An Algorithm of Mining Closed Frequent Itemsets

Haifeng Li[1,a]

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

[a]mydlhf@cufe.edu.cn

Keywords: Closed Frequent Itemset, Sampling, Data Mining.

Abstract. Closed frequent itemset is a perfect representation of frequent itemset. This paper tries to find an efficient solution to mine the closed frequent itemsets over databases by sampling technique. We employ the SCFI tree to record the data synopsis of the frequent itemsets, and propose an efficient algorithm SCFI to maintain the SCFI. We conduct the experiments over standard benchmark datasets, and the results show that sampling is an effective method to improve the performance when mining the closed frequent itemsets.

Introduction

Frequent itemset[1] mining is a traditional and but very important problem in data mining. An itemset is frequent if its support is larger than the minimum support specified by users. Traditional frequent itemset mining algorithms mainly considered the problem of mining transaction databases, in which the transactions are stored in disk to perform multiple scans over the databases. Han et al. reviewed the method of frequent itemset mining and discussed the research directions [2]. Frequent Itemsets are huge when the given threshold is low; consequently, the condensed representations of frequent itemsets including closed frequent itemsets[3], free frequent itemsets[4], maximal frequent itemsets[5], and non-derivable frequent itemsets[6] were proposed.

Closed frequent itemsets are one of the condensed representations, which can completely store the non-redundant cover of frequent itemsets, and thus can reduce the memory cost, and we will introduce the details in Section 2.

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 $\Lambda_r(X)=\Lambda(X)/|D|$. 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.

An itemset is a closed frequent itemsets(CFI) if it is frequent and it is closed. The concept of closed itemset is based on the two following functions f and g, Function f returns the set of itemsets included in all the transactions in T, while function g returns the set of transactions supporting a given itemset. An itemset I is closed if and only if c(I) = f(g(I)) = I.

$$f(T) = \{i \in L \mid \partial t \in T, i \in t\}$$
$$g(I) = \{t \in D \mid \partial i \in I, i \in t\}$$

Example 1. Given a simple database D as shown in Table 1 and an absolute support 3, the frequent itemsets are {a, b, c, d, e, ab, ac, ad, bc, bd, be, cd, ce, de, abc, abd, acd, bcd, cde, abcd}. The closed frequent itemsets are {bcd:3, b:4, cd:4, e:3}.

Closed Frequent Itemset Mining Algorithm

SCFI tree

We try to use a novel index to record the frequent itemsets, our goal is to quickly retrieve the itemsets when we perform mining. Consequently, we construct a tree-based index named SCFI tree(Sampling Closed Frequent Itemset Tree). In the tree, we denote each node n_X as 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 implementation, we use a pointer to link the child node to its parent node. From 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. We show the tree of the database of Table 1 in Figure 1. The bold rectangles represent the closed frequent itemsets.

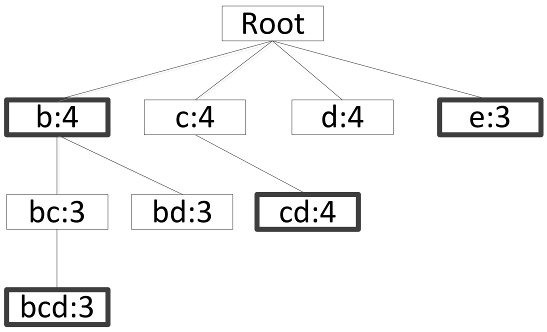

Figure 1 SCFI tree for λ =3

Sampling the Databases

We employ a simple sampling method to get the samples from the database, that is, we randomly take the samples until we get the transactions of count specialized by the users. During this process, we make sure that the sampling is normal distributed. In the experiments, we will use different counts to evaluate our algorithm. This is a key method to reduce the runtime; we try to find the relationship between the sampling rate and the runtime time, as well the result accuracy. This can help us to do the further improvement in our future works

Algorithm 1. SCFI Method

Explore Function

```
Input {r: root of the SCFI tree, minsup: minimum support}
Output { r: root of the SCFI tree }
for child in r.children
        compute the support of child
        if child.support >= minsup
                Explore(child, minsup)
```

GetClosedItemset Function

```
Input {r: root of SCFI tree}
Output {c: the collection of closed frequent itemsets}
if (r has no children or r.support > all r.children.supports)and r not in c
        append r into c
for child in r.children
        GetClosedItemset(child, c)
```

SCFI Algorithm

In this algorithm, we will mine the closed frequent itemsets with two steps. First, We propose a breadth-first algorithm to conduct the mining. In this algorithm, we first generate a root node, and then we create the children nodes of the root, which represent the distinct items. For each child, we recursively 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 itemsets with support larger than the threshold are generated, we achieve all the frequent itemsets. Second, we retrieve the SMFI tree and find the itemset iff 1) it has no children; or 2) it has children, but the supports of its children are smaller than its support. We use a collection to maintain such itemsets. Also, when a new itemset will be inserted into the collection, we will compare it to the itemsets in the collection, if it is not covered, it will be append in the collection. We show the details in Algorithm 1.

