时态约束下的数据挖掘问题及算法

毛国君, 刘椿年
(北京市多媒体与智能软件重点实验室, 北京工业大学计算机学院, 北京 100022)

摘 要: 对于一个大型数据库而言, 为了提高挖掘效率, 必须考虑减少数据库的扫描次数, 同时使内存需求量保持在一个适当的水平上. 把时态约束应用到事务数据库的挖掘中, 可以获得更好的效率. 本文首先利用时态区间代数操作实现原始数据库的过滤和挖掘时态区间的合并; 然后在定义项目序列集操作的基础上, 提出一个称为 TISS2DM 的发现频繁项目序列集的高效算法; 最后讨论了这个算法的效率.

关键词: 数据挖掘; 关联规则; 时态区间; 频繁项目序列
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2003) 11-1690-05

A Method of Data Mining Based on Temporal Constraints

MAO Guo-jun, LIU Chun-nian
(Beijing Municipal Key Laboratory of Multimedia and Intelligent Software Technology,
School of Computer Science, Beijing University of Technology, Beijing 100022, China)

Abstract: For a large database, an efficient algorithm for mining association rules should try to reduce the number of the database passes and control the size of necessary memory. Temporal constraints can help us to get higher efficiency in data mining. This paper first defines temporal interval operators, and uses them to filter the database and merge temporal intervals for mining. Then, it gives a new effective algorithm called TISS2DM for mining temporal association rules. Finally, the performance of this algorithm is discussed.

Key words: data mining; association rule; temporal interval; frequent itemsequence

1 引言

关联规则挖掘是数据挖掘研究中最活跃的方向之一. 设 $I=\{i_1, i_2, \ldots, i_m\}$ 是一个项目集合, 事务数据库 $D=\{t_1, t_2, \ldots, t_n\}$ 中的每个事务 t_i 都是 I 上的一个子集. 显然, 使子集中的元素有序排列, 既可以提高效率又不影响问题解决, 因此本文使用术语"项目序列"[1]来替代大多数文献使用的"项目(子)集"[2,3,6]. 这样, 关联规则挖掘问题可以分解为两个子问题: (1) 寻找频繁项目序列集. 设 $I_1 \subseteq I$, 项目序列 I_1 在 D 上的支持度是包含 I_1 的事务在 D 中所占的百分比, 即 $support(I_1) = |\{t \in D \mid I_1 \subseteq t\}| / |D|$. 如果用户给定 minsupport, 则 D 中出现的满足 support 不小于 minsupport 的项目序列称为频繁项目序列. (2) 生成关联规则. 所谓规则的可信度为 $confidence(I_1 \Rightarrow I_2) = support(I_1 \cup I_2) / support(I_1)$, 其中 $I_1, I_2 \subseteq I$, $I_1 \cap I_2 = \varnothing$. 通过用户给定 minconfidence, 就可以利用频繁项目序列集生成 confidence 不小于 minconfidence 的关联规则.

关联规则挖掘的典型应用是购物篮分析问题. 如果把一个项目序列看作是用户一次购买物品的记录, 那么挖掘出的诸如"牛奶⇒面包"这样的关联规则就可以用于指导进货等商业行为. 近年的关联规则应用已经深入到因特网用户行为、商业客户分析等诸多商业领域. 寻找频繁项目序列集是挖掘关联规则的关键步骤. Apriori 算法[2]是最具代表性的算法, 它是通过多次扫描数据库来完成的, 因而具有昂贵的 I/O 代价. 最近提出了一些减少数据库扫描次数的方法, 比较著名的是两次数据库扫描算法 FP2Tree(Han, 2000)[3]. 作者也曾给出了一个一次扫描算法 ISS2DM[1](2002). 这些方法明显改善了 I/O 效率, 但是对于大容量数据库而言, 它的内存需求等成为主要矛盾. 将约束条件应用到数据挖掘中, 是大幅度提高效率的重要途径[4,5]. 对于大容量数据集而言, 用户关心的往往是某一时间区域的数据, 因此时态约束可以用于对数据集的预先过滤等[6~8]. 1999 年, 欧阳和蔡给出了时态约束关联规则挖掘方法 TCAR[6], 被挖掘出的规则形式为"A ⇒ B, [n_1, n_2]", 它刻画了在[n_1, n_2]时态区间内有效的规则 A ⇒ B. 这样的时态语义约束下的关联规则挖掘技术可以更精准地用于具有时间标记的商品交易等数据的分析中.

