Mining Long High Utility Itemsets in Transaction Databases

GUANGZHU YU[1] , SHIHUANG SHAO[1], DAOQING SUN[1] and BIN LUO[2]
[1] Information and Technology College, DongHua University, ShangHai, CHINA
[2] Automation College, Guangdong University of Technology, Guangzhou, CHINA

Abstract: Although support has been used as a fundamental measure to determine the statistical importance of an itemset, it can't express other richer information such as quantity sold, unit profit, or other numerical attributes. To overcome the shortcoming, utility is used to measure the semantic importance and several algorithms for utility mining have been proposed. However, existing algorithms for utility mining adopt an Apriori-like candidate set generation-and-test approach，and are inadequate on databases with long patterns. To solve the problem, this paper proposes a hybrid model and a novel algorithm, i.e., inter-transaction, to discover high utility itemsets from two directions: existing algorithms such as UMining [1] seeks short high utility itemsets from bottom, while inter-transaction seeks long high utility itemsets from top. To avoid the costly process of extending short itemsets step by step, inter-transaction find long itemsets directly by intersecting relevant transactions. Experiments on synthetic data show that the new algorithm achieves high performance, especially in high dimension data set.

Key-Words: utility; long high utility itemset; intersection transaction; partition; hybrid model

1 Introduction

Traditional association rule mining (ARM) aims to find all itemsets that have support above a user defined threshold. It treat all the items equally by assuming that the utility of each item is always 1 (item is present) or 0 (item is absent). Under this kind of model, a wholesale of an itemset is treated in the same way as a small sale: corresponding support is added by one. Obviously, it's unrealistic and will lead to some useful pattern missed. For example, in a transaction database, there are 1000 sale records of milk which occupy 10% of the total transaction number, contributing 1% of the total profit. On the other hand, there are 600 sale records of birthday cake that occupy 6% of the total number, contributing 5% of the total profit. If the support threshold is 8%, according to traditional mining algorithm for finding frequent itemsets, milk will be reported as a frequent item and birthday cake will be ignored. But in fact, the market professional must be more interesting to birthday cake because it contributes a larger portion to total profit than milk. The example shows that support is not sufficient to reflect user's interestingness.

According to Expectancy Theory [2], we have the well-known equation "motivation=probability * utility", which says motivation is determined by the utility of making a decision and the probability of success. In many fields such as retailing, web log technique, users are not only interested in the frequency of occurrence of an itemset (support), but also their utility. So a decision-oriented ARM algorithm should output both the support and the utility of all interesting patterns. For this reason, utility based ARM has been proposed to discover all itemsets in a database with utility values higher than a user specified threshold.

Table 1 is an example of a simplified transaction database where the total utility value is 162. The number in each transaction in table 1 is the sales profit of each item. if s(X) and u(X) represent the support and utility of itemset X respectively, then u(A,B)=43, s(A,B)=5, u(A,B,C)=54, s(A,B,C)=3, u(A,B,C,D)= 45, s(A,B,C,D)=2, u(A,B,C,D,E)=57, s(A,B,C,D,E)=2.

	A	B	C	D	E
T1	0	0	5	0	1
T2	2	3	0	0	0
T3	3	5	15	7	4
T4	0	0	4	7	2
T5	4	5	8	0	0
T6	9	4	0	0	2
T7	6	0	8	3	6
T8	0	0	0	6	3
T9	3	0	0	9	5
T10	3	5	6	1	8

Table1. A transaction database

If support threshold is 0.3 and utility threshold is 50, {A,B} is a frequent but not a high utility itemset, {A,B,C} is both a frequent and high utility itemset, {A,B,C,D} is neither a frequent nor high utility itemset and {A,B,C,D,E} is a high utility but non-frequent itemset.

From above example, we can draw a conclusion: downward closure property, which states if an itemset is frequent by support, then all its nonempty subsets must also be frequent by support, doesn't apply to utility mining. Relevant studies have shown that utility constraint is neither anti-monotone nor monotone nor succinct nor convertible [3] [4]. Because of this property, most algorithms for frequent pattern mining can't be used to find high utility itemsets.

