I.J. Information Technology and Computer Science, **2014, 07, 15-23**
Published Online June 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2014.07.03

Improved Apriori Algorithm for Mining Association Rules

Darshan M. Tank
Department of Information Technology, L.E.College, Morbi-363642, India
Email: dmtank@gmail.com

Abstract—Association rules are the main technique for data mining. Apriori algorithm is a classical algorithm of association rule mining. Lots of algorithms for mining association rules and their mutations are proposed on basis of Apriori algorithm, but traditional algorithms are not efficient. For the two bottlenecks of frequent itemsets mining: the large multitude of candidate 2-itemsets, the poor efficiency of counting their support. Proposed algorithm reduces one redundant pruning operations of C_2. If the number of frequent 1-itemsets is n, then the number of connected candidate 2-itemsets is C_n, while pruning operations C_n. The proposed algorithm decreases pruning operations of candidate 2-itemsets, thereby saving time and increasing efficiency. For the bottleneck: poor efficiency of counting support, proposed algorithm optimizes subset operation, through the transaction tag to speed up support calculations.
Algorithm Apriori is one of the oldest and most versatile algorithms of Frequent Pattern Mining (FPM). Its advantages and its moderate traverse of the search space pay off when mining very large databases. Proposed algorithm improves Apriori algorithm by the way of a decrease of pruning operations, which generates the candidate 2-itemsets by the apriori-gen operation. Besides, it adopts the tag-counting method to calculate support quickly. So the bottleneck is overcome.

Index Terms—association rule mining, frequent itemset generation, support and confidence

I. Introduction

A. Basic Concepts

Many business enterprises accumulate large quantities of data from their day to day operations. For example, huge amounts of customer purchase data are collected daily at the checkout counters of grocery stores. Table 1 illustrates an example of such data, commonly known as market basket transactions. Each row in this table corresponds to a transaction, which contains a unique identifier labeled TID and a set of items bought by a given customer. Retailers are interested in analyzing the data to learn about the purchasing behavior of their customers. Such valuable information can be used to support a variety of business-related applications such as marketing promotions, inventory management, and customer relationship management.

Association analysis is useful for discovering interesting relationships hidden in large data sets [1]. The uncovered relationships can be represented in the form of association rules or sets of frequent items. The following rule can be extracted from the data set shown in Table 1:

{Diapers} → {Beer}

Table 1. An example of market basket transactions

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

The rule suggests that a strong relationship exists between the sale of diapers and beer because many customers who buy diapers also buy beer. Retailers can use this type of rules to help them identify new opportunities for cross selling their products to the customers.

Besides market basket data, association analysis is also applicable to other application domains such as bioinformatics, medical diagnosis, Web mining, and scientific data analysis. In the analysis of Earth science data, for example, the association patterns may reveal interesting connections among the ocean, land, and atmospheric processes. Such information may help Earth scientists develop a better understanding of how the different elements of the Earth system interact with each other [2] [3].

B. Itemset and Support Count

Let I = {i1, i2. . . id} be the set of all items in a market basket data and T = {t1, t2. . . tN} be the set of all transactions. Each transaction ti contains a subset of items chosen from I. In association analysis, a collection of zero or more items is termed an itemset. If an itemset contains k items, it is called a k-itemset. The null set is an itemset that does not contain any items.

The transaction width is defined as the number of items present in a transaction. A transaction tj is said to contain an itemset X if X is a subset of tj. An important property of an itemset is its support count, which refers to the number of transactions that contain a particular itemset.

C. Association Rule

An association rule is an implication expression of the form X → Y , where X and Y are disjoint itemsets, i.e.,

Copyright © 2014 MECS

$X \cap Y = \emptyset$. The strength of an association rule can be measured in terms of its support and confidence. Support determines how often a rule is applicable to a given data set, while confidence determines how frequently items in Y appear in transactions that contain X. The formal definitions of these metrics are

$$\text{Support, } s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$\text{Confidence, } c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

D. Support and Confidence

Support is an important measure because a rule that has very low support may occur simply by chance. A low support rule is also likely to be uninteresting from a business perspective because it may not be profitable to promote items that customers seldom buy together. Support is often used to eliminate uninteresting rules. Support has a desirable property that can be exploited for the efficient discovery of association rules.

Confidence measures the reliability of the inference made by a rule. For a given rule $X \rightarrow Y$, the higher the confidence, the more likely it is for Y to be present in transactions that contain X. Confidence also provides an estimate of the conditional probability of Y given X.

E. Frequent Itemset Generation

A lattice structure can be used to enumerate the list of all possible itemsets. In general, a data set that contains k items can potentially generate up to $2^K - 1$ frequent itemsets, excluding the null set. Because k can be very large in many practical applications, the search space of itemsets that need to be explored is exponentially large.

