基于垂直数据分布的关联规则高效发现算法*

欧阳为民[1] 蔡庆生[2]

[1](安徽大学计算中心 合肥 230039)
[2](中国科学技术大学计算机系 合肥 230027)

摘要 文章分析了在KDD研究中现有的关联规则发现算法关于频繁项目集的生成与测试方法,提出了一种新的基于垂直数据分布的关联规则发现算法.该算法无需复杂的Hash数据结构,仅需对整个数据库作两次遍历,从而既方便了实现,又提高了效率.

关键词 关联规则,频繁项目集,等价类.

中图法分类号 TP311

关联规则是R. Agrawal等人[1,2]首先提出的KDD研究的一个重要课题.它可作如下形式化定义:令$I=\{i_1, i_2, \ldots, i_m\}$为项目集,$D$为事务数据库,其中每个事务$T$是一个项目子集($T\subseteq I$),并另有一个唯一的顾客标识符TID.我们说事务$T$包含项目集$X$,如果$X\subseteq T$.关联规则是形如$X\Rightarrow Y$的逻辑蕴含式,其中$X\subset T$,$Y\subset T$,且$X\cap Y=\varnothing$.如果事务数据库中有$s\%$的事务包含$X\cup Y$,那么,我们说关联规则$X\Rightarrow Y$的支持为$s$;如果事务数据库中包含$X$的事务中有$c\%$的事务同时也包含$Y$,那么,我们说关联规则$X\Rightarrow Y$的信任为$c$.

文献[2~7]分别提出了多种关联规则发现算法,其中最著名的是R. Agrawal提出的Apriori算法[2].大多数算法的主要缺点是要对数据库作多次遍历,从而导致较大的I/O负载.而且,这些算法都使用了Hash数据结构,对其进行维护与搜索又要增加额外的负载.Partition算法[7]虽然只需对数据库作两次遍历,但随着划分块数量的增加,局部频繁集的数量也将相应增加.DIC(dynamic itemset counting)算法[6]可以在一次遍历中对长度不同的若干项目集进行计数,从而提高了每次遍历的效率.但它实现起来不甚方便,而且仍然需要Hash数据结构.我们的目标是,算法既要高效,实现起来又要方便.为此,我们提出了等价类概念,并将数据库的数据分布由通常的水平方式改为垂直方式,从而提出基于垂直数据分布的关联规则发现算法.

本文第1节提出项目集聚类技术——等价类概念.第2节讨论数据的分布方式.第3节提出基于垂直数据分布的关联规则高效发现算法.第4节是实验结果.最后总结全文.

1 项目集聚类

考察集合$\{A,B,C,D,E\}$,其所有子集构成的子集空间如图1所示.注意,应忽略所有空集.频繁集以实线框表示,最大频繁集(某频繁集是最大的,如果它不是任何其他频繁集的子集)以粗实线框表示,非频繁集以虚线框表示.某频繁集的子集必是频繁的,所有的频繁集形成一个边界,如图1中的粗线所示,边界之下是频繁集,边界之上是非频繁集.显然,最佳的关联规则发现算法应仅仅生成并测试频繁集,而避免任何非频繁集的生成并测试.这就是说,算法应能有效地确定频繁集边界的结构.幸运的是,由最大频繁集导出的子集空间正好与边界结构相对一致.这些子空间也如图1所示.

一般地,我们是无法预知最大频繁集的,但是,我们可以设法得到尽可能小的、包含最大频繁集的超集.为此,我们引入等价类概念来对项目集进行聚类.考察Apriori算法中的候选生成方法.第k次遍历所需的候选集

---

* 本文研究得到国家自然科学基金和国家教育部博士点基金资助.作者**欧阳为民**,1964年生,博士,副教授,主要研究领域为KDD,机器学习,人工智能及其应用.**蔡庆生**,1938年生,教授,博士生导师,主要研究领域为机器学习,知识发现,人工智能.