Furthermore, all existing algorithms for utility mining are Apriori-like algorithms. They employ a bottom-up, breadth-first search, iteratively generate candidate (k+1)-itemsets from k-itemsets and are inadequate on datasets with long patterns. To the best of our knowledge, there is no efficient algorithm for mining long high utility itemsets by far. To solve the problem, we propose a hybrid top-down/bottom-up search model and a partitioning-based algorithm, i.e. inter-transaction. Under the hybrid model, existing algorithm such as two-phase searches the short high utility itemsets in a bottom-up manner, while inter-transaction searches long high utility itemsets in a top-down manner, they complement each other.

Inter-transaction is based on the fact that long transactions usually have few common items, which means the intersection of multiple long transactions is usually very short. Since existing algorithms are efficient for short itemsets, we emphasize on introducing the inter-transaction.

The remainder of the paper is organized as follows: section 2 overviews related work, section 3 formally defines relevant terms and notations; section 4 introduces the new algorithm. In section 5, experimental results are presented and in section 6, we summarize our work.

2 related work

Lots of researches have been conducted to improve the usefulness of traditional ARM, but most of them are utility-related, not utility-based. Value added association rules [5] [6] extends traditional association rules by taking into consideration semantics of data. The difference between [5] and [6] is that price and quantity of supermarket sales are considered in the former, while the later try to attach a value to every item in the database and use the added values to rank association rule. Quantitative association rules mining [7] [8] introduce statistical inference theory into data mining field to find extraordinary and therefore interesting phenomena in database.

Weighted association rules gives up treating all the items and all the transactions uniformly by assigning different weights to items [9] or transactions [10]. These weights essentially reflect users' preferences. [10] also proposed a mixed weighted association rules model, which incorporate both vertical weighted association rules and horizontal weighted association rules.

Shen Y. D. proposed an objective-oriented apriori (OOApriori) model [3]. He puts utility constraint into apriori algorithm so that some frequent high utility itemsets could be found. Chan R. et al. also proposed a utility mining algorithm to mine top-k frequent high utility closed patterns [11]. To reduce search space, he developed a new pruning strategy based on a weaker but anti-monotonic condition to prune low utility itemsets.

Barber B. uses itemset share as a measure to overcome the shortcoming of support [12]. Item share is defined as a fraction of some numerical values. It can reflect the impact of the sales quantities of items on the cost or profit of an itemset, it should be regarded as a utility.

A formal definition of utility mining and theoretical model were proposed by Yao H [1] [13]. In his UMining algorithm, utility upper bound property is used to reduce the size of candidate set. In order to narrow search space furthermore, support upper bound property is used in a heuristics model to predict whether an itemset should be added into the candidate set. Unfortunately, the heuristics mode can't guarantee an accurate prediction. Yao H also summarized the mathematic properties of utility constraint in [4]. Liu Y. proposed a two-phase algorithm to mine high utility itemsets [14]. In Liu's model, transaction weighted downward closure property is used to reduce search space.

3 definitions

Utility of an itemset is a subjective term dependent on user and application; it could be measured in terms of profit, cost, risk, aesthetic value or other expressions of user preference. For easy understand, we refer to utility of an itemset as the economic utility such as sales profit.

Let $I=\{i_1, i_2, \ldots, i_m\}$ be a set of items, $D=\{T_1, T_2, \ldots, T_n\}$ be a transaction database. Each transaction T_q in database D ($T_q \in D$) is a subset of I, i.e., $T_q \subseteq I$. To simplify notation, we sometimes write a set $\{ i_1, i_2, \ldots, i_k \}$ as $i_1 i_2 \ldots i_k$. Adapting from the notations described in [1], [14] and [15], we have following definitions:

Definition1. The transaction utility of item x in transaction T_q, denoted $u(x, T_q)$, is the utility brought on by item x when transaction T_q occur. Take example for table 1, u(A,1)=0, u(A,2)=2.

Definition 2. The transaction utility of itemset X in transaction T_q, denoted $u(X, T_q)$, is the sum of transaction utility of item x contained in X, i.e.,

$$u(X, T_q) = \sum_{x \in X \wedge X \subseteq T_q} u(x, T_q) \quad (1)$$

For example, in table 1, u(AB,2)= u(A,2)+ u(B,2) =5, u(ABC,5)= u(A,5)+u(B,5)+u(C,5) =4+5+8=17.