A brute-force approach for finding frequent itemsets is to determine the support count for every candidate itemset in the lattice structure. To do this, we need to compare each candidate against every transaction, an operation that is shown in Fig. 1. If the candidate is contained in a transaction, its support count will be incremented. Such an approach can be very expensive because it requires O (NMw) comparisons, where N is the number of transactions, $M = 2^K - 1$ is the number of candidate itemsets, and w is the maximum transaction width.

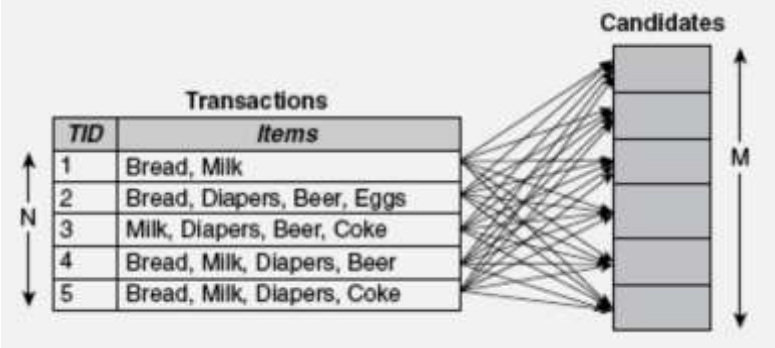

Fig 1. Counting the support of candidate itemsets

There are several ways to reduce the computational complexity of frequent itemset generation.

1) Reduce the number of candidate itemsets (M). The Apriori principle, described in the next section, is an effective way to eliminate some of the candidate itemsets without counting their support values.
2) Reduce the number of comparisons. Instead of matching each candidate itemset against every transaction, we can reduce the number of comparisons by using more advanced data structures, either to store the candidate itemsets or to compress the data set.

II. The Apriori Principle

Support measure helps to reduce the number of candidate itemsets explored during frequent itemset generation. The use of support for pruning candidate itemsets is guided by the Apriori Principle. **"**If an itemset is frequent, then all of its subsets must also be frequent." Considering the itemset lattice shown in Fig. 2, suppose {c, d, e} is a frequent itemset. Any transaction that contains {c, d, e} must also contain its subsets, {c, d}, {c, e}, {d, e}, {c}, {d}, and {e}. As a result, if {c, d, e} is frequent, then all subsets of {c, d, e} must also be frequent [6].

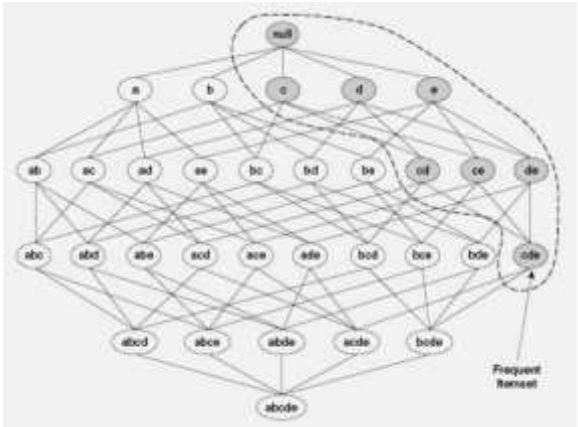

Fig. 2. An illustration of the Apriori principle

Conversely, if an itemset such as {a, b} is infrequent, then all of its supersets must be infrequent too. As

illustrated in Fig. 3, the entire sub graph containing the supersets of {a, b} can be pruned immediately once {a, b} is found to be infrequent. This strategy of trimming the exponential search space based on the support measure is known as support-based pruning. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support for its subsets. This property is also known as the anti-monotone property of the support measure.

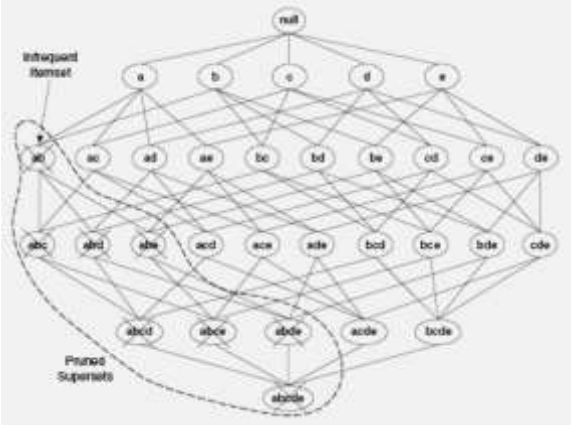

Fig. 3. An illustration of support-based pruning

Any measure that possesses an anti-monotone property can be incorporated directly into the mining algorithm to effectively prune the exponential search space of candidate itemsets.

A. Frequent Itemset Generation in the Apriori Algorithm

Apriori is the first association rule mining algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets. Fig. 4 provides a high-level illustration of the frequent itemset generation part of the Apriori algorithm for the transactions shown in Table 1. Considering the support threshold is 60%, which is equivalent to a minimum support count equal to 3.

