Frequent Itemset Mining Algorithm based on Sampling Method

Haifeng Li[1,a], Ning Zhang[1], Yuejin Zhang[1]

[1]School of Information Central University of Finance and Economics Beijing, China 100081

[a]mydlhf@cufe.edu.cn

Keywords: Frequent Itemset, Sampling, Data Mining.

Abstract. Frequent itemset mining is an important technique in data mining. This paper employ the sampling method to improve the performance. An in-memory index is presented to store the data information, which is maintained by our proposed algorithm FIMS. We conduct the experiments over two datasets and find that when the sampling rate is reduced, the mining performance will be more efficient.

Introduction

Frequent itemset mining is a traditional and important problem in data mining. An itemset is frequent if its support is not less than a minimum support specified by users. Traditional frequent itemset mining approaches have mainly considered the problem of mining static transaction databases. In these methods, transactions are stored in secondary storage so that multiple scans over the data can be performed. Three kinds of frequent itemset mining approaches over static databases have been proposed: reading-based[4], writing-based[7], and pointer-based[8]. [6] presented a comprehensive survey of frequent itemset mining and discussed research directions.

Frequent Itemsets are huge when the given threshold is low; consequently, the condensed representations of frequent item-sets including closed itemsets[10], maximal itemsets[2,9], free itemsets[3], approximate k-sets[1], and non-derivable itemsets[5] were proposed.

The rest of this paper is organized as follows: In Section 2 we present the preliminaries. Section 3 presents the data structures, and illustrates our algorithm in detail. Section 4 evaluates the performance with theoretical analysis and experimental results. Finally, Section 5 concludes this paper.

Table 1 Simple Database

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

Preliminaries

Given a set of distinct items $\Gamma = \{i_1,i_2,\ldots,i_n\}$ 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 $X = \{x_1,x_2,\ldots,x_m\}$ is $x_1x_2\ldots x_m$. A database $D = \{T_1,T_2,\ldots,T_v\}$ is a collection wherein each transaction is a subset of Γ, namely an itemset. Each transaction $T_i(i =1\ldots v)$ is related to an id, i.e., the id of T_i 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 $\Lambda(X)= \{|T \,||T\in D\wedge X\subseteq T\}$; the relative support (RS) of an itemset X is the ratio of AS with respect to |D|, denoted $\Lambda_r(X)=\Lambda(X)$. Given a relative minimum support λ ($0 \leqslant\lambda\leqslant 1$), itemset X is frequent if $\Lambda_r(X) \geqslant\lambda$. Table 1 is a simple database.

Example 1. Given a simple database D as shown in Table 1 and an absolute support 2, the frequent itemsets are {a, b, c, d, e, ab, ac, ad, bc, bd, be, cd, ce, de, abc, abd, acd, bcd, cde, abcd}.

FIMS Algorithm

FIMS tree To quickly match the itemset and perform incremental mining, we design an in-memory index named FIMST. In the FIMST, each node n_X denotes an itemset X. n_X is a 2-tuple < item, sup >, in which item denotes the last item of the current itemset X, and it is sorted by the support order under the same parent; sup is the support of X. In our data structure, we can see that if node n_X is the parent of node n_Y , then itemset Y is the superset of itemset X; also, all the nodes denote the frequent itemsets, and the infrequent nodes are deleted. We show the FIMST of the database in Table 1 in Figure 1.

