# HI-Tree: Mining High Influence Patterns Using External and Internal Utility Values

Yun Sing Koh[1] and Russel Pears[2]

[1] Department of Computer Science, University of Auckland, New Zealand,
ykoh@cs.auckland.ac.nz,
[2] School of Computing and Mathematical Sciences, AUT University, New Zealand,
rpears@aut.ac.nz

**Abstract.** We propose an efficient algorithm, called HI-Tree, for mining high influence patterns for an incremental dataset. In traditional pattern mining, one would find the complete set of patterns and then apply a post-pruning step to it. The size of the complete mining results is typically prohibitively large, despite the fact that only a small percentage of high utility patterns are interesting. Thus it is inefficient to wait for the mining algorithm to complete and then apply feature selection to post-process the large number of resulting patterns. Instead of generating the complete set of frequent patterns we are able to directly mine patterns with high utility values in an incremental manner. In this paper we propose a novel utility measure called an influence factor using the concepts of external utility and internal utility of an item. The influence factor for an item takes into consideration its connectivity with its neighborhood as well as its importance within a transaction. The measure is especially useful in problem domains utilizing network or interaction characteristics amongst items such as in a social network or web click-stream data. We compared our technique against state of the art incremental mining techniques and show that our technique has better rule generation and runtime performance.

**Keywords:** Prefix-Tree, High Influence Patterns, FP-Growth

## 1 Introduction

The accumulation of massive data sets has increased substantially as the ability to record and store more data has increased rapidly. This has led to the development of various techniques that enable us to obtain valuable information from massive data sets. Although frequent pattern mining [4] plays an important role in data mining applications, there is one major limitation: it treats all items with the same importance. However, in reality, each item in a supermarket for example, has a different importance based on its profit margin. In turn, items having high and low selling frequencies may have low and high profit values, respectively. For example, some frequently sold items such as bread, milk, and chips may have lower profit values compared to that of infrequently sold higher profit value items such as vodka and caviar. This means that finding only frequent patterns in a dataset cannot fulfill the objective of identifying the most valuable customers that contribute to the major part of the total profits in a retail business.

Focusing on high utility items and itemsets has two major benefits: firstly, a smaller set of potentially useful patterns and rules are presented to the user; secondly, the execution time overhead for pattern and rule generation are substantially reduced. Our experimental results clearly illustrate the advantages of the influence factor measure in achieving these two benefits.

A recent approach using a utility mining [11, 5, 7] model was proposed to discover knowledge from a dataset. Using utility mining, several important business objectives such as maximizing revenue or minimizing inventory costs [1] can be considered, and in turn knowledge about itemsets contributing to the majority of the profit can be discovered. Utility mining allows us to represent real world market data. This can be applied in other areas, such as stock tickers, web server logs, network traffic measurements, and data feeds from sensor networks. However, one limitation of these techniques is that we need to have predefined utility values from domain experts for each item within the dataset. This may be impossible if no domain knowledge is available. Moreover, it is difficult for domain experts to constantly update these utility values. Unlike current techniques, we propose a technique that automatically determines the utility value as a dataset evolves incrementally. In situations where there is a lack of domain knowledge, our technique derives an influence factor for each item based on its connections with other items in the dataset. The influence factor measure is based on two factors: the external utility and internal utility of an item. For internal utility, an item that appears multiple times in a single transaction is given a higher internal weighting compared to an item that only appearing once within a transaction. This gives more importance to items that appear frequently within a single transaction over items that appear less frequently. For external utility, we consider the strength of the item based on connections from other items associated with it or co-occurring from it. Such a basis for the derivation of influence factor is intuitive for domains such as social networks and web server logs. In a web server log for an online retailer, we would be interested in identifying how many times a particular product page is viewed in a single session, which will represent internal weighting component. We would also be interested in product pages browsed through together with that particular product, which would account for the external weighting component. Products that are constantly viewed together with co-associated products could be expected to have a major impact on marketing strategies. Most previous work in these areas is based on static datasets and hence do not investigate strategies for constantly changing data. In such dynamic data environments one or more transactions could be deleted, inserted, or modified in the dataset. Using a form of incremental pattern mining, we can use previously mined results, and thereby avoid unnecessary computation when the dataset is updated or the mining threshold used is changed.

Motivated by the limitations existing in previous approaches, we propose an efficient solution to overcome the problems in existing work by proposing a technique to automatically compute the utility of a pattern with the influence factor measure. A further contribution of this research is a new tree structure called the HI-Tree that supports the “build once mine many times” property for high utility pattern mining in an incremental data environment.