Definition 3. The partition utility of itemset X in partition P_i, denoted $u(X, P_i)$, is the sum of the transaction utility of itemset X in partition P_i, i.e.,

$$u(X, P_i) = \sum_{T_q \in P_i \wedge X \subseteq T_q} u(X, T_q) \quad (2).$$

For more details about partitions, refer to [15].

Definition 4. The utility of X in database, denoted $u(X)$, is the sum of transaction utility of itemset X in database, i.e.,

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in P_i \wedge P_i \subseteq D} u(X, P_i)$$
$$= \sum_{T_q \in D \wedge X \subseteq T_q} u(X, T_q) \quad (3)$$

Examples can be seen in section 1.

Definition 5. The utility of transaction T_q, denoted $u(T_q)$, is the sum of transaction utility of item x in transaction T_q, i.e.,

$$u(T_q) = \sum_{x \in T_q} u(x, T_q) \quad (4)$$

Definition 6. Transaction identifier list, denoted tidlist, is a set of transaction ID.

Definition 7. Intersection transaction, denoted T(tidlist), is an itemset obtained by the intersection of transactions listed in tidlist. For example, let T_1=ABDF, T_2=ADFG, then intersection transaction T(1,2)=$T_1 \cap T_2$=ADF, corresponding tidlist is {1,2}. If |tidlist|=k, T(tidlist) is also called k-intersection transaction, k is called the current support of k-intersection transaction. Pay attention to the difference between current support k and support s. the support of an itemset is the maximal number of transactions containing the itemset, while current support is the number of parts of the transactions.

Definition 8. A long transaction is the transaction that includes more than minlen items. minlen is a user defined value. Otherwise, called short transaction.

Definition 9. A high utility itemset is the itemset with utility value higher than a user specified threshold, i.e., minutil. Otherwise, we say the itemset is low.

Definition 10. A long high utility itemset is the high utility itemset with length longer than minlen.

Definition 11. A locally high utility itemset is an itemset in partition p_i with partition utility value higher than the local utility threshold minutil/n. n is the partition number.

4 Inter-transaction algorithm

As we know, each itemset is determined either by a transaction or by a group of transactions. If we let any two transactions intersect each other, we can obtain all itemsets (2-intersection transactions) with support higher than 2. Similarly, we can obtain all itemsets (k-intersection transactions) with support higher than k by intersecting any k transactions ($1 \le k \le N$, N is the number of transactions). Theoretically, we can obtain all itemsets by intersecting relevant transactions.

In real database, transaction number N can easily reach to several millions and there will be 2^N intersection transactions at the worst situation! To solve the problem, two methods are used in our algorithm. One is to divide database into multiple partitions with each partition containing fitting amount of transactions, then build a global candidate set from locally high utility itemsets, and finally test the entire candidate set, just like that described in [15]. The correctness of the partition method is guaranteed by following lemma:

Lemma suppose D is a transaction database, P=P_1P_2, …, P_j is a set of partitions of D. If $X \subseteq I$ is a high utility itemset, it appears as a locally high utility itemset in at least one of the partitions.

Proof. Let X be a high utility itemset, then u(X) $\geqslant$minutil. Divide D into n partitions, then X may fall into m partitions ($1 \leqslant m \leqslant n$). Assume B=Max(u(X,$P_i$)) denote the biggest utility value of X in all partitions, By definition 4, we have

$$u(X) = \sum_{X \subseteq T_q \wedge T_q \in P_i \wedge P_i \subseteq D} u(X, P_i) \le mB$$

If $B < \frac{\text{minutil}}{n}$, then

$$u(X) < \frac{m}{n}\text{minutil} \le \text{minutil}$$

But this is a contradiction.

Another method is to filter out all short (intersection) transactions. The rational behind the method is that short transactions have no effect on the support and utility of long itemsets. Since the intersection of long transactions is usually very short, large amount of short intersection transactions will be pruned out in time. This is why our method is so efficient for long patterns.

