基于前缀的 Apriori 算法

粟莉萍,杨文伟
(广东工业大学 计算机学院,广东 广州 510006)

摘 要:通过对 Apriori 算法的研究和分析,结合算法存在的缺陷,利用“桶”技术及压缩组合项集技术,对频繁项集提出了前缀概念,并提出了基于前缀的频繁项集挖掘算法。该算法将具有同一前缀的频繁项集的子集合作为一个节点,由频繁 k-项集的子集合直接产生候选 $(k+1)$-项集,从而省略了连接步中判断 I_1、I_2 是否能连接。同时,该算法使得整个程序中节点数目减少,这样不仅减少了内存消耗,而且提高了查找 C_k 和 L_k 的速度,尤其便于大型数据库的分布式处理。经实验证实,改进后的算法是可行的。

关键词:Apriori 算法;关联规则;频繁项集;“桶”技术;压缩组合技术
中图分类号:TP311.13 **文献标识码**:A **文章编号**:1674-7720(2011)04-0075-04

Apriori algorithm based on prefix

Su Liping, Yang Wenwei
(College of Computer, Guangdong University of Technology, Guangzhou 510006, China)

Abstract: In view of Apriori algorithm research and analysis, combined with the existence of the algorithm, according to “Bucket” technology and compression technology, the paper puts forward the prefix concept to frequent itemsets and frequent itemsets mining algorithm based on prefix. The new algorithm lets the subset of frequent itemsets having a same prefix as a node, and has a decrease in number of nodes, thus reduces the storage space and improves the speed of finding C_k or L_k. Through the node generates candidate K-frequent itemsets directly, omits the judge step in join step. Besides, the algorithm performs well in distributed processing, especially for large databases. The experimental results show the proposed method is effective.

Key words: Apriori algorithm; association rules; frequent itemsets; “Bucket” technology; compression technology

关联规则挖掘概念最早由 Agrawal 等人在 1993 年提出[1]。1994 年,Agrawal 等人建立了用于事务数据库挖掘的项集格空间理论[2],并提出了著名的 Apriori 算法,后其成为基本的关联规则挖掘算法。其核心原理是频繁项集的子集是频繁项集,非频繁项集的超集是非频繁项集。

关联规则挖掘算法的设计可以分解为两个子问题:

(1)找到所有支持度大于最小支持度的项集(itemset),称之为频繁项集(frequent itemset);

(2)由频繁项集和最小可信度产生规则。

其中,提高整个过程效率的关键在于提高问题(1)的效率。针对问题(1),本文对 Apriori 算法的实现提出了基于前缀的频繁项集挖掘算法。主要针对大型数据库,通过减少项集占用内存和分段处理,使设备资源在有限的情况下有效地实现频繁项集挖掘。

1 Apriori 算法简介及评价

1.1 Apriori 算法简介

Apriori 算法是较为经典的关联规则挖掘算法,该算法利用逐层迭代搜索的方法挖掘数据间的关联关系。其基础思想是重复扫描数据库,根据项集格空间理论,可以由频繁 k-项集迭代地产生候选 $(k+1)$-项集;再扫描数据库验证其是否属于频繁 $(k+1)$-项集。

为了方便计算,令所有的事务和项集中的项都按一定的原则排序,即对任意事务 $d \in D$ (D 记为事务数据库),$d[1]<d[2]<\cdots<d[m]$;对任意频繁项集 I,$I[1]<I[2]<\cdots<I[k]$;对任意候选项集 c,$c[1]<c[2]<\cdots<c[k]$。其主要步骤为:初始时扫描数据库一次,产生 L_1;然后重复连接步和剪枝步[3],直到频繁项集集合为空。其中连接步为:对任意 $I_1 \in L_k$、$I_2 \in L_k$,若 $(I_1[1]=I_2[1]) \wedge (I_1[2]=I_2[2]) \wedge \cdots \wedge (I_1[k-1]=I_2[k-1]) \wedge I_1[k]<I_2[k]$,则令 $c_{k+1}=I_1 \cup I_2[k]$,