2 时态区间代数及预处理

图 1 给出了一个本文所讨论的事务数据库示例, 它包含

收稿日期: 2002-12-23; 修回日期: 2003-05-28
基金项目: 国家自然科学基金(No. 60173014); 北京市自然科学基金(No. 4022003); 北京市教委资金

三个属性: 事务号(Tid) 、时态区间(Interval) 和项目序列(Item2Sequence) . 这里的时态区间反映了对应项目序列发生或被收集的时间范围, 并已转换成易于处理的整数区间形式[6] .

Tid	Interval	ItemSequence
1	[10, 50]	A, B, C, D
2	[30, 60]	B, C, E
3	[70, 80]	A, B, C, E
4	[70, 120]	B, D, E
5	[70, 90]	A, B, C, D
6	[300, 500]	A, B, C, D, E

图 1　样本事务数据库

我们首先从时态区间代数空间的形式化开始, 定义两个基本时态区间操作. 我们做这些工作的主要目的有两个: 其一是通过使用它们对数据库进行过滤以减少进入内存数据的容量; 其二是通过对时态区间的合并使由于过滤后生成的时态区间碎片合并成互不相交的挖掘时态区间集, 并对每个挖掘时态区间单独通过内存演算来生成关联规则. 这样, 就可以增强对大型源数据库的处理能力.

定义 1(时态区间的代数空间)　一个时态区间的代数空间可以用三元组 T= (TI, TD, TO) 来刻画, 其中含义如下:

u 时态区间变量集 TI: 一个时态区间变量形如 $I_i = [I_i^-, I_i^+]$, 其中 I_i^- [I_i^+, I_i^- 和 I_i^+ (i= 1, 2, , , n) 是定义在 TD 上的时间点变量;

u 时态解释域 TD: 时态区间变量对应的时间点定义范围, 即 P $I_i = [I_i^-, I_i^+]$ I $T_I I_i^-$, I_1^+ I TD, I_i I TD @TD;

u 时态区间变量间的操作集 TO: 定义在 TD 上的关于时态区间变量的操作集.

下面我们将定义两个关于时态区间变量的操作, 之后将应用它们完成数据挖掘的预处理工作.

定义 2(时态区间变量操作)　设 $I_1 = [I_1^-, I_1^+]$ 和 $I_2 = [I_2^-, I_2^+]$ 是定义在某个时态解释域上的两个时态区间变量, 它们的时态交(H_T) 、时态并(U_T) 运算定义为

u 时态交(H_T) : 如果 $I_1^+ < I_2^-$ 或 $I_2^+ < I_1^-$, 则 $I_1 H_T I_2$= ª ; 否则 $I_1 H_T I_2 = [I_3^-, I_3^+]$, 其中 $I_3^- = \max\{I_1^-, I_2^-\}$, $I_3^+ = \min\{I_1^+, I_2^+\}$;

u 时态并(U_T) : 如果 $I_1^+ < I_2^-$ 或 $I_2^+ < I_1^-$, 则 $I_1 G_T I_2$= ª ; 否则 $I_1 G_T I_2 = [I_3^-, I_3^+]$, 其中 $I_3^- = \min\{I_1^-, I_2^-\}$, $I_3^+ = \max\{I_1^+, I_2^+\}$.

不带约束的关联规则挖掘空间描述为一个五元组 W= (I, D, O, U, R)[1], 而时态约束下的关联规则挖掘空间可以描述为一个六元组 W= (I, T, D, O, U, R), 其中 I, D, O, U 和 R 与文献[1]的含义类似, 分别是 W 所涉及的项目定义域、事务数据库(如图 1 所示)、D 中项目序列的操作集、用户给定的限制参数及约束条件和 D 中所蕴涵的带有时态约束的关联规则集. T 为定义 1 所刻画的时态区间空间, 即 W 所涉及的时态区间变量集及其操作.

