Research Paper
Volume : 5 | Issue : 8 | Special Issue August-2016 • ISSN No 2277 - 8179

A Survey On Various Methods For High Utility Itemset Mining

Engineering

KEYWORDS: association rule mining; frequent pattern growth; high utility itemset mining; negative item values

Geethu P T — Computer Science and Engineering PG Student, NSS College of Engineering Palakkad, India

Anuraj Mohan — Assistant Professor, Computer Science and Engineering NSS College of Engineering Palakkad, India

ABSTRACT

Data mining concepts and techniques helps in uncovering interesting patterns hidden in large data sets. Frequent pattern mining is an important task in data mining and it find itemsets whose support count is not less than a minimum support count. However in frequent pattern mining the importance of an item is not considered. Every itemset is associated with a value like quantity, profit, cost, or other values which indicates the importance of the itemset in that database and it is called as the utility of that itemset. Utility of an itemset is usually the measure of how useful an itemset is. Utility mining aims at finding such itemsets that have much importance among a set of transactions or itemsets that yields much profit to a business application. High utility itemset mining finds out itemsets whose utility is not less than a minimum utility threshold. With the emergence of the new application, the data processed may be in continuous data streams. Data stream or stream data is where data flow in and out of an observation platform dynamically. Data stream is composed of a continuous set of transactions. Utility mining in data streams produces high utility itemsets which contribute more to the total utility among a set of recent data. These results are very important and have wide range of applications such as in retail marketing, web click stream analysis etc.

I. INTRODUCTION

Data mining aims at extracting higher level hidden information from profusion of raw data. Data mining, which has been used in various data domains aims at discovering interesting knowledge from large amount of data stored in a database, data warehouse or from some other information repositories. Data repositories include relational databases, data warehouses, transactional databases, data streams, World Wide Web and also advanced database systems. Data streams or stream data is where data flow in and out of an observation platform dynamically. Data streams have some unique features which include (i) Huge or possibly infinite volumes of data (ii)Dynamically changing data (iii) Data flowing in and out in a fixed order (iv)Data streams allow only one or a small number of data scans (v) Demanding fast (often real time) response time.

Mining data stream involves the efficient discovery of general patterns and dynamic changes within stream data. Traditional data mining methods have been focused on frequent pattern mining over different kinds of databases such as transactional databases, streaming databases and also to various application domains. These data mining technique were based on support count. The objective of frequent itemset mining is to find a set of items which frequently appears in a transactional database and have a support count (whose value indicates the number of transactions in which that itemset appears) not less than a minimum support count. The restraint of frequent itemset mining is (i) it ignores the number of occurrence of an item within a transaction and (ii) it assumes that all items have the same importance or weight. Therefore the frequency of an itemset is not an adequate indicator of interestingness. In reality the benefit of frequent itemset mining is challenged in most of the research areas(for eg: in retail marketing etc). It has been noticed that in many real time application domains, some infrequent itemsets may contribute more than frequent itemsets. In practice some item or itemsets with low support count in a data set may bring high profit due to their high price or high frequencies inside the transactions. Identifying such itemsets with high profit is more important, but these itemsets are often missed by frequent pattern mining.

The limitations of frequent itemset mining leads researchers towards utility based mining approach which allows the user to better express his or her views concerning the usefulness of itemsets as utility and then find itemsets with utility values higher than a given threshold. The utility of an itemset is the measure of how useful an itemset is. Usually the utility value represents the importance of an itemset which can be measured in terms of profit, cost, quantity or other information depending upon the user preferences. The utility mining is an emerging research area and has a wide range of applications such as in retail marketing, web click-stream analysis, online e-commerce management, finding important pattern in bio medical applications etc. The importance in finding utility of an itemset can be explained with the following real world example. Consider in a real world market database customer A has brought 5 Book, 8 Pen and 4 erasers, Customer B has brought one golden ring and customer C has brought 5 loaf of bread. According to the real world profit value, the profit value of ring is much more than the profit value of other items. Business men get more profit from customer B, even though the selling frequency is only one. Therefore finding high utility pattern is more important than finding only frequent patterns.

Given a data set of transactions, high utility itemset mining find itemsets whose utility value is above a threshold. An itemset is called a high utility itemset (HUI) if its utility is not less than a minimum utility threshold.

An important problem in HUI mining method is that, here the user has to supply the minimum utility threshold. But it is a difficult task for the user to specify a minimum utility threshold. If the threshold is set too low, then a large number of HUIs can be found which is not only time and space consuming but also make it hard to analyze the mining result. Contrary if the threshold had a higher value, then very few or no HUIs will be found, which mean that some interesting pattern will be missed.