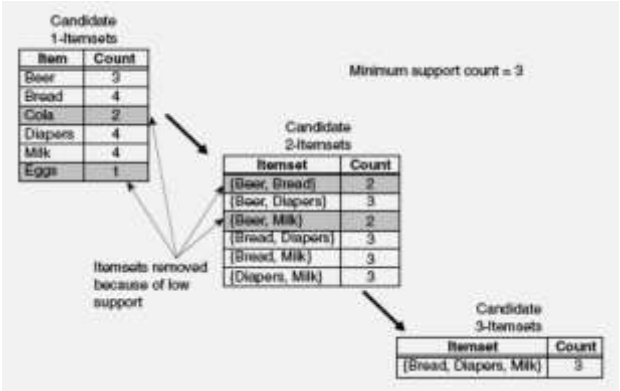

Fig. 4. Illustration of frequent itemset generation using the Apriori algorithm

Initially, every item is considered as a candidate 1-itemset. After counting their supports, the candidate itemsets {Cola} and {Eggs} are discarded because they appear in fewer than three transactions. In the next iteration, candidate 2-itemsets are generated using only the frequent 1-itemsets because the Apriori principle ensures that all supersets of the infrequent 1-itemsets must be infrequent. Because there are only four frequent 1-itemsets, the number of candidate 2-itemsets generated by the algorithm is $\binom{4}{2}$ = 6. Two of these six candidates, {Beer, Bread} and {Beer, Milk}, are subsequently found to be infrequent after computing their support values. The remaining four candidates are frequent, and thus will be used to generate candidate 3-itemsets. Without support-based pruning, there are $\binom{6}{3}$ = 20 candidate 3-itemsets that can be formed using the six items. With the Apriori principle, we only need to keep candidate 3-itemsets whose subsets are frequent. The only candidate that has this property is {Bread, Diapers, Milk}.

The effectiveness of the Apriori pruning strategy can be shown by counting the number of candidate itemsets generated. A brute-force strategy of enumerating all itemsets (up to size 3) as candidates will produce

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$$

candidates. With the Apriori principle, this number decreases to

$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$$

candidates, which represents a 68% reduction in the number of candidate itemsets. The pseudo code for the frequent itemset generation part of the Apriori algorithm is shown below.

Apriori algorithm for frequent itemset generation

1. $k = 1$
2. $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times \min\sup \}$

{Find all frequent 1-itemsets}

3. **repeat**
4. $k = k + 1$
5. C_k = *apriori-gen* (F_{k-1}) *{Generate candidate itemsets}*
6. **for** *each transaction* $t \in T$ *do*
7. C_t = *subset* (C_k, t)

{Identify all candidates that belong to t}

8. **for** *each candidate itemset* $c \in C_t$ *do*
9. $\sigma(c) = \sigma(c) + 1$

{Increment support count}

10. **end for**
11. **end for**

12. $F_k = \{c \mid c \in C_k \wedge \sigma(c) \geq N \times \min\sup\}$
{Extract the frequent k-itemsets}
13. ***until*** $F_k = \varnothing$
14. ***Result*** = $\bigcup_k F_k$

Procedure apriori_gen (F_{k-1}***)***

1. ***for*** *each itemsets* $f_1 \in F_{k-1}$ *do*
2. ***for*** *each itemsets* $f_2 \in F_{k-1}$ *do*
3. ***if*** ($f_1[1] = f_2[1]$) $\wedge$ ($f_1[2] = f_2[2]$) $\wedge$ ... $\wedge$ ($f_1[k-2] = f_2[k-2]$) $\wedge$ ($f_1[k-1] = f_2[k-1]$) *then*
4. $c = f_1 \otimes f_2$
{join step: generate candidates}
5. ***for*** *each* $(k-1)$*-subsets s of c do*
6. *if* ($s \notin f_{k-1}$) *then*
7. *delete c*
{prune step: remove candidates}
8. *else*
9. *add c to* C_k
10. ***end for***
11. ***return*** C_k
12. ***end if***
13. ***end for***
14. ***end for***

Procedure subset (C_k **, t)**

1. ***for*** *all candidates* $s \in C_k$ *do*
2. *if t contains s*
3. *subset = subset +* $\{s\}$
4. ***end for***

C_k denote the set of candidate k-itemsets and F_k denote the set of frequent k-itemsets:

- The algorithm initially makes a single pass over the data set to determine the support of each item. Upon completion of this step, the set of all frequent 1-itemsets F_1 will be known (steps 1 and 2).
- Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent (k − 1)-itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called apriorigen.
- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6–10). The subset function is used to determine all the candidate itemsets in C_k that are contained in each transaction t.
- After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than minsup (step 12).
- The algorithm terminates when there are no new frequent itemsets generated, i.e., $F_K = \Phi$ (step 13).

The frequent itemset generation part of the Apriori algorithm has two important characteristics.