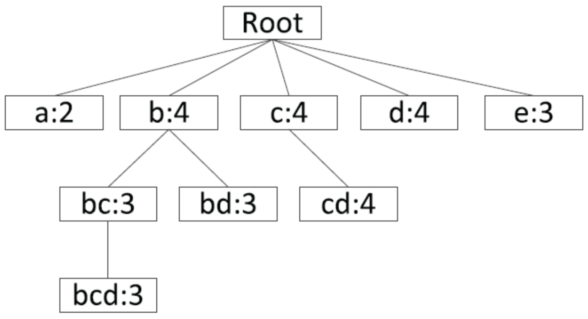


Fig. 1 FIMS tree for λ =3

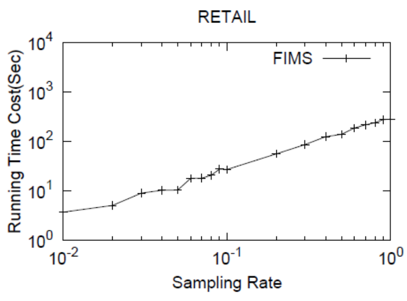
```
Algorithm 1 FIMS Algorithm
Require: $n_X$: node of $FIMST$ denote itemset $X$; $D$: Transaction Database; $\lambda$: minimum support; $\tau$:sampling rate;
1: for each $n_X$'s right sibling node $n_Y$ do
2:     generate itemset $X \cup Y$;
3:     compute the support of $X \cup Y$, $\Lambda(X \cup Y)$ by sampling rate $\tau$;
4:     if $\lambda \leq \Lambda(X \cup Y)$ then
5:         set $n_{X \cup Y}$ as $n_X$'s child node;
6: for each new generated node $n_{X \cup Y}$ do
7:     call FIMS($n_{X \cup Y}$,D,$\lambda$, $\tau$);
```

Sampling the Databases To improve the performance, we take sample from the database. A basic idea is to use a sampling rate to obtain the samples with a normal distribution. Thus, we can run the mining algorithm with a higher speed. Note when we take the samples, different sampling rate may result in a various accuracy, which will be evaluated in our experiments.

Algorithm Description We propose a breath-first algorithm to perform the mining. Algorithm 1 shows the details. As can be seen, we first generate a root node, then we create the children nodes of the root, which represent the distinct items. Furthermore, we generate $X \cup Y$ for itemset X with its sibling itemset Y, and we compute the support, if the support is larger than the minimum support, we will generate a child node $n_{X \cup Y}$ for n_X . After all the children nodes are generated, we will recursively call the FIMS algorithm for the children nodes.

Experiments

We conducted the experiments to evaluate the performance of FIMS. Since the sampling rate may impact the accuracy of the mining results, then we will use it as the major element to conduct the evaluation.



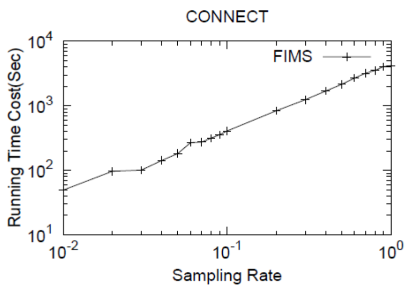


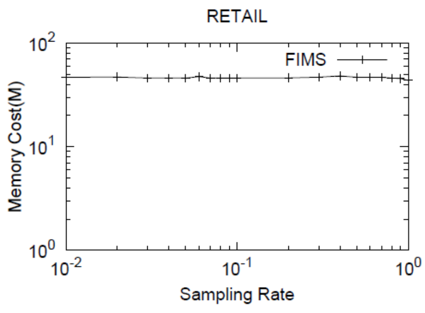
Fig. 2 Runtime Cost for minsup = 0.01 Fig. 3 Runtime Cost for minsup = 0.9

Running Environment and Datasets All the algorithms were implemented with Python, compiled with **Wingide** running on Microsoft Windows 7 and performed on a PC with a 3.60GHZ Intel Core i7-4790M processor and 12GB main memory.

Table 2 Dataset Characteristics

DataSet	Trans Count	Average Size	Min Size	Max Size	Items Count	Trans Correlation
Retail	88 162	10	1	76	16470	1598
Connect4	67 557	43	43	43	129	3

We employed 2 real-world datasets to generate the samples. The retail dataset contained the sale transactions from a super market, and the Connect4 dataset contained all legal 8-ply positions in the game of Connect Four in which neither player has won yet, and in which the next move is not forced. The detailed data characteristics are shown in Table 2.