The remainder of this paper is organized as follows. In Section 2, we describe related studies. In Section 3, we describe the high influence item problem. In Section 4, we describe our proposed tree structures for high utility pattern mining. In Section 5, our experimental results are presented and analyzed. In Section 6 we conclude the paper and discuss the future research.

## 2 Related Work

In utility itemset mining or utility pattern mining, every item in an itemset is associated with an additional value, an external utility is attached to an item, showing its utility (e.g. profit). With such a utility-based dataset, high utility itemsets (or patterns) are mined, including those satisfying the minimum utility. Mining high utility itemsets is much more challenging than discovering frequent itemsets, because the fundamental downward closure property in frequent itemset mining does not hold in utility itemsets. Several algorithms are available. UMining[11] was proposed for mining high utility patterns, but it cannot extract the complete set. IHUP [1] maintains the high utility patterns in an incremental environment, avoiding multiple scans of the dataset. UP-Growth[10] also uses a tree structure, UP-Tree, to mine high utility patterns. Compared to IHUP, UP-Growth is more efficient, since it further reduces the number of promising patterns which cannot be pruned in IHUP. The Two-Phase [8] algorithm was developed to find high utility itemsets using the downward closure property of Apriori. The authors have defined the transaction-weighted utilization (TWU) property. For the first dataset scan, the algorithm finds all the one-element transaction-weighted utilization itemsets, and based on that result, it generates the candidates for two-element transaction-weighted utilization itemsets. This algorithm suffers from the same problem as the level-wise candidate generation-and-test methodology. The expected utility approach usually overestimates the importance of the itemset, especially in the beginning stages, where the number of candidates approaches the number of all the combinations of items. This trait renders the approach impractical whenever the number of distinct items is large and the utility threshold value is low. Furthermore, updating the utility values is a difficult task in an incremental dataset.

Research into developing techniques for incremental and interactive mining has been carried out in the area of traditional frequent pattern mining, and they have shown that incremental prefix-tree structures such as CanTree [6], CP-tree [9] are quite efficient in using currently available memory. Current efficient dynamic dataset updating algorithms are designed for mining datasets when data deletion is performed frequently in any dataset subset. However, these solutions are not applicable to incremental or interactive high utility pattern mining approaches. Therefore, we propose new tree structures for incremental high utility pattern mining. Our algorithm inserts items into a tree based on the influence factor of the item, and uses the FP-growth mining operation.

## 3 Preliminaries

In this section, we present some background concepts that are used throughout the paper. Moreover, we formalize the definition of influence factor of a particular item.

## 3.1 Online Frequent Itemsets Mining

A transaction data stream $S = \{T_1, T_2, \ldots, T_L\}$ is a continuous sequence of transactions where $L$ is the $tid$ of the latest incoming transaction $T_L$. A transaction sliding window, $SW$, in the data stream is a window that slides forward for every transaction. The window at each slide has a fixed number, $w$, of transactions where $w$ is the size of the window. In the current window $SW_{L-w+1} = \{T_{L-w+1}, T_{L-w+2}, \ldots, T_L\}$, $L - w + 1$ is the window identifier. The support of an itemset $X$ over $SW$, denoted as $supp(X)_{SW}$ is the number of transactions in $SW$ that contains $X$ as a subset. An itemset $X$ is called a frequent itemset if $supp(X)_{SW} \geq minsup$. Given a sliding window $SW$, and a $minsup$, the problem of online mining of frequent itemsets in recent transaction data streams is to mine the set of all frequent itemsets using one scan.

## 3.2 Influence Factor

In our algorithm we use confidence measure of a discovered itemset to measure its influence. Given a high influence item $x$, it may lead to a potentially high impact rule $x \rightarrow y$ whereby the purchase of a high influence (high impact) item $x$ triggers the purchase of another high influence item $y$. We divide the influence factor for an item into two components: internal and external utility values. An item $x$ receives an internal utility value based on the number of times it occurs within a single transaction. Most association rule mining techniques only consider whether an item appears or not within a transaction in a binary sense. The information arising from multiple occurrences of an item within a single transaction is disregarded. For example when generating rules such as a customer who buy bread $\rightarrow$ buy milk we do not consider the quantity of purchase of each of the items, such information may lead to more interesting rules being uncovered. We include this information by including an internal utility component. An item $x$ also receives an external utility value based on the connections between item $x$ and its neighborhood. We measure the external utility of an item $x$ based on the sum of the conditional probability gain from $x$ with the set of items that co-occurred with $x$.

