一种新的改进的先验算法

夏 磊

（南昌工学院信息中心 江西 南昌 330108）

[摘 要] 先验算法是数据挖掘关联规则中的经典算法。因为先验算法需要多次扫描数据库，所以运行速度太慢。为了提高运行效率，本文在先验分析的基础上对先验算法进行了改进。改进的思想是将事务数据库转换成相应的0-1矩阵，将它的每个向量和随后的向量进行内积运算以获得支持，与给定的MinSupport相比，如果向量小于MinSupport，则会删除行和列，从而减小分级矩阵的大小，提高运行速度。由于改进后的算法在运行时只需扫描一次数据库，因此运行速度更快。实验表明，该改进算法是有效可行的。

[关键词] 先验关联规则；改进的先验算法；数据挖掘

相关设计介绍

目前，对关联规则挖掘问题的算法研究较多。大量数据项集之间存在相互关系，可以检测到一些异常[1]。关联规则也被称为关联模式，它根据事务中某些项的出现来诱导其他某些项的出现。同一事务中的关联规则问题可以描述为：

设$I=\{i_1,i_2,\cdots,i_n\}$是元素集合，$D=\{t_1,t_2,\cdots,t_n\}$是数据库中所有事务的集合，每个事务只由标识符tid标识。假设A是一个项目集，并且仅当A=>B的时候，事务B包含事务A，关联规则的形式可以表示为：A=>B可以表示在$A\subset B$且$A\cap B=\varnothing$的情况下，一个事务的发生可以诱导其他事务也发生在该事务中，关联规则有两个重要的阈值支持sup和信赖con。使用两个阈值来约束规则A=>B。支持sup表示d包含事务$A\cup B$，信任conf表示d如何包含事务a

$$conf(A\Rightarrow B)=p(A|B)=\frac{supprotsont(A\cap B)}{supportcont(A)}$$，其中sup port_cont（A）表示事务$A\cup B$包含的数目，sup port_cont（A）包含项集A的事务编号。

最小支持minsup表示项目集在统计意义上是最小的重要性。最小置信minconf表示规则的最小可靠性。如果数据项满足$X.sup\geqslant min\,sup$，X被称为大数据集。最低限度的支持和信心是由用户提供的。

强规则表示它满足了最小支持和最小信任的需要，称为强规则。频繁项集：当项集的支持大于或等于给定的最小支持时，该项X.sup集被称为频繁项集。对于给定的事务D，挖掘规则是查找支持的关联规则和大于给定最小支持和最小信任的用户的置信度。

先验算法

先验算法。1993年，数据库专家阿格拉瓦提出了一种先验算法，该算法是挖掘布尔关联规则频繁项集的基本算法，也是最有影响的挖掘关联算法。

利用先验算法嵌入关联规则挖掘的步骤主要分为两步：第一步，从数据库或仓库中查找所有频繁项集；第二步，根据频繁项集生成关联规则。第二步相对容易实现，但第一步相对难分两步实现。但是，挖掘关联规则的总体性能主要取决于第一步。目前，大多数研究主要集中在第一个问题上。

先验属性：

（1）a11的常数项非空子集必须是常数；

（2）如果K维数据项集X_k的任何一个K-L维子集X_kUX不是频繁项集，那么X_k也不是频繁项集。

（3）X_k是K维度常见项目，如果所有的设置F-K-L维度常见项目集包含K-L维度子数小于K，那么X_k可以是K维度常见项目。

证明：数据项集合X_k的k-l维度子集的数目为k-l。如果所有集合k-l维数频繁项集包含k-l维数子集数小于k，则X_k的k-l项集不频繁，由结论（2）可以看出k维数数据项集本身也是频繁项集。

该算法通过连接和修剪步骤搜索频繁项集。

瓶颈问题或先验算法。开放式算法在数据挖掘研究中具有里程碑式的作用，是一种经典的关联规则挖掘算法，但先验算法的执行效率非常理想。特别是对于大型数据库的操作，这个问题变得更加突出。先验算法中有两个主要的瓶颈问题。

首先，它可能产生大量的候选项集。原因是频繁的k-1项集在连接操作时会产生更大的k项集，候选k项集来自Lk-1以指数方式增长，大量候选项集需要花费大量时间来验证项集是否频繁。其次，在计算项目集的支持度时，需要反复扫描事务数据库D，扫描过程会大大增加系统的开销，数据库的大小会影响算法的性能。

改进的开放算法

定义1.由向量空间定义线性代数内积，任意二维矢量$\alpha=(x_1,x_2,\cdots,x_n)$和$\beta=(y_1,y_2,\cdots,y_n)$，$\alpha$和$\beta$的内积定义为$\alpha\cdot\beta=\sum_{i=1}^{n}x_iy_i$，连接的改进思想是当k-1项集生成k项集并实现L（k-1）连接本身，如果两个项目集的k-2项不同，则给出连接操作。从而减少了机器的计数，提高了算法的运行速度。

定义2.事务数据库$D=\{t_1,t_2,\cdots,t_n\}$，M项$I_j(j=1,2,\cdots,m)$，和n×m矩阵A，$A(D)=a_ijn\times m$，是这样定义的$a_{ij}=\begin{cases}1 & I_j\in T_i\\ 0 & I_j\notin T_i\end{cases}(i=1,2,\cdots m)$。取矩阵A（D）称为事务数据库D对应的0-1矩阵。假设事务数据库D有五个事务记录，即如表1所示。d对应的0-1矩阵如图1所示。

表1 事务数据库D

T_{id}	Items
T_1	I_1,I_2,I_3
T_2	I_2,I_3,I_4,I_5
T_3	I_1,I_2,I_3,I_5
T_4	I_2,I_3,I_4,I_5
T_5	I_1,I_3,I_5

$$\begin{bmatrix}1&1&1&0&0\\0&1&1&1&1\\1&1&1&0&1\\0&1&1&1&1\\1&0&0&0&1\end{bmatrix}=\begin{bmatrix}a1\\a2\\a3\\a4\\a5\end{bmatrix}$$

图1 对应0-1矩阵Ai

候选集和频繁集的改进算法如下：对于给定的事务数据库D，将其转换为相应的0-1矩阵Ai。可以计算出每一项支持的D列SUP。矩阵的每个向量由后面的向量实现内积运算，以及确定产品价值是否大于或等于最小支持。如果大于或等于最小支撑，则转到步骤（3）。如果两个项目集的顶部（K-2），则执行LK-KX_{11}的连接操作。

两者都相等，然后需要对连接操作生成的候选项集执行修剪操作，方法是对矩阵进行扫描，判断候选对象是否频繁。如果不频繁，直接删除，得到LK，删除冗余事务，得到AK。

重复步骤（3），直到无法生成新项集，可以获得所有频繁的项集。

结论

因为原始算法一旦产生频繁的项集就需要扫描数据库。如果数据库很大，那么传统的先验算法的运行时间将保持较长的运行时间。然而，改进的算法在修剪时与矩阵有关，改进后的算法只扫描一次数据库，节省了扫描时间，提高了运行速度。