Enhanced Association Rule Mining Algorithm to Extract High Utility Item sets from a Large Dataset

K. Karpagam

Assistant Professor, Computer Science Department,

H.H. The Rajah's College(Autonomous),

Pudukkottai, Tamil Nadu, India.

kkarpaga05@gmail.com

Abstract— **Data mining aims at bringing out the hidden information from an extensive data set using data mining techniques according to the requirements. Association rule mining identifies itemsets that frequently occur in data set and frames association rules by taking all items equally. But many differences exist among the items that play a vital role in decision making. By taking one or more values of items as utilities, the utility mining technique works on finding the itemsets with more significant utilities. In the proposed paper, we present a utility mining algorithm named IUM (Improved Utility Mining) algorithm that finds high utility itemsets, and also low utility itemsets from an extensive data set, and the experiments state that the proposed algorithm performs better than existing algorithms in case of running time.**

Keywords— **Association rules, frequent itemsets, low utility itemset, high utility itemset**

I. Introduction

Data mining is the process of searching for new information and pattern from a big dataset[1]. It is the process of discovering useful knowledge by automatic or semiautomatic means from a collection of data [2]. Data mining techniques are derived after long term vigorous research and development. Association rule mining finds new associations or correlations among large sets of data. The frequent itemsets identified by ARM does not take into account any properties of the items except the frequency of occurrence[6]. But the items vary in many angles in the real world. The difference between the properties of items has a more significant influence on decision making based on association rules. Hence, standard ARM cannot adequately meet the actual requirement arising from different applications [8]. By considering the various factors of items as utilities, utility mining aims at detecting the high utility itemsets. In all stages of the data mining process (e.g., acquisition, extraction, and application), there is always a utility factor involved, which will have a considerable impact on the final extracted results. The foremost goal of Utility-based Data Mining is to amplify and boost the expected utility related to the entire mining process, thereby enhancing the utility results in one or more of the mining stages.

For more composite circumstances, many models and methodologies are built based on prior experience faced after knowing the costs/benefits involved in a decision. Using these models, we predict the "best" decision to take on complex situations. Apart from all these, our final decision ought to be the one which was predicted from earlier experience to attain maximum utility value.

II. RELATED STUDY

Of late, the utility mining has emerged as an essential research topic in data mining verticals. The process of mining and unearthing high utility itemsets from datasets refers to discovering the itemsets with high profits. Here in utility itemset mining [5], the connotation of itemset utility is interestingness or profitability of an item to the users. The utility of items in the transactional datasets comprises of two features [9]:

1) The prominence of distinct items, which is called external utility.
2) The prominence of items in transactions, which is called internal utility.

The utility of an item set is rightly expressed as the product of its external utility and its internal utility. An itemset is said to be a high utility itemset if and only if its utility value is greater than or equal to user-specified minimum utility threshold. If not, it is said to be a low utility itemset. *Utility-based data mining* permits a user to aptly express their views concerning the usefulness of patterns related to utility values and then identifying patterns with utility values higher than the threshold utility value. Some of the existing algorithms are discussed below.

A. Two-phase algorithm

Two-phase algorithm [3] was introduced by [Liu et al. in] for generating itemsets of high utility. The two-phase algorithm effectively cuts down the number of candidates and generates the full set of high utility itemsets. In the first phase, only the combinations of high transaction weighted utilization itemsets are added into the candidate set at each level during the level-wise search. In the second phase, the database is scanned again to eliminate the itemsets that are overestimated. Anyhow, it requires the complete database to be scanned again whenever new transactions are included in the database.

First Phase: Let E={$e_1, e_2, \ldots, e_m$} be an itemset and DS={$Tr_1, Tr_2, \ldots, Tr_n$} be a transaction database. The items of each transaction Tr_i belongs to the set E. The utility of item e_x in a particular transaction Tr_y, is represented as $u(e_{px}Tr_y)$.
Transaction Utility, The transaction utility of a transaction Tr_y, denoted as $tu(Tr_y)$, is the total of the utilities of each item in Tr_y,

$$tu(T_y) = \sum_{i_x \in T_y} u(i_x, T_y)$$

Second Phase: The transaction-weighted utilization of an item set M, denoted as twu(M), is the sum of the transaction utilities of all the transactions containing M:

$$twu(M) = \sum_{M \subseteq T_q} tu(T_q)$$

B. HU-Miner Algorithm

Two In the HU-Miner algorithms [4], each itemset holds a utility-list. By scanning the database two times, the information about the utility of a database can be built, and it can be loaded in the initial utility-list. As the first step, the transaction-weighted utilities of all items are cumulated by scanning the database. If the derived transaction-weighted utility of an item is not greater than or equal to the desired minimum utility, the item is eliminated from the subsequent mining process. The items whose transaction-weighted utilities are greater than or equal to the minimum utility are sorted and stored in transaction-weighted- utility-ascending order.

