第16卷 第4期
2006年4月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol. 16 No. 4
Apr. 2006

挖掘关联规则中 Apriori 算法的研究与改进

胡吉明，鲜学丰
（河海大学 计算机及信息工程学院，江苏 南京 210098）

摘 要：关联规则是数据挖掘中重要的研究课题。对关联规则挖掘算法 Apriori 算法的关键思想以及性能进行了研究，指出该算法的某些不足，并且提出了一种产生候选项集的新算法。该算法提高了原算法的效率。

关键词：数据挖掘；关联规则；频繁项集
中图分类号：TP301.6　　**文献标识码**：A　　**文章编号**：1005－3751(2006)04－0099－03

Research and Improvement on Apriori's Algorithm in Mining with Association Rules

HU Ji-ming, XIAN Xue-feng
(College of Computer & Information Engineering, Hohai University, Nanjing 210098, China)

Abstract: Association rule is an important and issue in data mining. Based on the study of the principle and efficiency of the Apriori algorithm, point out the defect and present a new algorithm that the efficiency of mining with association rules could be improved through employing a new method of producing candidate set.

Key words: data mining; association rules; frequent item－set

0 前 言

关联规则是 Agrawal 等人[1]提出的数据挖掘领域中的一个重要课题。关联规则揭示数据间的相互关系，关联规则挖掘试图从一组给定的数据项以及事务数据库（每个事务是一个数据项的集合）中，筛选出数据项集在事务数据库中出现的频度关系。关联规则挖掘可以发现大量数据中数据项集之间有价值的关联或相关联系。例如，从大量商务事务记录中筛出的关联关系，可以用于购物篮分析；而且，关联规则挖掘还可用于其他的一些应用，例如诊断系统、决策支持系统、电信系统、分类分析系统等。

1 关联规则挖掘的描述

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是所有数据项的集合，设 $D=\{T_1,T_2,\cdots,T_n\}$ 为所有事务的集合，即一个事务数据库，其中的每个事务 T 是一个数据项子集，即 $T\subseteq I$。每个事务可以用惟一的标识符 TID 来标识。

设 X 为一个数据项集合，称为项集，包含 k 个数据项的项集称为 k－项集。当且仅当 $X\subseteq T$ 时，称为事务 T 包含 X。

事务数据库 D 中数据项集 X 的支持度为：$\text{support}(X)=P(X)=|T_X|/|D|$，其中 $T_X=\{T\in D\mid X\subseteq T\}$。在用户给定的支持度阈值 minisupport 下，若 $\text{support}(X)\geqslant \text{minisupport}$，则称 X 为频繁项集，否则 X 为非频繁项集。但为下文便于叙述，数据项集 X 的支持度是用 D 中包含 X 的事务数量来表示。

一个关联规则是“$X\Rightarrow Y$”形式的蕴含式，其中 $X\Rightarrow I$，$Y\Rightarrow I$ 且 $X\cap Y=\varnothing$。如果 D 中包含事务 $X\cup Y$ 的百分比为 s，则称 s 为关联规则 $X\Rightarrow Y$ 的支持度，它是概率 $P(X\cup Y)$。如果 D 中包含 X 的事务同时已包含 Y 的百分比为 c，则称 c 为关联规则 $X\Rightarrow Y$ 的信任度，它是条件概率 $P(Y\mid X)$。即：

$$\text{support}(X\Rightarrow Y)=P(X\cup Y)=\text{support}(X\cup Y)$$

$$\text{confidence}(X\Rightarrow Y)=P(Y\mid X)=\text{support}(X\cup Y)/\text{support}(X)$$

挖掘关联规则的问题就是要生成所有满足 $\text{support}(X\Rightarrow Y)\geqslant \text{minisupport}$ 和 $\text{confidence}(X\Rightarrow Y)\geqslant \text{miniconfidence}$ 的关联规则，其中 minisupport 和 miniconfidence 分别为用户给定的最小支持度阈值和最小信任度阈值。同时满足这两个条件的关联规则称为强关联规则。