$c_{k+1} \in C_{k+1}$，其关键在于快速判断 I_1、I_2 的前 $k-1$ 项都相同，以提高速度。剪枝步：扫描数据库，对任意 $c \in C_{k+1}$ 计数，若计数≥最小支持度计数，则 $c \in L_{k+1}$，其关键在于如何快速查找 c，减少查找时间。

1.2 相关关联规则算法的评价

由于 L_k 和 C_{k+1} 数目可能很大，因此涉及的判断和查找的计算量将会很大；此外多次扫描事务数据库，需要很大的 I/O 负载；同时，L_k 和 C_{k+1} 占据的大量存储空间中，有很大一部分是重复的。

针对 Apriori 算法的性能瓶颈，许多的研究者在 Apriori 算法的基础上提出了很多解决方法；同时，也有许多研究者提出了非基于分层搜索的频繁项集挖掘算法。其中基于分层搜索的算法，主要从减少候选项集的规模并提高查找速度及扫描数据库次数和规模两方面考虑。如 Park 基于散列技术和事务压缩技术提出了 DHP 算法[4]，有效缩减了 2 候选项集的规模和扫描事务量，减少内存消耗，但此方法对大型数据库如何合理地构建 Hash 桶时比较难把握。针对多次扫描数据库的问题，有人提出了基于 Tid 表[5]、基于矩阵[6]、基于位阵[7]等的频繁项集挖掘算法。基于 Tid 表的频繁项集挖掘算法利用得到 L_1 后重组数据库，生成频繁项集表，只需要 2 次访问数据库。基于矩阵、位阵的算法是利用矩阵来存储事务数据库，只需 1 次访问数据库，同时利用矩阵、位阵的特性，提高了运算速度。无论是基于频繁项集表，还是基于矩阵位阵的频繁项集挖掘算法，都需要占用大量内存来一次性存储频繁项集表和事务数据库。此外，对于基于频繁项集表的算法，一个重组后规模为 n 的事务，根据排列组合原理将生成 $(2n-1)$ 个规模大于 1 的子集，再根据互补子集原理及栈原理，得出在最优情况下时间复杂度为 $O(2n)$，显然生成频繁项集表的时间消耗也不小。因此，此类型算法不适于大型的事务数据库。

在非基于分层搜索的算法中，主要以 FP_Growth[8]算法及其各种改进算法为主。这类算法，需要 2 次访问数据库。通过第 1 次访问数据库，得到 L_1，并按支持度计数的递减顺序排序，再采用“分治策略”构造 FP_Tree，最后由 FP_Tree 挖掘出频繁项集。同基于矩阵的算法一样，该算法需要大量内存空间存储 FP_Tree；此外，删除某一项时，对与此相关的节点支持度计算进行调整将花掉不少时间，这主要是由于在 Tree 中只能由父节点直接查找子节点，而不能由子节点查找父节点。因此，对于大型数据库，此类算法也不适合。

而对于 Apriori 算法，可以考虑对每一轮的 L_k 重组项，利用 SQL 优化查询访问数据库，来减少了每轮扫描的事务量及提高查找速度，从而提高整体性能。

2 改进的 Apriori 算法

Apriori 算法主要依赖于迭代性质产生频繁项集。候选 $(k+1)$-项集 c_{k+1} 的产生是在判断频繁 k-项集 I_1、I_2 能够连接的基础上产生的。显然，在按照单个频繁项集为一个节点的情况下，需要大部分时间来判断 I_1、I_2 是否能够连接。如果频繁项集不是很大，则这个连接也不会花很多时间；但若频繁项集很大，这个判断过程将会花费很多时间。同时，在计算候选项集计数时，也将花费很多时间用于查找频繁项集。

2.1 数据结构

Apriori 算法数据结构中的类主要包括以下几种：

(1)*LkSet* 所有候选 k-项集或频繁 k-项集集合，关键属性 isets 为 *LkISet* 集合，*Items* 为当前 L_k 中所有的项集合，*Iflags* 为对应 *Items* 的简约表示，*min* 最小支持度计数；

