International Journal of Engineering & Technology, 7 (4.19) (2018) 1007-1010

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Framework for High Utility Pattern Mining using Dynamically Generated Minimum Support Threshold

Shankar B. Naik[1]*, Jyoti D. Pawar[2]

[1] *Directorate of Higher Education, Goa*
[2] *DCST, Goa University*
**Corresponding author Email: xekhar@rediffmail.com:*

Abstract

In this paper we have proposed a framework which uses high utility itemset mining to store data stream elements in a compressed form and then detect events from the sliding window. This approach promises to reduce the memory requirements when applied to frequent pattern mining in data streams.
In addition to this, a method to dynamically define the value of minimum support threshold based on data in the data stream is presented.

Keywords: *Data mining, high utility itemset, data stream, closed itemset, frequent itemset.*

1. Introduction

With recent advancements in computational devices and software, there is an increasing demand for data mining methods which can handle huge streams of data. A data stream is a huge sequence of data elements which arrive at high rate. A data stream is often generated from sensors, financial markets, social networks, etc[2][3][4][5].
An example of a data stream is the posts which are posted on microblogging websites. The number of active users on microblogging sites is huge and so are the messages posted by them. There are many agencies that are interested in mining knowledge from microblogging data such as the current topic of discussion between users on microblogging websites. Finding answers to such queries is challenging due to the large amount of microblogging data generated online. In this case we consider a data stream as a stream whose elements are messages. Each element of the data stream is an itemset where each item is a word.
In this paper we use an approach to compress the data stream elements by using high utility itemset mining. High utility itemset mining associates a profit value, called as utility, to an itemset. This utility of an itemset could be profit on retail price, frequency of its occurrence, etc.
In this paper we present utility of an itemset in terms of the amount of memory that is saved by replacing the itemset by its compressed form in the sliding window. This approach will reduce the amount of memory required to store a sliding window.
In addition to this, we present a framework to define the value of minimum support threshold from the distribution of itemsets in the data stream dynamically, specifically the sliding window. Replacing the user given fix value of minimum support with a dynamic one has the following two advantages. Firstly, the value is defined based on the data in the sliding window and not merely specified by the user who otherwise has no idea about the distribution of itemsets across the data stream. Secondly, the value changes as the window slides across the data stream. Since the minimum support value is specific for a sliding window, it would actually be most suitable one, in that sliding window, to identify the frequent itemsets. It eliminates the situation of having a constant minimum support value for all sliding window across the entire data stream. Since data streams in terms of their data are dynamic in nature, a dynamic value of minimum support threshold is more suitable.
This paper is organized as follows. Related works has been presented in section 2. The problem is defined in section 3. Section 4 describes the proposed approach. The experimental study is presented in section 5 with section 6 concluding the paper.

2. Related work

High utility itemset mining has been worked upon in [8] and [9]. The approach in [8] compresses the elements using codes generated based on their occurrences in the previous batch. Using codes of a previous batch to compress the elements of the current batch is not suitable. So also, the codes are generated and stored for every batch. The approach in [9] compresses elements by dividing the data stream into same sized batches. It finds frequent patterns for each batch and uses these patterns to compress the elements of the batches. Both the approaches compress the elements of the entire data stream in batches.
In this paper we propose a framework which compresses elements in a sliding window incrementally. By incremental we mean that compression is done while an element enters or leaves the sliding window and not at the end of a batch of elements altogether. The compression of elements is based on the utility of the element in the current sliding window. The approach presented in this chapter is suitable for processing data stream using a sliding window model as it allows more elements to be stored in a sliding window for a fixed memory budget.

3. Problem Definition

3.1. Preliminaries

Let $D = (T_1,T_2,...)$ be a data stream where T_i is an itemset. Let *SW* be a sliding window of size *w* which slide over the data stream *D*. The support of an itemset *X*, denoted as *supp*(*X*), is the number of element containing *X* in the sliding window *SW*.
The idea is to replace the itemsets in elements of the sliding window by links to the itemsets themselves stored as closed itemsets in the intermediate summary data structure. A link to an itemset can be a pointer or a code of the itemset stored in the summary data structure. Not all the itemsets in the sliding window are replaced but the ones which are chosen based their utility in the sliding window. The method to find the utility of an itemset is described in the subsequent subsections.

3.2. Utility of an itemset

The utility of an itemset is the profit offered by an itemset. In this chapter we define utility of an itemset as based on two factors. The first factor is the memory saved by replacing the itemset in the sliding window by a link. This is calculated as the difference between the size of the itemset and the size of the link to the itemset. The second factor is the frequency of occurrence of the itemset in the sliding window, which is the support of the itemset.
Let u_i be the memory size of an item. The memory size of an itemset *X* is given by

$$u(X) = \sum u_i \quad (1)$$

where u_i is the memory size of items in *X*. Let δ be the size of the link which could replace the itemset *X* in the sliding window. The utility of an itemset *X* is given by *U*(*X*) as

$$U(X) = u(X) - \delta \quad (2)$$

where $u(X) - \delta$ is the memory saved by replacing *X* in the sliding window and *u*(*X*) is the memory required to store *X* into the intermediate summary data structure. The term *U*(*X*) multiplied by *sup(X)* reflects the total memory saved by replacement of the itemset *X* in the sliding window. A negative value of utility means the cost of replacement is more and the itemset should not be replaced. If the utility is zero then its a no-profit no-loss situation.