挖掘关联规则主要包含两个子问题：

①从事务数据库 D 中生成所有的频繁项集；

②根据获得的频繁项集生成强关联规则。

事实上，挖掘关联规则的整个执行过程中第一个子问题是核心。当找到所有的频繁项集后，相应的关联规则将很容易生成，Apriori 算法主要是处理第一个子问题。

收稿日期：2005－08－22
作者简介：胡吉明（1960－），男，浙江绍兴人，副教授，硕士生导师，研究方向为 High Availability Intranet Web GIS 应用，并行处理技术。

2 Apriori 算法及其思想

2.1 Apriori 算法的思想

在已有的关联规则发现算法中,最著名的是 Agrawal 等人[1]于 1993 年提出的 Apriori 算法。Apriori 算法是一种宽度优先算法,算法步骤如下:

① 扫描事务数据库 D,对遇到的每个事务分析其中出现的数据项,如果第一次遇到该数据项,则加入候选 1 - 项集的集合 C_1,并将它的计数值设置为 1;如果该数据项已加入 C_1,则将它的计数值加上 1,这样就得到了候选 1 - 项集的集合 C_1(其中每个数据项集只包含一个数据项)。扫描 C_1,删除那些出现计数值小于 minisupport 的项集,这样就得到了 1 - 频繁项集的集合 L_1。

② 一般地,假设 L_{k-1} 已生成,现在可用它来生成 L_k,L_{k-1} 与自身进行连接(L_{k-1} 中的每个项集与其他项集相互连接),得到候选 k - 项集的集合 C_k。

③ 对 C_k 进行剪枝,从 C_k 中删除所有($k-1$) - 子集不全包含在 L_{k-1} 中的项集。

④ 扫描数据库事务 D,对于其中的每一个事务,如果它包含 C_k 中的候选项集 c,则将 c 的计数值加 1(在扫描之前,初始值为 0)。扫描 C_k,删除那些出现计数值小于给定支持度的项集,这样就得到了 k - 频繁项集的集合 L_k。

⑤ 重复 ② 到 ④,直到 L_k 为空。

⑥ 对 L_1 到 L_k 取并集即为最终的频繁集 L。

2.2 Apriori 算法

下面给出 Apriori 算法[1~3]。算法的第一步仅仅对所生成的 1 - 项集计数,以生成 1 - 频繁项集的集合 L_1。接下来的第 k 步,由两个部分构成。首先,在第 $k-1$ 步找到 L_{k-1},用于生成 C_k,生成方法是使用后面所描述的 apriori - gen 函数。然后,扫描数据库,算出 C_k 中候选项的支持度。

1) L_1 = {frequent 1 - itemset}

2) for ($k = 2; L_{k-1} \neq \emptyset; k++$) do begin

3) C_k = apriori - gen(L_{k-1}); //生成所有长度为 k 的候选项集

4) for each transaction $t \in D$ do begin

5) C_t = subset(C_k, t); //t 中包含的候选项集

6) for each candidate $c \in C_t$, do

7) c.count ++

8) end

9) $L_k = \{c \in C_k \mid c.\text{count} >= \text{minisupport}\}$

10) end

11) return($\bigcup_k L_k$)

Apriori 候选项生成:

apriori - gen 函数主要针对 L_{k-1}。该函数返回一个所有 k - 频繁项集的集合的超集。函数如下所示。首先,在联接步,将 L_{k-1} 与自身进行连接:

```
insert into C_k
select p.item1, p.item2, ..., p.item(k-1) > q.item(k-1)
from L_{k-1} p, L_{k-1} q
where p.item1 = q.item1, ..., p.item(k-2) = q.item(k-2), p.item(k-1) < q.item(k-1);
```

接着,在剪枝步,删除所有的项集 $c \in C_k$,这些 c 的一些($k-1$) - 子集就不出现在 L_{k-1} 中:

```
for all itemset c ∈ C_k do
  for all (k-1) - subsets s of c do
    if (s is not in L_{k-1}) then
      delete c from C_k
```

例如,设 L_3 为((1,2,3),(1,2,4),(1,3,4),(1,3,5),{2,3,4)) 联接步之后,C_4 则为(1,2,3,4),(1,3,4,5))。因为项集(1,4,5)不在 L_3 中,剪枝步将删除项集(1,3,4,5),那么留在 C_4 中的只有(1,2,3,4)。

