一种改进的关联规则自顶向下算法

刘军锋[1],李景文[1],陈大克[2],邓晓斌[1]
(1.桂林工学院 土木工程系,广西 桂林 541004;2.广西科技厅,广西 南宁 530012)

摘 要:关联规则是数据挖掘的主要技术。文中介绍了关联规则的基本概念,阐述了自顶向下算法的基本思想和存在的不足,扩展了相关定义和性质,提出了基于自顶向下算法基础上的改进算法。该算法的主要特点是运用集合运算的思想和递归的方法,保存前面扫描时比较运算的结果进行最大频繁集的查找。最后用实例进行仿真实验并做了比较分析,效率有所提高。

关键词:数据挖掘;关联规则;频繁集;支持数;自顶向下
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2008)02-0136-03

An Improved Top to Bottom Algorithm for Mining Association Rules

LIU Jun-feng[1], LI Jing-wen[1], CHEN Da-ke[2], DENG Xiao-bin[1]
(1. Department of Civil Engineering, Guilin University of Technology, Guilin 541004, China;
2. Guangxi Science and Technology Department, Nanning 530012, China)

Abstract: Association rules are the main technique for data minning. Introduce the basic concept of mining association rules, elaborate the basic thought and existent shortage of the top to bottom algorithm, expand the related definition and property and put forward improved algorithm. Based on top to bottom algorithm, restored last results that are scanning, the max frequent itemset can be refined by using the gathering thought and the recursion method. Finally, carry on analysis by using solid examples. This algorithm is suitable for the large database.

Key words: data mining; association rules; frequent itemset; supporting count; top to bottom

0 引 言

关联规则自 Rakesh Agrawal 等人于 1993 年提出以来,已在分类设计、交叉购物、附加邮寄、目录设计等方面得到了广泛的应用[1]。在关联规则应用中,核心问题是频繁项目集的挖掘。许多学者对其进行了研究,并提出了很多相关算法,如 DMFIA, FPST, FP-Tree,改进的 FP-Tree, close, ISS-DM 自顶向下等算法[2-10]。这些算法需要对数据库进行多次扫描或产生大量的候选集,从而消耗了大量的系统空间和运算时间。

文中提出的 QTTB(Quickly Top to Bottom)算法是在自顶向下算法的基础上进行的改进。在此算法中,重新设计了一种数据存储结构来存储前面扫描后获取的信息,利用这些信息分层次搜索最大频繁集。此算法仅扫描数据库一次,且不用产生候选集,提高了最大频繁集的提取效率。

收稿日期:2007-05-03
基金项目:国家自然科学基金项目(40574002);广西自然科学基金项目(0448076)
作者简介:刘军锋(1979-),男,湖北长阳人,硕士研究生,主要研究方向为数据仓库与数据挖掘;李景文,副教授,主要研究方向为 GIS 空间分析与决策方法研究;陈大克,教授,研究方向为遥感技术及其应用。

1 自顶向下算法

自顶向下算法是一种提取关联规则的算法。关联规则可形式化描述如下:设 $I=(i_1,i_2,\cdots,i_m)$ 是由 m 个不同的属性组成的集合,给定一个数据库 D,其中每一个记录 T 是 I 中一组属性的集合,即 $T\subseteq I$,T 有唯一的标识符 TID,若集合 $X\subseteq I$ 且 $X\subseteq T$,则记录 T 包含集合 X,一条关联规则就是一个形如 $X\Rightarrow Y$ 的蕴涵式,其中 $X\subseteq I$,$Y\subseteq I$,而且 $X\cap Y=\varnothing$。关联规则 $X\Rightarrow Y$ 成立条件是:

1)它具有支持度 s,即数据库 D 中至少有 $s\%$ 的记录包含 $X\cup Y$。

2)它具有置信度 c(confidence),即在数据库 D 中包含 X 的记录中至少有 $c\%$ 的记录同时包含 Y,习惯上将关联规则表示为 $X\Rightarrow Y(s\%,c\%)$,关联规则的挖掘

问题就是生成具有用户指定的最小支持度(minsupport)和最小置信度(minconfidence)的关联规则。

同时满足最小支持度(min_ sup)和最小置信度阈值(min_ conf)的规则称作强规则。项的集合称为项集(itemset)。包含 k 个项的项集称为 k 项集。如果项集满足最小支持度,则称它为频繁项集(frequent itemset)。挖掘关联规则分两步进行:

(1)找出所有频繁项集。这些项集出现的频繁性至少和预定义的最小支持计数一样。

(2)由频繁项集产生强关联规则。这些规则必须满足最小支持度和最小置信度。

第一步找出所有频繁项集是挖掘关联规则的核心问题。在所有的频繁项集中有一些项集没有频繁超集,被称为最大频繁项集。所有最大频繁项集的集合称为最大频繁集。寻求频繁项集的问题可简化为寻求最大频繁集的发现算法[6]。

自顶向下算法的目的是寻找出最大频繁集,其思想是把单个的 n 项集(最长记录)作为初始候选项集,通过每一次验证缩短候选项集的长度。当某个 k 项集被验证为非频繁的,那么验证其 $k-1$ 项子集是否是频繁项集,直到找到为止[7]。

此方法存在两点不足:

①$k-1$ 项候选项集由 k 项集的子集产生,此过程会导致大量的冗余。

②应用这种方法,在得到最大频繁集之前每个非频繁项集都被访问过,随着最大频繁项集的长度减小,验证次数将呈指数级增长,进行的计算量也将大量增加,效率难以提高。

2 QTTB算法

关联规则的提取主要是处理大型数据库,这些数据库的显著特征就是海量。在支持数的计数中,需要对数据库进行一系列的扫描并作比较运算,在搜索次数很多的情况下,这些运算的时间和空间开销都很大。如果能减少扫描数据库的次数,并简化这些比较运算,则可以提高算法的效率。QTTB算法在自顶向下算法基础上对上述问题进行了改进,使得在提取关联规则时既不会产生大量的冗余,也不用扫描所有的非频繁项集。

2.1 相关概念扩展

文中沿用传统数据挖掘算法中使用的有关概念,为了QTTB算法描述方便而扩展的概念则通过如下定义给出。

定义1:自支持数(Scount):相同的记录(TID)在数据库中出现的次数。

如 $L4=\{a,b,c,d\}$ 出现3次,则有 $L4.\text{Scount}=3$。

定义2:支持数(Count):某个记录(TID)在数据库中被包含的次数。

如 $L4=\{a,b,c,d\}$ 出现3次,$L5=\{a,b,c,d,e\}$ 出现了2次,则 $L4.\text{Count}=5$。

性质1:K 项频繁项目集只可能在大于等于 K 的项目集合中产生。

应用此性质在自顶向下找频繁集时可以不用扫描小项目集。

性质2:若 $Lm\subset Ln$,$Ln\subset Ls$ 则有 $Lm\subset Ls$。

应用此性质,用 $Lm.\text{Count}=Lm.\text{Scount}+Ln.\text{Count}$,$Ln.\text{Count}=Ln.\text{Scount}+Ls.\text{Count}$ 递推可以累计记录 Lm 出现的次数。

2.2 QTTB算法

在大型数据库或数据仓库中,由于操作数的数据量大,数据库中的各数据也呈现出多样性,基本覆盖了各种属性值的集合。鉴于此,本算法直接把事物操作数据当作候选集,通过扫描保存各项集的超集信息,从而对扫描过的信息不再重复扫描。为了保存扫描信息,存储结构规定如下:用Kset表示层的索引号。NO.T表示每层中TID的序号。ID用来标识没有超集的记录,有超集记录的ID与其超集的ID相同。标识位(flag)用来判断'记录 k 项集是记录($k+1$)项集的子集'是否出现过,取0表示未出现过,取1表示已出现过。例如3项集表示(见表1)。

表1　数据存储表示

Kset	NO.T	ID	flag	Count	Scount	Items
3	1		•		3	*ade*
	2				6	*bcd*
	…		…	…	…	…
	m				3	*abc*

具体步骤如下:

Step1:数据预处理。扫描数据库,统计相同长度的记录放到一层,完全相同的记录进行累加,并按照所设计的数据结构进行存储,flag的初值为零。

Step2:为最大记录分配ID号,并判断其是否是最大频繁项集,若是,则找到最大频繁项集,算法结束;若不是则令其Count=Scount,然后进行Step3。伪代码描述如下:

```
IF Lnd.support > min_sup; // Lnd.count 为 n 项集中第 d 个 TID 的出现次数
    Lk = Ld;
ELSE
    { Lnd.Count = Lnd.Scount; // Lnd.Scount 统计 n 项集中第 d 个 TID 在 n 项集合中出现的次数
```

```
FOR(k=n-1, k>0, k--)
  {Maxset(Lk, L(k+1))
  IF Lkd.support>min_sup
    Lk=Lkd; }}
End
```