4. Framework for high utility itemset mining using closed itemsets

4.1. The intermediate summary data structure

The intermediate summary data structure is used to store the temporary results generated while processing the data stream. The intermediate summary data structure presented in this paper is a table *ItemList* with four fields- *Id*, *Support*, *ItemSet* and *Utility* as shown in figure 1.

4.2 The approach

The approach uses the algorithms described in [6] and [7] to generated closed itemsets. The algorithm in [7] maitains a list if all closed itemsets in its summary data structure. When an element of ths data stream enters the sliding window the intermediate summary data structure is updated either by inserting the new closed itemsets in the intermediate summary data structure or increasing the supports of the already existing itemsets by one. When the itemset *X* is inserted into or updated in the *ItemList*, the approach calculates *U*(*X*) and stores it in *ItemList* table.

The approach then selects the itemsets to be replaced by the link to themselves in the *ItemList* table.
The selections is based on their utility values. The itemsets with utilities above zero may be replaced.

Data Stream

tid	Itemset
1	{a,b}
2	{a,d,e,f}
3	{a}
4	{b,c}
.	.
.	.

ItemList

Id	ItemSet	Support	Utility

Figure 1: Data stream and the intermediate summary data structure

4.3 Dynamic generation of the minimum support threshold s_0

Most of the approaches use minimum threshold values that are fixed and specified by the users. Irony being that the users have no idea about the distribution of data in the sliding window and are not in a position to specify a proper value of minimum support threshold. This is mostly true in the case of data streams produced by social websites and micro-blogging websites. A lower value of minimum support threshold may lead to a large number of frequent patterns which are trivial. Similarly a high value of minimum support threshold may make frequent patterns escape from the algorithm in showing them as frequent. It is a good idea to provide to the user a suitable value of minimum support threshold based on the data in the sliding window.
In this section we present a method which generates the value of minimum support threshold based on the data in the sliding window itself. The user is presented with three values of minimum support, that are s_0, s_{0+} and s_{0-}, where $s_{0-} \leq s_0 \leq s_{0+}$.

4.3.1 Defining values of s_0, s_{0+} and s_{0-}

The value of s_0 is defined in the following equation as

$$s_0 = \frac{\sum supp(X)}{N}, \forall X \epsilon ItemList \quad (3)$$

where *N* is the total count of itemsets in *ItemList*. The value assigned to s_0 is the average of the supports of all the itemsets in *ItemList*. However, for certain datasets the value of s_0 would be low enough for trivial patterns to become frequent or too high for significant patterns to become frequent. Hence we define two values s_{0-} and s_{0+} as below

$$s_{0-} = s_0 - \sigma \quad (4)$$

and

$$s_{0+} = s_0 + \sigma \quad (5)$$

where σ denotes the standard deviation of the supports of all itemsets present in *ItemList* at that time. s_{0+} is a more strict minimum support threshold.The range $[s_{0-},s_{0+}]$ reflect the variations of supports of different itemsets in *ItemList*.
However, we do not eliminate the impact of the ability of the user to decide the value of minimum support threshold. The values generated by out approach may not reflect the user's mind. Nevertheless, the user can then use his/her discretion to specify the value of minimum support threshold based on the s0,s0− and s0+.

4.4 Detection of frequent events in the sliding window

Each element in the sliding window represents a message posted on a micro-blogging website. It is a set of items which are words. The approach generates closed frequent itemsets from the set of

closed itemsets in the intermediate summary data structure by comparing their supports with the minimum support threshold s_0. These closed frequent itemsets will have the set of words which are mostly occurring in the messages posted on social websites and micro-blogging websites for the period of which elements are present in the sliding window.

5. Experiments

Experiments were separately conducted for high utility mining and frequent pattern mining using dynamic generation of minimum support threshold. All experiments were carried out on 2.26GHz Intel Core i3 PC with 3 GB memory and with Windows 7 operating system. The proposed algorithm is implemented in C++ language. The programs were compiled using GNU GCC compiler.

5.1 High utility mining using closed itemsets

These experiments were carried on synthetic dataset. This dataset was generated using IBM Synthetic Data Generator [2]. Table 1 below contains the details of the dataset.

Table 1: Dataset parameters

Parameters	Value
Total transaction count	200K
Transaction item count (average)	10
Distinct item count	200

5.1.1 Varying sliding window size

The experiment was done by varying the size of sliding window from 1 and 50000 with intervals of 1*K*. The results are shown in figure 2. We observed that there is a significant amount of memory saved using our approach. The amount of memory saved in storing the elements of sliding window is directly proportional to the size of the sliding window.

