactrions	Apriori			FP-Growth	
		Time (ms)	Passes	Frequent items	Time (ms)	Passes
50	1000	203	2	8	172	2
	10000	1594	3	11	1047	2
	100000	15708	3	12	10172	2
100	1000	250	2	2	125	2
	10000	1109	2	2	1047	2
	100000	10344	2	4	10172	2

Minimum support: 0,2

Items	Transactions	Apriori			FP-Growth	
		Time (ms)	Passes	Frequent items	Time (ms)	Passes
50	1000	391	5	143	203	2
	10000	2875	5	188	1203	2
	100000	588891	10	1044	10656	2
100	1000	297	5	80	125	2
	10000	2704	5	79	1063	2
	100000	2719	5	79	1078	2

No. of transactions: 100.000

Min support	**Apriori**		**FP Growth**		**Frequent itemsets**
	Run time (ms)	**Passes**	**Runtime (ms)**	**Passes**	
0,1	62630	10	11422	2	1320
0,3	27016	5	10266	2	117
0,5	15708	3	10172	2	12
0,7	5390	1	9921	2	1
0,9	5281	1	4828	1	0

Association Rules:

Items	Transactions	Confidence	Time (ms)	Association rules	Frequent itemsets
50	1000	0,5	15	2	8
	10000	0,5	15	8	11
	100000	0,5	15	12	12
100	1000	0,5	16	0	2
	10000	0,5	16	0	2
	100000	0,5	16	0	4
50	1000	0,9	62	40	143
	10000	0,9	62	34	188
	100000	0,9	516	16237	1044
100	1000	0,9	15	38	80
	10000	0,9	15	48	79
	100000	0,9	0	48	79

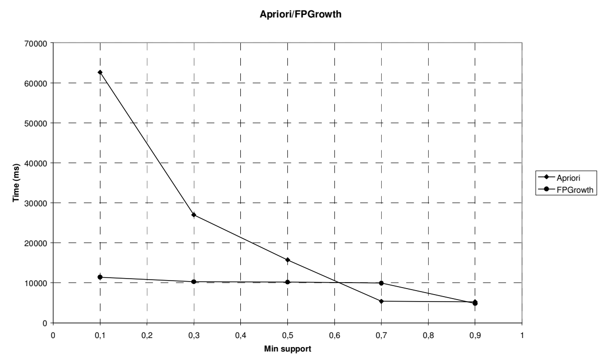

Figure 2 - Execution time as a function of minimum support

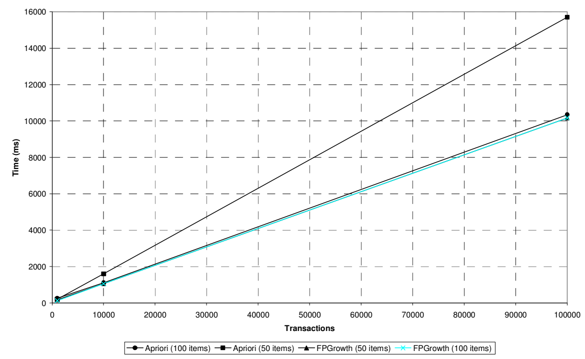

Figure 3 - Execution time vs. No. of transactions

Conclusion

Experimental data analysis shows the following:

Apriori:

- Poor results regarding the execution time are due to the fact that the algorithm requires repeated passes over the database; these are actually disk accesses.
- The number of passes over the database depends on the number of frequent items found until a certain point in execution. It can be easily seen that, if the number of database passes is smaller then the execution time is considerably reduced. The table below depicts the fact that, if one pass over the database is required, the Apriori algorithm performs better than the FPGrowth

No. of transactions : 100.000

Minimum support	**Apriori**		**FP Growth**		**Frequent itemsets**
	Run time (ms)	**Passes**	**Run time (ms)**	**Passes**	
0,1	62630	10	11422	2	1320
0,3	27016	5	10266	2	117
0,5	15708	3	10172	2	12
0,7	**5390**	**1**	**9921**	**2**	**1**
0,9	5281	1	4828	1	0

FP Growth

- The execution time is considerably smaller due to the fact that it requires only 2 passes over the database.
- The memory requirements of this algorithm are largr than in the case of Apriori, because the FP tree is built and kept in the main memory. In the case of the databases used in the example the amount of used memory could not be measured, reaching about 16 mbytes, but in case of large databases the memory requirements could go over 512 Mbytes.

References

1. Imiclinski T. Swami A. Agrawal, R. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD Conference Washington DC, USA, 1993.
2. Man Hon Wong Chan Man Kuok, Ada Fu. Mining fuzzy association rules in databases. SIGMOD Rec., 27(1):41-46, 1998.
3. Ng V.T. Fu A.W. Yongjian Fu Cheung D.W., Jiawei Han. A fast distributed algorithm for mining association rules. In In 4th International Conference on Parallel and Distributed Information Systems (PDIS '96), pages 31-43. IEEE Computer Society Technical Committee on Data Engineering, and ACM SIGMOD, 1996.
4. Hand Heikki, Mannila Padhraic, Smyth David. Principles of Data Mining. A Bradford Book The MIT Press Cambridge, 2001. Fondi di Ricerca Salvatore Ruggieri - Numero 558 d'inventario.
5. Attlila Gyenesei. Mining weighted association rules for fuzzy quantitative items. In Principles of Data Mining and Knowledge Discovery, pages 416-423, 2000.
6. Chi S.C. Wang S.L. Hong T.P., Kuo C.S. Mining fuzzy rules from quantitative data based on the apriotitid algorithm. In Proceedings of the 2000 ACM symposium on Applied computing, pages 534-536, 2000.
7. Philip S. Yu Jong Soo Park, Ming-Syan Chen. An effective hash-based algorithm for mining association rules. In Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, pages 175-186, San Jose, Canada, 1995.
8. J. C. Shafer R. Agrawal. Parallel mining of association rules. Ieee Trans. On Knowledge And Data Engineering, 8:962-969, 1996.
9. Ramakrishnan Srikant Rakesh Agrawal. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, Proc. 20th Int. Conf. Very Large Data Bases, VLDB, pages 487-499. Morgan Kaufmann, 12-15 1994.

10. Ramakrishnan Srikant Rakesh Agrawal. Mining quantitative association rules in large relational tables. In H. V. Jagadish and Inderpal Singh Mumick, editors, Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, pages 1-12, Montreal, Quebec, Canada, 4-6 1996.

11. J. D. Ullman S. Tsur S. Brin, R. Motwani. Dynamic itemset counting and implication rules for market basket data. In Proceedings ACM SIGMOD International Conference on Management of Data, pages 255-264, 1997.

12. Navathe S. Savasere A., Omiecinski E. An efficient algorithm for mining association rules in large databases. In Proc. of Intl. Conf. on Very Large Databases (VLDB), Zurich, Sept. 1995.