A solution to this threshold setting problem is to mine top k HUIs in which the user supplies k, the number of HUI to be returned. A benefit in mining top k pattern is that it is easier for the user to indicate how many patterns he would like to see than specifying a utility threshold. In addition here the number of returned patterns will be under control and it helps to analyze the result easily.

Most researches on high utility itemset focuses on static databases. With the emergence of the new application, the data processed may be in the continuous data streams. A data stream is composed of a continuous set of transactions. HUI mining in stream data have tremendous application in various domains such as retail marketing, e-commerce management etc.

Almost many applications consider the itemsets utility value as positive. But in some cases an itemset may be associated with negative item values. Discovery of HUIs considering negative value produces more prominent results.

The rest of this paper is organized as follows. Section 2 depicts the problem definition in detail. The literature survey is given in section 3. In section 4 we conclude the paper.

II. PROBLEM DEFINITION

Let $I = \{i_1, i_2, i_3 \ldots\ldots i_m\}$ be a set of items and each item $i_j \varepsilon I$ is associated with a positive number called $p(i_j)$, called its external utility. For an item the external utility may be its price, profit or other values according to the user preferences.

Let D be a set of N transactions: $D = \{T_1, T_2, \ldots\ldots, T_n\}$ such that for ¥ T_j D, $T_j = \{(i, q(i,T_j))\}, i \varepsilon I, q(i,T_j)$ is the quantity of item i in transaction T_j and it is called the local utility. For example the local utility of an item represents the number of times the item present in that transaction.

Utility of an item i in a transaction T_j is denoted as $U(i,T_j)$ and it is the product of the local utility and its external utility.

ie, $U(i,T_j) = q(i,T_j) * p(i)$ were $q(i,T_j)$ is the quantity of item in transaction T_j and p(i) is the external utility.

Utility of an itemset X in a transaction T_j is denoted as $U(X, T_j)$ and defined as

$$U(X, T_j) = \Sigma_{i\ \varepsilon\ X}\ U(I, T_j)$$

Utility of an itemset in a data set D of transaction is denoted as $U_D(X)$ and defined as

$$U_D(X) = \Sigma_{x\ \ Tj \wedge Tj\ \varepsilon\ D} \Sigma_{i\ \varepsilon\ X} U(I, T_j).$$

An itemset X is called a high utility itemset (HUI) on a dataset D, if and only if $U_D(X) \geq min_util$ were min_util is called minimum utility threshold.

In a data stream environment, transactions come continuously over time and are usually processed in batches. A Batch B_i consists of transactions arriving continuously in a time period. ie $B_i = \{T_j, T_{j+1}, \ldots\ldots, T_m\}$. A sliding window consist of m most recent batches were m is called the size of window, denoted as winsize. Suppose the first batch in a sliding window is Bi, then the window can be represented as $SWi = \{B_i, B_{i+1}, \ldots\ldots, B_{i+winsize-1}\}$. As a new batch form up in a data stream, and if the sliding window is full, then the window removes its oldest batch and adds the new batch in to the window. The problem here is:

For each sliding window SW_i in a data stream the problem is to find the top k high utility itemsets in SW_i, which is returned in the descending order of their utility value, where k is a positive integer given by the user.

III. LITERATURE SURVEY

High utility itemset (HUI) mining in stream data is an emerging topic and so many new data structures, techniques and algorithms are used for the effective processing of stream data. The two major approaches for frequent pattern mining uses apriori algorithm and a frequent pattern growth approach.

A. Apriori based approaches

Apriori algorithm uses generation and test approach which requires multiple scan of the original database. A set of candidate itemsets will be generated and then these candidate itemsets are tested to determine the high utility itemsets. It proceeds by identifying the frequent individual item in the database and extending them to larger and larger itemsets as long as those itemset appear sufficiently often in database. Several researches had studied the application of apriori based approaches for high utility itemset mining.

In [1] R. Agarwal et al proposed an apriori algorithm for the discovery of association rules between items in a large database of sales transactions. The problem here is to find out association rules which satisfies a user specified minimum support and minimum confidence. The algorithms for discovering large itemsets usually need to make multiple passes over the data. This algorithm generates the candidate itemsets (ck) to be counted in a pass by only using the itemsets found large in the previous pass without considering the transactions in the database. In the second phase the database is scanned and the support of candidates in ck is counted. This step generates association rules from frequent itemsets. Hash trees are used for storing the Candidate itemsets (ck). Each node in a hash tree contains either a list of itemsets or a hash table. Here the number of candidate itemsets generated will be large. The search space will be exponential to the number of items occurring in database.