But how should we choose the number of partitions? Let u be total utility value, and coefficient a be the minimum acceptable ratio of the utility value of an itemset to the total utility value in the database. Suppose we divide the database D into n partitions, the local utility threshold ($\frac{\text{minutil}}{\text{n}} = \frac{a*u}{\text{n}}$) should be far larger than the average transaction utility ($\frac{\text{u}}{\text{N}}$), denoted as $\frac{\alpha*\text{u}}{\text{n}} >> \frac{\text{u}}{\text{N}}$. Otherwise, large amount of locally high utility itemsets would be generated. Let S be partition size, we have:

$$S = \frac{N}{n} >> \frac{1}{a} \quad (5)$$

Inequation (5) contradicts the goal of partition method (reducing the amount of intersection transaction in a partition). Experiments show that it's applicable for S to be between $5/a$ and $10/a$ in the context of our datasets.

Inter-transaction can be described as follows:

Input: A transaction database D, minutil, minlen

Output: All long high utility itemsets.

Method:

```
1. Divide database D into n partitions;
2. For every partition $P_i \in D$, call Subroutine
   gen-LHU-itemsets to obtain all locally long high
   utility itemsets;
3. Obtain a candidate set C by union of all locally
   long high utility itemsets;
4. Scan the database again to calculate the utility
   and support of each itemset $c \in C$. if utility value
   c.utility≥minstil, output its utility and support;
```

The algorithm is very similar to partition algorithm [15], one of the main differences between them is that the former seeks balance between keeping a higher local utility threshold and reducing the amount of intersection operation, while the later choose partition size in terms of main memory size, such that at least those itemsets and other information that are used for generating candidates can fit in main memory.

Gen-LHU-itemsets is responsible for generating locally long high utility itemsets in a partition. In the subroutine, tidlist is used to record which transactions are involved in an intersection transaction. If s(T(tidlist)) and u(T(tidlist)) represent the current support and utility of T(tidlist) respectively, T(tidlist).tidlist represents transaction identifier list associated with T(tidlist), let tidlist= tidlist1 ∪ tidlist2 (tidlist1≠tidlist2), we have:

$$\text{T(tidlist)} = \text{T(tidlist1} \cup \text{tidlist2)}$$
$$= \text{T(tidlist1)} \cap \text{T(tidlist2)} \quad (6)$$

$$\text{T(tidlist).tidlist} = \textit{tidlist} = \text{tidlist1} \cup \text{tidlist2} \quad (7)$$

$$\text{s(T(tidlist))} = |\text{T(tidlist).tidlist}|$$
$$= |\text{tidlist1} \cup \text{tidlist2}| \quad (8)$$

$$\text{u(T(tidlist), tidlist)} = \sum_{Tq \in tidlist} u(T(tidlist), T_q) \quad (9)$$

To compute the utility of an itemset in a k-intersection transaction, we assume all the transactions listed in tidlist form a partition, so that we can use equation (9) to compute the utility, which stems from equation (2). The subroutine is described as follows:

Input: A partition P_i, minutil, minlen

Output: All locally long high utility itemsets in P_i.

Method:

```
1. Take a partition $P_i$ and calculate the utility of
   each (intersection) transaction independently
   according to equation (4) for individual
   transaction or equation (9) for intersection
   transaction. If $u(T_q) \geq$ minutil/n, put $T_q$ into
   candidate set, call subroutine mine_single_trans;
2. Perform all the intersections of any two
   (intersection) transactions;
3. If there is no long itemset, subroutine ends;
4. Prune out all short intersection transactions,
   merge repetitious intersection transactions,
   update corresponding tidlist according to
   equation (7). go to step 1;
```

Subroutine mine_single_trans tries to discover all locally long high utility itemsets an intersection transaction contains. It can be described as follows:

Input: T_q, minutil, minlen

Output: All locally long high utility itemsets in T_q

Method:

```
1.  Sort the transaction $T_q$ decreasingly by its utility
    value: $T_q = t_0\ t_1\ t_2\ \ldots\ t_{k-1}\ t_k\ \ldots\ t_{L-1}$,
    satisfying $u(t_i, T_q) \geq u(t_j, T_q) (i \leq j)$;
2.  Let k=L-1;
3.  p=0;  // position of the first item of X in $T_q$
4.  Let $X = t_p t_{p+1} \ldots t_{p+k-1}$. if u(X) ≥minutil/n, add X
    into candidate set, go to step 5, otherwise,
    subroutine ends;
5.  For j=1 to k do begin
6.    Count=0;
7.    For i=p to L-k-1 do begin
8.      Replace $t_{k-j}$ in itemset X with $t_{k+i}$, obtaining
        a new itemset X'. If u(X') ≥minutil/n, add X'
        into candidate set, count increases by one; if
        u(X')<minutil/n, break (exit loop);
9.    End;
10.   If count=0, break;
11. End;
```