This measure is applicable to many different types of applications and datasets including web click stream data and network data. To analyze these types of data it is necessary to utilize connections between the items whilst considering the importance of the item by itself. Other types of applications that would benefit are social network analysis applications for Twitter where the importance (weight) of a person is measured on the basis on the number of followers re-tweeting the person's messages (number of connections) and the number of tweets produced by a person (individual importance of a person). Thus it is reasonable for us to combine these two factors to generate an influence factor when no domain knowledge can be found or when the data is constantly evolving. For example the number of followers may constantly change and the number of tweets produced may also fluctuate over time. Given a dynamic environment setting a constant utility value would not suffice. Our influence factor allows us to mimic the importance of an item whilst adapting to the network and data changes.

**Definition 1 (Internal Utility).** *The internal utility of an item, $x$, $I(x)$, is given by*

$$I(x) = \frac{1}{|t|} \sum_{\exists x \in t} \frac{|x \in t|}{|F|}$$

*where $t$ a transaction and $t \in SW$, and $F$ is the largest set of an item with multiple occurrences within the same transaction in the neighborhood of $x$.*

For example given that we have three transactions $\{a, a, b, b, b, c\}$, $\{a, b, c\}$, $\{a, c\}$ in $SW$. If we are considering the $I(b)$, $|F|$ is 3 whereby item $b$ occurred at most three times within the SW. The $I(b)$ is $\frac{1}{3}\left(\frac{3}{3} + \frac{1}{3} + \frac{0}{3}\right) = \frac{4}{9}$.

**Definition 2 (Directed Influence Factor).** *The direct Influence factor of an item, $x$*

$$\epsilon_x = I(x) \sum_{y \in N_x} \big( - Pr(xy) \log Pr(y|x) - Pr(x, \neg y) \log Pr(\neg y|x) \big)$$

*where $Pr(x|y)$ is the conditional probability for $x$ and $y$ and $\neg x$ represents the absence of $x$. $I(x)$ is the internal utility for item $x$.*

Here $N_x$ represents the neighborhood of $x$, which is the set of items that occur with $x$. For example, given in the sliding window item $x$ co-occurs with $a$, $b$, $c$, this means that $N_x = \{a, b, c\}$ as shown in Figure 1. Here item $x$ received an external utility value from its neighborhood:

$$E(x) = \sum_{y=\{a,b,c\}} \big( - \Pr(xy) \log \Pr(x|y) - \Pr(\neg xy) \log \Pr(\neg x|y) \big).$$

This particular measure is analogous to information gain. The main reason information gain is not used by itself is it does not consider the internal utility of an item. The directed influence factor would work in directed datasets, where the interactions within the dataset follows a specific directions. For example in web-clicks incoming and outgoing links represents the directionalilty of the interactions.

However there are undirected datasets, where the interactions withn the dataset does not follow a specific direction, such as market basket and retail datasets. This measure does not take into account the directionality of the items which is more representative of the importance of an item within the structure.

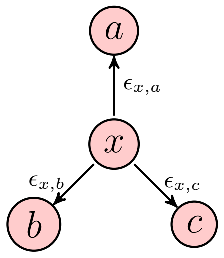
**Definition 3 (Undirected Influence Factor).** *The undirected influence factor of an item, $x$*

$$\epsilon_x = \frac{1}{2} \sum_{y \in N_x} \Big( \big( - Pr(xy) \log Pr(y|x) - Pr(x, \neg y) \log Pr(\neg y|x) \big).I(x) +$$
$$\big( - Pr(xy) \log Pr(x|y) - Pr(\neg x, y) \log Pr(\neg x|y) \big).I(y) \Big)$$

*where $Pr(\neg x|y)$ is the conditional probability for $\neg x$ and $y$.*

Figure 2 shows the undirected Influence. This measure is stricter compared to the directed Influence factor. As per the previous measure a higher $\epsilon_x$ value for an item $x$ represents a higher influence factor.

A high $\epsilon$ value represents a high Influence factor, potentially giving rise to more interesting rules. Instead of using high Influence factors in place of utility scores, we use them to obtain a projection list of how incoming items for a particular transaction will be inserted into a tree structure for mining.



