iFUM - Improved Fast Utility Mining

S. Kannimuthu
Assistant Professor
Sri Krishna College of Engineering and Technology, Coimbatore, India

Dr. K. Premalatha
Professor
Bannari Amman Institute of Technology, Sathyamangalam, India

S. Shankar
Associate Professor
Sri Krishna College of Engineering and Technology, Coimbatore, India

ABSTRACT
The main goals of Association Rule Mining (ARM) are to find all frequent itemsets and to build rules based of frequent itemsets. But a frequent itemset only reproduces the statistical correlation between items, and it does not reflect the semantic importance of the items. To overcome this limitation we go for a utility based itemset mining approach. Utility-based data mining is a broad topic that covers all aspects of economic utility in data mining. It takes in predictive and descriptive methods for data mining. High utility itemset mining is a research area of utility based descriptive data mining, aimed at finding itemsets that contribute most to the total utility. The well known faster and simpler algorithm for mining high utility itemsets from large transaction databases is Fast Utility Mining (FUM). In this proposed system we made a significant improvement in FUM algorithm to make the system faster than FUM. The algorithm is evaluated by applying it to IBM synthetic database. Experimental results show that the proposed algorithm is effective on the databases tested.

General Terms
Algorithms, Performance, Process, Results.

Keywords
ARM, Data Mining, FUM, HUI, iFUM, UMining, Utility Based Data Mining.

1. INTRODUCTION
Data mining can be regarded as an algorithmic process that takes data as input and yields patterns, such as classification rules, itemsets, association rules, or summaries, as output [9]. The most significant tasks in data mining are the process of discovering frequent itemsets and association rules [1]. Numerous efficient algorithms are available in the literature for mining frequent itemsets and association rules. Incorporating economic utility considerations in data mining tasks is gaining popularity in recent years. The aim of utility-based data mining is to integrate utility considerations in both predictive and descriptive data mining tasks. Certain association rules enhance the business value and the data mining community has acknowledged the mining of these rules of interest since a long time. Several business applications have been found to benefit from the discovery of frequent itemsets and association rules from transaction databases. The goal of Frequent Itemset Mining is to find items that co-occur above a user given value of frequency, in the transaction database. In the Frequent Itemset Mining problem, the occurrence of each item in a transaction is represented by a binary value without considering its quantity or an associated weight such as price or profit [8]. However, quantity and weight are significant for addressing real world decision problems that require maximizing the utility in an organization.

Interestingness measures can play an important role in knowledge discovery. These measures are intended for selecting and ranking patterns according to their potential interest to the user [6]. In practice, the frequency of occurrence may not express the semantics of applications, because the user's interest may be related to other factors, such as cost, profit, or aesthetic value.

Utility based data mining refers to allowing a user to conveniently express his or her perspectives concerning the usefulness of patterns as utility values and then finding patterns with utility values higher than a threshold [7]. A pattern is of utility to a person if its use by that person contributes to reaching a goal

The remaining parts of the paper are organized as follows. In section 2, high utility itemset mining process is described. Existing algorithms were discussed in section 3. In section 4, proposed iFUM algorithm is presented. In section 5, experimental results were presented and analyzed. Section 6 concludes the paper

2. HIGH UTILITY ITEMSET MINING PROCESS
A frequent itemset is a set of items that appears at least in a pre-specified number of transactions. Formally, let $I = \{i_1, i_2, ..., i_m\}$ be a set of items and $DB = \{T_1, T_2, ..., T_n\}$ a set of transactions where every transaction is also a set of items (i.e. itemset). Given a minimum support threshold MST an itemset S is frequent iff:

$$\frac{|\{TR \mid S \subseteq TR, TR \subseteq D, S \subseteq I\}|}{|D|} \geq MST \quad (1)$$

2.1 High Utility Itemsets (HUI)
An itemset with utility value greater than the minimum threshold utility as specified by the user depending upon his context of usage is called as the high utility itemset [11]. A well known model for mining such high-utility itemset was defined by Yao et al which is a generalization of the share-mining model [2, 3]. The following is the set of definitions given by Yao et al and is illustrated by a simple example.