1) First, it is a level-wise algorithm; i.e., it traverses the itemset lattice one level at a time, from frequent 1-itemsets to the maximum size of frequent itemsets.
2) Second, it employs a generate-and-test strategy for finding frequent itemsets. At each iteration, new candidate itemsets are generated from the frequent itemsets found in the previous iteration. The support for each candidate is then counted and tested against the minsup threshold. The total number of iterations needed by the algorithm is $K_{\max}+1$, where $K_{\max}$ is the maximum size of the frequent itemsets.

B. Candidate Generation and Pruning

The apriori-gen function shown in Step 5 of algorithm generates candidate itemsets by performing the following two operations:

1) **Candidate Generation:** This operation generates new candidate k-itemsets based on the frequent (k − 1) -itemsets found in the previous iteration.
2) **Candidate Pruning:** This operation eliminates some of the candidate k-itemsets using the support-based pruning strategy.

Considering a candidate k-itemset, X = { $i_1, i_2, \ldots\ldots, i_k$ }, the algorithm must determine whether all of its proper subsets, X − $\{i_j\}$ ($\forall_j$ = 1, 2, . . . ,k), are frequent. If one of them is infrequent, then X is immediately pruned. This approach can effectively reduce the number of candidate itemsets considered during support counting. The complexity of this operation is $O(k)$ for each candidate k-itemset [6] [7].

In principle, there are many ways to generate candidate itemsets. The following is a list of requirements for an effective candidate generation procedure:

1) It should avoid generating too many unnecessary candidates. A candidate itemset is unnecessary if at least one of its subsets is infrequent. Such a candidate is guaranteed to be infrequent according to the antimonotone property of support.
2) It must ensure that the candidate set is complete, i.e., no frequent itemsets are left out by the candidate generation procedure. To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets.
3) It should not generate the same candidate itemset more than once. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

Copyright © 2014 MECS

III. Proposed Algorithm For Mining Association Rules

A. Enhancement of Subset Procedure

Procedure subset (C_k, t)

1. *for all candidates* $s \in C_k$ *do*
2. *if t contains s*
3. *subset = subset +* $\{s\}$
4. *end for*

Improved Procedure subset (C_k, t)

1. *for all candidates* $s \in C_k$ *do*
2. *If (first item of* $s \geq \min$ *)* $\bigcap$ *(last item of* $s \leq \max$ *)*
3. *subset = subset +* $\{s\}$
4. *end for*

Its running time is $O(|D|^k)$, this is a very large number. Considering k = 3, the transaction t (x2, x6, x7, x9), the itemsets s (x4, x6, x10). Refinement of this sentence: "if t contain s during a judge are the following conditions: if (t contains x4) $\bigcap$ (t contains x6) $\bigcap$ (t contains x10) then….." Obviously, t does not contain s, but still have to compare one by one [8].

The proposed algorithm improves the subset operation through adding a tag column on the transaction database, of which values are the first and the last numbers of itemsets. If (first item of x > = min) $\bigcap$ (last item of x <= max), then t contains s. Otherwise t does not contain s.

B. Proposed Apriori Algorithm

The Proposed algorithm can be divided into two steps. First, the algorithm finds out all frequent itemsets. Then it generates all association rules from frequent itemsets.

Input: A database D and min-support
Output: All the frequent itemsets of the database

1. $k = 1$
2. $L_k = \{i \mid i \in I \wedge \sigma(\{i\}) \geq N \times \min\sup\}$

{Find all frequent 1-itemsets}

3. $C_2 = L_1 \otimes L_1$
4. ***for*** *all candidates* $c \in C_2$ *do*
5. C_t *= subset* (C_2, t)
6. ***for*** *all candidates* $c \in C_t$ *do*
7. $\sigma(c) = \sigma(c) + 1$
8. ***end for***
9. $L_2 = \{c \mid c \in C_k \wedge \sigma(c) \geq N \times \min\sup\}$
10. ***end for***
11. ***repeat***
12. $k = 3$
13. C_k = *apriori-gen* (L_{k-1})

{Generate candidate itemsets}

14. ***for*** *each transaction* $t \in T$ *do*
15. C_t *= subset* (C_k, t)

{Identify all candidates that belong to t}

16. ***for*** *each candidate itemset* $c \in C_t$ *do*
17. $\sigma(c) = \sigma(c) + 1$

{Increment support count}

18. ***end for***
19. ***end for***
20. $L_k = \{c \mid c \in C_k \wedge \sigma(c) \geq N \times \min\sup\}$

{Extract the frequent k-itemsets}

21. ***until*** $L_k = \varnothing$
22. ***Result*** = $\bigcup_k L_k$

As C_2 is connected by L_1 and pruning two-generated, of which pruning operations to test whether a subset of C_2 is frequent itemsets. Because $C_2 = L_1 \otimes L_1$, the subset of the set must be frequent. Therefore, the proposed algorithm generates C_2 by the direct connection of L_1, rather than by pruning operation [14] [17].

