A Review of the Maximal Frequent Itemset Mining Algorithms over Dynamically Changed Data

Haifeng Li[1, a]

[1]School of Information, Central University of Finance and Economics, Beijing, China

[a]mydlhf@139.com

Keywords: maximal frequent itemset, data mining, stream.

Abstract. Maximal frequent itemset mining is a very important method in mining frequent itemsets, which will reduce the mining meory cost and supply a better understanding of the rules generated by the frequent itemsets. In this paper, we review the maximal frequent itemset mining algorithms over a stream, which is an unlimited and dynamically changed data.

Introduction

Frequent Pattern was proposed by Agrawal in 1993[1], when data is high relative or the minimum support is set much lower, massive frequent patterns are generated, the count is even bigger than that of the original transactions; thus, frequent patterns contain redundant information. Given a set of distinct items $\Gamma = \{i1,i2,\ldots,in\}$ where $|\Gamma| = n$ denotes the size of Γ, a subset $X \subseteq \Gamma$ is called an itemset; suppose $|X| = k$, we call X a k-itemset. A concise expression of itemset $X = \{x1,x2,\ldots,xm\}$ is $x1x2\ldots xm$. A database $D = \{T1,T2,\ldots,Tv\}$ is a collection wherein each transaction is a subset of Γ, namely an itemset. Each transaction Ti($i = 1\ldots v$) is related to an id, i.e., the id of Ti is i. The absolute support (AS) of an itemset X, also called the weight of X , is the number of transactions which cover X, denoted $\wedge(X) = \{|T||T \in D \wedge X \subseteq T\}$; the relative support (RS) of an itemset X is the ratio of AS with respect to |D|, denoted $\wedge r(X) = \wedge(X)/|D|$. Given a relative minimum support λ ($0 \leqslant \lambda \leqslant 1$), itemset X is frequent if $\wedge r(X) \geqslant \lambda$. Table 1 is a simple database.

Researchers began to find the condense representations of frequent patterns for data concision. The maximal frequent pattern is the most effective representation since the count of maximal frequent patterns is much smaller when the minimum support is low, which can efficiently reduce the computing cost and storage cost; furthermore, they are easier to understand for users. A maximal itemset is a largest itemset in a database D, that is, it is not covered by other itemsets. A maximal frequent itemset is both maximal and frequent in D, i.e., given an relative support λ, an itemset X is maximal frequent itemset if $\wedge r(X) \geqslant \lambda \wedge !Y|Y \supset X$.

Table 1 Simple Database

ID	Itemsets
1	a b c d e
2	a b c d
3	b e
4	c d e

Data Structures

Vertical Data Format Given the distinct items set $Q=\{i_1,i_2,\ldots,i_m\}$, traditional dataset D is composed of transactions with ids. Each transaction is an itemset T in Q, that is, $D=\{(id_1,T_1),(id_2,T_2),\ldots,(id_n,T_n)\}$, which is called the horizontal data format. The data format in Table 1 is horizontal data format, in which a support is computed by scanning all the dataset, which results in a huge runtime cost. Consequently, the dataset can be converted to another format, that is, each item corresponds to the id collection in which each transaction covers it, i.e., $Dv=\{(i_1,idlist_1),(i_2,idlist_2),\ldots,(i_m,idlist_m)\}$, which is called the vertical data format. Vertical data format

can quickly compute the support according to the idlist of a pattern, i.e., for a pattern $X=\{i_1,i_2,\ldots,i_x\}$, its idlist is computed by $idlist_1 \wedge idlist_2 \wedge \ldots \wedge idlist_x$, and the support of X is the ids count in its idlist. As an instance, in Table 1, $idlist_{\{a\}}=(1,2)$, $idlist_{\{b\}}=(1,2,3,4)$, then $idlist_{\{ab\}}=idlist_{\{a\}} \wedge idlist_{\{b\}}=(1,2)$, as a result, the support of ab is 2.

Since the size of the distinct items set Q is unchanged, a bitmap format can be employed to represent the idlist, consequently, the bits number in the bitmap equals to the size of Q. Therefore, we can build a binary value bit_t for each item i_t. In bit_t, the id_{th} number is 1 if this id is existed in $idlist_t$, otherwise 0. This can compresses the idlists and raise computing efficiency through bitwise operators. For an example in Table 1, each pattern idlist can represented by a binary value with 5 numbers; thus, $bit_{\{a\}}=11000$, $bit_{\{b\}}=11110$, then $bit_{\{ab\}}=bit_{\{a\}} \wedge bit_{\{b\}} = 11000$.

If a tree structure is introduced to build the relationship between sub-pattern and super-pattern, a new structure, diffset, can be used for further pruning. For two patterns X and Y and X in Y, then the transactions covering Y cover X, that is, $idlist_Y$ in $idlist_X$. Generally, $idlist_X \setminus idlist_Y$ is far less than $idlist_Y$; thus, we can only store $idlist_X \setminus idlist_Y$ for pattern Y, which is called the diffset of Y, denoted $diffest_Y$. For an instance, $idlist_{\{a\}}=(1,2)$, since the idlists of ab, ac, ad are all (1,2), the diffsets are all empty; on the other hand, $idlist_{\{ae\}}=(1)$, then $diffset_{\{ae\}}=(2)$.