A transaction independent numerical value say y_p defined by the user which reflects the importance or the profit value of an item say i_p is defined as the external utility of the item i_p. External utilities are stored in a utility table. For example, external utility of item B in Table 2 is 10.

A transaction dependent numerical value say x_p defined by the user which reflects the quantity Q of an item say i_p in a transaction say T is defined as the internal utility of the item i_p. For example, internal utility of item E in transaction T5 is 2 as mentioned in Table 1.

Utility function f is a function of two variables commonly defined as the product of internal and external utility as given below

$$f(x, y) : x_p \times y_p \quad (2)$$

Table 1 Database with 10 Transactions and 5 Distinct Items

TID	A	B	C	D	E
1	0	0	18	0	1
2	0	6	0	1	1
3	2	0	1	0	1
4	1	0	0	1	1
5	0	0	4	0	2
6	1	1	0	0	0
7	10	0	0	1	1
8	3	0	25	3	1
9	1	1	0	0	0
10	0	6	2	0	2

Table 2 External Utilities of Items from Database given In Table 1

Item	A	B	C	D	E
Profit	3	10	1	6	5

The utility of item i_p in transaction T is the quantitative measure computed with utility function as defined above

$$\text{(i.e.) } u(i_p, T) = f(x_p, y_p), i_p \in T \quad (3)$$

For example: utility of item E in transaction T5 is $2 \times 5 = 10$. The utility of itemset S in transaction T is defined as follows

$$U(S) = \sum_{i_p \in s} u(i_p, S) \quad (4)$$

For example: utility of itemset {C, E} in transaction T_1 is $u(\{C,E\}, T_1) = u(\{C\}, T_1) + u(\{E\}, T_1) = 18 \times 1 + 1 \times 5 = 23$. On the basis of the above definitions, high utility itemset can be defined as follows

Itemset S is of high utility if $U(S) \geq \text{minUtil}$ where minUtil is user defined utility threshold in percents of the total utility of the database. High utility itemset mining is the task of finding set H defined as

$$H = \{S \mid S \subseteq I, U(S) \geq \min Util\} \quad (5)$$

where 'I' is the set of items (attributes).

3. EXISTING ALGORITHMS

3.1 UMining Algorithm

One of the well known algorithm used for mining all high utility itemsets is UMining [10]. Figure 1 briefly describes the Umining algorithm.Further details of the Umining algorithm and detailed description of the function Scan, CalculateAndStore, Discover, Generate, and Prune can be found in [10].

Input : -database T
-constraints minUtil
Output : -all high utility itemsets H

1. I = **Scan(T);**
2. C1= I;
3. k = 1;
4. C_k = **CalculateAndStore(C_k,T, f);**
5. H = **Discover(C_k, minUtil);**
6. while (|Ck| > 0 and k <= K)
7. {
8. k = k + 1;
9. Ck = **Generate(C_{k-1}, I);**
10. C_k = **Prune(C_k, C_{k-1}, minUtil);**
11. C_k = **CalculateAndStore(C_k,T, f);**
12. H = H $\bigcup$ **Discover(C_k, minUtil);**
13. }
14. return H;

Fig 1: Pseudo Code of the Umining Algorithm

3.2 FUM Algorithm

FUM [5] algorithm is used for mining all high utility itemsets. Figure 2 briefly describes the FUM algorithm which generates high utility itemsets using CombinationGenerator.

It is simpler and executes faster than Umining algorithm, when large number of itemsets are identified as high utility itemsets and as the number of distinct items increases in the input database.

The CombinationGenerator(T) is a method which is used to generate all the combinations of the items. It takes the Itemld and the level as the input which is generally denoted by the variable loop. The factorial computation method is defined in this, to generate the factorial of a given number.

