文章编号:1001 - 9081(2005)06 - 1385 - 03

基于逆向 FP-树的频繁模式挖掘算法

赵艳铎,宋斌恒
(清华大学 软件学院,北京 100084)
(yanduozh@eyou.com)

摘 要:提出了一种称为逆向 FP-合并的算法,该算法逆向构造 FP-树并通过在其中寻找频繁扩展项集与合并子树来挖掘频繁模式。新算法在时空效率方面均优于 FP-增长算法,其中时间效率提高了 2 倍以上。此外,新算法还具有良好的伸缩性。

关键词:数据挖掘;频繁模式;逆向 FP-树;逆向 FP-合并算法;频繁扩展项
中图分类号: TP311.12 **文献标识码**:A

Algorithm for mining frequent patterns based on converse FP-tree

ZHAO Yan-duo, SONG Bin-heng
(*School of Software, Tsinghua University, Beijing* 100084, *China*)

Abstract: It proposed an algorithm for mining frequent patterns by finding the frequent extensions and merging sub-trees in a conversely constructed FP-tree. The performance of the algorithm is superior to the one of FP-Growth both in time and space consuming. It runs over two times faster than the FP-Growth and has a good scalability.

Key words: data mining; frequent pattern; conversed FP-tree; conversed FP-merging algorithm; frequent extension item

0 引言

FP-增长算法[2]是一种本质上不同于 Apriori 算法[1]的挖掘频繁模式的有效算法。FP-增长算法只需两次扫描数据库,而且不用产生大量候选项集,将发现长频繁模式的问题转换为递归发现一些短模式,然后连接后缀。该方法降低了搜索开销,大约比 Apriori 算法快一个数量级。

FP-增长算法开辟了有效挖掘频繁模式的新途径。然而,它的时间和空间效率仍需要进一步改进。文献[3]改进了 FP-树,提出了一种基于被约束子树的挖掘方法,改进了树中结点的存储结构,节省了 1/3 的树空间,同时挖掘过程中不产生条件 FP-树,从而需要更少的辅助空间,大大提高了挖掘的时空效率。文献[4]、[5]、[6]也都提出了一种有效的挖掘频繁模式的方法,其时间效率都比 FP-增长提高了 1 倍左右。

本文提出了一种称为逆向 FP-合并算法的新方法,在该算法中,我们构造逆向 FP-树,树中结点的存储引入了文献[3]中的思想,节省了树空间;然后挖掘每棵子树,通过寻找频繁扩展项集与合并子树来挖掘频繁模式。实验表明,本文的算法明显优于 FP-增长算法,挖掘速度较 FP-增长算法提高了 2 倍以上,而且挖掘过程中不需要太多的辅助空间。

1 问题的描述

1.1 基本概念

项是一个文字,在交易数据库中,可以代表商品;在分类时,可用代表属性的值。

项的集合称为项集,包含 k 个项的项集称为 k-项集。项集出现的频率是包含项集的事务数,又称为项集的支持度计数。给定最小支持度 min_sup,如果项集支持计数大于或等于 min_sup 与 D 中事务数的乘积,则称项集满足最小支持度 min_sup。如果项集满足最小支持度,则称该项集为频繁项集,频繁 k - 项集的集合通常记作 Lk。

给定项集 $I = \{i1, i2, \cdots, im\}$,事务数据库 $D = \{T1, T2, \cdots, Tn\}$,其中每个事务 $Ti(i \subset [1,2,\cdots,n])$ 包含事务 ID 号 TID 和一个 I 中项的子集。I 的子集也是项集或模式,根据定义,项集 X 的支持度计数 $sup\_count(X)$ 是 D 中包含项集的事务数;X 的支持度 $support(X) = sup\_count(X)/|D|$,其中 $|D|$ 是 D 中事务的个数。

1.2 关联规则的挖掘

关联规则的挖掘可分为如下两个过程:1)找出所有频繁项集;2)根据频繁项集产生强关联规则。

在这两个过程中,第二步比较容易,挖掘关联规则的总体性能由第一步决定。在事务数据库中挖掘频繁模式的问题可以描述为:给定事务数据库 D 和最小支持度阈值 min_sup,挖掘所有的频繁模式。

2 逆向 FP-合并算法及实现

本文所介绍的算法与 FP-增长算法有很多相似之处,同样分为两个过程:构造逆向 FP-树和挖掘逆向 FP-树。所不同的是,本算法所构造的 FP-树是基于支持度计数升序排列的频繁 1 - 项集,为了以示区别,将本文构造的树称为逆向 FP-树,挖掘时,挖掘第一棵子树,然后将第一棵子树的所有子树与逆向 FP-树的其他分支合并,同时删除第一棵子树,然后对新的逆向 FP-树递归调用挖掘过程,直至树只有一棵子树,且已经挖掘完该子树。