IV. Working Of Proposed Algorithm

A transactional database is showed in Table 2, and the minimum support threshold is 20%.The result of mining association rules which is generated by proposed algorithm is illustrated as follows.

Table 2. Transactional Database D

Transaction No.	Itemsets
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 3. The Database in the proposed algorithm

Transaction No.	Itemsets	Tag
T100	I1, I2, I5	1, 5
T200	I2, I4	2, 4
T300	I2, I3	2, 3
T400	I1, I2, I4	1, 2, 4
T500	I1, I3	1, 3
T600	I2, I3	2, 3
T700	I1, I3	1, 3
T800	I1, I2, I3, I5	1, 2, 3, 5
T900	I1, I2, I3	1, 2, 3

Copyright © 2014 MECS

The first to deal with a database in the form of Table 3, the last one marked with the corresponding transaction of the smallest item serial number and the largest serial number. After the first scan the database, looking for L_1, you can compare the various tag of the transaction, because L_1 is not in [2, 4], or in [2, 3], and so on to avoid many comparison of the operation and thereby saving time.

After calculating support, L_1 = {I1, I2, I3, I4, I5}. According to the improved algorithm, C_2 connected directly by L_1 is no need for pruning operations. Thus, C_2 = {(I1, I2), (I1, I3), (I1, I4), (I1, I5), (I2, I3), (I2, I4), (I2, I5), (I3, I4), (I3, I5), (I4, I5)}. As (I1, I2) in [1, 5], it includes the transaction T100 and increases the corresponding support. And so on, L_2 = {(I1, I2), (I1, I3), (I1, I5), (I2, I3), (I2, I4), (I2, I5)}. After pruning after connecting C_3 = {(I1, I2, I3), (I1, I2, I5)}. Itemsets (I1, I2, I3) in [1, 5], and so increases in support. Because it is not in [2, 4] or [2, 3] and therefore do not increase support. Finally, L_3 = {(I1, I2, I3), (I1, I2, I5)}. C_4 can not be find as the algorithm end. Clearly, the proposed algorithm reduces the number of unnecessary operations, streamline the collection of frequent generation and improve the efficiency of the algorithm.

V. IMPLEMENTATION

A. *Class File Detail*

1) Apriori.cs

It contains class and methods used to implement the APriori Market Based Analysis Data Mining Algorithm. Apriori class implements the APriori algorithm for market based analysis. This class implements the APriori algorithm and creates association between items in a transaction.

2) DataAccessLayer.cs

Implements data access to relational databases and XML data stores.

3) DataMining.cs

A class that provides data mining services using C#.NET, ADO.NET, XML.NET and a Market Based Analysis Data Mining Algorithm.

4) Itemsets.cs

An abstract class that represents a set items from a shopping cart. Defines the Level of an Itemset, its Items and Support Count. The level of an Itemset is the Hierarchy of an itemset or how many items make up the Itemset. A one level Itemset contains one item. A two level Itemset contains two items.

5) DataTransformationServices.cs

DataTransformationServices is used to transform data contained in tables and relationships like the Northwind database to a single table containing a TransactionID and Transactions field which is used for the C#.NET market based sales data mining.