3 Apriori 算法的缺陷及现有改进方法

3.1 Apriori 算法的缺陷

根据对 Apriori 算法的分析,关联规则挖掘可以比较有效地产生关联规则,但是也存在算法效率不高的严重缺陷[4]。

主要原因如下:

① 数据库扫描的次数过多,寻找每个 k - 频繁项集($k = 1,2,\cdots,k$) 都需要扫描数据库一次,共需要扫描 k 次。因此当数据库或者 k 太大时,算法的耗时将太大甚至无法完成。

② 算法的剪枝步,$\forall c \in C_k$,判断 c 的 k 个($k-1$) - 子集是否都在 L_{k-1} 中,若找到一个($k-1$) - 子集不在 L_{k-1} 中就淘汰 c。因为这个过程会多次扫描 L_{k-1},特别是当生成的 C_k 很大时,算法的效率并不理想。

3.2 现有改进方法

为了提高 Apriori 算法的性能,已经有许多变种对 Apriori 进一步改进和扩展。可以通过以下几个方面对 Apriori 算法进行改进[5]:

① 通过减少扫描数据库的次数改进 I/O 的性能。

② 改进产生频繁项集的计算性能。

③ 寻找有效的并行关联规则算法。

④ 引入抽样技术改进生成频繁项集的 I/O 和计算性能。

⑤ 扩展应用领域。如:定量关联规则、泛化关联规则及周期性的关联规则的研究。

比较有成效的是:由 Park 等人提出基于 Hash 的算法,用于有效地寻找频繁项集。由 Savasere 等人提出的基于划分的算法,将交易数据集划分为几个部分,每一个部分可以容纳于主存,每一部分独立地生成频繁项集。此方法是高度可并行的,只须将每一部分分配在每一个处理机上即可用于产生关联规则。文中主要是改进了算法的第二个缺陷,提出了一种产生候选项集的集合 C_k 的新算法,使剪枝步的计算量减小以提高算法的效率。

4　生成候选项集的集合 C_k 的新算法

4.1　算法思想

生成 C_k 的新算法的基本思想:在 L_{k-1} 中的一个项集 l 与 L_{k-1} 中所有项集进行连接,把连接得到的不同 k - 项集存入 TQ,然后立即确定包含项集 l 的所有符合剪枝后的候选 k - 项集。这样就省略了在 L_{k-1} 中寻找 k - 项集的所有 $(k-1)$ - 子集的费时剪枝操作,只要对包含项集 l 能产生的数据项集 TQ 进行扫描即可,因此使剪枝部分扫描的集合减小,从而减小了总的计算量,因而提高了算法效率。

定理 1:$\forall X \in Q$。如果 $|Q(X)| \neq k-1$,则 X 不是频繁项集。其中 Q 表示 L_{k-1} 中的一个项集 l 与 L_{k-1} 中所有 $(k-1)$ - 项集连接生成的 k - 项集的集合,$|Q(X)|$ 表示 X 在 Q 中出现的次数。

证明:假设 X 为 k - 频繁项集,则它的 k 个 $(k-1)$ - 子集均在 L_{k-1} 中。则在由 X 生成的 $(k-1)$ - 子集中,任何一个 $(k-1)$ - 子集与其他所有 $(k-1)$ - 子集相连接,X 必出现 $k-1$ 次(因为任何一个 $(k-1)$ - 子集是从 X 中删除一个数据项得到的,不同的 k 个 $(k-1)$ - 子集是通过删除不同的元素得到的,所以任何一个 $(k-1)$ - 子集定能从其他 $(k-1)$ - 子集中找到被删除的数据项)。X 是频繁项集均有 $|Q(X)| == k-1$,这与条件矛盾,所以 X 不是频繁项集。

推论 1:$\forall X \in C_k$,则必有 $|Q(X)| == k-1$,并且只有在它第一次出现的地方出现 $k-1$ 次。

证明:$\forall l \in L_{k-1}$,l 与 L_{k-1} 中所有 $(k-1)$ - 项集连接时,所有包含 l 的 k - 项集都会出现。根据定理 1,从中可以判断出包含 l 的所有候选 k - 项集。假设 X 是包含 l 的候选 k - 项集。为了优化 L_{k-1} 与自身进行连接操作,下次连接时就可以不再包含 l,又因为 l 不出现在下次连接中,所有同一个 X 不可能再出现 $k-1$ 次。由此推论 1 成立。