12. If there isn't any high utility k-itemset, subroutine ends; if all the k-itemsets verified in step 8 are high utility itemsets，p increases by one, go to step 4 until all the k-itemsets are verified;
13. Let k=k-1, go to step 3, until k=minlen

The subroutine uses Quick Sort method to sort the items descendingly by utility value so that we can produce and check only necessary itemsets, pruning out low utility itemsets as many as possible. Suppose $Y_i=X\cup t_i$, $Y_j=X\cup t_j$, if i<j, then $u(Y_i) > u(Y_j)$. if Y_i is low, Y_j must be low. Furthermore, if all the k-itemsets are low, all (k-l)-itemsets must be low ($1\le l\le k-1$). Following example can show how mine_single_trans works:

Let T_q=ABCDEF, corresponding utility values are 6, 4, 5, 1, 3, 2. After sorting, T_q can be expressed as ACBEFD, relevant utility values can be write as U=654321. Here the length of T_q is 6, i.e., L=6. if minutil=18, minlen=3, itemsets will be examined in the order shown in table 2.

Itemset Utility	comments
u(ACBEF)= 20	add ACBEF to H_i
u(ACBED)= 19	add ACBED to H_i
u(ACBFD)= 18	add ACBFD to H_i
u(ACEFD)= 17	stop finding 5-itemsets
u(ACBE) =18	add ACBE to H_i
u(ACBF) =17	stop finding 4-itemsets
u(ACB) =15	Algorithm end

Table2. The process of calculating utility

According to step 1) of gen-LHU-itemsets, only a small amount of (intersection) transactions need to call the subroutine and thus won't cause high computational cost.

5 Experimental results

All the experiments were performed on a 2GHz XEON server with 2GB of memory, running windows 2003. Program was coded in Delphi 7. The synthetic databases used in our experiments are T40.I30.D8000K with the number of items varying from 0.5k to 4K, which were generated by IBM quest data generator [16]. Because the generator only generates the quantity of 0 or 1 for each item in a transaction, we use Delphi function "RandG" to generate random numbers with Gaussian distribution, which mimic the quantity sold of an item in each transaction.

Figure 1 shows that our algorithm scales linearly with the number of transactions. Figure 2 shows the performance when varying the number of items. Different from other algorithms, the performance of inter-transaction increases with the increase of the number of items. In figure 3, minlen is the minimum length of itemsets the algorithm can discover within a reasonable time. The experiment result indicates that the larger the number of items, the smaller the minlen, and the less the tasks left for its cooperator such as UMining. That's to say, inter-transaction can complete more works in a sparse dataset. Figure 4 shows the execution time decreases as the utility threshold increases.