Both bitmap format and diffset aim to reduce the runtime or storage cost, nevertheless, they have their own weakness. When the dataset is sparse, most of bitmap format data is redundant, which will waste vast storage space; similar, the size of diffset is larger than that of idlist, and the format conversion adds extra computing cost.

Frequent Pattern Tree Frequent pattern tree(FP-tree) is proposed by Han et al, which can efficiently compress the itemsets, this is due to the items sharing; also, the pattern support is computed more quickly; furthermore, a batch processing strategy can be used when memory is not enough to store all patterns. In FP-tree, each node is composed of 5 parts: the node name, node support, node link, child node pointers, parent node pointer; in addition, an items head table, composing of item name, item support, and item link, is employed to effective traverse the FP-tree, in which the item name is sorted in a descending criteria order by the item support. In a FP-tree, only the nodes whose support is higher than the minimum support are stored. Figure 1 is the FP-tree of dataset in Table 1 when the minimum support is 1. In such a data structure, the support of a pattern can be computed by one FP-tree scan. For example, for pattern abcd, its support is computed as follows: The item a is found out from the item head table firstly, then the patterns head with a are obtained by the item link, in Figure 1, the first searched node is a:1, which is as a beginner to traverse end at the root node, that is, the pattern aedcb is obtained, which covers abcd; thus, the temp support of abcd is 1, continue with this method, the pattern between the second node a:1 and the root node is adcb, which also covers abcd; thus, the final support of abcd is 1+1=2.

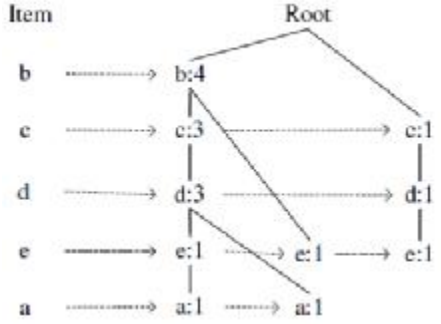

Fig. 1 FP-Tree

Maximal Frequent Itemset Mining Algorithms over Dynamically Changed Data

UMFIA algorithm[2] proposed a FP-tree based maximal frequent pattern updating algorithm. This algorithm proved that the support of the new item generated from the new database was lower than the support of the original item generated from the original database; thus, the new item can be directly inserted into the FP-tree as a leaf node; furthermore, it also analyzed the relation between the maximal frequent itemsets in the original database and the new one, that is, the union of these maximal frequent itemsets are definitely the superset of all the maximal frequent itemsets in the whole database; thus, the initial maximal pattern candidates are not composed of all the distinct itemsets, but the items in the union of items in the original maximal patterns and the new maximal patterns.

IMFI algorithm[3] further presented IMFI algorithm, which not only considered the maximal frequent patterns relationship between the original database and the new one, but also considered the relationship of other patterns. This consideration can reuse the mined information. IMFI classified the patterns into six categories. for X in {D, d, D U d}, where D denotes the original database, d denotes the added database, and D U d denotes the overall database; in addition, MFI_X, F_X, and I_X separately denote sets of the maximal frequent patterns, the frequent patterns and the infrequent patterns. When a pattern q belongs to one of the six categories in D or d, it maybe becomes the maximal frequent pattern in D U d. If q satisfies the first possibility, then it is definitely a maximal frequent pattern in D U d; if q is in the second or the eighth possibility, and it is frequent in D U d, then it is a maximal frequent pattern; otherwise, if q is infrequent and q satisfies one of the rest possibilities, then the subsets of q may be the maximal frequent patterns in D U d. Based on the heuristic rules, IMFI conducted a fast pruning. Further, in this algorithm, the SG-Tree, a data structure similar to R-Tree, was employed to maintain the pattern bitmaps. This structure can quickly locate the patterns; thus, the support computing is speeded up.

DSM-MFI algorithm [4] proposed the DSM-MFI method, in which a batch processing technique is used. The stream data is split into groups for separate mining, and the results are merged. This paper uses the SFI-forest to store the frequent items and frequent pattern synopsis, for each arrived transaction $T=x_1x_2\ldots x_m$, the projection $x_{i+1}x_{i+2}\ldots x_m$ of each item x_i is maintained in the SFI-forest, and the infrequent patterns are pruned to reduce the memory cost. Even though the support of frequent itemsets is not accurate, it can be handled by the specified error parameter. When a user is querying with a threshold, this method summarizes all the data in SFI-forest and compute the maximal frequent itemsets in real time.

estDec+ algorithm estDec+ is an improved algorithm from estDec, also presented by [5]. To guarantee the data synopsis can be stored in memory, the algorithm introduces the approximate argument to combine the neighbor nodes in the compacted prefix tree CP-tree. To conveniently combine, split and rearrange the nodes when new transactions arrive, a node maintains the index of its parent nodes and the maximal possible support and minimal possible support, as well as the combined items.