The combination generation is based on the concept proposed by Kenneth H. Rosen, Discrete Mathematics and its applications [4]. First the items for which the combination is to be generated is put in the form of an array. Then the getNext() method is called until there are no more combinations left. The getNext() method returns an array of integers, which tells the order in which to arrange the original array of letters.

Task: Discovery of High Utility Itemsets
Input: Database DB {Set of Transactions}
Transaction $T \in DB$ Minimum Utility value threshold minUtil
Output: High Utility Itemsets H

1. Compute the utility value $\forall$ single itemset
2. For each $T \in DB$
3. begin
4. if T $\notin$ S {where S $\subseteq$ DB | S = [0 .. T-1]}
5. begin
6. Candidateset = CombinationGenerator(T)

7. For each C ∈ CandidateSet
8. begin
9. if ($C \notin H$) $\wedge$ ($U(C,T) \geq \min Util$)
10. H.add (C);
11. end
12. end
13. end
14. return (H);

CombinationGenerator(T) - Generate all possible combinations of itemset ∈ T

Fig 2: Pseudo code of the FUM algorithm

Let us consider Table 1 and Table 2 as input to the proposed FUM algorithm. we compute the utility values of all single itemsets say A, B, C, D and E in step 1(as explained in section 2). In step 2, we begin a loop for processing each and every transaction present in the DB one by one. In step 4, the algorithm generates the itemsets in the current transaction. For e.g. In Table 1, the first transaction is represented as CE according to FUM algorithm, since only those two items were purchased in that transaction. FUM algorithm omits the remaining items A, B and D. In a similar way, the remaining transactions are processed. The algorithm also checks (step 4), whether a transaction defined by an itemset purchased in it, repeats its occurrence in a later transaction. If a later transaction also contains same itemset purchased in any of the previous transactions, then that transaction is ignored from processing. In Step 6, the candidate itemsets are generated using the CombinationGenerator(T) function, which takes itemset, purchased in a particular transaction as input and generate the various possible combinations of the itemset. In the consecutive steps, the algorithm analyzes each candidate belonging to the candidate itemsets generated. In step 9, the algorithm computes the utility value of each and every candidate, U(C,T) as described in section 2. If the utility value of a candidate is found to be more than the minimum utility threshold, which is given as input by the user, (say a sales manager) then that particular candidate is added to the set of High Utility Itemsets {H} in step 10 of the algorithm. The condition C $\notin$ H in step 9 simply ensures no duplicate high utility itemsets are generated.

4. PROPOSED iFUM ALGORITHM

The core step of FUM algorithm is CombinationGenerator(T) which takes significant time to compute. In the existing system FUM combination generation is performed for itemsets and its subset without checking one important condition. This is illustrated as follows,

A set X is a subset of a set Y if every element of X is also an element of Y. Such a relation between sets is denoted by $X \subseteq Y$

The CombinationGenerator output for itemset AB are $\{A, B, AB\}$ and itemset ABC are $\{A, B, C, AB, BC, AC, ABC\}$. If we have already computed the utility value for any of the subsets of ABC and if any of the subset repeats itself as a later transaction, then it is not necessary to generate the subsets again (which is implemented in step [4]), say for AB and also not necessary to calculate the utility value for the same. Existing FUM algorithm fails to check this condition so it generates the combinations for the already generated subset of the itemsets too, if it repeats in a later transaction of the input database. Our proposed algorithm avoids these extra computations and enhances FUM efficiency. The proposed iFUM algorithm is illustrated in Figure 3.

In step 1 of the iFUM algorithm, we compute the utility values of all single itemsets. In step 2, we begin a loop for processing each and every transaction present in the DB one by one. In step 4, the algorithm generates the itemsets in the current transaction. The algorithm checks (step 4) whether a transaction defined by an itemset purchased in it, repeats its occurrence in a later transaction. If a later transaction also contains same itemset purchased in any of the previous transactions, then that transaction is ignored from processing and the condition also checks if the utility value for any of the subsets is computed already, then it is not necessary to generate the subsets again. Remaining steps are the same as in FUM algorithm.

