SuffixMiner: Efficiently Mining Frequent Itemsets in Data Streams by Suffix-Forest[1]

Lifeng Jia, Chunguang Zhou, Zhe Wang, and Xiujuan Xu

College of Computer Science, Jilin University,
Key Laboratory of Symbol Computation and
Knowledge Engineering of the Ministry of Education, Changchun 130012, China
jia_lifeng@hotmail,.com cgzhou@jlu.edu.cn

Abstract. We proposed a new algorithm SuffixMiner which eliminates the requirement of multiple passes through the data when finding out all frequent itemsets in data streams, takes full advantage of the special property of suffix-tree to avoid generating candidate itemsets and traversing each suffix-tree during the itemset growth, and utilizes a new itemset growth method to mine all frequent itemsets in data streams. Experiment results show that the SuffixMiner algorithm not only has an excellent scalability to mine frequent itemsets over data streams, but also outperforms Apriori and Fp-Growth algorithms.

1 Introduction

A data stream is a continuous, huge, fast changing, rapid, infinite sequence of data elements. The nature of streaming data makes the algorithm which only requires scanning the whole dataset once be devised to support aggregation queries on demand. In addition, this kind of algorithms usually owns a data structure far smaller than the size of whole dataset. As mentioned above, the single scan requirement of streaming data model conflicts with the demand of accurate result. Consequently, an estimation mechanism is proposed in the Lossy Counting algorithm [1] which is based on the well-known Apriori property [2] to harmonize such a conflict. Frequent itemsets in data streams are found when a maximum allowable error ε as well as a minimum support θ is given. The information about the previous mining result is maintained in the lattice which contains a set of entries of the form $(e,f,\triangle)$ where e is an itemset, f is the count of itemset e, and $\triangle$ is the maximum possible error count of itemsets e. For each entry $(e,f,\triangle)$of an itemset e in lattice, if the itemset e is one of the itemsets identified by new transactions in the buffer, its previous count f is incremented by its count in new transactions. Subsequently, if the estimated count, $f+\triangle$, is less than $\varepsilon \cdot N$, such that N is the number of transactions, its entry is pruned from the lattice. On the other hand, when there is no entry in lattice for this new itemset e identified by the new transactions, a new entry $(e,f,\triangle)$is inserted into the lattice.

[1] This work was supported by the Natural Science Foundation of China (Grant No. 60433020) and the Key Science-Technology Project of the National Education Ministry of China (Grant No. 02090).

L. Wang and Y. Jin (Eds.): FSKD 2005, LNAI 3614, pp. 592 – 595, 2005.

Its maximum possible error ε is set to $\varepsilon \cdot N'$ where N' denotes the number of transactions that were processed up to the latest batch operation. Han et al. [3] developed a FP-tree-based algorithm, called FP-stream, to mine frequent itemsets at multiple time granularities by a novel titled-time windows technique.

2 SuffixMiner Algorithm

SuffixMiner is single-scan algorithm which utilizes the suffix-forest and is also batch-processed. As mentioned above, SuffixMiner chooses the suffix-forest to store the summary information of data streams. Now, we begin to explain our newly-devised algorithm step by step.

SuffixMiner Algorithm
Input: Minimal support threshold θ, Maximal estimated possible error ε.
Output: frequent itemsets in lattice.
For each block of transactions in the data stream

1. Construct the suffix-forest to store the summary information of streaming data;
2. Generate frequent itemsets in the current batch of transactions from the suffix-forest by counter sequence and depth first itemset growth method.
3. According to the estimation mechanism, insert the new frequent itemsets into the lattice or update the frequency of old frequent itemsets already in the lattice.
4. Pruning infrequent itemsets from the lattice based on the Apriori property.

Before explaining how SuffixMiner to avoid traversing suffix-trees during the phase of itemset growth, let introduce an important property of suffix-tree first.

Theorem 1: In the suffix-tree(I_i), its sub-tree whose root is I_k (k>i) must be a sub-tree of suffix-tree(I_k).