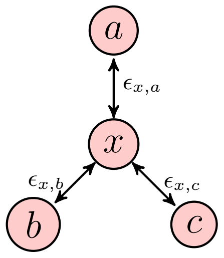


**Fig. 1.** Directed utility for the neighborhood of $x$

**Fig. 2.** Undirected utility for the neighborhood of $x$

## 4 High Influence Tree (HI-Tree)

In our proposed technique, we order incoming transactions based on a projection order which is based on descending $\epsilon$ values for the items within a transaction. We then insert the ordered transaction into a prefix tree which will be mined when needed. The main rationale for inserting transactions into the tree based on this order is that items with higher $\epsilon$ values will have higher influence factor and will lead to more concise rules. Items which are leaves have less impact on rules produced and should not be considered during the mining phase. In this section we will discuss the construction of our High Influence Tree (HI-tree). For incrementally maintaining the patterns in a sliding window, a summary structure called HI-tree was designed. The HI-tree construction has two phases: a tree construction phase followed by a tree mining phase.

### 4.1 HI-Tree Structure

The HI-tree has two parts: a prefix subtree which saves the patterns in the sliding window, and an item-index (header) table which indexes the items in the prefix tree. We also have a structure $\prec S$ which stores the projection order of the items based on decreasing value of influence factor, $\epsilon$.

**Definition 4 (Projection Order).** *Given a current sliding window $SW$ and a set of items $I$ in the sliding window, the projected order of items, $\prec S$, is defined by*

$$\prec S = (s_0, s_1, s_3, \ldots, s_k)$$

*where $s_k$ is an item, $s_k \in I$ for $1 \leq k \leq m$, and $\epsilon_k > \epsilon_{k+1}$.*

In an HI-Tree prefix subtree, an item can be represented as node(s) within the tree.

### 4.2 HI-Tree Construction

The tree construction phase is divided into two parts: insertion of transactions and resorting transactions. In the insertion step, items in a transaction are inserted into the tree based on the current predetermined order called $\prec S$. Given a transaction $T$, a data sequence $S$, called the projection of $T$ is obtained by arranging the items in $T$ according to a predefined order $\prec S$. In order to ensure the completeness of patterns for the data stream, it is necessary to store not only the information related to high Influence items, but also that of low Influence items since they may transit to high Influence status later on. Therefore, when a new transaction $T$ arrives, the completeness information of $T$ is incrementally updated in the HI-tree.

**Table 1.** Example of transactions within a sliding window

| TID | Transactions | TID | Transactions | TID | Transactions |
|---|---|---|---|---|---|
| $t_1$ | {a,b,c,d,e} | $t_4$ | {d,a,b} | $t_7$ | {a,b,d} |
| $t_2$ | {a,b,e} | $t_5$ | {a,c,b} | $t_8$ | {b,a,d} |
| $t_3$ | {b} | $t_6$ | {c,b,a,e} | $t_9$ | {c,e,d, f} |

**Table 2.** Example of projection list for $t_1$ to $t_3$

| TID | Transactions | Projection S |
|---|---|---|
| $t_1$ | {a,b,c,d,e} | {a,b,c,d,e} |
| $t_2$ | {a,b,e} | {a,b,e} |
| $t_3$ | {b} | {b} |

Suppose that the size of the sliding window $SW$ is $N = 9$, and that the window is equally divided into three blocks $(B = 3)$, each containing three transactions. A block is the unit at which re-sorting of transactions take place. For reasons of efficiency, the re-sorting operation does not take place with every new transaction as the change will be minimal and unnecessary overhead will b expended. The items of the transactions in the sliding window at a certain moment in time are shown in Table 1. Assume that all items in the HI-tree are arranged in descending order of the high Influence projection list for the sliding window $SW$. Given a projection list of $\prec S = (a, b, c, d, e)$ where the influence factor of $\epsilon_a > \epsilon_b > \epsilon_c > \epsilon_d > \epsilon_e$ , the projection of each transaction is shown in Table 3.

Figure 3 shows the resulting tree after transactions $t_1$ to $t_3$ are added. The main difference between our technique and previous techniques is that each transaction is sorted based on the order of the projected order $S$ before insertion into the tree.