5.1.2 Sliding the window across the data stream

The experiment was by sliding the sliding window across the data stream by one element. The readings were noted for each sliding window. We observe that there is a significant amount of memory saved using our approach. Also that the amount of memory saved for initial sliding windows is less as compared to the subsequent ones. This is because in the initial sliding windows the approach is in learning process about the utilities of itemsets which are based on their supports. Hence less number of itemsets are replaced (compressed).

5.1.3 Discussion

The approach presented by us need not always be promising high in terms of the amount of memory saved. The results vary according to the kind of data distribution in the dataset. They depend on the size of itemsets as well as on the frequency occurrence of large itmesets in the dataset. For instance, if the frequency of occurrence of large itemsets is high, then the savings on memory would be high.

5.2 Frequent pattern mining using dynamic generation of minimum support threshold

The experiment was performed on real dataset[1]. The dataset is a collection of ball-by-ball commentary of 577 IPL matches. The elements of the datasets are pre-processed to keep information about matchid, batting teams, bowling team, bowler, batsman, non-striker, runs scored, extra run, etc. The total number of elements in the dataset is around 13K.

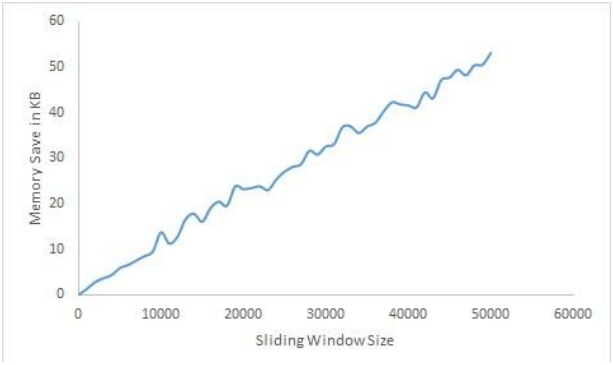

Figure 2: Memory saved against sliding window size

The following are the observations made. The value of s_0 generated using the proposed approach is 3.058415243 while s_{0+} is 6.52430226. A total of 11343 patterns were found to be frequent for the generated value of s_0 and 4565 patterns were found to be frequent for the generated value of s_{0+}. The value of s_{0-} was generated as -0.407471774 in this case.

5.2.1 Discussion

The above results depict very broad analysis on the dataset. The aim of the experiment is to generate frequent patterns in general by considering every value in the dataset element as an item.
Even though the value of s_0 is assigned close to 3 it is to be noticed that the highest support value is 63. However, the method can be further improved upon to have more strict values of s_0.

6. Conclusion

The major contribution in this section is design of an approach that stores element of a sliding window in a compressed form and detects events for the sliding window using the data from summary data structure.
The replacement of the items in a transaction by the link to the itemset in the intermediate summary data structure may generate false positive results. This could be avoided by generating closed frequent itemsets.
This approach also detects events in the sliding window by automatically estimating a suitable value for the minimum support threshold. This estimated value of minimum support threshold gives an idea about the data distribution in the sliding window and helps the user to specify his/her own minimum support threshold value. This approach is useful in detecting events from posts on micro-blogging websites.

References

[1] https://www.kaggle.com.
[2] Rakesh Agrawal, Tomasz Imielin´ski, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.
[3] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM, 2002.
[4] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.
[5] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.
[6] Naik, S. B., & Pawar, J. D. (2013, December). An efficient incremental algorithm to mine closed frequent itemsets over data streams. In *Proceedings of the 19th International Conference on Management of Data* (pp. 117-120). Computer Society of India.
[7] Naik, S. B., & Pawar, J. D. (2015, March). A quick algorithm for incremental mining closed frequent itemsets over data streams.

In *Proceedings of the Second ACM IKDD Conference on Data Sciences* (pp. 126-127). ACM.

[8] Matthijs Van Leeuwen and Arno Siebes. Streamkrimp: Detecting change in data streams. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 672–687. Springer, 2008.

[9] Xintian Yang, Amol Ghoting, Yiye Ruan, and Srinivasan Parthasarathy. A framework for summarizing and analyzing twitter feeds. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 370–378. ACM, 2012.

[10] Naik, S. B., & Pawar, J. D. (2012). Finding frequent item sets from data streams with supports estimated using trends. *Journal of Information and Operations Management*, *3*(1), 153.

[11] Naik, S. B., & Pawar, J. D. (2017, May). Clustering attribute values in transitional data streams. In *Computing, Communication and Automation (ICCCA), 2017 International Conference on* (pp. 58-62). IEEE.

[12] Naik, S. B., & Pawar, J. D. (2017, July). A single-pass algorithm for incremental mining patterns over data streams. In *Intelligent Computing, Instrumentation and Control Technologies (ICICICT), 2017 International Conference on* (pp. 565-569). IEEE.

[13] Naik, S. B., & Pawar, J. D. (2017, June). Mining association rules between values across attributes in data streams. In *Computational Intelligence in Data Science (ICCIDS), 2017 International Conference on* (pp. 1-6). IEEE.