Y. Liu et al in [2] proposed a two phase algorithm for finding HUIs. This two phase algorithm efficiently prunes down the number of candidates and obtains the complete set of HUIs. A transaction weighted utilization (TWU) and a model called transaction weighted utilization mining is proposed in phase1. A transaction weighted downward closure property is maintained here. Then a level wise search is performed were at each level the combinations of high TWU itemsets are added into the candidate itemset. Phase1 may over estimate some low utility itemsets. In phase2 one extra database scan is performed to filter the overestimated itemsets. This two phase algorithm performs very effectively in terms of computational cost and memory but this algorithm is focused on traditional datasets and is not suited for data streams.

M. Liu et al in [3] proposed a novel structure called utility list and an algorithm called HUI-Miner for mining high utility itemsets from transaction dataset. The utility list stores the utility information about an itemset and also the interested information about whether the itemset should be pruned or not. Instead of generating candidate high utility itemsets, the algorithm HUI-Miner mines high utility itemsets from the utility list. This is a single phase algorithm so there is no need to scan the database multiple times but it is very costly to create the utility list.

Conventional association rules mining cannot satisfy the demands emerging from certain real time applications. In apriori, candidate generation and test method significantly reduces the size of candidate sets, leading to a good performance gain. However it can suffer from two non-trivial costs which are the need to generate a large number of candidate sets and the need to repeatedly scan the database and check a large set of candidates by pattern matching. One solution to all these problems is the frequent pattern growth approach, which mines the complete set of frequently occurring itemsets from a large database without candidate generation.

B. Frequent Pattern Growth based approaches

Frequent Pattern Growth based method adopts divide and conquer strategy as follows: at first the database which represents frequent itemsets is compressed into frequent pattern tree (FP-tree) which retains the itemset association information. Then the compressed database is divided into a set of conditional databases, each of which is associated with a frequent item or pattern fragment and mines each such database separately to get the high utility itemsets. A large database is reduced into a smaller data structure, which stores the crucial information about the itemsets and thus avoids the repeated database scans and also save considerable amount of memory for storing the transactions. The most commonly used data structure used for finding HUI mining are FP-Tree [4], UP Tree [6], HUDS-Tree [11].

Table 1 shows an example database for frequent itemset mining. J. Han et al in [4] proposed Frequent Pattern tree (FP-Tree). In order to mine the frequent patterns, an initial

TABLE I. AN EXAMPLE DATABASE FOR FREQUENT ITEMSET MINING

TID	Items Bought	Frequent Items
100	F, A, C, D, G, I, M, P	F, C, A, M, P
200	A, B, C, F, L, M, O	F, C, A, B, M
300	B, F, H, J, O	F, B
400	B, C, K, S, P	C, P, B
500	A, F, C, E, L, P, M, N	F, C, A, M, P

scan is performed on the transaction database to identify the set of frequent items. The set of frequent items of each transaction is then stored in some compact structure.

[(F, 4) (C, 4) (A, 3) (B, 3) (M, 3) (P, 3)] are the list of frequent items derived during the first scan of database. The tree structure of frequent pattern tree consist of one root node labeled as "root" and a set of item prefix tree as the children of the root. Each node other than the root consists of two fields, item name and count. Item name registers which item is represented by this node and count registers the number of transactions represented by the portion of the path reaching from this node. The scan of the first transaction leads to the construction of the first branch of the tree {(F: 1), (C: 1), (A: 1), (M: 1), (P: 1)}. For the second transaction since the frequent item list (F, C, A, B, M) shares a common prefix, they are incremented by 1 and one new node (B: 1) is created and linked as new child of (A: 2) and another new node (M: 1) is created and linked as the child of (B: 1). Similarly for the following transactions new nodes will be created or the existing nodes will be updated.

Once the FP-Tree is built, a pattern growth approach called FP-Growth is used to mine the complete set of frequent patterns. By applying a pattern growth method costly candidate generation can be avoided, but FP-growth is not able to find high utility itemsets.

For utility mining we have to consider the count of items in each transaction too. Table 2 shows an example database for utility mining. Table 3 shows the utility value of each item in the transaction. Here the utility value represents the profit value of each item.