Experiments

We conducted the experiments to evaluate the performance of SCFI. In the experiments, we use the minimum support and sampling rate as the major elements to do the evaluation.

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
T40I10D100K	100 000	4	77	40	1000	24

We employed a real-life dataset named Retail, and a synthetic dataset named T40I10D100K as the evaluation dataset, which was generated by the IBM data generator. The detailed data characteristics are shown in Table 2.

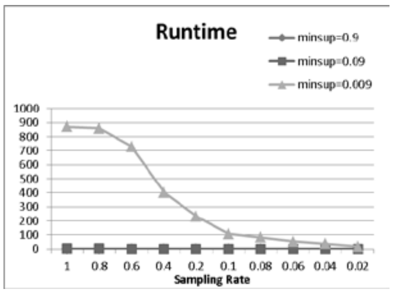

Fig. 2 Runtime Cost(S) for Retail

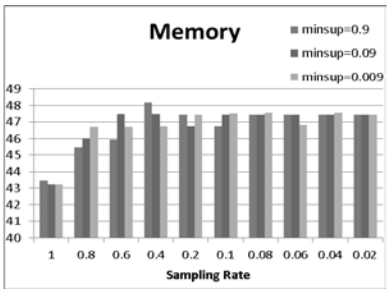

Fig. 3 Memory Cost(S) for Retail

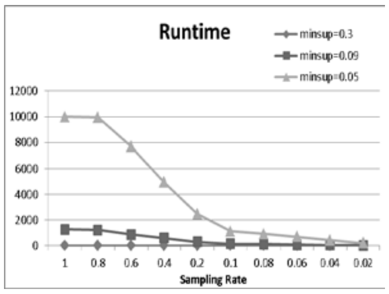

Fig. 4 Runtime Cost(S) for T40I10D100K

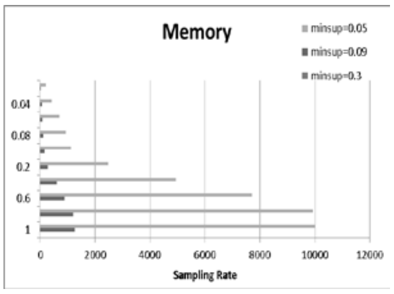

Fig. 5 Memory Cost(S) for T40I10D100K

We first compared the SCFI algorithm over different sampling rate and various minimum support. We can see from Figure 2 and Figure 4, when we decreased the sampling rate, the mining efficiency increased significantly over the dataset. The smaller the minimum support, the more effect over the performance the sampling method achieved. We argue that it is reasonable since little frequent itemsets were generated for such an argument. On the other hand, we can see from Figure 3 and Figure 5, the memory cost was stable when we changed the sampling rate, which is due to the fact that the index we need to maintain in the memory are almost the same; thus, if the results are same, the memory cost will be similar. Furthermore, we compared the accuracy of SCFI algorithm over different sampling rate. As can be seen from Figure 6, in the spare dataset Retail, the precision decreased when we reduce the sampling rate, but the recall was almost unchanged. On the other hand, when we perform the algorithm over the dense dataset T40I10D100K, we noticed that both the precision and the recall decreased in line with the sampling rate.

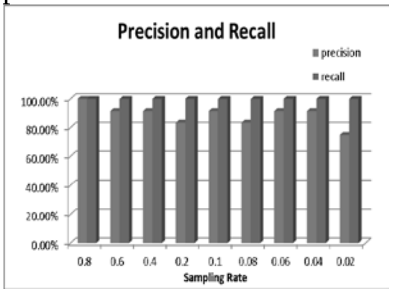

Fig. 6 Retail for minsup = 0.09

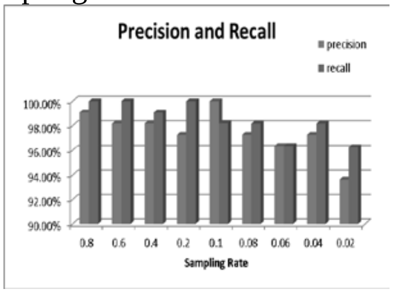

Fig. 7 T40I10D100K for minsup=0.09

Conclusions

In this paper we make a research on how to achieve the closed frequent itemset over a sampling database. We propose an in-memory efficient data structure SCFI tree to store the frequent itemsets, and introduce a collection to record the closed frequent itemsets. We use a breadth-first algorithm SCFI to achieve the mining results. A sampling method is employed to improve the performance. The extensive experiments are conducted, and the results show that when we reduce the sampling rate, we can achieve a significant improvement with a little accuracy loss.

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).

Refereneces

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

[2] 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

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

[4] 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

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

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