本文通讯联系人:欧阳为民,合肥 230039,安徽大学计算中心

本文1997-12-23收到原稿,1998-08-11收到修改稿

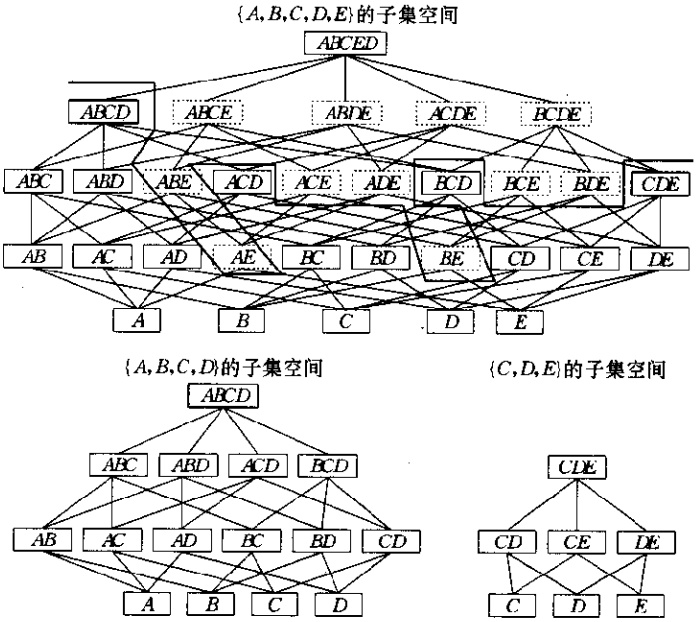

图1　项目集聚类

C_k 是通过链接频繁$(k-1)$-项目集 L_{k-1}而得到的. 于是，我们有 $C_k=\{x=p[1]p[2]\ldots p[k-1]q[k-1]\}$，对所有 $p,q\in L_{k-1}$，$p[1:k-2]=q[1:k-2]$. 为方便计，一般要求项目集中各项目之间保持某种偏序关系，这里，我们采用通常的字母序. 于是，$p[k-1]<q[k-1]$. $x[i]$表示项目集 x 的第 i 个项目，$x[i:j]$表示项目集 x 中的从第 i 个到第 j 个项目. 由图 1 可知，$L_2=\{AB,AC,AD,BC,BD,CD,CE,DE\}$，那么有 $C_3=\{ABC,ABD,ACD,BCD,CDE\}$. 我们按照前 $k-2$ 个项目(称为$(k-2)$-前缀)是否相同来对 L_{k-1}进行等价类划分，即具有相同$(k-2)$-前缀的项目集属于同一个等价类，记作

$$P_a=[a]=\{x[k-1]\in L_1 \mid a[1:k-2]=x[1:k-2]\}.$$

于是，候选 k-项目集可以通过链接同一等价类中的所有的元素对，然后再以类标识符作为各链接结果的前缀这种方法来生成. 对上述例子中的 L_2，我们得到如下等价类：$P_A=[A]=\{B,C,D\}$，$P_B=[B]=\{C,D\}$，$P_C=[C]=\{D,E\}$，$P_D=[D]=\{E\}$. 由等价类$[A]$，$[B]$和$[C]$链接生成的集合分别为$\{ABC,ABD,ACD\}$，$\{BCD\}$，$\{CDE\}$. 这 3 个集合彼此独立. 由于仅有 1 个元素的等价类不能产生新的候选，因而可以不再进一步加以考虑.

注意，在实际实现时，等价类的生成可以在候选生成的同时进行构造. 其方法如下：

Input：$C_k=\{x_1,x_2,\ldots,x_n\}$，其中各项目集按字典序排列，$C_k[i]$表示候选集 C_k 中的第 i 个项目集，$|C_k|$表示候选集 C_k 中所包含的项目集的个数；

Output：L_{k+1}的等价类；

```
Begin
(1)  for i=1 to |C_k| do {
(2)    [C_k[i]]=∅; /* 等价类[C_k[i]]初始化为空集 */
(3)    for j=i+1 to |C_k| do {
(4)      C=C_k[i][1]C_k[i][2]...C_k[i][k]C_k[j][k]; /* 候选链接，详见第 2 节 */
(5)      if C.support≥minsup then [C_k[i]]=[C_k[i]]∪{C};
(6)      }
(7)    }
End
```