B. *Screen Shots*

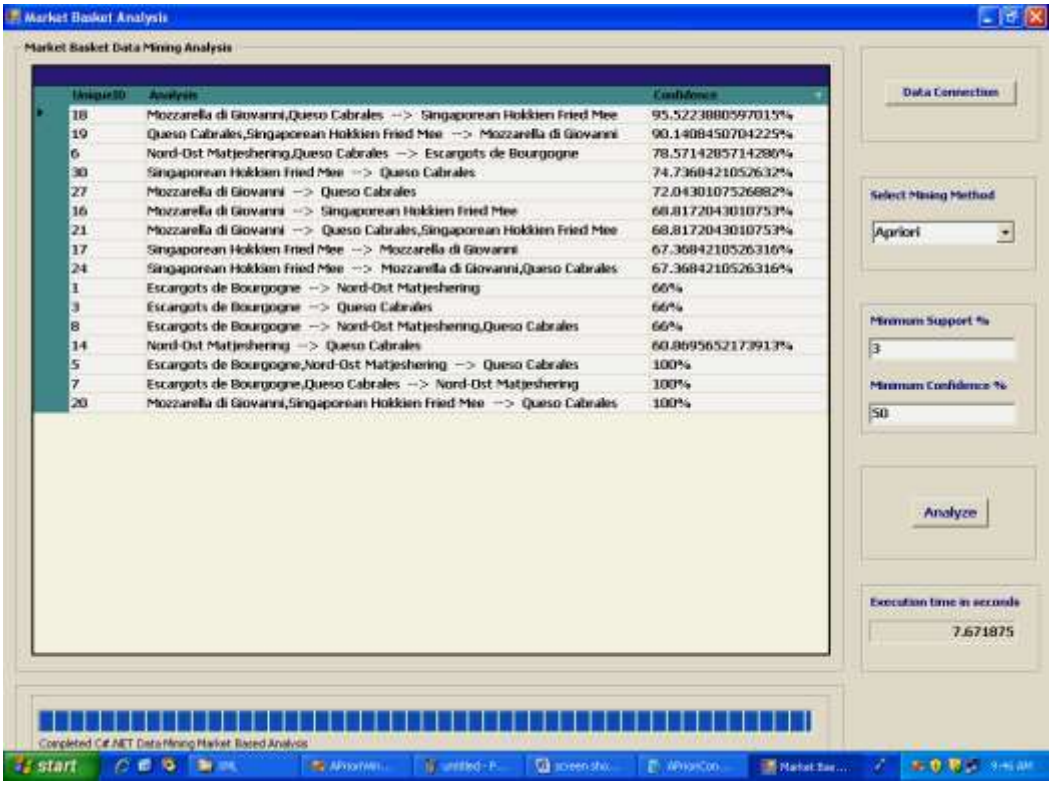

Fig. 5. Output of application using Apriori algorithm

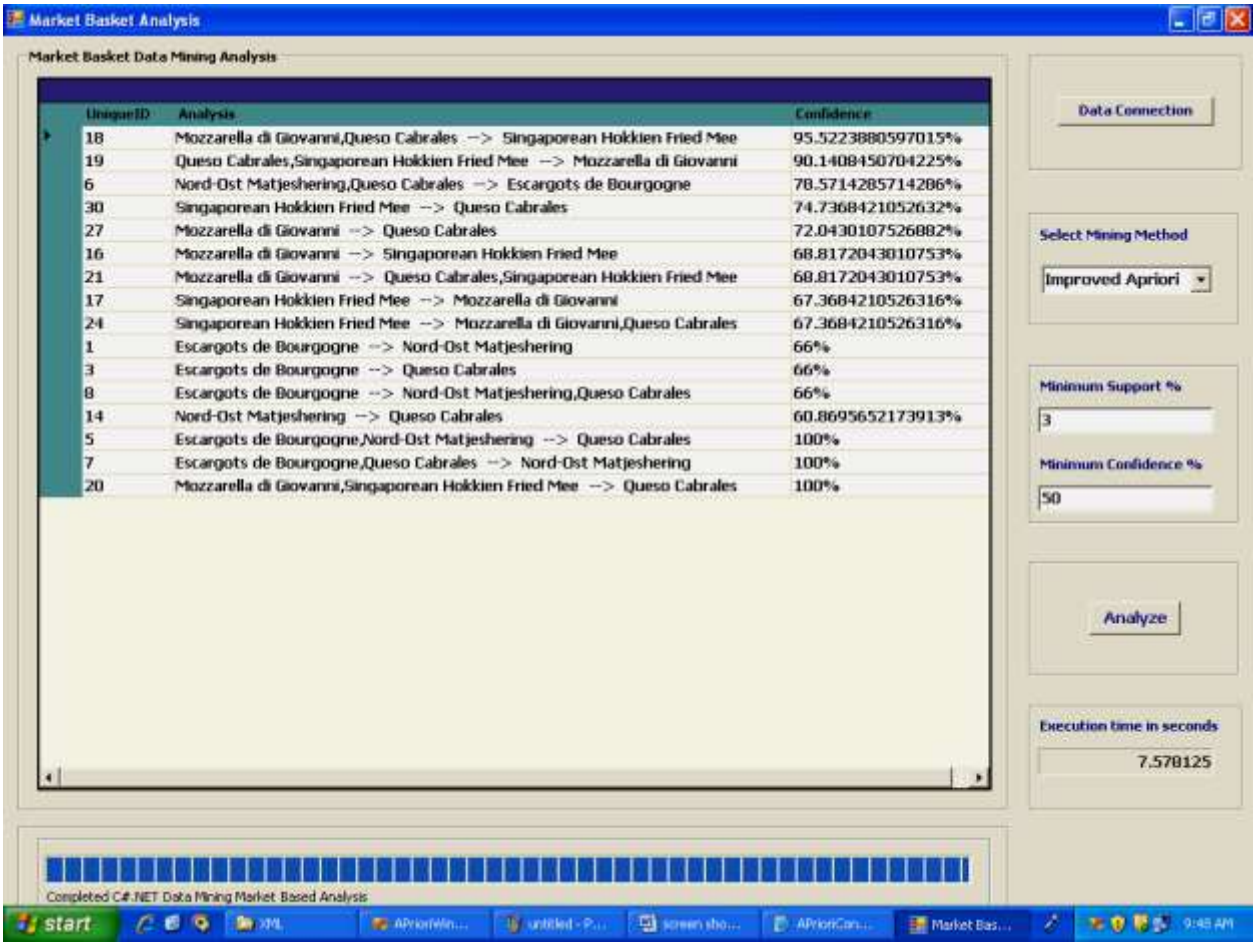

Fig. 6. Output of application using Improved Apriori algorithm

VI. Performance Study

In order to study the performance, the computer used to run the experiments had Intel(R) Pentium D CPU 2.80 GHz processor and 512 MB of memory and 512 MB of memory. The operating system used was Microsoft Windows XP Professional. Programs are coded in C#.NET on the platform of Visual Studio 2005. The performance of the Apriori algorithm is largely depends on the internal characteristic of the datasets and the number of records. To make the time measurements more reliable, no other application was running on the machine while the experiments were running.