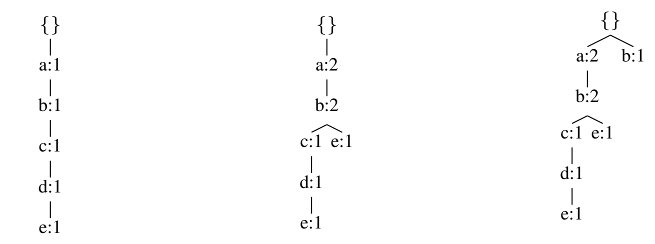


**Fig. 3.** Insertion of $t_1$ to $t_3$

In the re-sorting step, we update the projection order list and re-sort the tree based on the latest projection order list. Subsequent transactions are inserted according to the new projection order. Given a projection list of $\prec S = a, b, e, c, d$ where the influence factor of $\epsilon_a > \epsilon_b > \epsilon_e > \epsilon_c > \epsilon_d$; the projection of each transaction is also shown in the Table 3.

**Table 3.** Example of projection list for $t_4$ to $t_6$

| TID | Transactions | Projection S |
|---|---|---|
| $t_4$ | {d,a,b} | {a,b,d} |
| $t_5$ | {a,c,b} | {a,c,b} |
| $t_6$ | {c,b,a,e} | {,a,b,e,c} |

**Table 4.** Example of projection list for $t_7$ to $t_9$

| TID | Transactions | Projection S |
|---|---|---|
| $t_7$ | {a,b,d} | {a,b,d} |
| $t_8$ | {b,a,d} | {a,b,d} |
| $t_9$ | {c,e,d,f} | {d,e,c,f} |

In the re-sorting step for transactions $t_7$ to $t_9$ we update the projection order list and re-sort the tree based on the latest projection order list. Subsequent transactions are inserted according to the new projection order. Given a projection list of $\prec S = a, b, d, e, c$ where the Influence factor of $\epsilon_a > \epsilon_b > \epsilon_d > \epsilon_e > \epsilon_c$. The projection of each transaction is shown in the Table 4.

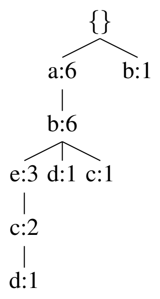


**Fig. 4.** Insertion of $t_4$ to $t_6$

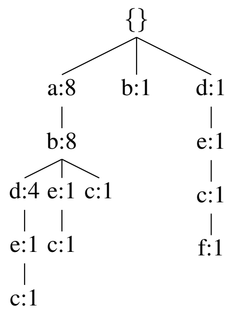


**Fig. 5.** Insertion of $t_7$ to $t_9$

## 4.3 HI-Tree Mining

The tree mining phase follows the FP-Growth mining technique. Once the HI-tree is constructed we use FP-growth to mine patterns. FP-Growth is used in the mining phase in CP-Tree. Instead of using minimum support threshold as seen in previous research, we push two different constraints into our algorithm. We propose two different approaches, each of which has its own benefit.

In the HI-Tree, the influence factor of any item is no smaller than those of its descendants. When we incrementally maintain the transactions, the less influential are also stored in the tree to ensure completeness of the patterns within a stream. Additionally as the sliding window moves, new transactions enter as the oldest transactions are removed. The transaction that represents the old transactions may cause errors in the mining results. Therefore after the current data segment has been processed and before a new data segment arrives, the insignificant patterns in the HI-tree should be deleted by performing a pruning operation. Keeping obsolete and low influence not only degrades performance, it also causes errors in the mining results.

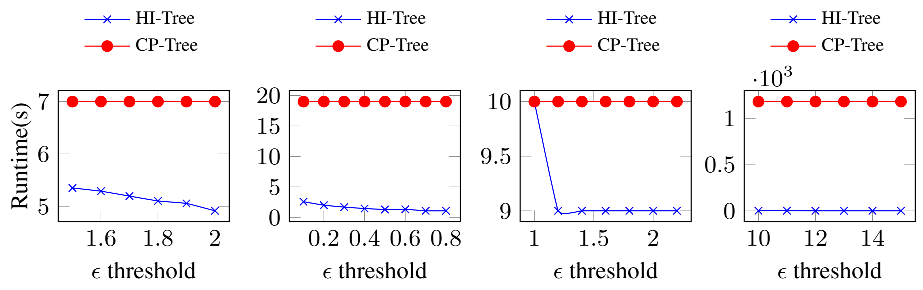
If the influence factor, $\epsilon$, of an item in the projection order list is less than a minimum $\epsilon_{min}$ threshold, then it is redundant and its items are not considered in the prefix tree. The rationale for this is that a low influence would not contribute to a strong rule, thus excluding items with low influence would provide us with a set of strong influential patterns.