INSTANCE algorithm [6] proposed a simple but effective algorithms INSTANCE, which only use arrays to store the maximal frequent itemsets, that is, an array u[i] is used to store the itemsets with support i(i is an integer no larger than the minimum support). Once a new transaction T arrives, the existing itemset will be compared to T, if the support is updated, it will be transferred to the array corresponding to its support, and the new covered itemsets are deleted. This algorithm maintains all the maximal frequent itemsets with support lower that the minimum support, as can be seen, the performance will be improved when the minimum support is small.

estMax algorithm [7] introduced the most efficient stream maximal frequent itemset mining method estMax. Along with the increment of stream data, if the relative minimum support is not changed, then the absolute minimum support will become larger, a maximal frequent itemset with an unchanged support will become infrequent. A large number of data check will spend computing cost. To address this problem, this method predict the maximal frequent itemset with the maximal life cycle, that is, to compute the number of arrived transactions which may lead the maximal frequent itemset to

infrequent. The maximal life cycle of an itemset e in the first k transactions $D_{\{k\}}$ can be computed by the ratio of the current support and the minimum support, i.e., $ML_{\{k\}}(e)= \Lambda_{\{k\}}(e)/ \lambda$. When new transaction T_{k+1} arrives, the status of an itemset can be guaranteed by the updated information: If $ML_{\{k\}}(e)>k+1$, and e is maximal frequent before $T_{\{k+1\}}$ arrives, then it is still maximal frequent; for the largest frequent itemset $e_{\{L\}}$ covered by $T_{\{k+1\}}$, if $ML_{\{k\}}(e_{\{L\}})<k+1 <= ML_{\{k+1\}}(e_{\{L\}})$, then it is the new maximal frequent itemset in $D_{\{k+1\}}$.

Max-FISM algorithm Recently, Farzanyar firstly developed a sliding window based algorithm named Max-FISM[8]. In this algorithm, an in-memory prefix-tree Max-Set is used to maintain the data synopsis. When the sliding window continues, the information of out-of-date transaction will be deleted from the Max-Set. In the Max-Set, the nodes are sorted by a novel order, that is, the value of the bitmap that the itemset occurs in the sliding window. If new transaction arrives, the Max-Set will prune some redundant itemsets with its maximum validity time, which is a measure to indicate an itemset will remain frequent without any additional occurrence in the future transactions. When the results need to be output, the support of each itemset will be computed and the maximal frequent itemsets will be stored in a list, in addition, the minimal infrequent itemsets are also stored in a list for further pruning.

Summary

This paper reviewed the state-of-the-art algorithms of mining the maximal frequent itemsets when over the dynamically changed data. As can be seen, an index should always be used in these algorithms, which can significantly improve the performance.

Acknowledgements

This research is supported by the National Natural Science Foundation of China (61100112, 61309030), Beijing Higher Education Young Elite Teacher Project (YETP0987). Key project of National Social Science Foundation of China(13AXW010), 121 of CUFE Talent project Young doctor Development Fund in 2014 (QBJ1427).

References

[1] R. Agrawal, T. Imielinski and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In Proceeding of ACM SIGMOD the International Conference on Management of Data, 1993.

[2] Y.Song, Y.Zhu, Z.Sun and G.Chen. An Algorithm and Its Updating Algorithm Based on FP-tree for Mining Maximal Frequent Itemsets. Journal of Software, Vol.14,No.9, 2003.

[3] W. Lian, D. W. Cheung and S. M. Yiu. Maintenance of Maximal Frequent Itemsets in Large Databases. In Proceeding of SAC the Annual ACM Symposium on Applied Computing, 2007.

[4] H. Li, S. Lee and M. Shan. Approximate mining of maximal frequent itemsets in data streams with differenet window models. In ESWA the Journal of Experts System with Applications, 2008.

[5] D. Lee and W. Lee. Finding Maximal Frequent Itemsets over Online Data Streams Adaptively. In Proceeding of ICDM the IEEE International Conference on Data Mining, 2005.

[6] G. Mao, X. Wu, X. Zhu and G. Chen. Mining Maximal Frequent Itemsets from Data Streams. In JIS the Journal of Information Science, 2007.

[7] H. J. Woo and W. S. Lee. estMax: Tracing maximal frequent item Sets over Online Transactional Data Streams. In TKDE the Transactions on Knowledge and Data Engineering, 2009.

[8] Z.Farzanyar, M.Kangavari and N.Cercone. Max-FISM: Mining maximal frequent itemsets over data streams using the sliding window model. In the Computers and Mathematics with Applications, Vol. 63, 2012.