过滤数据库以减少规则挖掘所对应的扫描空间, 是处理大容量源数据库的关键技术[8] . 我们知道, 随着源数据库 D 容量的增大, 利用有限内存资源通过一次数据库扫描来实现有效挖掘是困难的, 因此使用用户约束来缩减源数据集容量成为必须. 数据过滤的目的是为了增强对大容量数据库的挖掘能力. 当然, 它也必须付出为过滤处理所花费的 I/O 代价等. 给定用户挖掘时态区间 I_{user}, 可以利用上面定义的操作 H_T, 实现对数据库的过滤. 下面描述的过滤算子 filter (D, I_{user}, Dc) 实现对 D 中无效数据的剔除, Dc 作为形成关联规则的缩减数据集, 按着元组对应的项目序列的字典顺序被存储在硬盘中.

算子 1　filter(D, I_{user}, Dc)
1) FOR all t I D DO
2)　IF (t. Interval H_T I_{user} X ª)
3)　　t_1. Interval z　t. Interval H_T I_{user};
4)　　t_1. ItemSequence z　t. ItemSequence;
5)　　sort2insert(t_1, Dc)
6)　ENDIF
7) ENDFOR;

对 D 中的每个元组 t, 利用 t 的时态区间属性(标识为 t. Interval) 和用户挖掘时态区间 I_{user} 进行 H_T 运算以判别 t 中是否包含用户所关心的信息. 如果 t. Interval H_T I_{user} X ª , 上面的 3) 、4) 和 5) 生成一个新的元素 t_1, 并把它作为一个元组按项目序列的字典顺序插入到 Dc 中.

由于使用 H_T 运算对 D 的每个元组的 Interval 属性进行可能的压缩, 因此 Dc 中的时态区间可能是很零散的. 本文进一步考虑 Dc 中所涉及的时态区间的合并问题, 其目的是演化成最大的互不相交的挖掘时态区间集. 下面算子 merge(Dc, TS) 实现这一功能, 其中 TS 为合并后的挖掘时态区间集.

算子 2　merge(Dc, TS)
1)　TS= ª ;
2)　FOR all d I Dc DO
3)　　tt z d. Interval;
4)　　FOR all t_1 I TS DO
5)　　IF (tt U_T t_1 X ª)
6)　　　tt z tt G_T t1;
7)　　　delete(t_1, TS) M把 t_1 从 TS 中删除
8)　　ENDIF
9)　　ENDFOR
10)　　insert(tt, TS)　M把 tt 加入到 TS 中
11)　ENDFOR;

3　时态约束下的关联规则挖掘算法

如前所述, 使用 filter 算子可以对源数据库进行压缩, 而 merge 可生成互不相交的挖掘时态区间集. 现在将描述最大频繁项目序列集的生成算法(限于篇幅, 不对由频繁项目序列集演化关联规则的方法进行介绍, 感兴趣的读者可以参见文献[2]) . 我们先给出整个算法, 然后再对相关的算子给出解释.

图 2 中的 TISS2DM 算法的 2) 和 3) 完成了对源数据库的过滤和时态区间的合并(见第 2 节) . 对于 TS 中的每个挖掘时态

区间, 通过对 Dc 的一次扫描, 调用 join 过程逐个测试项目序列IS 并把需要的加入到 TISS 中, 再通过 make_fre 过程把生成的频繁项目序列加入到 TISS* 中.

```
算法 TISS2DM(挖掘基于时态约束的频繁项目序列集)
1) Input(minsupport, Iuser);
2) filter(D, Iuser, Dc);
3) merge(Dc, TS);
4) FOR tt I TS DO M对每个挖掘区间生成频繁项目序列集
5)     TISS= a; TISS* = a;
6)     FOR d I Dc DO
7)       IF(d.Interval HT tt X a)
8)         IS= d.ItemSequence;
9)         join(IS, TISS); M见后
10)        make_fre(IS, TISS, TISS*) M见后
11)      ENDIF
12)    ENDFOR
13)    Answer= Answer U(TISS*, tt)
14) ENDFOR
```

图 2 TISS2DM 算法描述

为了描述 join 和 make_fre 算子, 我们首先定义项目序列集上的/ 亚0 操作运算.