(2)*LkISet* 所有第一项相同的候选 k-项集或频繁 k-项集集合，关键属性 first 为项集的第一项，nodes 为 *LkNode* 集合；

(3)*LkNode* 具有相同前缀(记为 pres)的候选 k-项集或频繁 k-项集集合。其中 *LkNode* 还有两个关键的属性，一是 rigths，是节点中所有候选项集或频繁项集的最后一项的集合体；二是 degrees，是节点中所有候选项集或频繁项集的计数的集合体。

2.2 算法描述

(1)初始条件：所有事务和项集都按照一定的原则对项进行排序；扫描数据库，产生 L_1、*Items* 和 *Iflags*，其中 *Items* 为当前 L_k 中所有项的集合。

(2)根据得到的 L_1，由事务数据库直接产生 C_2，并对 C_2 进行剪枝产生 L_2，同时更新 *Items* 和 *Iflags*。

(3)由 L_k 连接产生 C_{k+1}：CkfromLk(begin,end)。

(4)扫描事务数据库 D，对 C_{k+1} 计数：Updatedegrees(D)；对任意 $d \in D$：LkfromCk(d)。其中在 Updatedegrees(D)中首先根据 Items 筛选有效地数据记录，然后在根据事务的规模决定是否进入函数 LkfromCk()。

(5)删除计数小于 *min* 的 c_{k+1}：DeleteByMin-(begin)。

(6)更新 *Items* 和 *Iflags*：UpdateItems()。

然后重复(3)~(6)步，直到 $L_k=\varnothing$。

以下是一些函数的具体描述：

```
(1)CkfromLk(begin,end)
If(end<1) end=Lk.iset.size();
  For each iset in Lk.isets(begin...end)
  {  For each node in iset
     { If(node.rights>=2)
       { cnode.pres=node.presnode.rigths.get(j);
         cnode.rigths=node.rigths(j+1...node.
         rigths.size()-1);}//cnode ∈ Ck+1;
     iset.remove(node); }
  isets.remove(iset); }
```

由于大型数据库的候选项集规模庞大，若一次性得到所有候选项集，再进行剪枝，可能会受到设备的限制，因没有足够大的内存而导致 OutOfMemoryError。通过增加 begin、end 参数，能够有效地控制当前候选项集的规

模，不过这样增加了计算支持度计数时访问数据库的次数。但是，通过这些参数可以很方便地运用到分布式处理，能够使各个块互补干扰，且所有块的频繁项集之和就为整个数据库的频繁项集。在合成 C_{k+1} 的同时，删除 L_k 中不需要的节点。

(2)LkfromCk(*d*)函数用来计算事务 *d* 对 C_k 计数的变化。对 $\forall c_k \in C_k$，若 $c_k \subset d$，则该项集支持度计数加 1。其具体表述为：

```
index=cnode.contain(d);
if(cnode.pres⊂d) index=j+1;
// d[j]=cnode.pres[k-1]
else index=-1;
if(index! =-1)
{ for each rights[i] in cnode.rights
    if(rights[i] in d[index…d.size()])
      cnode.degrees[i]++; }
```

(3)DeleteByMin(begin)函数用来修剪 C_k。其中参数 begin 用来控制 iset 的起始点。对于一次迭代，需要分段处理时，每一分段处理后得到的频繁项集都属于最终频繁项集，与其他分段是互补干扰的，因此 begin 用来确认当前分段的初始 iset，这样使得这次的 Updatedegrees(D)不会对前面分段产生影响，同时也提高了查找速度。其具体表述为：

```
For each iset in this.iset[begin… size()]
{ For each node in iset
    { For each degree[i] in node.degrees
      If(degree[i]<min)
      { node.degrees.remove[i];
        node.rights.remove[i]; }
      If(node.Isempty()) iset.remove(node);}
    If(iset.Isempty()) this.iset.remove(iset) }
```