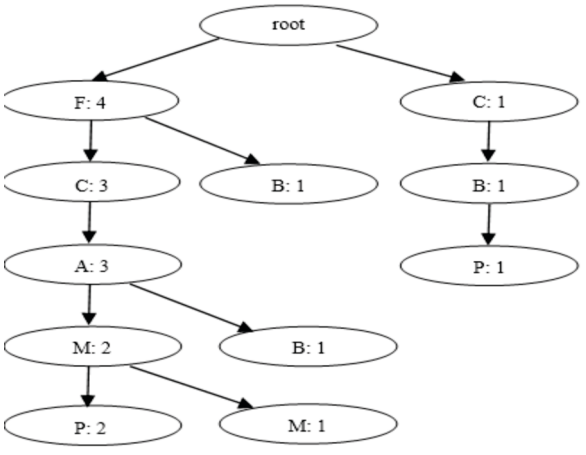


Fig. 1. The FP-Tree

TABLE II.
AN EXAMPLE DATABASE FOR Utility MINING

TID	Transaction	TU
T_1	(P,1)(R,10)(S,1)	17
T_2	(P,2)(R,6)(T,2)(V,5)	27
T_3	(P,2)(Q,2)(S,6)(T,2)(U,1)	37
T_4	(Q,4)(R,13)(S,3)(T,1)	30
T_5	(Q,2)(R,4)(T,1)(V,2)	13
T_6	(P,1)(Q,1)(R,1)(S,1)(W,2)	12

TABLE II. Profit Table

Item	P	Q	R	S	T	U	V	W
Profit	5	2	1	2	3	5	1	1

In [5] C. F. Ahmed et al proposed three novel tree structures to efficiently process incremental and interactive High Utility Pattern (HUP) mining. The first tree structure, Incremental HUP Lexicographic tree ($IHUP_L$-Tree) is arranged according to an items lexicographic order. It can capture the incremental data without any restructuring operation. The second tree structure is Incremental HUP Transaction frequency tree ($IHUP_{TF}$-Tree) , which obtains a compact size by arranging items according to their transaction frequency. The third tree, Incremental HUP Transaction weighted utilization tree ($IHUP_{TWU}$) is based on transaction weighted utility value of item in descending order. Here after scanning each transaction, the algorithm performs the insertion or deletion or modification operation to the tree constructed

Fig 2 shows the IHUP-Tree constructed for the database. Here the minimum utility threshold is taken as 40. Each non-root node in the tree has three fields. First field is the item name and it represents the item in a transaction. The first number beside the item name is the transaction weighted utility (TWU), which is calculated based on the Transaction Utility (TU) given in table 2 and the item utility given in the profit table(table 3) and the second one is its support count.

After the initial tree or any update to the tree is finished, the algorithm asks the user to input the mining threshold. Using the power of "build once mine many" property it can perform several mining operations using different minimum threshold, without rebuilding the tree. Here the user has to specify a minimum utility threshold, which is a difficult task.

V. S. Tseng et al in [6] proposed two novel algorithms as well as a compact data structure for efficiently discovering high utility itemsets from transactional databases. The two algorithms are named as Utility Pattern Growth (UP-Grōwth) and UP-Growth+ and the compact tree structure is called the Utility Pattern Tree (UP-Tree) which is used to maintain the information about the transactions. UP-Tree can be constructed with the two scans of the database. In the first scan Transaction Utility (TU) of each transaction and Transaction Weighted Utility (TWU) of each item is calculated. If an item's TWU is less than minimum utility threshold, then that item and its supersets are considered as unpromising items. During the

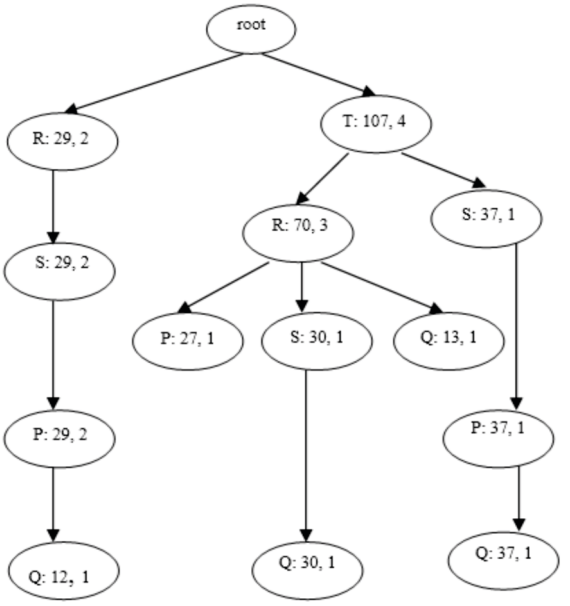


Fig. 2. An IHUP-Tree when min_util=40

second scan of database, transactions are added into a UP-Tree. When considering each transaction the unpromising items should be removed from the transaction and their utilities should also be eliminated from the transaction's TU.