上述定理和推论说明这样一个事实:若 $\forall X \in Q$,如果 $|Q(X)| \neq k-1$,则 X 一定不是频繁项集。那么 X 就不应该包含在 C_k 中,所以在后面的算法中仅仅将 $|Q(X)| == k-1$ ($c.\text{count} == k-1$) 的 k - 项集加入 C_k。

产生 C_k 的新算法具体步骤如下:

① 在 L_{k-1} 中的一个项集 l 与 L_{k-1} 中 l 之后的所有项集进行连接。

② 将连接结果依次存入 TQ,存储机制是:在存储前首先与已经存入 TQ 的 k - 项集进行比较,如果与某项集相同,则相应 k - 项集计数值加 1;如果没有相同的 k - 项集,则直接存入,并设计数值初值为 1。

③ 把 TQ 中所有计数值为 $k-1$ 的 k - 项集加入 C_k。然后循环执行 ①,②,③ 步,直到 L_{k-1} 中每个项集都相互连接完为止。最终得到的 C_k 就是要生成的候选项集的集合。

4.2　算法实现

新的候选集的生成过程:apriori - gen(L_{k-1})

```
for each itemset l ∈ L_{k-1} do begin
    TQ = ∅; //存储临时候选项集的集合
  for each itemset {l_2 | l_2 ∈ L_{k-1}, l_2在 l 的后面} do begin
   c = l join l_2; //连接,产生候选项集
   if(length(c) > k ) then
     continue;
   else
     if(exists (TQ, c ) and q∈TQ and q = = c ) then //
如果 c 在 TQ 中已存在
     q.count + + ; //计数初值加 1
     else{ //c 在 TQ 中第一次出现
       add c to TQ;
       q= find(TQ , c );
       q.count =1; //计数初值为 1
     }
end
For each itemset c ∈TQ do
     If c. count= = k - 1 then //将在 TQ 中出现 k - 1 次
的项集加入到 C_k
     add c to C_k
end
return C_k;
```

5　算法性能分析

Apriori 算法的剪枝步,删除所有的项集 $c \in C_k$,这些 c 的一些 $(k-1)$ - 子集就不出现在 L_{k-1} 中,即 $\forall c \in C_k$,判断 c 的 k 个 $(k-1)$ - 子集是否都在 L_{k-1} 中,如果有其中一个不在,则将 c 从 C_k 中删除。利用上述方法对 C_k 过滤一遍得到一个相对较小的候选项集的集合 C_k。在此方法中,对于 c 在最好的情况下只需扫描一次,即扫描一次 L_{k-1},已就是说第一个被扫描的 $(k-1)$ - 子集不在 L_{k-1} 中;最坏的情况是直到第 k 个 $(k-1)$ - 子集不在 L_{k-1} 中或者它的 k 个 $(k-1)$ - 子集都在 L_{k-1};综合考虑每个项集的平均检查次数为 $|L_{k-1}| \times k/2$ 次。这个过滤过程中平均计算量为 $|C_k| \times |L_{k-1}| \times k/2$。新算法的剪枝步,从 TQ 中淘汰所有 $c.\text{count} \neq k-1$ 的数据项集 c 只需要扫描一次 TQ,而每次循环得到的 TQ 都是是 L_{k-1} 中的一个项集 l 与 L_{k-1} 中所有 $(k-1)$ - 项集连接得到的所有不同 k - 项集的集合。所以经过 $|L_{k-1}|-1$ 次循环得到的所有 TQ 组成的集合 $\sum$TQ 是 L_{k-1} 与自身连接生成的所有不同 k - 项集的集合。$\sum \text{TQ} \subset C_k$(Apriori 利用 L_{k-1} 与自身连接生成的 k - 项集的集合)。$|L_{k-1}|-1$ 次循环总的扫描集合的大小为 $|\sum \text{TQ}|$,所以新算法平均计算量是 $|\sum \text{TQ}|$。

由 $|\sum \text{TQ}| < |C_k|$ 和 $|C_k| < |C_k| \times |L_{k-1}| \times k/2$,得新算法剪枝平均计算量 $|\sum \text{TQ}|$ 远小于 Apriori 算法的