上述算法中的候选生成方法与 Apriori 算法中的候选生成方法相比至少有三大优点.首先,前者不需要作任何比较运算,直接进行拼接即可;而后者却需要判断 p 和 q 的前 $k-1$ 个项目是否相同.显然,前者更为方便、快捷.其次,前者在候选生成后没有修剪步骤,而后者是有的.这样一来,是否会出于前者的候选大于后者的候选数,因而需要对更多的候选进行支持计数,从而导致效率下降呢?回答是否定的.事实上,两者的候选数是相等的,只不过后者经过修剪步骤后候选数可能有所减少.更重要的是,根据下一节提出的基于垂直数据分布策略的候选链接方法,某个候选一经生成,其支持数就立即可知,可以直接判断其是否是频繁的.因此,没有必要在候选生成后再进行修剪.第 3 个优点是,引入等价类概念后,我们可以独立地在各个等价类中分别进行频繁项目集的发现,因而便于算法的并行化.

在算法的任何中间步骤,当确定了频繁集 $L_k(k\geqslant 2)$ 以后,我们就可以从 L_k 生成潜在的最大频繁集.注意,对于 $k=1$,我们以整个项目集作为潜在最大频繁集.然而,对于 $k\geqslant 2$,我们便可以得到较精确的关于项目之间关系的知识.k 值越大,聚类越精确.每个等价类就是一个潜在的最大频繁集.例如,从等价类[A]和[C],我们最终可以分别得到真正的最大频繁集 $ABCD$ 和 CDE.

2 数据分布

KDD 系统的处理过程由若干步骤组成.最初一步是通过属性集聚焦或数据抽样来建立与发现任务相关的目标数据集.为了便于数据归约和映射,数据库的建立一般还需要某些技术来进行数据变换、不必要信息的删除和缺损数据(missing data)的补充.接下来,用户必须确定数据发掘任务,选择合适的算法.然后是对已发现的模式或知识进行评价、解释等.数据预处理阶段的一个重要问题是数据集中数据的表达与分布方式.这里,我们简要讨论数据分布的两种分布方式:水平分布与垂直分布.

2.1 水平数据分布

水平数据分布是一种常见的方式.在这里,数据集由一系列事务构成.每个事务有一个事务标识符 TID 以及相应事务所包含的项目表列(即项目集).使用这种格式,计算负载主要集中在支持计数上.对于平均含有 L 个项目的事务,在第 k 次循环期间,我们必须生成并测试该事务的所有 k-元素子集,看其是否包含在候选集 C_k 中.为了快速地进行子集检索,一般都采用复杂的 Hash 数据结构来存储各个候选.因而必须对 Hash 数据结构进行维护与搜索,从而又增加了额外的计算负载.而且,数据的水平分布迫使每次循环均需遍历整个数据库或每个局部划分.

2.2 垂直分布

数据的垂直分布是指:数据集由一系列项目构成,每一个项目均带有其相应的 TID 表列,即包含该项目的所有事务的标识符表,且按字典序的升序排列.垂直分布不存在水平分布的问题,其原因主要是:

(1) 因为 TID 表列以升序存储,所以候选 k-项目集的生成与支持可以通过简单地链接任意两个 $(k-1)$-子集得到,不需要复杂的 Hash 数据结构,无需对整个数据库进行遍历.我们不必生成某一事务的所有 k-子集,也不需要对 Hash 数据结构进行维护与搜索.

水平分布

TID	A	B	C	D	E	F
T1	1	0	1	0	0	0
T2	0	1	1	0	1	0
T3	1	1	1	0	1	0
T4	0	1	0	0	1	0
T5	0	1	1	0	1	1
T6	0	1	1	1	1	1

垂直分布

TID	A	B	C	D	E	F
T1	1	0	1	0	0	0
T2	0	1	1	0	1	0
T3	1	1	1	0	1	0
T4	0	1	0	0	1	0
T5	0	1	1	0	1	1
T6	0	1	1	1	1	1

图2 数据分布

(2) TID 表列包含了所有的关于某项目集的相关信息,垂直分布因此具有可本地化计算的优点.频繁项目集的计算可以在每一个聚类中独立进行.例如,考察图 2,该图对比了水平分布与垂直分布两种情况.为简单计,我们在图中对空值元素也作了表示.A 和 C 的 TID 表列分别为 $A.tids=\{1,3\}$,$C.tids=\{1,2,3,5,6\}$.于是,这两张 Tid 表列的链接仅需对其同时作一次遍历,即可得到 AC 的 TID 表列 $AC.tids=\{1,3\}$.我们只要在 TID 表列中简单地清点元素个数就可以立即确定相应项目集的支持.实际上,在上述链接生成过程中就可以进行支持计数,不必事后进行,从而可以省去对 $AC.tids$ 的一次遍历.如果满足最低支持,便将其加入频繁项目集 L_2 之中.