Fig. 3 shows an UP-Tree constructed for transactions in table 2. Each non-root node contains item name, TWU and count. Here in order to reduce the overestimated utility values of an item, two strategies called DLU (Discarding Local Unpromising Items) and DNU (Discarding Local Node Utilities) are applied. After constructing the UP-Tree a two phase algorithm is used for finding high utility itemsets. In phase1 PHUIs (Potential High Utility Itemsets) which is based on the overestimated utilities of itemsets are found. After finding all PHUIs phase2 identify high utility itemsets and their utility value from the set of PHUIs by scanning original database once. The proposed method works well for transactional databases but it cannot be applied for data streams.

H. Li et al in [7] proposed two one pass algorithms for mining high utility itemsets from data streams with in transaction sensitive sliding window. A Transaction Weighted Downward (TWU) closure property is used to mine the set of HUIs. In order to improve the efficiency of mining HUIs, two effective representation of item information, ie, Bit vector and TIDList and an extended lexicographical tree based summary data structure called LexTree-2HTU (Lexicographical Tree with 2-HTU Itemsets), based on item information is constructed. The two algorithms MHUI-BIT (Mining High

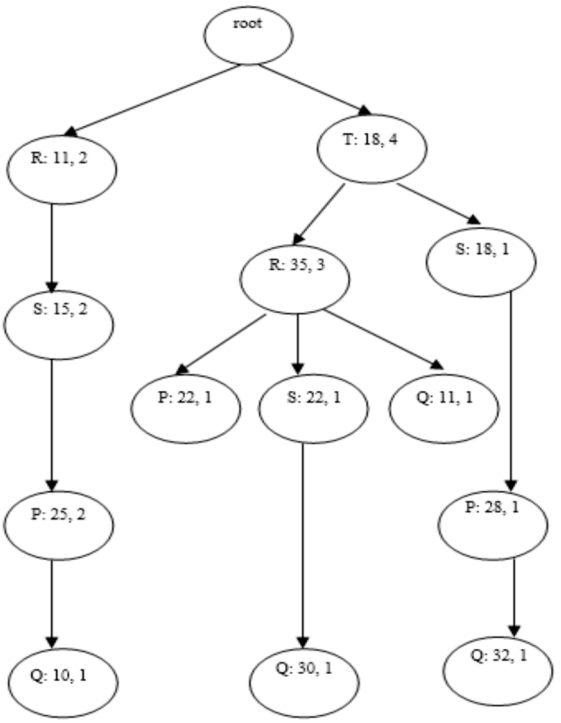


Fig. 3. A UP-Tree by applying strategies DGU and DGN

Utility Itemsets based on BIT vector) and MHUI-TID (Mining High Utility Itemsets based on TID list) are composed of three phases. (i) window initialization phase, which is activated when the number of transactions generated so far in a data stream is less than or equal to user defined window size. In this phase the item information, ie Bit vector and TID list and transaction utility of each transaction within current sliding window are generated. While the sliding window is full LexTree-2HTU is constructed. (ii) The second phase of mining HUI, ie , window sliding phase is activated while the window is full and a new transaction arrives. Two operations, one to update the item information and another one to update the summary data structure is performed in this phase. (iii) In high utility itemset generation phase, the proposed algorithm uses level-wise-method to generate a set of candidate K-HTU-itemset, ck from the previous pre-known (k-1)-HTU-itemsets. Then by using the item information K-HTU-itemsets are generated. The number of candidate itemset generated will be large.

B. shie et al in [8] proposed a novel framework named GUIDE (Generation of maximum high Utility itemsets from Data streams) to find maximal high utility itemsets from data streams with different models. ie, (i) Landmark model (ii) Sliding window model (iii) Time fading model. Landmark model stores the whole data from a specific time point called landmark and find patterns within the data. Sliding window model uses a fixed-sized window which slides with time to keep the data within fixed time or fixed number of transactions. Time fading model captures data from the landmark time to present. However in this model a time delay function is given to decrease the importance of out of date data. The basic idea is to effectively pick up the essential information in data streams and store them into a tree structure called MUI-Tree (Maximum high Utility Itemset Tree). Three methods are proposed for constructing the MUI-Trees for different models of data stream mining. They are $GUIDE_{LM}$ for landmark model, $GUIDE_{SW}$ for sliding window model, $GUIDE_{TM}$ for time fading model. Whenever a user queries for the MaxHUIs (Maximum HUIs), with a specified minimum utility threshold, the MUI Trees are traced and the MaxHUIs are generated. The generation of MaxHUIs consists of two steps. (i)Tracing MUI-Trees and generating HUIs in the nodes and (ii) checking MaxHUIs within the set of HUIs.