Proof: In the suffix-tree(I_i), when constructing a branch of sub-tree whose root is I_k, SuffixMiner must deal with a certain suffix-set$\{I_i,...,I_k,...,I_n\}$.Meanwhile, the suffix-set $\{I_k, ...,I_n\}$must appear together with this suffix-set$\{I_i,...,I_k,...,I_n\}$. Thus, SuffixMiner will construct a new branch of suffix-tree(I_k) to store the suffix-set $\{I_k,...,I_n\}$, or insert it into an already existed branch in suffix-tree(I_k) by updating the frequency of nodes in that branch. Consequently, the theorem 1 is proofed.

Based on theorem 1, we draw a conclusion: if a certain node(I_i) of suffix-tree(I_m) has a corresponding node(I_i) in the suffix-tree(I_k), this node(I_i) must have an ancestor node(I_k) in the suffix-tree(I_m). In order to recognize the corresponding identical nodes in different suffix-trees, we need to code all nodes of suffix-tree. SuffixMiner algorithm endows every node of suffix-tree the same serial number as that of it which exists in the corresponding complete suffix-tree. The nodes of complete suffix-tree are coded by the depth first. According to the method for coding the nodes of suffix-forest, in the suffix-tree(I_i), the serial number difference of node(I_n) and its ancestor node(I_n) is equal to the serial number of its corresponding node(I_n) in the suffix-tree(I_m)**.** Consequently, we can use this relationship to identify whether two nodes in different suffix-trees are corresponding nodes.

Theorem 2: In an assumed suffix-tree(I_1) with i kinds of child nodes$\{I_2, \ldots, I_{i+1}\}$, let $f(I_{i+1})$ and $c(I_{i+1})$ denote the frequency and the serial number of node(I_{i+1}) respectively. For each node(I_{i+1}) of suffix-tree(I_1), if it has a corresponding node(I_{i+1}) in suffix-tree(I_k), the serial number of this node(I_{i+1}) is stored in the serial number set $C_k(1<k<i+1)$. For all the nodes(I_{i+1}) of suffix-tree(I_1), let the serial number set $C = C_2 \bigcap \ldots \bigcap C_i$. if $C \neq \varnothing$, the frequency of (i+1)-itemset$\{I_1, \ldots, I_{i+1}\} = \sum f(I_{i+1})$, such that $c(I_{i+1}) \in C$; Otherwise, the frequency of (i+1)-itemset$\{I_1, \ldots, I_{i+1}\} = 0$.

Proof: In order to compute the frequency of (i+1)-itemset$\{I_1, \ldots, I_{i+1}\}$, we need to know all the nodes(I_{i+1}) are the common child nodes of node(I_1),…, node(I_i) in the suffix-tree(I_1). According to the conclusion of theorem 1 and the statement of theorem 2, the set C_k stores serial numbers of nodes(I_{i+1}) which have nodes(I_k) as ancestor nodes. Given the set C, $C = C_2 \bigcap \ldots \bigcap C_i$. So, if there is the occurrence of nodes(I_{i+1}) which are the common child nodes of node(I_1),…, node(I_i), the serial numbers of these nodes(I_{i+1}) must be stored in the set C($C \neq \varnothing$). Subsequently, the frequency of (i+1)-itemset$\{I_1, \ldots, I_{i+1}\}$ is equal to the sum of frequencies of nodes(I_{i+1}), i.e., $\sum f(I_{i+1})$, such that $c(I_{i+1}) \in C$. Otherwise, if the set C is empty($C = \varnothing$), the frequency of (i+1)-itemset$\{I_1, \ldots, I_{i+1}\}$must be equal to zero. Consequently, the theorem 2 is proofed.

Before detailing the counter sequence and depth first itemset growth method, we firstly describe two core operations of it.

Insert Itemset Growth(IIG): When SuffixMiner has already computed the frequency of i-itemset$\{I_1, \ldots, I_m, \ldots, I_k\}$(k>m>i), through the operation of insert itemset growth, SuffixMiner will compute the frequency of (i+1)-itemset$\{I_1, \ldots, I_p, I_m, \ldots, I_k\}$, such that I_p is newly inserted item (i<p<m<k).