基于上述描述，我们有如下TID表列的链接算法：

Algorithm join of two candidates;

Input: two candidate L-itemsets X and Y, and their ctid-list $X.tids$ and $Y.tids$;

Output: new candidate $(L+1)$-itemset C and its tid-list;

Begin

(1) $C=X[1]X[2]\ldots X[L]Y[L]$;

(2) $C.support=0$;

(3) $i=1;j=1;k=0$;

(4) while $i\leqslant|X|$ and $j\leqslant|Y|$ do {

(5) if $X.tids[i]==Y.tids[j]$ then {

(6) $k++$; $C.support++$;

(7) $C.tids[k]=X.tids[i]$;

(8) }

(9) else if $X.tids[i]<Y.tids[j]$ then $i++$;

(10) else $j++$;

(11) }

End

通过利用最低支持，可以更快地执行TID表列之间的链接. 例如，我们假定最低支持为100，正在链接的两个项目集是AB和AC，其支持分别为119和200. 一旦发现在AB中出现了20次失配，我们就可以中止链接，因为ABC的支持最大不可能超过119，而现在已有20项失配，这样，其支持肯定低于100. 采用这种优化方式，我们可以高效地进行链接运算.

然而，正如水平分布存在不足一样，垂直分布也是有缺点的. 对于小项目集的检查，垂直分布的代价要高于水平分布的代价，这是因为小项目集的TID表列没有对项目间的关系提供多少有用的信息，特别是在1-项目集中，其TID表列就根本没有这样的信息. 为此，我们在频繁1-项目集和频繁2-项目集的计算时仍采用水平数据分布方式，而在频繁2-项目集的计算时构造出每个频繁2-项目的TID表列，然后，对所有频繁k-项目集($k\geqslant3$)的计算均根据有关项目集的TID表列进行，即转而采用垂直数据分布方式. 这样，我们仅需对整个数据库遍历两次.

3　基于垂直数据分布的关联规则发现算法描述

本节首先给出一个例子来说明根据上述各节的概念与方法发现最大频繁集的过程，然后给出相应的算法描述.

例：根据如图2所示的数据库来发现最大频繁集$Answer_Set$.

假定用户最低支持为2，置$Answer_Set$为空. 首先，我们基于水平数据分布方式来发现频繁1-项目集F_1和频繁2-项目集F_2. 以所有项目构成候选1-项目集C_1，我们有$C_1=\{A,B,C,D,E,F\}$. 遍历数据库一次，计算各个候选的支持，取支持不低于2的所有候选构成频繁1-项目集$F_1=\{A,B,C,E,F\}$. 接着按照上节所述的同时构造等价类的候选生成方法，根据F_1来生成候选2-项目序列集C_2. 于是有，$[A]=\{B,C,E,F\}$，$[B]=\{C,E,F\}$，$[C]=\{E,F\}$，$[E]=\{F\}$，即$C_2=\{AB,AC,AE,AF\}\cup\{BC,BE,BF\}\cup\{CE,CF\}\cup\{EF\}$.

再次遍历数据库，计算各个候选的支持，并构造各个候选的TID表列. 从上述各个等价类中删除支持低于2的元素，得到$[A]=\{C\}$，$[B]=\{C,E,F\}$，$[C]=\{E,F\}$，$[E]=\{F\}$，从而有频繁2-项目集$F_2=\{AC,BC,BE,BF,CE,CF\}$.

然后，采用垂直数据分布方式，即根据有关项目集的TID表列来发现所有频繁k-项目集($k\geqslant3$)的计算. 依次根据各个等价类，分别进行频繁k-项目集($k\geqslant3$)的发现.

(1) 因为等价类$[A]=\{C\}$仅有一个元素，所以无法得到更长的频繁项目，因而不必进一步考察. 因此，

Answer_Set=*Answer_Set*∪{*AC*}.

