Efficient Mining of Weighted Frequent Patterns Over Data Streams

Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong
Database Lab, Department of Computer Engineering, Kyung Hee University
1 Seochon-dong, Kihung-ku, Youngin-si, Kyunggi-do, 446-701, Republic of Korea
{farhan, tanbeer, jeong}@khu.ac.kr

Abstract

By considering different weights of the items, weighted frequent pattern (WFP) mining can discover more important knowledge compared to traditional frequent pattern mining. Therefore, WFP mining becomes an important research issue in data mining and knowledge discovery area. However, the existing algorithms cannot be applied for stream data mining because they require multiple database scans. Moreover, they cannot extract the recent change of knowledge in a data stream adaptively. In this paper, we propose a sliding window based novel technique WFPMDs (Weighted Frequent Pattern Mining over Data Streams) using a single scan of data stream to discover important knowledge form the recent data elements. Extensive performance analyses show that our technique is very efficient for WFP mining over data streams.

1. Introduction

Even though frequent pattern mining [1, 2, 9, 11, 27] plays an important role in data mining and knowledge discovery techniques, it does not consider different semantic significances (weights) of the items. Weighted frequent pattern mining [3, 4, 5, 6, 7, 8, 26] is proposed to discover more important knowledge considering different weights of each item which plays an important role in the real world scenarios. For example, in a real world business database, frequency of gold ring is very low compared to the frequency of pen sold. Hence, knowledge about the patterns having low frequency but high weight remains hidden by finding only frequent patterns. The main contribution of the weighted frequent pattern mining is to retrieve this hidden knowledge from databases.

However, the existing weighted frequent pattern mining research works need multiple database scans to find out the weighted frequent patterns and therefore not eligible for real time data processing like data streams [12, 13, 14, 15]. A data stream, where data flows in the form of continuous stream, is a continuous, unbounded and ordered sequence of items that arrive in order of time. To find weighted frequent patterns from data streams, we no longer have the luxury of performing multiple data scans. Once the streams flow through, we lose them. However, old information may become obsolete and recent information may become important in a data stream. Hence, single-pass and sliding window [15, 16, 17, 18, 19] based mechanism is required to find out recent important knowledge from a data stream. In recent years, many applications generate data streams in real time, such as sensor data generated from sensor networks, transaction flows in retail chains, web record and click streams in web applications, performance measurement in network monitoring and traffic management, call records in telecommunications, and so on.

Motivated by these real world scenarios, in this paper, we propose a sliding window based novel technique WFPMDs (Weighted Frequent Pattern Mining over Data Streams). It can discover useful recent knowledge from a data stream by using a single scan. Our technique exploits a pattern growth mining approach to avoid the level-wise candidate generation-and-test problem. Besides retail market data, our technique can be well applied for the area of mining weighted web path traversal patterns. By considering different importance values for different websites, our algorithm can discover very important knowledge about weighted frequent web path traversals in real time using only one scan of data stream. Moreover, it is also useful in the area of bio-medical and DNA data analysis as different biological gene has different importance, therefore, by detecting the combination of weighted gene patterns special gene patterns can be detected for a particular disease and drugs can be made based on that criterion using only one scan of data stream. Other applications areas are telecommunication data, data feeds from sensor networks and stock market data analysis.

The remainder of this paper is organized as follows. In Section 2, we describe background. In Section 3, we develop our proposed technique for weighted frequent pattern mining over data streams. In Section 4, our experimental results are presented and analyzed. Finally, in Section 5, conclusions are drawn.

2. Background

2.1. Frequent pattern mining

The support/frequency of a pattern is the number of transactions containing the pattern in the transaction database.
The problem of frequent pattern mining is to find the complete set of patterns satisfying a minimum support in the transaction database. The downward closure property [1, 2] is used to prune the infrequent patterns. This property tells that if a pattern is infrequent then all of its super patterns must be infrequent. The Apriori [1, 2] algorithm is the initial solution of frequent pattern mining problem. But it suffers from the level-wise candidate generation-and-test problem and needs several database scans. The FP-growth [9] algorithm solved this problem by using FP-tree based solution without any candidate generation and using only two database scans. Other research [10, 11, 25, 27] has been done to efficiently mine frequent patterns. However, this traditional frequent pattern mining considers equal profit/weight for all items.

2.2. Weighted frequent pattern mining

The weight of an item is a non-negative real number which is assigned to reflect the importance of each item in the transaction database [3, 5]. For a set of items, $I = \{i_1, i_2, \ldots, i_n\}$, the weight of a pattern, $P\{x_1, x_2, \ldots, x_m\}$, is given as follows

$$Weight(P) = \frac{\sum_{q=1}^{n} Weight(x_q)}{\text{length}(P)}$$

A weighted support/frequency of a pattern is defined as the value that results from multiplying the pattern’s support with the weight of the pattern [3, 5]. So, the weighted support of a pattern, $P$, is given as follows

$$W\text{support}(P) = Weight(P) \times \text{Support}(P)$$

A pattern is called a weighted frequent pattern if the weighted support of the pattern is greater than or equal to the minimum threshold [3, 5].