Task: Discovery of High Utility Itemsets
Input: Database DB {Set of Transactions}
Transaction $T \in DB$
Minimum Utility value threshold minUtil
Output: High Utility Itemsets H
[1] Compute the utility value $\forall$ single itemset
[2] For each $T \in DB$
[3] begin
[4] if $(T \notin S) \wedge (T \not\subset S)$ {where S ⊆ DB | S = [0 .. T-1]}
[5] begin
[6] Candidateset = CombinationGenerator(T)
[7] For each C ∈ CandidateSet
[8] begin
[9] if $(C \notin H) \wedge (U(C,T) \geq \min Util$
[10] H.add (C);
[11] end
[12] end
[13] end
[14] return (H);

CombinationGenerator(T) - Generate all possible combinations of itemset ∈ T

Fig 3: Pseudo code of the iFUM algorithm

5. EXPERIMENTAL RESULTS

We have evaluated the performance of our algorithm and compared it with FUM and UMining algorithm. The experiments were performed on a 1.86 GHz Intel Celeron M CPU Processor with 1 GB RAM, and running on Windows XP. The algorithms were implemented in Java language. The data utilized in our experimental results is widely-accepted IBM synthetic data called T10I4D100K which is obtained from IBM dataset generator. This dataset contains 100,000 transactions and 1000 distinct items. T10I4D100K denotes the Average size of the transactions (T), Average size of the maximal potentially large itemsets (I) and the number of transactions (D). Utility values for the items were assigned randomly in the profit table.

The experiments were conducted by varying the number of

distinct items from 50 to 1000 that are totally available by keeping the Minimum Utility Threshold at 1% throughout the experiment and execution time was recorded for iFUM, FUM and UMining algorithms (refer Table 3).

The proposed approach results in a significant reduction in the execution time. The test results are illustrated through graph (refer Figure 4). From the Figure 4 and Table 3, it is observed that the execution time of iFUM is considerably less than the existing FUM algorithm.

Table 3. Comparison of iFUM, FUM and UMining Algorithms based on Number of Items

Number of Items	Execution Time (Seconds)		
	iFUM	FUM	UMining
50	1176	1246	12268
100	1169	1244	*
200	1250	1364	*
500	1109	1249	*
1000	1112	1184	*

*Indicates that, we have to manually stop the system as it hanged while executing the UMining algorithm. Hence the execution time could not be measured for 100, 200, 500 and 1000 items respectively for Umining Algorithm.

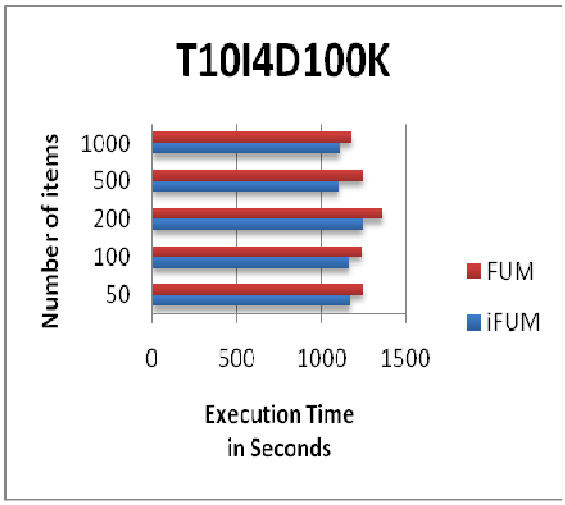

Fig 4: Comparison of iFUM and FUM algorithms based on number of items

Table 4. Comparison of execution time, High Utility Itemsets (HUI) and Minimum Utility Threshold in iFUM and FUM algorithms