C. F. Ahmed et al in [9] defines the problem of sliding window based Incremental HUP mining. Their main contribution is a novel tree structure called HUS-Tree and a new algorithm called HUPMS (High Utility Pattern Mining over Stream data) for incremental and interactive high utility pattern mining over data streams within a sliding window. HUS Tree is constructed to capture the stream data. It arranges the item in lexicographic order. A header table is maintained to keep an item order. Each entry in a header table maintains item id and TWU (Transaction Weighted Utility) value of an item. Item id and batch-by-batch TWU information is stored in each node of the HUDS tree. Once the HUS-Tree is build, HUMPS can take several minimum utility thresholds, min_util and mine the resultant patterns without rebuilding the tree. When the mining procedure receives a pattern and a prefix tree T, it recursively mine all the candidate patterns prefixing . Here items have TWU value less than min_util is deleted and conditional tree of HUS is created. This proposed method generates a large candidate itemset. Although many studies have been developed to HUI mining, it is difficult for uses to choose an appropriate minimum utility threshold. The choice of the threshold greatly influence the output size and also the performance of the algorithms. If the threshold is set too low, too many HUIs will be presented to the users and it is difficult for the user to comprehend the results. On the contrary if the threshold is set too high, no HUI will be found. To find an appropriate value for minimum utility threshold, user need to try different threshold values until getting the needed result. This process is both inconvenient and time consuming. To precisely control the output size and discover the itemsets with the highest utilities without setting the threshold, a promising solution is to refine the task of mining HUIs as mining top-K High Utility Itemsets (top-k HUIs). The idea is to let the user specify k, the number of desired itemsets instead of specifying the minimum utility threshold.

C.W. Wu et al in [10] proposed TKU (top-k utility itemset mining) for mining high utility itemsets without setting min_util. TKU is an extension of UP Growth, for mining high utility itemsets and it adopts the idea of UP-Tree [5] to maintain the information of transaction and top-k HUIs. A UP-Tree is constructed with two scans of the original database. In the first scan, the TU (Transaction Utility) of each transaction and the TWU (Transaction Weighted Utility) of each single item are calculated. Then items are inserted into the

header table in descending order of their TWU's. During the second database scan, transactions are re-organized and then inserted into UP Tree. The TKU algorithm uses an internal variable named Border minimum utility threshold (border_min_util) which is initially set to zero and raised dynamically after a sufficient number of itemsets with higher utilities has been captured during the generation of PTKHUIs (Potential Top-K High Utility Itemsets). Then the exact utilities of PTKHUIs are identified and top-K HUIs are examined by scanning the original database. This proposed method is for transactional databases not for data streams.

M. Zihayat et al in [11] proposed THUDS, to find top-k HUIs in data streams without specifying a minimum utility threshold. THUDS works based on a prefix-tree called HUDS Tree and two auxiliary list of utility values. HUDS Tree dynamically maintains a compressed version of the transaction in a sliding window. The two auxiliary list each maintain a utility list are used to dynamically adjust the minimum utility threshold. THUDS includes three steps: (i) HUDS Tree construction, to construct a HUDS Tree and two auxiliary list, (ii) HUDS Tree Mining, to discover top-k HUIs from the current sliding window, (iii) HUDS Tree Update- once a new batch arrives insert the transactions in the new batch into the tree and remove transactions in the oldest batch, if the sliding window has been filled up and update two auxiliary list. The structure of HUDS Tree is similar to that of FP Tree[4], UP Tree[6], HUS Tree[9]. These trees are used to compress a transactional database into a tree. A non-root node in a tree represents an item in the transactional database and a path from the root to a node compress the transaction that contains the item on the path. Since in a data stream environment transactions come continuously over time, they are usually processed in batches. A Batch B_i consist of transactions arriving continuously in a time period. A sliding window consist of m most recent batches, where m is called the size of the window. Fig. 3. Shows a sliding window whose window size is two. Therefore it contains two batches. Here to construct the HUDS Tree first four transactions in table 2 is considered. Each batch consist of two transactions and the sliding window consist of two most recent batches, B_1 and B_2. Fig. 4 represents a HUDS Tree constructed for the sliding window given in fig. 4.