## 5 Experimental Evaluation

All experiments were performed on a Intel Core i7 PC with 1.9GB of memory, running Windows 7. The experiments were performed on real world datasets. We compared the performance of HI-Tree using undirected Influence factor on the 4 datasets while varying $\epsilon_{min}$. In the initial experiments we compare the performance of our approach with the CP-tree approach [9] on runtime and size of rule base produced. Due to space constraints we show the overall end result to process the entire dataset as opposed to each individual block. The datasets used in our experiments were from both the UCI [2] and FIMI [3] repository.

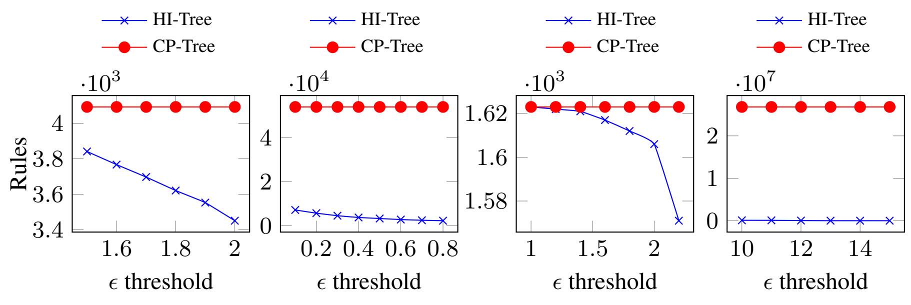
### 5.1 Varying minimum $\epsilon_{min}$ threshold

Figures 6 and 7 show the execution time and number of rules produced for each of the two approaches, HI-tree and CP-tree. For the BMS-POS dataset the window size was set to 50,000 and $\epsilon_{min}$ ranged from 1.2 to 2.0.



(a) BMS POS (b) Retail (c) Kosarak (d) Mushroom

**Fig. 6.** Runtime Performance



(a) BMS POS (b) Retail (c) Kosarak (d) Mushroom

**Fig. 7.** Rules Produced

For the Retail dataset the window size was set to 10,000 and $\epsilon_{min}$ ranged from 0.2 to 0.8. The dataset contained information on 5,133 customers who have purchased at least one product in the supermarket. Over the entire data collection period, the supermarket store carries 16,470 unique items. For the Kosarak dataset the window size was set to 100,000 and $\epsilon_{min}$ ranged from 1.0 to 2.2. The Kosarak dataset contains 990,000 sequences of click-stream data from an Hungarian news portal. For the Mushroom dataset the window size was set to 1,600 and $\epsilon_{min}$ ranged from 10.0 to 15.0. This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms with 8124 instances. Overall the runtime performance using our undirected Influence measure was far superior to that of CP-Tree using similar minimum support and confidence thresholds. In the next section we compute reduction ratios on the number of rules generated of the HI-tree approach in relation to the CP-tree approach.

## 5.2 Reduction Ratio

We show the implication of $\epsilon_{min}$ threshold using the reduction ratio of the rules produced as compared to the traditional incremental mining (CP-Tree) technique. This experiment is set to investigate the number of rules produced when undirected Influence measure is used on the datasets above. The reduction ratio in this case describes the ratio between the number of rules produced by CP-Tree versus the number of rules produced by HI-tree. Table 5 shows the number of rules generated by our algorithm

**Table 5.** Reduction Ratio

| | | | | | | |
|---|---|---|---|---|---|---|
| $\epsilon_{min}$ | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 2.0 |
| BMS-POS | 1.1 | 1.1 | 1.1 | 1.1 | 1.2 | 1.2 |
| $\epsilon_{min}$ | 0.1 | 0.2 | 0.3 | 0.4 | 0.6 | 0.8 |
| Retail | 7.5 | 9.5 | 11.9 | 14.4 | 19.2 | 23.7 |
| $\epsilon_{min}$ | 10.0 | 11.0 | 12.0 | 13.0 | 14.0 | 15.0 |
| Mushroom | 280.3 | 312.0 | 793.8 | 7620.1 | 9253.0 | 79807.4 |
| $\epsilon_{min}$ | 1.0 | 1.0 | 1.4 | 1.6 | 1.8 | 2.0 |
| Kosarak | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |

as compared to CP-Tree technique. From this table, it can be seen that when $\epsilon_{min}$ increases the reduction ratio also increases and vice versa. We notice that the ratio of rules produced varies between 1:1 to 1:79807 rules in the four datasets. A large reduction ratio means that the amount of rules produced by our technique is proportionally fewer as compared to CP-Tree. We do not always assume that a larger reduction ratio would be better for all cases; it only signifies that there were redundant rules produced by a traditional technique in such cases.