Replace Itemset Growth(RIG): When SuffixMiner has already computed the frequency of i-itemset$\{I_1, \ldots, I_m, \ldots, I_k\}$(k>m>i), through the operation of replace itemset growth, SuffixMiner will compute the frequency of i-itemset$\{I_1, \ldots, I_p, \ldots, I_k\}$, such that I_p is newly inserted item to replace the item I_m (i<p<m<k).

We stipulate that the priority of IIG operation is higher than RIG operation. We also stipulate that the sequence of newly-inserted item by both IIG and RIG operations is according to the counter item sequence.

3 Experimental Evaluation

We use synthetic data streams, T5.I4.D1000K and T10.I4D1000K, to simulate the stream environment. The default value of minimum support threshold θ is 0.1%. The maximal estimated possible error threshold ε is a tenth of θ . Results indicate the excellent scalability of SuffixMiner algorithm. We also compare the performance of Apriori and FP-Growth algorithms with that of SuffixMiner algorithm. Since the dataset includes 100 items, so it simulates the environment of dense data stream, and the execution time of SuffixMiner algorithm does not vary a great extent when θ varies. However, SuffixMiner still outperforms both Apriori and Fp-Growth algorithms which can be downloaded in the web: *http://www.cs.umb.edu /~laur/ARtool/*.

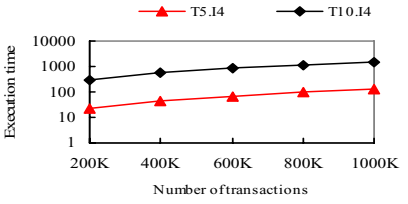

Fig. 1. The performance of SuffixMiner algorithm

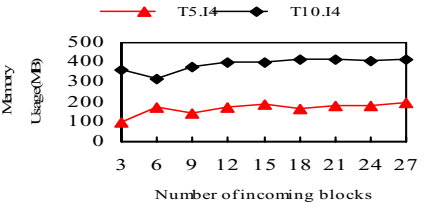

Fig. 2. The performance of SuffixMiner algorithm

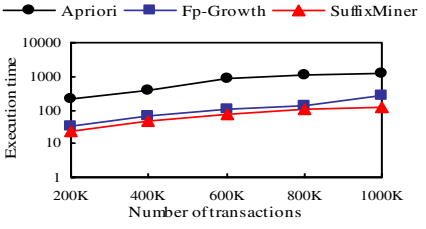

Fig. 3. The comparison of algorithms

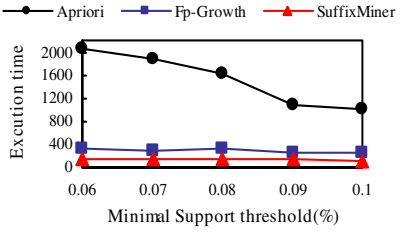

Fig. 4. The comparison of algorithms

4 Conclusion

We present a single-scan algorithm which utilizes the special property of suffix-trees to mine frequent itemsets while it is unnecessary to traverse the suffix-trees. Our algorithm also adopts a novel itemset growth method to avoid generating any candidate itemset. Experiments confirm the excellent abilities of SuffixMiner algorithm.

References

1. Manku, G. S. and Motwani, R.: Approximate Frequency Counts Over Data Streams. In Proceeding of the International Conference on Very Large Data Bases, Hong Kong, China (2002) 346-357
2. Agrawal, R. and Srikant, R.: Fast Algorithms for mining Association Rules. In Proceeding of the International Conference on Very Large Data Bases, Santiago de Chile, Chile (1994) 487-499
3. Giannella, C., Han, J., Pei, J., Yan, X. and Yu, P. S.: Mining Frequent Patterns in Data Streams at Multiple Time Granularities. Next Generation Data Mining, Chapter 3 (2002) 191-211