(2) 考察等价类[*B*]={*C*,*E*,*F*}.链接其中的所有元素对,得到候选 2-项目集中的各个等价类和其中各个候选的 TID 表列:[*BC*]={*E*,*F*},*BCE*.*tids*={2,3,5,6},*BCF*.*tids*={5,6};[*BE*]={*F*},*BEF*.*tids*={5,6}.

(2.1) 考察等价类[*BC*]={*E*,*F*}.因其各个候选的支持均不低于 2,故继续链接其元素对,得到一个候选 3-项目集中的等价类[*BCE*]={*F*},*BCEF*.*tids*={5,6}.由于该等价类中只有 1 个元素,所以在其中不会得到更长的频繁项目,因而不再对其进一步加以考察.因此,*BCEF* 为一个最大频繁项目集,*Answer_Set*=*Answer_Set*∪{*BCEF*}.

(2.2) 由于等价类[*BE*]={*F*}中只有一个元素,所以不会产生更长的频繁项目序列,因而不再对其进一步考察.但是,由于 *BEF* 包含在 *BCEF* 中,所以不认为是最大频繁项目集.

(3) 考察等价类[*C*]={*E*,*F*}.链接其元素对得到一个候选 2-项目集中的等价类及其 TID 表列:[*CE*]={*F*},*CEF*.*tids*={5,6}.由于该等价类中只有一个元素,所以不再对其进一步考察.由于 *BCE* 包含在 *BCEF* 中,所以不认为它是最大频繁项目集.

(4) 同理,考察等价类[*E*]={*F*},该类也没有产生新的最大频繁项目集.

至此,所有等价类考察完毕,最大频繁项目集的发现过程便告结束.基于上述描述,我们给出如下最大频繁项目集发现算法 FID:

Algorithm:最大频繁项目集发现算法 FID

Input:(1) *DB* 为定义在 $R=\{I_1,I_2,\ldots,I_m\}$之上的某关系数据库,其中属性 I_i 的值域为{0,1};

(2) min*sup* 为最低支持度;

Output:满足最低支持度的最大频繁项目集 *Answer_Set*;

```
Begin
(1)  C1=all items in DB, which are lexicographically sorted;
(2)  for each tuple t in DB do
(3)    Ct=get_subset(C1,t); /* all candidates contained in t */
(4)    for each candidate c∈Ct do c.support++;
(5)    }
(6)  for i=1 to |C1| do {
(7)    C2=∅;
(8)    for j=i+1 to |C1| do {
(9)      C=C1[i][1]C1[i][2]...C1[i][k]C1[j][k];
(10)     C2=C2∪{C};
(11)     }
(12)   }
(13) for each tuple t in R do
(14)   Ct=get_subset(C2,t); /* all candidates contained in t */
(15)   for each candidate c∈Ct do {c.support++;c.tid_list=c.tid_list∪t.tid}
(16)   }
(17) L2={c|c∈C2,c.support≥minsup};
(18) S=Get_equivalence_classes(L2);/* 按照等价类的概念进行生成 */
(19) for each equivalce class E in S do
(20)   Result_Set=Result_Set∪Bottom_Up(E);
(21) Answer_Set=Maximum itemsets in Result_Set;
(22) Return Answer_Set;
End.
```

```
Get_equivalence_classes(L2)
{
  S=∅;E=L2[1];
  x=L2[1][1];/* get the first element of the first frequent itemset in L2 */
  for i=2 to |L2| do
    if x==L2[i][1] then E=E∪{L2[i]};
    else {S=S∪{E};E=L2[i];x=L2[i][1];}
  Return S;}
Bottom_Up(Fk) /* Fk={I1,I2,...,Im} */
{if |Fk|=1 then {
  Result_Set=Result_Set∪get_element_of(Fk);
  Return Result_Set;
  }
for i=1 to m do{
      Fk+1=∅;
      for j=i+1 to m do{
        C=Join(Ii,Ij); /* 参见 2.1 节的 Join 算法*/
        if C.support≥minsup then Fk+1=Fk+1∪{C};
        }
    if Fk+1≠∅ then Result_Set=Result_Set∪Bottom_Up(Fk+1);
    Return Result_Set;
}
```

4 实验结果

我们用 Visual Foxpro 在内存为 16M 的 586 微机上实现了 Apriori 算法和 FID 算法，采用合成数据进行了算法比较测试. 测试数据库含有 5 个属性，每个属性有 100 个原始值. 首先测试算法的扩放性. 我们将测试数据库的元组数从 1 000 开始，逐次递增到 10 000. 最低支持设为 2%. 两种算法的扩放性能数据曲线如图 3 所示. 可见两种算法的扩放性能均较好，FID 算法计算量相对较少，因而执行时间短一些.