Further, there is no need to scan the database again. The utility-list of 2-itemset {xy} can be constructed by the intersection of the utility- list of {x} and that of {y}. The algorithm identifies the everyday transactions by comparing the tids in the two utility-lists. Suppose the lengths of the utility-lists are m and n respectively, and then (m + n) comparisons at most are enough for identifying everyday transactions because all tids in a utility-list are ordered. The process of identification is a 2-way comparison.

For each common transaction t, the algorithm will generate an element E and append it to the utility-list of {xy}. The tid field of E is the tid of t. The iutil of E is the sum of the iutils associated with t in the utility-lists of {x} and {y}. Suppose x is before y, and then the rutil of E is assigned as the rutil associated with t in the utility-list of {y}.

Given the utility-list of itemset X, if the sum of all the iutils and rutils in the utility-list is less than a given "minutil," any extension X' of X is not high utility.

For $\forall$ transaction $t \supseteq X'$:

$\because X'$ is an extension of $X \Longrightarrow (X'-X) = (X'/X)$

$X \subset X' \subseteq t \Longrightarrow (X'/X) \subseteq (t/X)$

$\therefore u(X',t) = u(X,t) + u((X'-X),t)$

$= u(X,t) + u((X'/X),t)$

$= u(X,t) + \sum u(i,t) \leq u(X,t) + \sum i \in (t/X)\ u(i,t)$

$i \in (X'/X) \qquad\qquad i \in (t/X)$

$= u(X,t) + ru(X,t)$

Suppose id(t) denotes the tid of transaction t, X.tids denotes the tid set in the utility-list of X, and X'.tids that in X', then:

$\because X \subset X' \Longrightarrow X'.tids \subseteq X.tids$

$\therefore u(X') = \sum \quad u(X',t)$

$id(t) \in X'.tids$

$\leq \sum id(t) \in X'.tids\ (u(X,t) + ru(X,t))$

$id(t) \in X'.tids$

$\leq \sum id(t) \in X.tids\ (u(X,t) + ru(X,t))$

$id(t) \in X'.tids$

$<$minutil.

III. Proposed Improved Utility Mining (IUM) algorithm

The proposed **Improved Utility Mining (IUM) algorithm** comprises of many subprocedures. The first procedure, "ArrangedItems," is to arrange the transaction items in the order of their utility values. From the transaction row E1, each transaction item is fetched using the delimiter space and placed in an array and iterated from the zero position to upper bound value of the array and nested with another iteration which starts from a position incremented by one to an upper bound value of the array. Now the fetched items from the corresponding iterations are compared to enable sorting. If the fetched item satisfies the condition given, then the corresponding values will be swapped. After the iteration loops end, the resultant sorted items will be stored in the array ArrangedItem.

PROCEDURE ArrangeItems (Elements E)

Input: ELEMENTS E
Output: ArrangedItems

```
1. Count no. of Items and store in ETot
2. For all E ∈ETot Do
   2.1 For all E+1 ∈ETot Do
      2.1.1 IF [ E+1< E ]
            Exchange [ E+1, E ]
3. RETURN ArrangedItem
```

The second procedure, "FindSets," finds utility values of all possible itemsets exist in the transaction dataset of various lengths, and the low utility itemsets are pruned away. The input for this procedure is the sorted transaction items. The sorted items, along with the corresponding costs, are first loaded into the memory. Next is to find the total item count present in the sorted transaction row. The second step is to **i**nitialize the permutation value pm=1 to iterate through the items to generate itemsets without repetition. Loop until the perm value is equal to the count found in step1, meanwhile inside the loop combine the items to form candidates. Check whether the pm value is equal to 1, to form two-item combinations. Initialize a TEMP array and store the two items combinations along with the utility costs.