定义 3(项目序列集上的亚操作) 设 $TISS_1$ 和 $TISS_2$ 是定义在某个项目集上的两个项目序列集变量, IS 是定义在这个项目集上的一个项目序列, 定义如下操作:

u 亚属于 I_{sub}: IS I $_{sub}TISS_1$ 当且仅当 v IS_1 I $TISS_1$, 使得 IS A IS_1;

u 亚包含 A_{sub}: $TISS_1$ A $_{sub}TISS_2$ 当且仅当 对P IS_1 I $TISS_1$] IS_1 I $_{sub}TISS_2$;

u 亚交 H_{sub}: $TISS_1$ H $_{sub}TISS_2$= {IS| IS I $_{sub}TISS_1$ 且 IS I $_{sub}TISS_2$};

u 亚并 G_{sub}: $TISS_1$ G $_{sub}TISS_2$= {IS| IS I $_{sub}TISS_1$ 或 IS I $_{sub}TISS_2$}.

从上面的定义可以看出, 这些亚操作在项目序列层次上刻画了项目序列集之间的关系. 当然, 项目序列集变量也可以使用普通的集合操作(如交、并、差、包含和属于等). 然而刻画这些亚操作, 对于形成频繁项目序列集是重要而有意义的. 例如, 虽然 AB| {ABCD, AD}, 但是 AB I $_{sub}$ {ABCD, AD}. 因此, 我们可以利用亚操作来完成项目序列的子序列的判别等.

下面来刻画 join 和 make_fre 算子. 为了直观和方便, 下面把一个项目序列的支持度简单记为它在数据库中的出现次数, 这很容易转换成引言中关于支持度的描述形式.

算子 3 join(IS, TISS)

```
1)  support(IS) z 1; flag z 0;
2)  FOR all IS1 I TISS DO
3)    IF IS= IS1
4)      support(IS1)= support(IS1)+ 1;
5)      flag z 1
6)    ENDIF
7)  ENDFOR
8)  IF flag= 0
9)      append(IS, TISS) M把追加到 TISS 后
10) ENDIF;
```

一个项目序列加入 TISS 后, 它或它的子项目序列可能达到频繁. 因此, make_fre 算子就是利用 IS 来测试它的子序列是否达到频繁, 并把那些频繁子序列加入到 TISS* 中.

算子 4 make_fre(IS, TISS, TISS*)

```
1) FOR all IS* I sub{IS} DO M对 IS 测频繁子序列
2)   sz 0;
3)   FOR all IS** I TISS DO
4)     IF IS* A IS**
5)       s= s+ support(IS**) M生成 IS* 的支持度
6)     ENDIF
7)   ENDFOR
8)   IF (s E minsupport)
9)     IF IS* | sub TISS*
10)      prune(IS*, TISS*); M见后
11)      TISS* = append(IS*, TISS*) M把 IS* 追加到 TISS* 后
12)    ENDIF
13) ENDIF
14) prune(IS*, TISS) M见后
15) ENDFOR;
```

既然我们的目标是希望得到最大频繁项目序列集, 那么我们在每次产生一个新的频繁项目序列 IS* 后, 就应该及时裁减 TISS 和 TISS* 中的 IS* 的子序列(调用 prune).

算子 5 prune(IS_1, $TISS_1$)

```
1)   FOR all IS2 I TISS1 DO
2)     IF IS2 I sub{IS1}
3)       TISS1= TISS1- {IS2}
4)     ENDIF
5)   ENDFOR;
```

为了直观地了解算法的执行过程, 我们对图 1 所示的数据库实施 TISS2DM 算法. 设 I_{user}= [50, 100], minsupport= 2(对应百分比 331 3%), 则过滤后的 Dc 为

Tid	Interval	ItemSequence
1	[50, 50]	A, B, C, D
2	[50, 60]	B, C, E
3	[70, 80]	A, B, C, E
4	[70, 100]	B, D, E
5	[70, 90]	A, B, C, D

图 3 对样本事务数据库的过滤结果

Merge 生成的挖掘时态区间集 TS= {[50, 60], [70, 100]}. 对[50, 60], TISS= {(ABCD, 1), (BCE, 1)}, TISS* = {BC}. 对[70, 100], TISS= {(ABCE, 1), (BDE, 1), (ABCD, 1)}, TISS* = {ABC, BD, BE}. Answer= {(BC, [50, 60], (ABC, BD, BE, [70, 100])}.