接着，我们比较算法在不同最低支持下的性能变化. 此时，测试数据库固定为 5000 元组，分如下 5 次变化最低支持，即 S1(2%)，S2(1.5%)，S3(1%)，S4(0.75%)和 S5(0.5%). 测试结果如图 4 所示. 可以看出，当支持下降时，执行时间上升，原因是过滤条件减弱了. 另外，我们还可以看出，FID 算法的执行时间比 Apriori 算法要少.

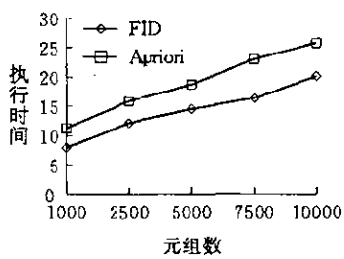

图3

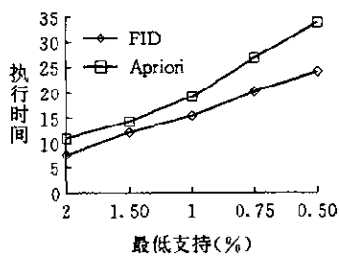

图4

从理论分析和实验结果可知，FID 算法的效率要优于 Apriori 算法.

5 结论与进一步工作

我们对现有的关联规则发现算法加以研究，提出了等价类概念，并为方便链接而将数据库的数据分布由通常的水平方式改为垂直方式，从而提出了发现频繁集的高效算法FID(frequent itemset discovery). 该算法实现方便，无需复杂的Hash数据结构，对整个数据库仅需遍历两次. 实验结果表明，FID算法的效率要优于Apriori算法.

本文所研究的仅是单层次的关联规则. 如何将本文所提方法应用于多层关联规则(multiple level association rules)和定量型关联规则(quantitative association rules)的发现是值得进一步研究的问题.

参考文献

1 Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. In: Proceedings of 1993 ACM SIGMOD International Conference on Management of Data. Washington, DC, 1993. 207～216

2 Agrawal R, Srikant R. Fast algorithm for mining association rules. In: Proceedings of 1994 International Conference on Very Large Databases. Santiago, Chile, 1994. 487～499

3 Agrawal R, Mannila H, Srikant R *et al*. Fast discovery of association rules. In: Fayyad U *et al* eds. Advances in Know ledge Discovery and Data Mining. New York: MIT Press, 1996. 307～328

4 Klemettinen M, Mannila H, Ronkainen P *et al*. Finding interesting rules from large sets of discovered association rules. In: Proceedings of the 3rd International Conference on Information and Knowledge Management. 1994. 401～407

5 Park J S, Chen M, Yu P S. An effective Hash based algorithm for mining association rules. In: Proceedings of 1995 ACM SIGMOD International Conference on Management of Data. 1995. 175～186

6 Brin S, Motwani R, Ullman J *et al*. Dynamic itemset counting and implication rules for market basket data. In: Proceedings of ACM SIGMOD Conference on Management of Data. 1997

7 Savasere A, Omiecinski E, Navathe S. An efficient algorithm for mining association rules in large databases. In: Proceedings of 1995 International Conference on Very Large Databases. Zürich, Swizerland, 1995. 432～444

An Efficient Algorithm for Discovering Association Rules Based on Vertical Data Layout

OU-YANG Wei-min[1] CAI Qing-sheng[2]

[1](*Computing Center Anhui University Hefei* 230039)

[2](*Department of Computer Science University of Science and Technology of China Hefei* 230027)

Abstract In this paper, the authors analyze the methods to generate and test frequent itemsets in existing algorithms in KDD research, and put forward a new efficient algorithm for discovering association rules based on vertical data layout. This algorithm has no need of Hash data structure and makes only two databases scans. As a result, the algorithm not only facilitates the implementation, but also improves the efficiency.

Key words Association rule, frequent itemsets, equivalence class.