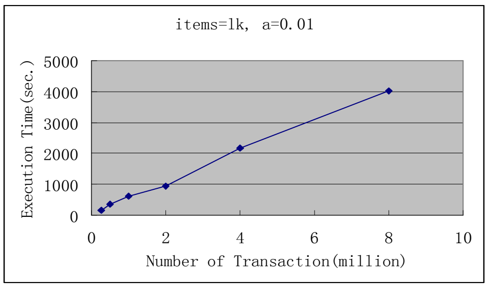

Fig.1 Scalability with transaction number

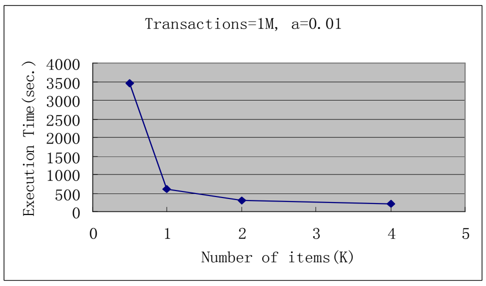

Fig.2 Scalability with item number

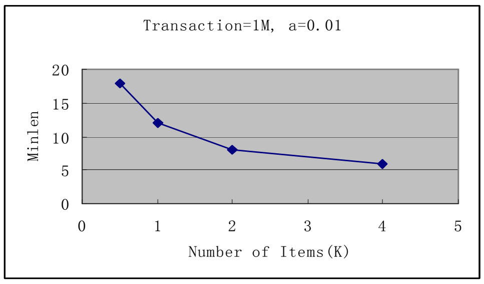

Fig.3 Effect of item number on minlen

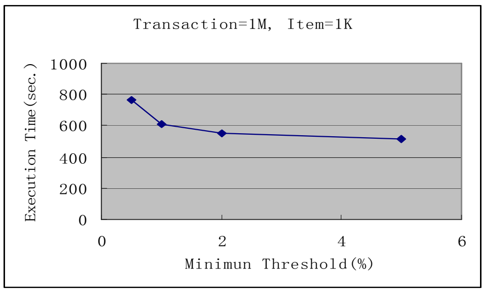

Fig.4 Scalability with threshold

6 conclusions

The paper decomposes the mining task into two parts (mining long high utility itemsets and short high utility itemsets) and emphasize on introducing a partition-base, top-down algorithm, i.e., inter-transaction, to discover long high utility itemsets. Besides the inherent advantages of partition algorithm [15], the new algorithm can obtain long itemsets directly from the intersection of relevant long transactions, without extending short itemsets step by step. In addition, new pruning strategies are used to cut down search space. It is very suitable for large high dimensional data.

Data skew, number of items and parameters such as threshold minlen affect its performance. How to choose threshold minlen is relevant to the hybrid model, and it's our future work.

References:

[1] Yao H. and Hamilton, H.J., Mining itemset utilities from transaction databases, Data & Knowledge Engineering, 59 , 2006, pp. 603 – 626

[2] V. H. Vroom, Work and Motivation, John Wiley, 1964

[3] Shen Y. D., Zhang Z. and Yang Q, Objective-oriented utility-based association mining, Proceedings of the 2002 IEEE International Conference on Data Mining, 2002, pp. 426-433

[4] Yao H. and Hamilton, H.J., A Unified Framework for Utility Based Measures for Mining Itemsets, Proceedings of the 2006 International Workshop on Utility-Based Data Mining 2006, 28-37, Philadelphia, PA.

[5] K. Wang, S. Zhou, J. Han, Profit mining: from patterns to action, Proceedings of International Conference on Extending Database Technology, 2002, pp. 70-87

[6] Lin TY, Yao YY, Louie E. Mining Value Added Association rules, Proceedings of PAKDD, 2002, pp. 328-333.

[7] Aumann Y., Lindell Y. , A Statistical Theory for Quantitative Association Rules, Journal of Intelligent Information Systems, 20, 2003, pp. 255–283.

[8] Geoffrey I. Webb, Discovering associations with numeric variables, Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, 2001, pp. 383-388

[9] C.H. Cai, Ada W.C. Fu, C.H. Cheng and W.W. Kong, Mining Association Rules with Weighted Items, Proceedings of the International Database Engineering and Applications Symposium, 1998, pp. 68-77

[10] Lu S.F., Hu H.P. and Li F, Mining weighted association rules, Intelligent Data Analysis, 5, 2001, pp. 211-225

[11] Chan, R., Yang, Q., and Shen, Y.D. , Mining high utility itemsets, Proceedings of the 3rd IEEE International Conference on Data Mining, 2003, pp.19-26

[12] Barber, B., and Hamilton, H. J., Extracting Share Frequent Itemsets with Infrequent Subsets, Data Mining and Knowledge Discovery, 7, 2003, pp. 153-185

[13] Yao H., Hamilton, H.J. and Butz, C.J. (2004). A Foundational Approach to Mining Itemset Utilities from Databases. Proceedings 2004 SIAM International Conference on Data Mining, 2004, pp. 482-486

[14] Liu, Y., Liao, W.-K., and Choudhary, A fast high utility itemsets mining algorithm, Proceedings of the First International Workshop on Utiliy-based Data Mining, 2005, pp. 90-99

[15] Savasere A, Omiecinsky E and Navathe S (1995). An efficient algorithm for mining association rules in large databases. 21st Int'l Conf. on Very Large Databases, 1995, pp. 432-444.

[16] http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html