4 实验与讨论

数据挖掘方法的适用性应是对的 I/O 代价、内存使用及 CPU 占用率等的综合考虑. 归纳 TISS2DM 的 I/O 代价主要有两方面: (1) 由 D 过滤成 Dc 需要的输入、输出工作; (2) 对每个挖掘时态区间扫描 Dc 所付出的 I/O 代价. 为了更好地刻画算法的性能, 下面我们给出两个与 I/O 代价相关的评价指标.

过滤效率用 feff = (+ D + - + Dc +) / + D + 来表示. 当 Dc 相对于 D 被大幅度压缩时, feff 势必很大, 因而对源数据库 D 的处理能力大大加强. feff 的高低与用户关心的挖掘时态区间大小 I_{user}、D 中数据相关的时态分布有关. 很显然, 当 I_{user} 覆盖 D 中的全部时态区间时, feff = 0, 意味着这种时态约束不起作用. 因此, 本文提出的方法针对的挖掘目标是: 在一个企业长期收集的时间跨度很大的业务数据(如数据仓库)中, 希望通过用户指定 I_{user} 来聚焦某一相对较短时期(如第一季度)的数据来进行规则生成.

另一个评价算法的指标是合并零散时态区间的效率, 用 meff = (N - + TS +) / N 来表示, 其中 N 为 Dc 中所涉及的时态区间数. 很显然, meff 越大, 扫描 Dc 的次数越少, 因而 I/O 代价越低.

我们在一个约 1MB 的数据库上进行了性能实验. 它是针对一个模拟超市的一年销售数据进行的, 其中所有商品都映射成符号标识, 通过随机函数产生相同大小的时态区间(按每天工作 12 小时、每 4 个小时为单位进行数据收集, 共需约 1000 个时态区间). 对这样的数据跟踪随着 I_{user} 增大时 feff 和 meff 的变化趋势, 图 4 显示了对应的实验结果.

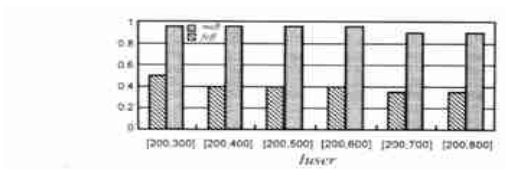

图 4　feff 和 meff 实验结果

图 4 说明, 在一个时态区间随机分布的事务数据库中, 有大量的数据被过滤掉(随着 I_{user} 的增长, feff 有所下降). 在我们的实验中, 在 I_{user} = [200, 300], [200, 400], [200, 500] 和 [200, 600] 时, 只产生 1 个挖掘时态区间, 当然 meff 较高. 在 I_{user} = [200, 700] 和 [200, 800] 时, 产生 2 个挖掘时态区间, meff 有所下降. 但是这增强了算法对大容量数据库和长 I_{user} 的适应能力. 这是因为随着数据库容量和 I_{user} 的增长, TS 中的挖掘时态区间数可能增多, 因此对每个 TS 中的挖掘时态区间进行扫描时, 可能使进入内存的项目序列数相对稳定. 为了进一步说明 TISS2DM 算法的内存占用情况, 我们从上面描述的数据库中抽取了不同容量的数据进行了实验. 图 5 给出了随着数据库规模增大, TISS 和 TISS* 占用内存的情况(最小支持度为 10%).

图 5 表明, TISS 和 TISS* 的存储空间远远小于原始事务数据库的大小, 而且相对稳定. 主要原因在于: 通过 filter 过滤掉大量用户不关心的数据; 通过 merge 生成互不相交的用户挖掘时态区间, 对每个挖掘时态区间单独扫描数据库, 因此使进入内存的项目序列减少; 通过 prune 算子及时裁减了 TISS 和 TISS*. 理论上讲, TISS 的空间上限值为 O(+ Dc +) 级别(以项目序列数计算), 而 TISS* 的理论上界是 $O(2^{LI})$ (LI 为 Dc 中所有挖掘时态区间中所涉及的项目数的最大值). 但是对于分布较均匀而且用户感兴趣数据的时态区间比较集中的挖掘问题, TISS 和 TISS* 对内存的需求都会比理论上界值要小的多(参见图 5 的实验结果).