(下转第 104 页)

```
    private InpProd1Model inpProd1Model;
    public ActionErrors InsertSpot ( InpProd1ActionForm
inpProd1Form){
        if(inpProd1Form.getname() == null || inpProd1 Form.
getname().equals("")){
            errors.add("name",new ActionError("inpProd001"));
            return errors;
            }
        resList = inpProd1Model.querry2(inpProd1Form.getname
());
       if (! resList.isEmpty()){
           InpProd1Bean inpProd1Bean = (InpProd1Bean)resList.
get(0);
       }
    }
}
```

在 InpProd1Edit 中对数据进行了校验处理,当数据不符合要求时就给出一定的错误提示信息。当数据通过了合法性校验后调用了 inpProd1Model,进行数据库操作。

4.4 InpProd1Model 的部分代码

```
public class InpProd1Model extends Model {
    public final static String querrySQL = "select name from prod-
uct where Size = ?"
    public List querry1(String Size) {
        super.setSql(querrySQL);
        clearPara();
        addPara(strSize);
        return super.execQuery();
    }
    protected Object row2Bean ( ResultSet rowData ) throws
SQLException {
    InpProd1Bean data = new InpProd1Bean();
        data.setname(rowData.getString(1));
        return data;
    }
}
```

在 inpProd1Model 中执行了检索操作,并根据传递过来的检索条件操作后返回符合条件的结果集。

在 InpProd1Bean 和 InpProd1ActionForm 的代码中主要是定义了多个变量,并且给出了变量的 Get,Set 方法,有关详细的代码就不列出了。

5 总 结

此程序的完整代码在 WebSphere5.1.1 开发环境中已经调试运行通过。对上述代码进行分析可以得出如下结论:应用 Struts 框架开发 Web 程序无需自己编写控制器 Servlet,Struts 本身已经建好了,因此大大提高了开发的效率,并且将程序界面和逻辑处理分开进行编码,这样就使程序的可维护性和可重用性得到了提高[5]。上例对 Action 进行了更一步的细化,将数据的校验部分和数据库处理部分从主要逻辑处理中分离出来,更增强了系统的灵活性和可伸缩性,使开发人员的分工更具体,极大地提高了程序的重用性,从而体现出利用框架进行 Web 应用程序开发的巨大优势,会在今后的程序开发中得到更加广泛的应用。

参考文献:

[1] 寇 毅,吴力文.基于 MVC 设计模式的 struts 框架的应用方法[J].计算机应用,2003,23(11):91-93.

[2] 张 杰.基于 struts 的 Web 应用程序设计[J].现代图书情报技术,2004(2):33-36.

[3] 冀旭钢.利用 struts 框架进行 Web 应用开发的研究[J].微机发展,2005,15(6):121-123.

[4] 徐云青,诸葛理绣.扩展 struts 框架的设计模式[J].微机发展,2005,15(5):80-83.

[5] 丁振国,任新杰.基于 struts 的 Web 应用开发研究[J].微机发展,2004,14(1):90-92.

(上接第 101 页)

对应平均计算量 $|C_k| \times |L_{k-1}| \times k/2$,所以新算法的效率比 Apriori 算法有了很好的改进。

6 结束语

提出了一种新算法产生候选集,通过改进剪枝部分被扫描项集的大小,从而使剪枝步的平均扫描量大为减少,因此提高了数据挖掘的效率。通过实验,在 Windows professional 2000 环境下利用 Visual C++ 6.0 实现了该算法,运行结果表明了该算法的有效性。

参考文献:

[1] Agrawal R, Imielinski T, Swanmi A. Mining association rules between sets of items in large database[A]. In Proc of the ACM SIGMOD Conference on Management of Data[C]. Washington DC: SIGMOD, 1993.207-216.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules[A]. In: Proceedings of the 20th International Conference on Very Large Databases[C]. Santiago, Chile: [s.n.], 1994. 487-499.

[3] 陆丽娜,陈亚萍.挖掘挖掘关联规则中 Apriori 算法的研究[J].小型微型计算机系统,2000,21(9):940-943.

[4] 罗 可,吴 杰.一种基于 Apriori 的改进算法[J].计算机工程与应用,2001,5(22):20-22.

[5] 郑丽英.基于 trie 的关联规则发现算法[J].兰州理工大学学报,2004,30(5):90-92.