PROCEDURE FindSets (ArrangedItem Ai)

Input: Array ArrangedItem

Output: Candidates CG after removing redundant combinations

```
1.  Ct = Total Items in ArrangedItem
2.  Sc = Support Count of single item
3.  Initialize Pm = 1
4.  While [ Pm <= Ct] do
 4.1. IF Pm = 1 do
    Add Tp = ArrangeItem[Pm]∪ArrangeItem[Pm+1]
    Prune Duplicates in Tp w.r.t. Sc
        Calculate UtilityCost UC
        CG 🡐Tp∪ UC
     ELSE
        Add CG = Tp∪SortedItem[Pm]
        Calculate UtilityCost UC
        STORE CG🡐CG∪ UC
 4.2      Increment Pm by 1
5.  Return CG
    .
```

The next procedure is "FUC" (FindUtilityCombination). The array from the previous procedure is given as input. The minimum utility threshold value is also given, and the high utility itemsets and Low utility itemset values are found. The second level of pruning concerning utility consideration is carried out hereafter, calculating the total transaction utility value. The total transaction utility value found is compared with the minimum utility threshold value. If found equal or greater, it is stored in high utility itemset results, and else it is stored in low utility itemset results.

PROCEDURE FUC(CANDIDATES RArray, Minimum Utility MU)

Input: Candidates RArray, Minimum Utility value MU

Output: UtilityItemsets

```
1.  For all I ∈RArray Do
    1.1   For all I+1 ∈RArray Do
        1.1.1 IF [ RArray[ I ] = RArray[ I+1 ]  ] do
    Calculate TransactionTotalUtility TTU
                Combine HighUtilityItems
    1.2  IF [ TTU >= MU ] Do
          Calculate Minsup
          Add HighUtilityItems in HIGHRESULT
        ELSE
          FINDLOWUTILITY [ TTU< MU].
          Add in LOWRESULT
```

Next is the main procedure which inherits and calls all the subprocedures explained above, and the resultant high utility itemsets and low utility itemsets are found and displayed for users.

PROCEDURE IUM (DATASET D, MininumUtility MU)

Input: Transaction database D, User specified minimum

```
        utility value MU
  Output: Set of High Utility Itemsets and Low Utility Itemsets

  1.   Scan D
  2.   For all R ∈ D do
    2.1 Split Transaction Data TD and Transaction Cost TC
    2.2 Insert TD in TDarray, Insert TC in TCarray
    2.3 For all R ∈TDarray do
        2.3.1 SORTdataItems(R)
        2.3.2   Reorder  TCarray according to TDarray
    2.4 For all  R∈TDarray do
        2.4.1  CG = FindSets(R)
        2.4.2 Add CG in ResultArray
  3. GUC(ResultArray, MU)
```

IV. PERFORMANCE AND ANALYSIS OF ALGORITHM

The proposed algorithm introduces two-level pruning techniques, and the first level pruning is done based on the support count of the items present in the transaction rows. The second level pruning is done based on the transaction utility values, and these two levels of pruning effectively reduce the number of candidates generated to a greater extent and which in turn reduces the search space, time consumption for execution, and the memory used for execution.

The proposed algorithm, along with the existing algorithms, are executed on the kosarak dataset sized 500 KB, and the number of items present is 320 with a transactional length of 3210 rows, and the average length is found to be 5.5. This characteristic of the dataset seems to be a sparse one, and the proposed algorithm is executed to check the execution time with varying utility values. The time taken for the algorithms to produce the results after applying it on the dataset is noted for varying utility threshold values.

The performance of the proposed algorithm is compared with the other two existing algorithms. The running time of the algorithms on the dataset Kosarak is noted. Running time was recorded, and it contains CPU time, input time, and output time. The results found are matched with the 2 Phase and HU-Miner algorithms and exhibited in the table below.

TABLE 1
EXECUTION TIME COMPARISON CONCERNING SIZE 500 KB

KOSARAK - size 500 KB					
Algorithm	**Minimum Threshold Utility %**				
	1%	**2%**	**3%**	**4%**	**5%**
2 Phase	260	180	100	65	29
HU-Miner	73	30	12	10	7
Proposed	35	21	10	8	5

When running time is measured, the minimum threshold value is varied for each dataset size. The lower threshold is, the number of high utility itemsets generated will be more significant, and thus the execution time will be more. The chart below showcases the difference in the execution time of the three algorithms.