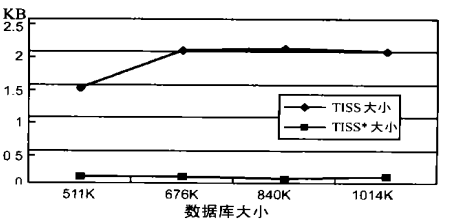

图 5　内存占用情况实验结果

TISS2DM 的 CPU 使用效率主要取决于项目序列的比较次数. 为了提高效率, 在扫描查询 D 而生成 Dc 的过程中, 我们已经对 Dc 中的元素按项目序列的字典顺序进行了排序. 这一工作可以通过所使用的 DBMS(数据库管理系统)的 DML 语句来实现, 而且大多数的 DBMS 采用优化的排序算法(如 22路归并排序等)来处理查询排序问题, 其占用 CPU 的时间复杂度可以控制在 O(+ Dc + * log + Dc +). 另外, join 过程是通过顺序扫描排序过的 Dc 来向 TISS 后面追加元素的, 因此保证了 TISS 是有序的. 很显然, make_fre 算子是影响 CPU 执行效率的关键因素. 在 make_fre(IS, TISS, TISS*) 中, 对每个 IS 的子序列串(最多 2^{+IS+} 个)需要在 TISS 中进行比较. 如果按最优化的查找算法(如 O(log N) 级别的对半方法), 处理一个 IS 所需要的比较次数不会大于 $O(2^{+IS+} * \log + Dc +)$. 由于我们对每个挖掘时态区间单独生成频繁序列集, 而且每次对一个挖掘时态区间中数据的处理量都小于 + Dc +, 因此整个算法执行时间的估计上限为 O(+ TS + * + Dc + * 2^{M} * log + Dc +), 其中 M 为 Dc 中的平均项目序列长度.

5 结语

本文讨论的是关联规则挖掘问题中的频繁项目序列集生成子问题. 它通过用户给出时态约束区间, 并利用时态区间代数操作, 对事务数据库进行过滤和挖掘时态区间合并等预处理, 以使挖掘过程集中在较小的用户感兴趣数据上. 对于一个特定的挖掘时态区间, 可以通过一次扫描过滤后的数据集, 在内存中通过项目序列集的操作算子来完成频繁项目集的生成. 我们的进一步工作将是探讨在其他时态约束形式下的生成关联规则方法以及通用约束条件的形式化表示等问题.

参考文献:

[1] 毛国君, 刘椿年. 基于项目序列集操作的关联规则挖掘算法[J]. 计算机学报, 2002, 25(4): 417- 423.

[2] Agrawal R, et al. Mining association rule between sets of items in large database [A]. Bunemuu P and Jajodia S eds. Proceedings of the ACM SIGMOD International Conference on Management of Data [C]. New

York: ACM Press, 1993. 207- 216.
[3] Han J W, et al. Mining frequent patterns without candidate generation [A]. Chen W, Naughton J, and Bernstein P A eds. Proceedings of 2000 ACM SIGMOD International Conference on Management of Data [C]. New York: ACM Press, 2000. 1- 12.
[4] Srikant R, et al. Mining association rules with item constrains [A]. Heckerman D, Mannila H, and Pregibon D eds. Proceedings of the 3rd International Conference on Knowledge Discovery in Databases and Data Mining [C]. Menlo Park, California: AAAI Press, 1997. 67- 73.
[5] Ng R, et al. Exploratory mining and pruning optimizations of constrained associations rules [A]. Laura M H and Ashutosh T eds. Proceedings of 1998 ACM-SIGMOD Conference on Management of Data [C]. New York: ACM Press, 1998. 13- 24.
[6] 欧阳为民, 蔡庆生. 在数据库中发现具有时态约束的关联规则 [J]. 软件学报, 1999, 10(5): 527- 532.
[7] 唐常杰, 等. 基于时态数据库的 Web 数据周期规律的采掘 [J]. 计算机学报, 2000, 23(1): 52- 59.
[8] Chen M S, et al. Data mining: An overview from a database perspective [J]. IEEE Transactions on knowledge and Data Engineering, 1996, 8(6): 866- 883.

作者简介:

毛国君 男, 1966 年生于内蒙古, 北京工业大学副教授, 主要研究领域为数据挖掘和分布式系统.

刘椿年 男, 1944 年生于江苏, 博士, 教授, 博士生导师, 主要研究领域为人工智能、知识工程及数据挖掘等.