Algorithm has been tested on the three different datasets described in the previous chapter. The performance measure is the execution time (seconds) of the algorithms on the datasets. The minimum confidence is set to 50%.

Table 4. Dataset characteristics

Datasets	Transactions	Distinct Items	Maximum Transaction Size	Average Transaction Size
Retail Sales	5000	200	8	3.0
Adventure Works	15000	500	12	5.0

A. *Experimental Analysis*

(1) Retail Sales Dataset

Table 5. The comparison of the runtime at different minimum support between the Apriori algorithm and the Improved Apriori algorithm

Minimum Support (%)	Execution Time (Seconds)		Rule Count
	Apriori	Improved Apriori	
1	8.42	8.29	23
2	8.21	8.14	23
3	8.21	8.15	23
4	7.92	7.71	17
5	7.65	7.65	16
6	7.90	7.65	16
7	7.75	7.65	16
8	7.35	7.21	10
9	7.14	6.81	3
10	6.76	6.68	0

Copyright © 2014 MECS

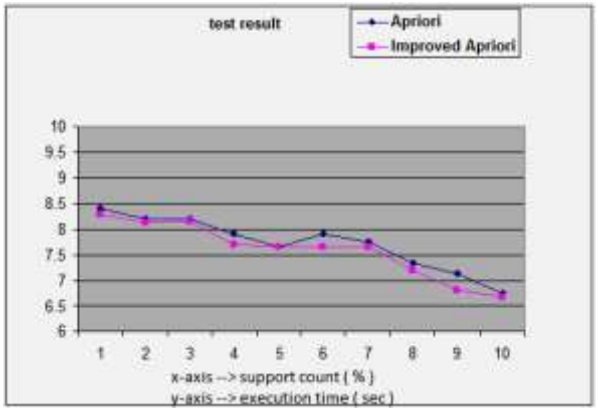

Fig. 7. The comparison of runtime at different minimum supports between the original Apriori and the improved Apriori

(2) Adventure Works Dataset

Table 6. The comparison of the runtime at different minimum support between the Apriori algorithm and the Improved Apriori algorithm

Minimum Support (%)	Execution Time (Seconds)		Rule Count
	Apriori	Improved Apriori	
10	534.67	533.54	1024
15	242.78	241.92	817
20	87.73	87.41	321
25	56.14	55.94	50
30	53.15	52.97	9
35	52.48	52.34	2
40	53.45	52.23	0
45	53.09	53.00	0
50	52.35	52.25	0

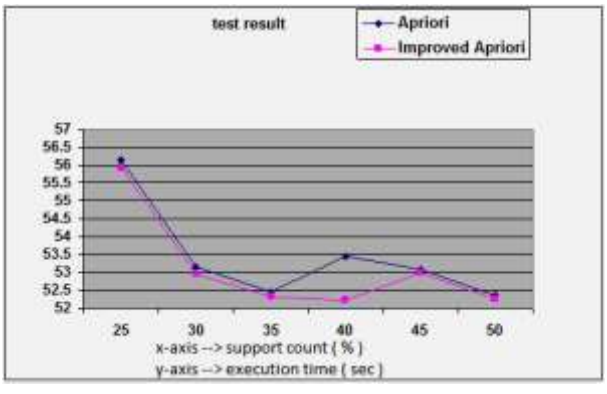

Fig. 8. The comparison of runtime at different minimum supports between the original Apriori and the improved Apriori

Fig. 7 and Fig. 8 show the execution times for the two datasets for increasing values of minimum support. As the minimum support increases, the execution time of the algorithm decreases because of decreases in the total number of candidates and large itemsets. From the above result, we can contrast the changes in efficiency. With the decrease of minimum support to some content, the Improved Apriori algorithm tends to prevail, and with the further decrease of minimum support, the advantage of the Improved Apriori algorithm can be greatly enhanced.

From the above figures, we can conclude that, with the reduction of the minimum support, the efficiency of the Improved Apriori algorithm can change correspondingly from a lower level to a higher level than that of the classic Apriori algorithm. When the value of minimum support count is small enough, the efficiency of the modified algorithm is improved. The total number of records and the characteristic of data will affect the performance of the algorithm.

VII. Conclusion And Future Work

The proposed algorithm for mining association rule, decreases pruning operations of candidate 2-itemsets, thereby saving time and increase efficiency. It optimizes subset operation, through the transaction tag to speed up support calculations. The experimental results obtained from tests show that proposed system outperforms original one efficiently.