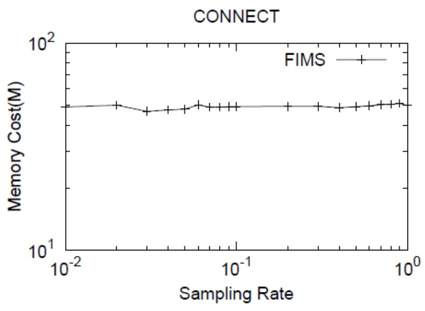


Fig. 4 Memory Cost for minsup = 0.01 Fig. 5 Memory Cost for minsup = 0.9

We set a fixed minimum support, and evaluated the FIMS algorithm over different sampling rate. As can be seen from Figure 2 and Figure 3, when we decreased the sampling rate, that is, from 0.9 to 0.01, the mining efficiency increased significantly over both datasets. Comparing the runtime over the two datasets, we can see that sampling method was more efficient when running over the dense dataset Connect. Furthermore, we can see from Figure 4 and Figure 5, the memory cost was almost unchanged when we alter the sampling rate. It is reasonable since the index we need to maintain in the memory are almost the same. On the other hand, we compared the accuracy of FIMS algorithm over different sampling rate. As can be seen from Table 3, the precision and recall decreased when we reduce the sampling rate. As a result, we can find that the performance is inversely proportional to the accuracy.

Conclusions

In this paper we studied the algorithm for mining frequent itemsets over a database with sampling method. An effective in-memory data structure, the FIMST, was used to record all the frequent itemsets. We proposed an algorithm FIMS to maintain the data synopsis in FIMST. In the algorithm, we use the sampling method to increase the speed of computing the support. Our experiments showed that our sampling method can significantly improve the performance with a relatively high accuracy.

Table 2 Precision and Recall over Datasets

	Retail		Connect4	
Sampling Rate	Precision(%)	Recall(%)	Precision(%)	Recall(%)
0.9	99.38	98.76	99.78	99.03
0.8	98.75	98.14	99.44	99.35
0.7	98.75	97.53	99.9	96.85
0.6	99.38	98.15	99.5	98.89
0.5	98.12	99.37	95.99	99.59
0.4	98.12	96.91	96.25	99.04
0.3	96.88	96.27	96.09	98.77
0.2	96.25	96.86	97.18	97.33
0.1	93.12	93.71	97.8	94.41
0.09	95.62	89.47	95.25	94.06
0.08	93.75	93.17	96.14	94.84
0.07	94.38	88.3	94.13	92.9
0.06	97.5	88.64	99.23	86.35
0.05	90.62	94.16	91.62	97.74
0.04	92.5	87.57	89.09	97.83
0.03	90.62	81.01	82.73	96.57
0.02	83.75	77.01	94.99	75.2
0.01	84.38	66.18	90.7	71.44

Acknowledgement

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] F.Afrati, A.Gionis, and H.Mannila, Approximating a Collection of Frequent Sets, in Proc. SIGKDD’2004

[2] G.Yang. The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns. in Proc. SIGKDD’2004.

[3] J.Boulicaut, A.Bykowski, and C.Rigotti, Free-sets: a condensed representation of boolean data for the approximation of frequency queries, Data Mining and Knowledge Discovery, 7 (2003) 5-22

[4] R.Agrawal, and R.Srikant, Fast algorithms for mining association rules, in: Proc. VLDB’1994.

[5] T.Calders, and B.Goethals, Mining All Non-Derivable Frequent Itemsets, in: Proc. PKDD’2002

[6] J.Han, H.Cheng, D.Xin, and X.Yan, Frequent pattern mining: current status and future directions, Data Mining and Knowledge Discovery, 17 (2007) 55-86

[7] J.Han, and J.Pei, Mining frequent patterns by pattern-growth: methodology and implications, in: Proc. SIGKDD’2000

[8] S.Kevin, and R.Ramakrishnan, Bottom-Up Computation of Sparse and Iceberg CUBEs, in: Proc. SIGMOD’1999.

[9] G.Mao, X.Wu, X.Zhu, and G.Chen, Mining Maximal Frequent Itemsets from Data Streams, Journal of Information Science 33 (3) (2007) 251-262

[10] N.Pasquier, Y.Bastide, R.Taouil, and L.Lakhal, Discovering frequent closed itemsets for association rules, in: Proc. ICDT’1999