In the very beginning some weighted frequent pattern mining algorithms MINWAL [6], WARM [7], WAR [8] have been developed based on the Apriori algorithm [1, 2] using the candidate generation-and-test paradigm. Hence, these algorithms require multiple database scans and result in poor mining performance. WFIM [3] is the first FP-tree based weighted frequent pattern algorithm using two database scans over a static database. They have used a minimum weight and a weight range. Items are given fixed weights randomly from the weight range. It has arranged the FP-tree [9] using weight ascending order and maintained the downward closure property on that tree. The WCloset [26] algorithm is proposed to calculate the closed weighted frequent patterns. To extract the more interesting weighted frequent patterns, the WIP [5] algorithm introduces a new measure of weight-confidence to measure strong weight affinity of a pattern.

The WFIM [3] and WIP [5] algorithms show that the main challenge of weighted frequent pattern mining is that the weighted frequency/support of an itemset (or a pattern) does not have the downward closure property. Consider that item “a” has weight 0.6 and frequency 4, item “b” has weight 0.2 and frequency 5, and itemset “ab” has frequency 3. According to equation (1), the weight of itemset “ab” will be $0.6 + 0.2 = 0.4$, and according to equation (2) its weighted frequency will be $0.4 \times 3 = 1.2$. The weighted frequency of “a” is $0.6 \times 4 = 2.4$, and of “b” is $0.2 \times 5 = 1.0$. If the minimum threshold is 1.2, then pattern “b” is weighted infrequent but “ab” is weighted frequent. WFIM and WIP maintain the downward closure property by multiplying each itemset’s frequency by the maximum weight. In the above example, if item “a” has the maximum weight of 0.6, then by multiplying it with the frequency of item “b”, 30 is obtained. So, pattern “b” is not pruned at this early stage, and pattern “ab” will not be missed. At the final stage, this overestimated pattern “b” will finally be pruned by using its actual weighted frequency.

Some research works [20, 21, 25, 27] have developed single-pass mining algorithms based on traditional frequent pattern mining. Some other mining algorithms [12, 13, 14] have been developed to find out frequent patterns over a data stream in real time. Research has been done to mine recent frequent patterns from data streams using sliding window [15, 16, 17, 18, 19]. However, these algorithms are not applicable for weighted frequent pattern mining.

The existing weighted frequent pattern mining methods need at least two database scans and therefore not suitable for stream data mining. Moreover, they cannot find important knowledge from the recent data. Hence, we propose a sliding window based novel technique for single-pass weighted frequent pattern mining over data streams.

3. Our proposed technique

3.1. Preliminaries

A data stream may have infinite number of transactions. A batch of transactions contains a nonempty set of transactions. Figure 1 shows an example of transaction data stream divided into four batches with equal length. A window can be composed of fixed number of non-overlapping batches. In our example data stream, we consider that one window contains three batches of transactions. Therefore, window1 contains batch1, batch2 and batch3. Similarly window2 contains batch2, batch3 and batch4.

The weighted support of a pattern $P$ can be calculated over a window $W$ by multiplying its support in $W$ with its weight. Therefore, pattern $P$ is weighted frequent in $W$ if its weighted support is greater than or equal to the minimum threshold. For example, if minimum weighted threshold is 2.0, “ab” is a weighted frequent pattern in window2.
frequency in batch2, batch3 and batch4 are 1, 2 and 1 respectively. Accordingly, it has total frequency of 4 in window2. Its weighted support in window2 is \(4 \times 0.55 = 2.2\), which is greater than the minimum weighted threshold.

### 3.2. Tree construction

In this section, we describe the construction process of our tree structure to capture stream data using a single pass. The header table is maintained to keep an item order in our tree structure. Each entry in a header table explicitly maintains item-id, frequency and weight information for each item. However, each node in a tree only maintains item-id and frequency information for each batch. To facilitate the tree traversals adjacent links are also maintained (not shown in the figures for simplicity) in our tree structure.

Consider the example data stream of Figure 1(a). At first we create the header table and keep all the items in weight ascending order. After that, we scan the transactions one by one, sort the items in a transaction according to header table sort order and then insert into the tree. The first transaction \(T_1\) has the items “a”, “b”, “c”, “d”, “g” and “h”. After sorting, the new order is “c”, “d”, “h”, “g”, “b” and “a”.