A non-root node in a HUDS Tree contains the following fields. nodeName, nodeCounts, nodePutils, nodeMtus. nodeName represents the name of the item represented by the node. nodeCounts field is an array with winsize elements, where winsize is the number of batches in the sliding window. Each element in nodeCounts corresponds to a batch in the current sliding window and registers the number of transactions in the batch falling on to the path from the root to node. If X is the itemset represented by the path, the

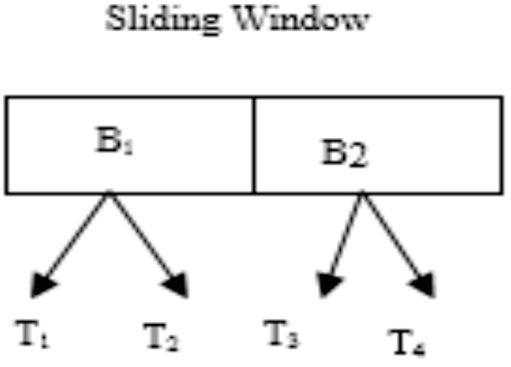


Fig. 4. Sliding window consisting of two batches.

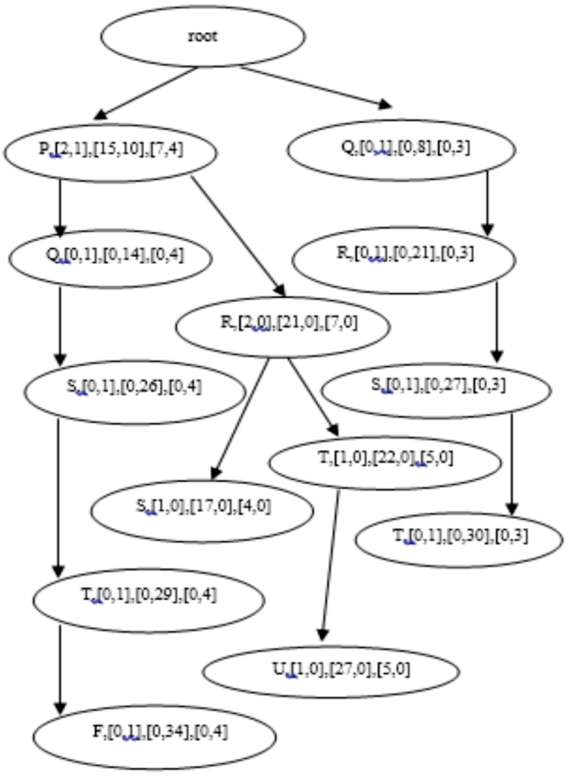


Fig. 4. HUDS Tree Constructed for transactions in table 2

nodePutils field is an array of winsize elements, each corresponding to a batch falling on to the path. Similarly nodeMtus is an array of the minimum transaction utilities (MTU) of X in the transactions falling onto the path for all batches of the sliding window.

After constructing the HUDS tree next step is to mine the top-k HUIS. An efficient method to find the top-k itemsets is first to use an efficient method for finding potential itemsets whose utility is above a threshold. The value of threshold should be such that it should not miss any top-k HUIs and its value must be equal to or close to the utility value of k^{th} item in the top-k HUIs. The threshold is initialized using the maxUtilList. maxUtilList contains a list of utility values corresponding to maximum utility value among a set of items in each level of HUDS Tree. Then the threshold is adjusted using MUIlist and using the mintopkUtil of last window. Given a set of already generated HUIs, MUIlist contains top-k list of MUI values of these HUIs. mintopkUtil of sliding window is the minimum of the utilities of the itemset in top-k HUI set in the previous sliding window. After a HUDS Tree is built or updated for a sliding window, a two phase procedure is used to find top-k HUIs in the sliding window. In first phase the HUDS Tree is mined to generate a set of potential top-k HUIs, that satisfy a dynamically changing minimum utility threshold. The main objective of this phase is to find out as few PTKHUIs as possible while not missing any top-k HUIs. This uses a pattern growth approach and PrefixUtil (an overestimated utility value for an item which is closer to the actual utility value of the item) and TWU are used to prune the search space. In the second phase the exact utilities of PTKHUIs are computed and top-k HUIs are returned.

Algorithm 2 shows the first phase in which PTKHUIs is generated. Here for each entry t in the header table, if its prefix utility is greater than minimum utility threshold then a conditional pattern base is generated by tracing all the prefix path from t. Unpromising items having utility value less than min_util is removed. After that Conditional Header tree is created with the remaining items and algorithm 2 is recursively called to find longer PTKHUIs. Once the PTKHUIs are generated algorithm 1 finds the TopKHUIs by performing an extra database scan to calculate the actual utilities.