## 5.3 Scalability Test and Rule Validation

To test the scalability of HI-Tree by varying the number of transactions, we used the Kosarak dataset, since it is a large sparse dataset with a large number of distinct items and transactions. We divided this dataset into blocks, each containing 100,000 transactions. Then we tested the performance on Kosarak by mining incrementally with the current block accumulated with previous blocks and setting $\epsilon_{min} = 1.0$. The time in the y-axis of Figure 8 specifies the total required time with increasing dataset size. Clearly, as the size of the dataset increases, the overall time increases. However, as shown in the figure, HI-Tree showed a linear scalability over the dataset size.

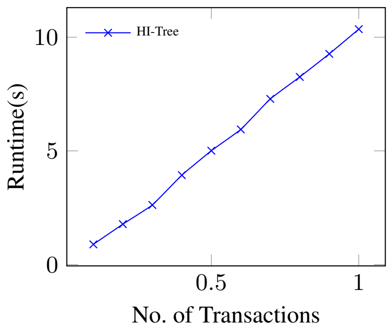


**Fig. 8.** Scalability

**Table 6.** Rule Validation

| Datasets | $\epsilon_{min}$ | Conf | Lift | IGain | Kappa |
|---|---|---|---|---|---|
| BMS-POS | 1.5 | 0.95 | 1.59 | 0.45 | 0.02 |
| | 1.6 | 0.95 | 1.59 | 0.45 | 0.02 |
| | 1.7 | 0.95 | 1.58 | 0.45 | 0.02 |
| | 1.8 | 0.95 | 1.58 | 0.45 | 0.02 |
| | 1.9 | 0.95 | 1.58 | 0.45 | 0.02 |
| | 2.0 | 0.95 | 1.58 | 0.44 | 0.02 |
| Kosarak | 1.0 | 0.96 | 3.13 | 0.81 | 0.06 |
| | 1.2 | 0.96 | 3.13 | 0.81 | 0.06 |
| | 1.5 | 0.96 | 3.13 | 0.81 | 0.06 |
| | 1.8 | 0.96 | 3.13 | 0.81 | 0.06 |
| | 2.0 | 0.96 | 3.09 | 0.80 | 0.06 |
| Mushroom | 10 | 0.97 | 1.22 | 0.19 | 0.71 |
| | 11 | 0.97 | 1.21 | 0.19 | 0.69 |
| | 12 | 0.97 | 1.13 | 0.12 | 0.59 |
| | 13 | 0.97 | 1.07 | 0.07 | 0.43 |
| | 14 | 0.97 | 1.06 | 0.06 | 0.40 |
| | 15 | 0.97 | 1.08 | 0.07 | 0.40 |
| Retail | 0.1 | 0.95 | 13.97 | 1.10 | 0.03 |
| | 0.2 | 0.96 | 9.41 | 1.06 | 0.02 |
| | 0.4 | 0.96 | 3.26 | 0.98 | 0.01 |
| | 0.6 | 0.96 | 3.30 | 1.01 | 0.01 |
| | 0.8 | 0.96 | 3.52 | 1.07 | 0.02 |

To assess the validity of the rules produced we examined the average confidence, Lift, Information Gain and Kappa value of rules produced on the four datasets, as shown in Table 6. Based on the results from these objective measures, all of the measures agree that the set of rules generated by our technique is interesting. A sample of the top ten rules produced by HI-Tree and CP-Tree in Table 7. Overall, we observe that the average size of the precedent terms of rules produced by HI-Tree is shorter, thus making the rules more actionable, in general.

## 6 Conclusion and Future Work

In this research we proposed the HI-Tree approach, an incremental approach for pattern mining of influential rules. We developed two measures that automatically estimated a utility value for items using the both the internal and external characteristics of an item. Our experimental results show that potentially high influence rules can be efficiently generated from the HI-Tree with a single scan of the dataset. Our experimentation on real world data revealed that mining performance is enhanced significantly since both the search space and the number of rules generated are effectively reduced by the proposed strategies.

In the future we plan to incorporate an adaptive sliding window size in our approach. By adapting the sliding window size we have greater control over changes in the computation of influence factor measure as well as the ability to react to changes faster when the changes have occurred within the underlying data stream. We will also consider the effects of drift, which will allow us to analyze items that have had rapid fluctuations in their influence factor values.