3 实验及性能分析

本数据来源于 http://grouplens.org 网站。首先预处理数据：select user,isbn from bxxbookratings where user in (select user from bxxusers) and isbn in (select isbn from bxxbooks)，最终得到记录 1 031 177 条，其中共有 92 107 个 user 和 269 862 种 book，事务的平均规模为 11.2。

运行环境：MyEclipse6.01；PC 内存：2GB；绘图环境：Matlab7.0。

由于内存的消耗主要用于存储候选项集和频繁项集，因此内存消耗主要取决于当前情况下候选项集和频繁项集的个数及 *k* 值。假设当前情况下，频繁 *k*-项集个数及节点个数分别为 N_k、P_k，候选 $(k+1)$-项集个数节点个数分别为 N_{k+1}、P_{k+1}，改进前后的内存消耗分别记为 *SA*、*OSA*，*A* 与 *OA* 差记为 *S*，则：

$N_k \geqslant P_k$；$N_{k+1} \geqslant P_{k+1}$；

$SA \approx N_k \times (k+1) + N_{k+1} \times (k+2)$；

$SOA \approx (N_k + N_{k+1}) \times 2 + P_k \times (k-1) + P_{k+1} \times k$；

$S \approx (N_k - P_k) \times (k-1) + (N_{k+1} - P_{k+1}) \times k$；

由此可知，任何情况下，改进后的 Apriori 算法内存消耗都不可能多于改进前的 Apriori 算法内存消耗；且随着事务数据库越稠密，节点个数与项集个数差越大，*S* 越大；此外，随着 *k* 的增加，*S* 越大，即改进后的算法空间占用越少。因此，对比实验主要针对时间消耗进行分析。

3.1 对比实验

对于同一事物数据库，频繁项集挖掘的效率和结果主要取决于最小支持度阈值；最小支持度阈值越大，运行越快，得到的频繁项集越少。对于同一事物数据库，*min* 越小，每次迭代产生的频繁项集和候选项集越多。图 1 所示为对于同一事务数据库，随 *min* 的不同，所需时间的对比情况。

图 1 整体运行时间随 *min* 变化的比较图

对于规模相同、稠密度不同的事务数据库，在 *min* 相同时，事务数据库越稠密，每次迭代产生的频繁项集和候选项集越多。此种性质类似于同一事务数据库不同 *min* 时的性质。因此，对于不同稠密度事务数据库的比较实验，可以参照同一事物数据库不同 *min* 的比较实验。由图 1 可知，事务数据库越稠密，改进的 Apriori 算法优势越明显。表 1 给出了 *min*=12 时，候选 *k*-项集和频繁 *k*-项集的个数及其节点个数；图 2 给出了 *min*=12 时，两种算法在每次迭代中各个步骤所花时间的比较情况。

表 1 项集个数表

K	*Cnode*	*Cnum*	Node	Num
3	159 495	10 054 226	65 027	489 335
4	424 308	4 645 035	169 835	552 398
5	382 563	1 753 530	161 113	339 368
6	178 255	490 190	94 750	161 942
7	67 192	138 821	45 489	69 548
8	24 059	41 314	19 348	26 764

根据算法自身的特点可知，DeletebyMin()只需要一次遍历所有候选项集的支持度计数；改进后的 CkfromLk()只需要一次遍历所有频繁项集，而非改进时，还需要判断两个频繁项集是否能连接，而存在某些频繁项集多次访问；Updatedegrees()与事务相关联，大量候选项集需要多次访问。结合算法的特点，从理论及实际上，证明了总