The current mining methods require users to define one or more parameters before their execution; however, most of them do not mention how users can adjust these parameters online while they are running. It is not feasible for users to wait until a mining algorithm to stop before they can reset the parameters. This is because it may take a long time for the algorithm to finish due to the continuous arrival and huge amount of data. For further improvement, we may consider either let the users adjust online or let the mining algorithm auto-adjust most of the key parameters in association rule mining, such as support, confidence and error rate.

References

[1] Mining Association Rules between Sets of Items in Large Databases by R. C. Agarwal, Imielienski T., and Swami A.
[2] Efficiently Mining Long Patterns from Databases by R. Bayardo. In Proc. of 2006 ACM-SIGMOD Intl. Conf. on Management of Data
[3] Fast Discovery of Association Rules. By Agrawal, A.,Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A.
[4] A New Improvement on Apriori Algorithm by Lei Ji, Baowen Zhang, Jianhua Li. – June 2008
[5] The analysis and improvement of Apriori algorithm by HAN Feng, ZHANG Shu-mao, DU Ying-shuang
[6] The Research of Improved Apriori Algorithm for Mining Association Rules by Fangyi Wang, Erkang Wang, Bowen Chen
[7] The Optimization and Improvement of the Apriori Algorithm by Yiwu Xie, Yutong Li, Chunli Wang, Mingyu Lu (International Symposium on Intelligent Information Technology Application Workshops)
[8] An Efficient Frequent Patterns Mining Algorithm based on Apriori Algorithm and the FP-tree Structure by Bo Wu, Defu Zhang, Qihua Lan, Jiemin Zheng (Third 2008 International Conference on Convergence and Hybrid Information Technology)

Copyright © 2014 MECS

[9] UCI Repository of Machine Learning Databases by Blake, C.L. and Merz, C.J. (Dept. of Information and Computer Science, University of California at Irvine www.ics.uci.edu/mlearn/MLRepository.html)
[10] Synthetic Data Generation Code for Associations and Sequential Patterns. http://www.almaden.ibm.com/software/quest/Resources/index.shtml Intelligent Information Systems, IBM Almaden Research Center
[11] R.Agrawal and R.Srikant. Mining sequential patterns. In P.S.Yu and A.L.P.Chen, editors, Proc.11th Int.Conf. Data Engineering, ICDE, pages3–14.IEEE Press,6–10.1995.
[12] H.Mannila,H.Toivonen,and A.I.Verkamo. Discovering frequent episodes in sequences. In Proceedings of the First International Conference on Knowledge Discovery and Data Mining, pages210–215.AAAI Press, 1995.
[13] K.Hatonen, M.Klemettinen, H. Mannila, P.Ronkainen, and H.Toivonen. Knowledge discovery from telecommunication network alarm databases. In S.Y.W.Su, editor, Proceedings of the twelfth International Conference on Data Engineering, February 26–March 1,1996,New Orleans, Louisiana ,pages115–122,1109 Spring Street, Suite 300,Silver Spring, MD20910, USA,1996.IEEE Computer Society Press.
[14] A. Inokuchi, T.Washio and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, pages13–23. Springer-Verlag, 2000.
[15] M.Kuramochi and G.Karypis. Frequent sub graph discovery. In Proceedings of the first IEEE International Conference on Data Mining, pages313–320, 2001.
[16] Park J S, Chen Ming-Syan, Yu Philip S. Using a hash-based method with transaction trimming for mining association rules[J].IEEE Transactions on Knowledge and Data Engineering,1997,9(5):487-499.
[17] Savasere A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in large databases. Proceedings of the 21st International Conference on Very large Database, 1995.
[18] Brin S, Motwani R, Ullman J D,et al. Dynamic Itemset counting and implication rules for market basket data. ACM SIGMOD International Conference on the Management of Data, 1997.
[19] Luo Ke,Wu Jie. Apriori algorithm based on the improved. Computer Engineering and application, 2001, 20:20-22.
[20] Li Xiaohong,Shang Jin. An improvement of the new Apriori algorithm [J].Computer science, 2007,34 (4) :196-198.
[21] Gu Qing-feng, SONG Shun-Lin. The improvement of Apriori algorithm and in SQL applications. Computer engineering and design 2007,28(13):3060-3233.
[22] Luo Jiawei,Wang Yan. Apriori algorithm with a fully connected the improvement [J]. Computer applications, 2006. 26 (5):1174-1177.

Author's Profiles

Darshan M. Tank: Lecturer of Information Technology at L E College, Morbi, Gujarat, India. My areas of interest include Business Intelligence, Data and Knowledge Mining, Real-Time Data Warehouse, Decision Support System and Information Retrieval.

How to cite this paper: Darshan M. Tank,"Improved Apriori Algorithm for Mining Association Rules", International Journal of Information Technology and Computer Science(IJITCS), vol.6, no.7, pp.15-23, 2014. DOI: 10.5815/ijitcs.2014.07.03