![Figure 1. Example of a transaction data stream with weight table](image)

![Figure 2. Tree construction for window1](image)
Figure 2(a) shows the tree after inserting batch1. Figure 2(b) shows the tree after inserting batch2. In the same way batch3 is inserted into the tree. Figure 2(c) shows the final tree for window1.

When the data stream moves to batch4, it is necessary to delete the information of batch1 because window2 does not contain it. Therefore, information of batch1 is actually garbage information for window2. We delete the information of batch1 shown in Fig. 3(a). Some nodes do not have any information for batch2 and batch3. As a result, they are deleted from the tree. Other nodes’ frequency counters are shifted one bit left in order to remove the frequency information of batch1 and represent the last frequency information for batch4. As a consequence, the three frequency information of any node represents batch2, batch3 and batch4. Figure 3(b) shows the tree after inserting batch4.

3.3. Mining process

In this section, we describe the mining process of our proposed WFPMDs technique. As discussed in Section 2, the main challenge in this area is the weighted frequency of an itemset does not have the downward closure property and to utilize this property we have to use the global maximum weight. The global maximum weight, denoted by GMAXW, is the maximum weight of all the items in the current window. For example, in Figure 1(b), the item “d” has the global maximum weight 0.6 for window1 and window2.

The local maximum weight, denoted by LMAXW, is needed when we are doing the mining operation for a particular item. As the tree is sorted in weight ascending order, we get advantage in the bottom up mining operation. For example, after mining the weighted frequent patterns prefixing the item “d”, when we go for mining operation prefixing the item “b”, then the item “d” will never come in any conditional trees. As a result, now we can easily assume that the item “b” has the maximum weight. This type of maximum weight in mining process is known as LMAXW. As LMAXW is reducing from bottom to top, the probability of a pattern to be a candidate is also reduced.

Suppose we want to mine the recent weighted frequent patterns in the data stream presented at Figure 1. It means that we have to find all the weighted frequent patterns in window2. Consider the minimum threshold = 1.8. Here the GMAXW = 0.6 and after multiplying the frequency of each item with GMAXW, the weighted frequency list is <c:1.2, f:0.6, d:1.8, h:0.6, g:2.4, e:1.2, b:3.0, a:3.0>. As a result, the candidate items are “d”, “g”, “b” and “a”. Now we construct the conditional trees for these items in a bottom up fashion and mine the weighted frequent patterns.

At first the conditional tree of the bottom-most item “d” (shown in Figure 4(a)) is created by taking all the branches prefixing the item “d” and deleting the nodes containing an item which cannot be a candidate pattern with the item “d”. For item “d”, LMAXW = 0.6 and we can get the weighted frequency list for item “d” by multiplying the other
item’s frequency with $LMAXW$. Obviously this weighted frequency is the maximum possible weighted frequency of an itemset prefixing item “a”. Hence, we have to take all the patterns as a candidate having maximum weighted frequency greater than or equal to minimum threshold. Accordingly, the weighted frequency list for the item “a” is $<d_{1.8}, g_{1.8}, b_{2.4}>$ (we should not consider the global non-candidate items “e”, “f”, “h” and “e”). and candidate patterns “ad”, “ag”, “ab” and “a” are generated here. In the similar fashion, conditional tree for the pattern “ab” is created in Figure 4(b) and candidate patterns “abcdef” and “abg” are generated.

For item “b”, the $LMAXW = 0.5$ as item “a” will not come out here. The weighted frequency list is $<d_{1.5}, g_{2.0}>$. The key point is that, the maximum weighted frequency of item “a” with item “b” is $3 \times 0.5 = 1.5$, as $LMAXW$ reduces from 0.6 to 0.5. Now, without further calculation we can prune “d”. But if $LMAXW$ is 0.6 at this place, the weighted frequency of “d” is $3 \times 0.6 = 1.8$ and as a result it becomes a candidate. This is one advantage of our tree structure. The conditional tree of item “b” contains only one item “g” (shown in Figure 4(c)) and the candidate pattern “bg” is generated. For item “g” the $LMAXW = 0.4$ and the weighted frequency list is $<d_{0.8}>$. As a result, we do not have to create any conditional tree for the item “g”. We have to test all the candidate patterns with their actual weights and the weighted frequency and mine the actual weighted frequent patterns. The actual weighted frequent patterns in window2 are $<a_{3.0}, b_{2.5}, ab_{2.2}, bg_{1.8}>$.