2.1 构造逆向 FP-树

构造逆向 FP-树的主要步骤如下:

1) 扫描数据库 D 一遍,根据最小支持度阈值得到频繁 1-项集,对频繁 1-项集按其支持计数由小到大排列;

收稿日期:2004 - 11 - 26;**修订日期**:2005 - 02 - 23

作者简介:赵艳铎(1981 -),男,河北青县人,硕士研究生,主要研究方向:数据挖掘与机器学习; 宋斌恒(1963 -),男,内蒙古凉城人,副教授,主要研究方向:网络计算与算法研究.

2) 再次扫描数据库 D 一遍,构造逆向 FP-树,过程如下:扫描数据库 D,获得下一个事务;删除事务中的非频繁项,将剩余的项按支持计数由小到大排列;将排列好的各项插入逆向 FP-树中。

例: 对表 1 所示的事务数据库 D 构造逆向 FP-树

项 a,b,c,d,e,f,g,h,m 和 p 的支持计数分别为 7,3,6,2,2,5,2,3,4,2,如果最小支持度阈值 $min\_sup = 3/7 = 42\%$,即支持计数大于等于 3,则频繁 1- 项集 $L1 = \{b,h,m,f,c,a\}$(按支持计数升序排列),构造逆向 FP- 树如图 1 所示。

表 1 事务数据库 D

TID	项集
1	*a, c, e, h, m, p*
2	*a, b, c, f, m*
3	*a, b, d, f, m*
4	*a, b, c, f*
5	*a, c, e, f, h*
6	*a, c, f, g, m*
7	*a, c, d, g, h, p*

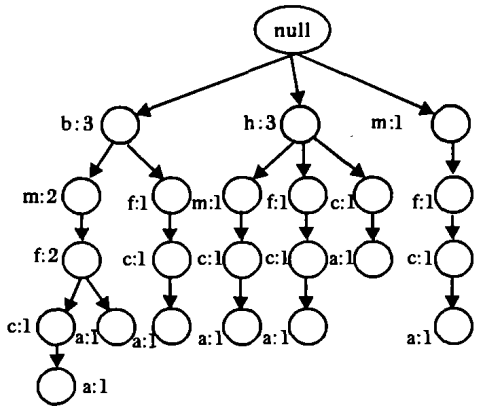

图 1 表 1 构造的逆向 FP-树

2.2 挖掘逆向 - FP-树

仍然以表 1 所示事务数据库为例,对图 1 所示的逆向 FP-树进行挖掘,过程如下:

1)挖掘第 1 棵子树;

第 1 棵子树的定义是指所有子树中根结点在 $L1$ 中最靠前的子树,本例即是以 b 为根的子树,由于 b 是逆向 FP-树中支持度计数最小的项,所以,所有包含 b 的频繁项集都可以从该子树中挖掘出来。

挖掘过程如下:

a) 寻找 b 的频繁扩展项集;

定义 给定事务数据库 $D = \{T1,T2,\cdots,Tn\}$,频繁 1- 项集 $L1 = \{i1,i2,\cdots,im\}$,如果 $\{i1,i2\}$ 是频繁 2- 项集,则称 $i2$ 是 $i1$ 的频繁扩展项,所有 $i1$ 的频繁扩展项组成的集合称为 $i1$ 的频繁扩展项集。

可以通过该子树中各项的出现频度来判断该项是否是 b 的频繁扩展项。本例中 b 的频繁扩展项集是 $\{f,a\}$。

b) 构造一棵只包含 b 及其频繁扩展项的新子树,本例的新子树只有一个分支 $\{b,f,a\}$;