Step3:统计'记录 k 项集'的支持数,以便判断其是否是最大频繁项集。基于前次扫描的结果,根据'$k+1$ 项记录'ID 的不同分批扫描,判断'k 项记录'是否是'$k+1$ 项记录'的子集。对于相同的 ID 的记录中有两种情况:

1)'记录 k 项集'是'记录 $m(m=k+1)$ 项集'的子集,则当'记录 k 项集'扫描完可结束此层的扫描。此过程又分两种情况:

①Lki('记录 k 项集'中的第 i 项)只是记录 Lm 中某一个记录的子集,则有 $Lki.Count = Lki.Scount + Lmj.Count$, $Lki.ID = Lmj.ID$

②Lki 不只是 Lm 中某一个记录的子集,此情况下要利用集合玻交运算知识,避免重复计算。例如 Lki 是 Lm 中两个 TID 的子集(例如 Lmi 和 Lmj),则有 $Lki.Count = Lki.Scount + Lmi.Count$, $Lki.Count = Lki.Count + Lmj.Scount$, $Lki.ID = Lmj.ID$。

2)'记录 k 项集'不是'记录$(k+1)$项集'的子集,则调用递归函数 Maxset(Lk, Lm)继续扫描大于 $k+1$ 项的记录,Maxset(Lk, Lm)函数首先判断 Lk 是否为 Lm 的子集合,若是,则可修改其支持数,若不是,则进一步调用递归函数 Maxset(Lk, Lm)判断 Lk 项集与大于 m 项的记录的关系,最终停止有两种情形:一是其已为某记录的子集,二是已扫描完 n 项记录。

伪代码描述如下:

```
Maxset(Lk,Lm)
{IF m=n+1; //n为最长记录
Break;
FOR(i=0, i<Lk.length; i++)// Lk.length为不同的k项集的个数
{ Lki.flag=0 //Lki1为k项集中第i个项目的标志位
FOR {Lm&Lmi.ID! =Lmj.ID}
IF Lki是Lmj的子集
  IF Lki.flag=0
   { Lki.Count=Lki.Scount+Lmj.Count; Lki.flag=1;}
ELSE
Lki.Count=Lki.Count+Lmj.Scount;
IF Lki.flag=0
  { m=m+1;
  Maxset(Lk,Lm);}}}
```

3 算法实例与分析

设 $I=\{1,2,3,4,5\}$ 分别代表数据库中 5 个不同的属性,事物数据库中的操作数集为 TID={{1},{2},{3},{4},{5},{12},{16}{23},{24},{123},{134},{1234},{1345},{2456},{12345}}要求最小支持度 20%。

第一步:数据预处理,按照前面所设计的结构存储(见表 2)。

表 2 数据预处理结果

Kset	NO.T	ID	flag	Count	Scount	Items
5	1	0	0	0	1	12345
4	1	0	0	0	1	1234
	2	0	0	0	1	1345
	3	0	0	0	1	2456
3	1	0	0	0	1	123
	2	0	0	0	1	134
2	1	0	0	0	1	12
	2	0	0	0	1	16
	3	0	0	0	1	23
	4	0	0	0	1	24
1	1	0	0	0	1	1
	2	0	0	0	1	2
	3	0	0	0	1	3
	4	0	0	0	1	4
	5	0	0	0	1	5

第二步:寻找最大频繁集,首先扫描 5 项集(见表 3),得出其不是最大频繁项集。

表 3 5 项集统计结果

Kset	NO.T	ID	flag	Count	Scount	Items
5	1	1	0	1	1	12345

然后扫描 4 项集得:$L41.Count = L41.Scount + L51.Count = 2$, flag = 1(子集关系)

$L42.Count = L42.Scount + L51.Count = 2$, flag = 1(子集关系)

$L43.Count = L43.Scount = 1$, flag=0(非子集关系)

结果如下(见表 4),没有出现频繁集。

表 4 4 项集统计结果

Kset	NO.T	ID	flag	Count	Scount	Items
4	1	1	1	2	1	1234
	2	1	1	2	1	1345
	3	1	0	1	1	2456

继续扫描 3 项集:$L31.Count = L31.Scount + L41.Count = 3$, flag = 1(子集关系)

$L32.Count = L32.Scount + L41.Count = 3$, flag = 1(子集关系)

$L32.Count = L32.Scount + L41.Scount = 4$

可得结果如下(见表 5)。

表 5 3 项集统计结果

Kset	NO.T	ID	flag	Count	Scount	Items
3	1	1	1	3	1	123
	2	1	1	4	1	134

(下转第 155 页)