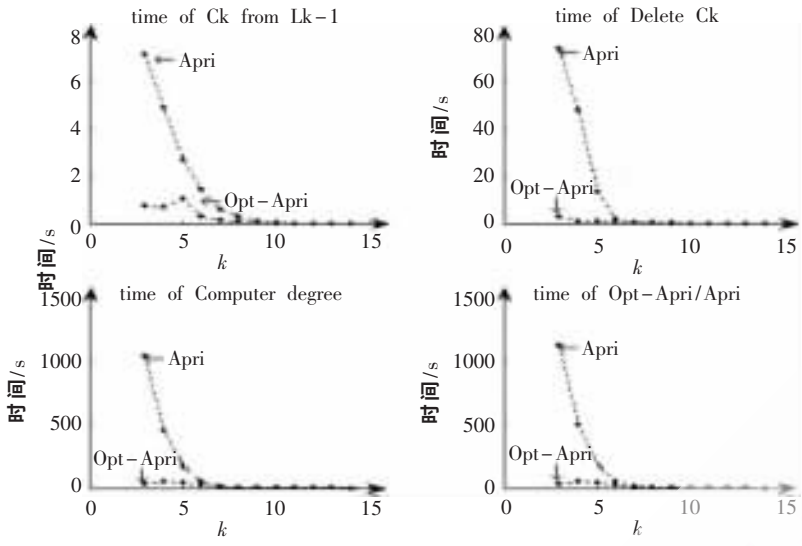

图 2 各个步骤运行时间随 k 变化的比较图

体运行时间主要取决于计数步，而随着数据集越稠密，改进后的算法优势更明显。

3.2 模拟分布式处理

令 min=12、k=4，平均分为 n 段(n=1，…，8)进行分段处理，以模拟分布式处理，得到结果如图 3 所示。

图 3 分段处理结果

从图 3 看出，在一定误差范围内，剪枝步和合成候选项集并没有随着 n 的变化而变化；计数所花时间随着 n 的增加有细微的增加；访问数据库所花时间随着 n 的增加大而成倍数增加；总体时间的变化主要取决于访问数据库所花时间。

在深入研究 Apriori 算法及其相关算法的基础上，结合“桶”技术、压缩组合原理、数据重组等思想，针对大型数据库提出了基于前缀的频繁项集挖掘算法，并且根据实际情况，对频繁 k-项集的产生采用了直接从数据库得出的、有别与其他频繁项集产生的特殊处理方法。理论和实验表明，改进后的 Apriori 算法在时间和空间上都有改进，且能够进行分段处理并运行到分布式处理中。在用于分段处理时，如何确定有分段使运行效果最优，还有待进一步研究。

参考文献

[1] AGRAWAL R,IMIELINSKI T,SWAMI A. Database mining:a performance perspective[J]. IEEE Transactions on Knowledge and Data Engineering,1993,5(6):914-925.

[2] AGRAWAL R,SRIKANT R.Fast algorithms for mining association rules in large databases[C]. In Proc.Of the 20th Int.Conf.on Very Large Data Bases(VLDB),Santiago,Chile,Septemer, 1994(2):478-499.

[3] HAN Jiawei,MICHELINE K.数据挖掘概念与技术[M].北京：机械工业出版社,2006.

[4] PARK J S,CHEN M S,YU P S.Using a hash-based method with transaction trimming for mining association rules[J].IEEE Trans on Knowledges Data Engineering,1997,9(5):813-825.

[5] 向程冠,姜季春,陈梅,等.AprioriTid 算法的改进[J].计算机工程与设计,2009,30(15):3581-3583.

[6] 胡斌,蒋外文,蔡国民,等.基于位阵的更新最大频繁项集算法[J].计算机工程,2007,28(2):59-61.

[7] 王锋,李勇华,毋国庆.基于矩阵的改进的 Apriori 算法[J].计算机工程与设计,2009,30(10):2435-2438.

[8] Liu Yongmei,Guan Yong.FP_growth algorithm for application in research of market basket analysis[J].Computational Cybernetics,2008.ICCC 2008.IEEE International Conference on,2008: 269-272.

(收稿日期：2010-07-28)

作者简介：

粟莉萍，女，1985年生，硕士研究生，主要研究方向：网络与分布式信息处理。

杨文伟，男，1958年生，副教授，硕士生导师，主要研究方向：计算机网络与分布信息处理。