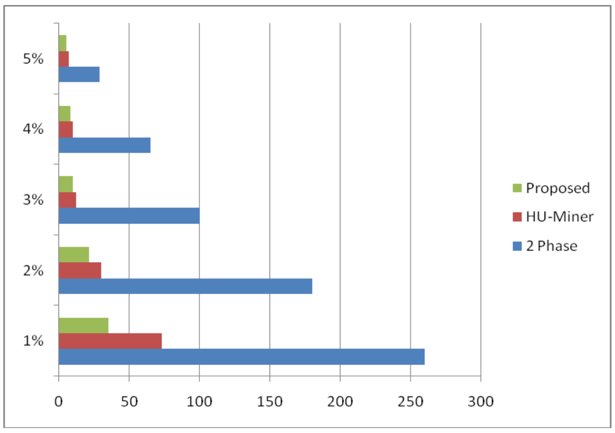

Fig: 1 Execution time Comparison on kosarak – 500 KB

From the above table and figure, it is evident that the execution time for the proposed algorithm is better than the other two algorithms taken for comparison.

V. Conclusion

In this paper, a new algorithm named "IUM" for mining high utility itemsets from transaction datasets is proposed. Here in the proposed algorithm, several strategies to enhance the performance of utility mining were carried out and successfully implemented. In the experiments, the benchmarked data sets were used to perform a thorough performance evaluation related to running time, candidate generation, and memory usage. Results exhibit that the strategies employed by the proposed algorithm considerably improved the overall performance by reducing the execution time.

References

[1] Sadak Murali, Kolla Morarjee, A Survey on Efficient Algorithm for Mining High Utility Itemsets, IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 5, Oct-Nov,2013.
[2] Rachna Somkunwar, A study on various data mining Approaches of Association Rules, International Journal of Advanced Research in Computer Science and Software Engineering, September 2012, pp, 141-144.
[3] Ying Liu, Wei-kengLiao, Aloc Choudhary, A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets, PAKDD 2005, LNAI 3518, pp. 689 – 695, 2005. © Springer-Verlag Berlin Heidelberg 2005.
[4] Mengchi Liu, Junfeng Qu, Mining High Utility Itemsets without Candidate Generation, CIKM'12, October 29–November 2, 2012, Maui, HI, USA.
[5] P.Asha, Dr.T.Jebarajan, G.Saranya, A survey on Efficient Incremental Algorithm for Mining High Utility Itemsets in Distributed and Dynamic Database, International Journal of Emerging Technology and Advanced Engineering, Volume 4, Issue 2, January 2014.
[6] Ying Liu, Wei-keng Liao, Alok Choudhary, A Fast High Utility Itemsets Mining Algorithm, UBDM '05, August 21, 2005, Chicago, Illinois, USA.
[7] Frequent Itemset Mining Dataset Repository. http:// fimi.ua.ac.be/, 2012.
[8] Dr. T. Christopher, Character-based Weighted Support threshold Algorithm using Multi-criteria decision-making technique, International Journal on Computer Science and Engineering, Vol. 02, No. 04, 2010, pp. 965-971.
[9] Ms. Jyoti Ashokkumar Aidale, An Improved Up-Growth Algorithm for Mining High Utility Itemsets from Transactional Databases, International Journal of Scientific & Engineering Research, Volume 5, Issue 5, May 2014.
[10] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Proc. Int'l Conf. Very Large Data Bases, pages 487–499, 1994.
[11] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong and Y.-K. Lee. Efficient tree structures for high utility pattern mining in incremental databases. IEEE Transactions on Knowledge and Data Engineering, 21(12):1708–1721, 2009.
[12] B. Barber and H. J. Hamilton. Extracting share frequent itemsets with infrequent subsets. Data Mining and Knowledge Discovery, 7(2):153–185, 2003.
[13] A. Ceglar and J. F. Roddick. Association mining. ACM Computing Surveys, 38(2), 2006.
[14] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery, 15(1):55–86, 2007.
[15] "Data mining Concepts and Techniques" by Jiawei Han, Micheline Kamber, Morgan Kaufmann Publishers, 2006.