c) 如果新的子树只有一个分支,则输出包含 b 的频繁项集;如果新的子树包含多个分支,则对该子树调用逆向 FP-树挖掘过程。

2) 将 b 的子树合并到 b 的兄弟节点,同时删除以 b 为根的子树。

本例中 b 有 2 个子树,将以 m 为根的子树与一级节点 m 合并,如图 2 所示。

因为一级节点中不包含 f,所以以 f 为根的子树成为根 null 的新子树,合并后的树如图 3 所示,此时第一棵子树是以 h 为根的子树。

3) 对合并后的树重复上述步骤,直至所有的分支都挖掘完。

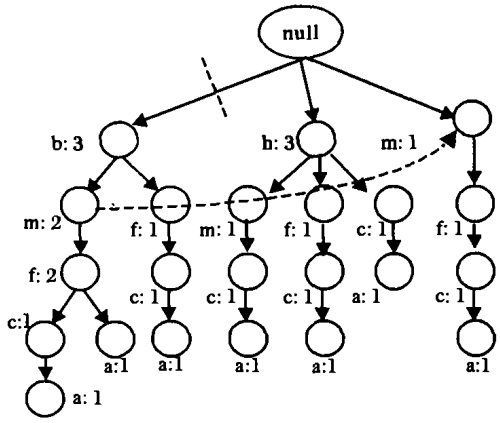

图 2 将 b 的子树与其兄弟节点合并

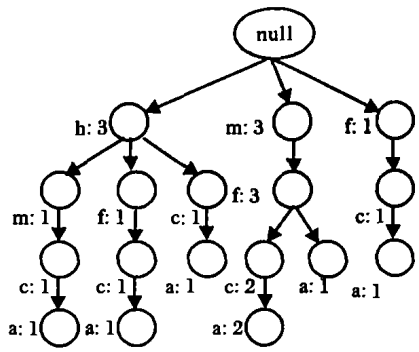

图 3 第一次合并后的树

2.3 算法实现

本文介绍的逆向 FP-树中结点类定义如下:

```
Class Item{
  private:
    char * value;
    int supp;
    set < Item > * children;
  public:
    Item();
    Item( char * v, int s);
    char * getValue();
    int getSupport();
};
```

该结构只存储了结点的项值、支持度计数和孩子结点,因此节省了树空间。

逆向 FP-树的构造过程与 FP-树的构造过程类似,这里不再赘述,下面给出挖掘逆向 FP-树的算法。

```
void converfp_mine( set < Item > * tr) {
  int i = 0;
  if tr 只有一个子树
    输出频繁项集;
    return;
  for( ; i < num; i ++) {                    //num 是频繁 1-项集的个数
    set < Item > : : iterator it;
    for( it = tr - > begin(); it ! = tr - > end(); it ++) {
      if ( it - > getValue() = = fp1[ i]) {
        //fp1 按支持度计数升序存放频繁 1 - 项集
        sub_mine( it);                       //挖掘第一棵子树
        merge( it);                          //合并子树
        break;
      }//end if
    } // end for
  } //end for
  fp_mine( tr);
}
```

其中 sub_mine 用于挖掘第一棵子树,merge 用于删除第一棵子树,同时把其所有子树合并到逆向 FP-树中,合并算法比较简单,下面给出挖掘第一棵子树的函数 sub_mine 的伪代码:

```
void sub_mine( set < Item > : : iterator it) {
  set < Element > * efp;
  //Element 是一种包括项值和支持度的数据结构
  efp = getEFP( it);                          //获取频繁扩展项集
  set < Item >  * newtr;
  newtr = ReBuild( efp,  it)                  //根据 efp 重新建树
  if ( newtr 只有一个分支)
    输出频繁项集;
  else
    converfp_mine( newtr);
}
```

其中 getEFP 用于获取根结点的频繁扩展项集,ReBuild 函数用于重新构造新的逆向 FP-树,构造过程与 FP-树的构造过程类似,这里不再赘述。

2.4 算法的正确性

要证明算法正确,需要证明如下两点:1)算法输出的项集都是频繁的;2)算法输出了所有的频繁项集。

根据扩展频繁项集的定义和算法的挖掘过程可以证明算法输出的项集都是频繁的;要证明本算法能够输出所有的频繁模式首先证明逆向 FP-树中包含所有频繁模式信息:

因为逆向 FP-树的构造思想与 FP-树相同,FP-树中包含了事务数据库 D 中的所有频繁模式信息,所以逆向 FP-树中也包含了所有频繁模式信息。

下面使用数学归纳法证明挖掘逆向 FP-树的算法能够输出所有频繁模式,证明过程如下:

证明:

设 $P = \{i1,i2,\cdots in\}$ 为任意频繁项集,且 $\{i1,i2,\cdots,in\}$ 按支持度计数升序排列。

∵ $L1$ 是所有频繁 1- 项集

∴ 当 $n = 1$ 时,本算法能够输出所有的频繁 1- 项集

设 $n = k$ 时,算法能够输出所有的频繁 k-项集;

如果存在 $ik + 1 \subset L1$,其支持度计数 $supp[k + 1] \leqslant supp[n](1 \leqslant n \leqslant k)$,使得项集 $\{i1,i2,\cdots ik,ik + 1\}$ 是频繁 k +1 - 项集,下面证明本文介绍的逆向 FP-合并算法能够输出该项集。

∵ 项集 $\{i1,i2,\cdots ik,ik + 1\}$ 中各项按支持度计数升序排列,所以在逆向 FP- 树中,$ik + 1$ 是 $i1$ 的子孙节点。