4. Experimental results

To evaluate the performance of our proposed technique, we have performed several experiments on both IBM synthetic datasets ($T104JD100K$, $T401JD100K$) and real-life kosarak dataset from frequent itemset mining repository [24]. Table 1 shows the characteristics of these datasets. These datasets do not provide the weight values of each item. As like the performance evaluation of the previous weight based frequent pattern mining [3, 4, 5, 7, 8, 26] we have generated random numbers for the weight values of the items, ranging from 0.1 to 0.9. Our programs were written in Microsoft Visual C++ 6.0 and run with the Windows XP operating system on a Pentium dual core 2.13 GHz CPU with 1GB main memory.

We show the performance of our technique by using different window and batch sizes in the above mentioned datasets. We have divided the $T104JD100K$ dataset into 10 batches. Therefore, each batch contains 10,000 transactions. We use window size = 3 batches. In a sliding window environment window1 contains the first three batches. Accordingly, window2 contains second, third and fourth batches. The other windows are formed in the same way. Next, we have divided this dataset into 5 batches. Therefore, each batch contains 20,000 transactions. We use window size = 3 batches. Figure 5 shows the comparison curves of these two scenarios with mining operations by using different minimum thresholds. Figure 6 shows the performance curves of WFPMDs in $T401JD100K$ dataset. Here we have used batch size = 10K and 15K and window size = 4 batches. This figure also shows that we have performed mining operations by using different minimum thresholds.

![Figure 4. Mining process](image)

![Figure 5. Performance on the $T104JD100K$ dataset](image)

**Table 1. Dataset characteristics**

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Size (MB)</th>
<th>No. of trans.</th>
<th>No. of distinct items</th>
<th>Avg. trans. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T104JD100K$</td>
<td>5.83</td>
<td>100,000</td>
<td>870</td>
<td>10.1</td>
</tr>
<tr>
<td>$T401JD100K$</td>
<td>14.7</td>
<td>100,000</td>
<td>942</td>
<td>20.61</td>
</tr>
<tr>
<td>kosarak</td>
<td>30.5</td>
<td>990,002</td>
<td>41,270</td>
<td>8.1</td>
</tr>
</tbody>
</table>


The kosarak dataset contains web click-stream data of a Hungarian on-line news portal [24]. As shown in Table 1, it is a big dataset containing almost one million transactions and 41,270 distinct items. Figure 7 shows the effectiveness and efficiency of our technique in kosarak dataset by using different window and batch sizes. Here we have used batch size = 100K, 150K and 200K; window size = 3 batches, and minimum threshold range of 2% to 6%. In figure 7, our technique also shows its scalability by handling around 1 million transactions and 41,270 distinct items.

With modern technology, main memory space is no longer a big concern. In this paper, we made the same realistic assumption as in many studies [15, 20, 21, 22, 23, 25] that we have enough main memory space (in the sense that the trees can fit into the memory). The constructed trees for T10I1D100K (W=3B, B=20K), T40I10D100K (W=4B, B=15K) and kosarak (W=3B, B=200K) datasets require 6.482MB, 28.836MB and 71.265MB memory respectively. Hence, our tree structure can be efficiently applied for the sliding window based single-pass weighted frequent pattern mining using the recently available gigabyte-range memory.

We show the comparison of our technique with the existing WFIM algorithm in the kosarak dataset. The existing WFIM algorithm is not suitable for stream data mining due to scanning a database at least twice. In the first scan, it finds all single-element candidate patterns and in the second scan it performs the tree creation and mining operation. Moreover, it cannot keep batch by batch information in the tree for sliding window-based stream data mining. Therefore, multiple executions of WFIM (i.e. one execution in each window) are needed in order to compare it with WFPMDS. For example, to mine the resultant weighted frequent patterns from window1, WFIM needs to scan window1 twice. Its tree structure is designed to represent the information of window1 with respect to a particular user given minimum threshold. As a consequence, its tree structure cannot be used when the window slides from window1 to window2. Hence, to mine the results in window2, WFIM needs to build its tree structure by scanning window2 twice. Figure 8 shows the runtime comparison between WFIM and WFPMDS algorithms in the kosarak dataset. Window size = 4 batches, batch size = 50K and minimum threshold range of 2% to 6% are used in figure 8. This figure shows that WFPMDS outperforms WFIM significantly with respect to execution time.

5. Conclusions

In this paper, we propose a sliding window based novel technique for weighted frequent pattern mining over data streams. Our main goal is to discover recent weighted frequent patterns from stream data. By using an efficient tree structure, our proposed technique WFPMDS can capture the recent data form a data stream. It requires only a single-pass of data stream for tree construction and mining operations. Therefore, it is quite suitable to apply in real time data processing to discover valuable recent knowledge. Moreover, the tree structure used by our technique is easy to construct and handle. Extensive performance analyses show that our technique is very efficient for weighted frequent pattern mining over data streams.
References


[24] Frequent itemset mining dataset repository. Available at (http://fimi.cs.helsinki.ﬁ/data/)