Minimum Utility Threshold	iFUM		FUM	
	HUI	Execution Time Seconds	HUI	Execution Time in Seconds
0.25	62122	1137	62122	1235
0.5%	13620	1133	13620	1221
1%	5134	1126	5134	1209
5%	489	1125	489	1191
10%	2	1109	2	1177

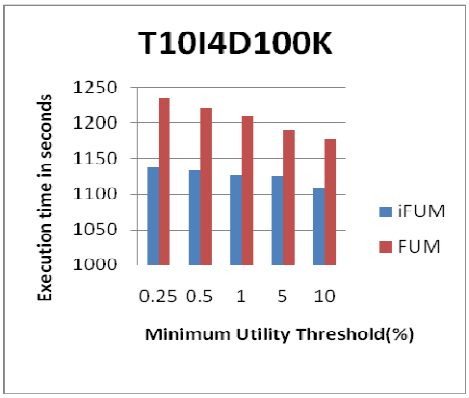

Fig 5: Comparison of Execution time, Minimum Utility Threshold in iFUM and FUM Algorithms

The experiment was also conducted by varying Minimum Utility Threshold (refer Table 4) by keeping 1000 distinct items as fixed. It can be observed that execution time of iFUM algorithm proved to be less than existing FUM algorithm as seen in Figure 5.

6. CONCLUSION

In this paper, existing UMining and FUM algorithms were discussed. We proposed the improved version of FUM algorithm, iFUM for mining all High Utility Itemsets. The proposed algorithm is compared with existing popular algorithms like UMining and FUM by using IBM synthetic data set. The experimental result shows that iFUM algorithm is faster than other existing algorithms in the literature. The iFUM algorithm also scales well as the number of distinct items increases in the input database as shown by the experimental results obtained.

7. REFERENCES

[1]Agrawal R, Srikant R, "Fast algorithms for mining association rules", Proceedings of 20th International Conference on Very Large Databases, Santiago, Chile, pp. 487-499, 1994.

[2]Carter C, Hamilton H J, Cercone N, "Share based measures for itemsets", Proceedings of First European Conference on the Principles of Data Mining and Knowledge Discovery, pp. 14-24, 1997.

[3]Hilderman R J Carter C L Hamilton H J Cercone N, "Mining market basket data using share measures and characterized itemsets", Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 159-170, 1998.

[4] Kenneth H. Rosen, "Discrete Mathematics and Its applications", Mc Graw Hill., 4th edition, 298-300.

[5] S.Shankar, Dr.T.Purusothaman, S.Jayanthi "A Fast Algorithm for Mining High Utility Itemsets", IEEE International Advance Computing Conference (IACC 2009) Patiala, India, 6-7 March 2009

[6] Shankar.S and T.Purusothaman, "A Novel Utility Sentient Approach for Mining Interesting Association Rules", IACSIT International Journal of Engineering and Technology Vol.1, No.5, December, 2009, ISSN: 1793-8236

[7] S Shankar, T Purusothaman, "Discovering imperceptible associations based on Interestingness: a utility-oriented data mining approach", Data Science journal, volume 9, 24 February 2010

[8] Shankar.S and T.Purusothaman, "Utility Sentient Frequent Itemset Mining and Association Rule Mining: A Literature Survey and Comparative Study", International Journal of Soft Computing Applications ISSN: 1453-2277 Issue 4 (2009), pp.81-95

[9] Yu-Chiang Li, Jieh-Shan Yeh, Chin-Chen Chang "Isolated items discarding strategy for discovering high utility itemsets", Elsevier Journal, Data & Knowledge Engineering 64 (2008) 198–217.

[10] Yao H and Hamilton J, "Mining itemset utilities from transaction databases", Data & KnowledgeEngineering, pp. 59: 603-626, 2006.

[11] Yao H, Hamilton H J, Butz C J, "A foundational approach to mining itemset utilities from databases", Proceedings of the Third SIAM International Conference on Data Mining, Orlando, Florida, pp. 482-486, 2004.