又∵ 逆向 FP- 合并算法能够输出频繁模式 $\{i1,i2,\cdots ik\}$,而 $ik + 1$ 是 $\{i1,i2,\cdots ik\}$ 的频繁扩展项,$\{i2,i3,\cdots ik,ik + 1\}$ 是 $i1$ 的频繁扩展项集。

所以逆向 FP- 合并算法一定也能输出频繁模式 $\{i1,i2,\cdots,ik,ik + 1\}$。

经过上述证明可以得出如下结论:使用本文的逆向 FP-合并算法能够正确输出给定事务数据库和最小支持度阈值的所有频繁模式。

3 实验结果

为了更好地说明本文算法的效率,我们在同样的实验环境下将本文算法与 FP-增长算法进行比较。

实验环境:P Ⅲ 1.4G,256M 内存,Win2003 Server,Visual C++ .NET。

实验数据集 D1 来自网络防火墙的日志,包含 10K 个事务,500 个不同项,事务平均长度为 35;数据集 $D2$ 来自网络 IDS 的日志,包含 100K 个事务,5K 个不同项,事务平均长度为 18。

实验结果如图 4 所示。

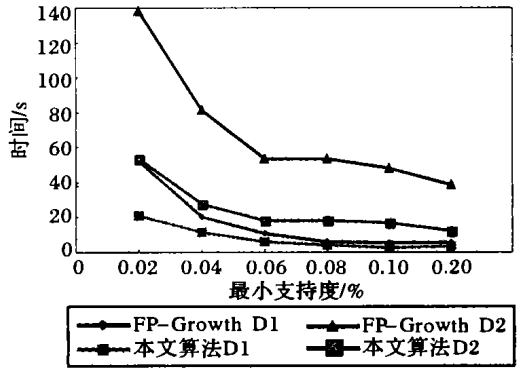

图 4　对数据集 D1 和 D2 两种算法优劣比较

从图中可以看出,对于我们测试的相对稀疏的数据集来说,本文的算法在时间上明显优于 FP-增长算法,尤其在支持度较小时,所需时间仅为 FP-增长算法的 35% 左右;而且随支持度的减小,本文算法的增长趋势也较平缓。

为了测试算法的可伸缩性,我们由 $D2$ 随机采样,对 10K,20K,…,100K 的事务数据集,最小支持度取 1% 进行挖掘,两种算法的比较如图 5 所示。

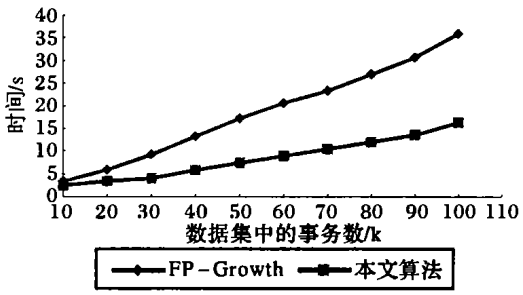

图 5　比较两种算法的伸缩性

从图中可以看出,本文算法增长趋势较 FP-Growth 算法平缓,所以具有良好的伸缩性。

4 结语

频繁模式挖掘是数据挖掘研究的重要内容,本文算法是对 FP-增长算法的一种改进,实验表明,本算法优于 FP-增长算法,而且具有良好的可伸缩性。但是,如何更有效地挖掘频繁模式,仍然是一个值得进一步研究的问题。

参考文献:

[1] HAN J, KAMBER M. 数据挖掘: 概念与技术[M]. 范明, 孟小峰, 等译. 北京: 机械工业出版社, 2001.

[2] HAN J, PEI J, YIN Y. Mining frequent patterns without candidate generation[A]. Proceeding of 2000 ACM-SIGMOD International Conference on Management of Data[C]. USA: ACM Press, 2000. 1 - 12.

[3] 范明, 李川. 在 FP-树中挖掘频繁模式而不生成条件 FP-树[J]. 计算机研究与发展, 2003, 8: 1216 - 1222.

[4] 张力飞, 朱骁峰, 何炎祥. 利用网格服务的分布式频繁模式挖掘算法[J]. 计算机工程与应用, 2004, 7: 179 - 181, 220.

[5] 刘君强, 潘云鹤. 一种基于树的频繁模式挖掘算法[J]. 系统工程理论与研究, 2003, 5: 108 - 112.

[6] CONG G, OOI BC, TAN K-L. Go Green: Recycle and Reuse Frequent Patterns[A]. Proceeding of 20th ACM - SIGMOD International Conference on Data Engineering (ICDE' 04)[C]. Computer Society 2004. 1 - 12.