**Table 7.** Sample of top 10 rules from CP-Tree and HI-Tree

| **CP-Tree (Top 10 rules from block 1)** |
|---|
| gill-spacing=c stalk-color-above-ring=w stalk-color-below-ring=w $\rightarrow$ stalk-shape=e |
| stalk-shape=e $\rightarrow$ gill-spacing=c stalk-color-above-ring=w stalk-color-below-ring=w |
| bruises=t stalk-shape=e $\rightarrow$ gill-spacing=c stalk-color-above-ring=w stalk-color-below-ring=w |
| gill-spacing=c ring-type=p stalk-color-above-ring=w stalk-color-below-ring=w $\rightarrow$ stalk-shape=e |
| gill-spacing=c stalk-color-above-ring=w stalk-color-below-ring=w $\rightarrow$ ring-type=p stalk-shape=e |
| ring-type=p stalk-shape=e $\rightarrow$ gill-spacing=c stalk-color-above-ring=w stalk-color-below-ring=w |
| stalk-shape=e $\rightarrow$ gill-spacing=c ring-type=p stalk-color-above-ring=w stalk-color-below-ring=w |
| bruises=t ring-type=p stalk-shape=e $\rightarrow$ gill-spacing=c stalk-color-above-ring=w stalk-color-below-ring=w |
| bruises=t stalk-shape=e $\rightarrow$ gill-spacing=c ring-type=p stalk-color-above-ring=w stalk-color-below-ring=w |
| gill-spacing=c stalk-color-above-ring=w stalk-color-below-ring=w stalk-surface-above-ring=s $\rightarrow$ stalk-shape=e |
| **HI-Tree (Top 10 rules from block 1)** |
| ring-type=p $\rightarrow$ stalk-surface-above-ring=s |
| gill-attachment=f ring-type=p $\rightarrow$ stalk-surface-above-ring=s |
| ring-type=p $\rightarrow$ gill-attachment=f stalk-surface-above-ring=s |
| ring-type=p veil-type=p $\rightarrow$ stalk-surface-above-ring=s |
| ring-type=p $\rightarrow$ stalk-surface-above-ring=s veil-type=p |
| ring-type=p veil-color=w $\rightarrow$ stalk-surface-above-ring=s |
| ring-type=p $\rightarrow$ stalk-surface-above-ring=s veil-color=w |
| ring-number=o ring-type=p $\rightarrow$ stalk-surface-above-ring=s |
| ring-type=p $\rightarrow$ ring-number=o stalk-surface-above-ring=s |

## References

1. Ahmed, C., Tanbeer, S., Jeong, B.S., Lee, Y.K.: Efficient tree structures for high utility pattern mining in incremental databases. Knowledge and Data Engineering, IEEE Transactions on 21(12), 1708–1721 (2009)
2. Bache, K., Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`
3. Goethals, B.: Frequent itemset mining dataset repository (2013), `http://fimi.ua.ac.be/data/`
4. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Min. Knowl. Discov. 8(1), 53–87 (Jan 2004)
5. Lan, G.C., Hong, T.P., Tseng, V.S., Wang, S.L.: Applying the maximum utility measure in high utility sequential pattern mining. Expert Syst. with App. 41(11), 5071 – 5081 (2014)
6. Leung, C.K.S., Khan, Q.I., Li, Z., Hoque, T.: Cantree: a canonical-order tree for incremental frequent-pattern mining. Knowl. Inf. Syst. 11(3), 287–311 (Apr 2007)
7. Lin, C.W., Hong, T.P., Lan, G.C., Wong, J.W., Lin, W.Y.: Incrementally mining high utility patterns based on pre-large concept. Applied Intelligence 40(2), 343–357 (2014)
8. Liu, Y., Liao, W.k., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Proceedings of the 9th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining. pp. 689–695. Springer-Verlag, Berlin, Heidelberg (2005)
9. Tanbeer, S.K., Ahmed, C.F., Jeong, B.S., Lee, Y.K.: Cp-tree: a tree structure for single-pass frequent pattern mining. In: Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining. pp. 1022–1027. Springer-Verlag (2008)
10. Tseng, V.S., Wu, C.W., Shie, B.E., Yu, P.S.: Up-growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 253–262. ACM, New York, NY, USA (2010)
11. Yao, H., Hamilton, H.J.: Mining itemset utilities from transaction databases. Data Knowl. Eng. 59(3), 603–626 (Dec 2006)