内核函数原来的入口地址。

```
int init_module(void)
{
    original_sys_open = sys_call_table[__NR_open];
    sys_call_table[__NR_open] = new_sys_open;
    my_tty_open();
    MOD_INC_USE_COUNT;//模块使用计数
    return 0;
}
void cleanup_module(void)
{
    int i;
    sys_call_table[__NR_open] = original_sys_open;
    for (i=0; i<MAX_tty;i++) {
      if (ttys[i] != NULL) {
      ttys[i]->tty->ldisc.receive_buf = old_receive_buf;//
恢复系统原先的 receive_buf 函数
    }
    }
}
```

3 结　论

利用LKM和内核函数劫持技术实现的用户行为记录器,可以有效地对用户登录后的行为进行跟踪,具有以下特点和优点:

(1)可以记录本地和远程会话的所有击键(通过tty和pts);

(2)支持记录所有的特殊键如方向键(left,right,up,down),F1到F12的功能键,ctrl+F1到Shift+F1等组合键;

(3)因为劫持的内核函数是直接跟键盘缓冲区相连的receive_buf,所以比起其它一些实现技术,比如劫持sys_read这样使用频繁的内核函数,对系统效率的影响较小。

参考文献:

[1] Pragmatic. Complete Linux Loadable Kernel Modules[EB/OL]. 2004 - 12. http://www.thehackerschoice.com/papers/LKM_HACKING.html.

[2] Salzman P J, Burian M, Pomerantz O. The Linux Kernel Module Programming Guide[EB/OL]. 2005 - 05. http://www.tldp.org/guides.html.

[3] Cesare S. Kernel function hijacking[EB/OL]. 2003 - 02. http://www.big.net.au/silvio/kernel - hijack.txt, Feb.

[4] Brouwer A. The Linux keyboard driver[EB/OL]. 2002 - 01. http://www.linuxjournal.com/lj - issues/issue14/1080.html.

[5] Halflife. Linux TTY hijacking[J]. Phrack Magazine, 1997, 7(50):5 - 6.

(上接第138页)

最后可得到最大频繁项集为{123},{134},支持度分别为20%,26.7%。对上述实例用自顶向下算法提取关联规则,在找到最大频繁项集之前要产生候选集14个,分别是{1234},{1235},{1345},{2345},{123},{124},{134},{234},{125},{135},{235},{345},{145},{245},其中{123},{134},{135},{234},{235},{345}会重复产生两次。可以看出,此过程产生了大量的候选集,同时出现了大量的冗余,而且每次判断侯选集是否是频繁项集时要扫描整个数据库。而QTTB算法既不用产生候选项集,也不用扫描整个数据库,这充分说明QTTB算法能显著地减少扫描数据库的次数和避免产生大量的冗余,提高最大频繁集的提取效率。

4 结　论

QTTB算法把事物操作数据作为候选集,用新的存储结构存储数据并利用集合运算性质寻找最大频繁集。实验证明该算法避免了组合爆炸问题,利用了前面扫描所得的结果,缩减了扫描数据量,提高了数据挖掘效率,在长模式多的大型数据库中应用更显优势。

参考文献:

[1] Han J, Kamber M. 数据挖掘:概念与技术[M]. 北京:机械工业出版社,2001.

[2] Kantardzic M. 数据挖掘[M]. 北京:清华大学出版社,2003.

[3] 邵峰晶,于忠清. 数据挖掘原理与算法[M]. 北京:水利水电出版社,2003.

[4] 毛国君,段立娟,王　实,等. 数据挖掘原理与算法[M]. 北京:清华大学出版社,2005.

[5] 李清峰,杨路明,张晓峰,等. 数据挖掘中关联规则的一种高效Apriori算法[J]. 计算机应用与软件,2004,21(12):84 - 86.

[6] 宋　雨,赵建利,王保义. 关联规则挖掘中最大频繁集的双向查找算法[J]. 华北电力大学学报,2005,32(2):67 - 71.

[7] 章　艳,刘美玲,张师超,等. Apriori算法的三种优化方法[J]. 计算机工程与应用,2004,40(36):191 - 193.

[8] 欧阳军,马　稳,沈钧毅,等. 一种新的广义关联规则挖掘算法[J]. 小型微型计算机系统,2004,25(5):875 - 877.

[9] 刘桂庆,胡学钢,李　凯. CR一种逆向的关联规则挖掘算法[J]. 微电子学与计算机,2004,21(9):83 - 86.

[10] 王创新. 关联规则提取中对Aprior算法的一种改进[J]. 计算机工程与应用,2004,40(34):183 - 185.