Algorithm 1
Input : HUDSTree, Min_Util, K
Output: TopKHUI
1: Generate a set of potential top k HUI (PTKHUI) by calling algorithm 2 with HUDSTree and min_util

2: Scan the transactions in the sliding window to get actual utility, U(X) , for each itemset x in the PTKHUI

3: TopKHUI 0;

4: **for** each itemset X in the PTKHUI **do**

5: **if** U(X) ≥ min_util **then**

6: Insert(X, U(X)) into TopKHUI

7: **if** size of TopKHUI > K **then**

8: Remove last element from TopKHUI,

9: **return** TopKHUI

Algorithm 2
Input : HUDSTree, min_util, K
Output: PTKHUI
1: **for** each entry t in the header table **do**

2: **if** utility(t), U(t) ≥ min_util **then**

3: Generate potential top K HUI (PTKHUI)

4: Cond. Patternbase All prefix path of node t in the tree

5: remove unpromising items having utility value, U(X) < min_util

6: construct conditional HUDSTree and its header table

7: **If** HUDSTreee is not empty **then**

8: Call Algorithm 2 with conditional HUDSTree, min_util, K

9: return PTHKHUI.

In some applications an itemset may be associated with negative item values. C. J. chu et al in [12] proposed HUINIV-Mine High Utility Itemsets with Negative Item Values mining high utility itemsets from large databases, by considering negative item values. HUINIV-Mine is based on the principle of a two phase algorithm. By removing item with negative values from a transaction in a large database to deal with the transaction weighted utilization itemset (TUI) generated. Each item of the itemset that have a negative value will not be a part of HUI. A filtering procedure is then applied which deals with filtering negative itemsets and generating HUIs with negative values from large databases. By applying this concept to data streams top-k HUIs can be mined from the data streams which will produce efficient mining results for various applications.

IV. CONCLUSION

Frequent mining is an important research work, were only the support count of an itemset is considered. In some applications infrequent item will be contributing more to the overall profit. HUI mining discovers such itemset. Adapted versions of apriori and the pattern growth based approach are the two different methods used for HUI discovery. In this paper the various approaches, algorithms, data structures used for high utility itemset mining is analyzed .A major problem in HUI mining is the threshold setting problem , for which top-k HUI mining is a solution, were k is the number of HUIs the user wish to see. By considering the negative profit value associated with an item, the exact utility values can be found.

REFERENCES

[1.] R. Agrawal and R Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Conf. Very Large Data Bases (VLDB), 1994.

[2.] Y. Liu, W. Liao and A. Chowdhury, "A fast high utility itemsets mining algorithm," Proc. Utility Based Data Mining Workshop (UBDM), 2005.

[3.] M. Liu and j. Qu, "Mining High Utility Itemsets without Candidate Generation," ACM International Conference on information and Knowledge Management (CIKM'12), 2012.

[4.] J.Han, J.pei, Y.Yin, R.Mao, "Mining Frequent Pattern Without candidate generation:A Frequent Pattern Tree Approach," Kulwer Academic Publishers. Data Mining and Knowledge Discovery, 8, 53-87, 2004

[5.] C. F. Ahmed, S. K. Tanbeer, B Jeong and Y. Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases," IEEE Trans. Knowledge and Data Eng., vol.21, no.12, Dec 2009.

[6.] V. S. Tseng, B Shie, C. Wu and P. S. Yu, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases," IEEE Trans. Knowledge and Data Eng., vol.25, no.8, Aug 2013

[7.] H. Li, H. Huang, Y. Chen, and Y. Liu and S. Lee, "Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams," Eight IEEE International Conference on Data Mining (ICDM), 2008

[8.] B. Shie, P. S. Yu, V.S. Tseng, "Efficient algorithms for mining maximal high utility itemsets from data streams with different models," *Expert Syst. Appl.* 39 (2012) 12947-12960

[9.] C. F. Ahmed, S. K. Tanbeer, B. Jeong and H. Choi, "Interactive mining of high utility patterns over data streams," Expert Syst. Appl. 39(2012) 11979-11991.

[10.] C. W. Wu, B. Shie, P. S. Yu, V. S. Tseng, "Mining Top-K High Utility Itemsets," *ACM Knowledge Discovery and Data Mining (KDD'12),* 2012

[11.] M. Zihayat and A. An, "Mining top-k high utility patterns over data streams," *Information Science 285 (2014)* 138-161, 2014

[12.] C. Chu, V. S. Tseng and T. Liang, "An efficient algorithm for mining high utility itemsets with negative item values in large databases," Applied Mathematics and Computation 215 (2009) 767-778, 2009