文章编号: 1001- 9081(2005)07- 1577- 03

一种基于关系矩阵的关联规则快速挖掘算法

胡慧蓉[1], 王周敬[2]
(1. 广东商学院 信息学院, 广东 广州 510320; 2. 厦门大学 自动化系, 福建 厦门 361005)
(hhryeah@163.com)

摘 要: 首先对关联规则挖掘问题进行了简单的回顾, 然后应用关系理论思想, 引入了项目可辨识向量及其"与"运算, 设计了一种快速挖掘算法 SLIG, 将频繁项目集的产生过程转化为项目集的关系矩阵中向量运算过程。算法只需扫描一遍数据库, 克服了 Apriori 及其相关算法产生大量候选集和需多次扫描数据库的缺点。实验证明, 与 Apriori 算法相比, SLIG 算法提高了挖掘效率。

关键词: 关联规则; 频繁集; 可辨识向量; 可辨识矩阵
中图分类号: TP311.13 文献标识码: A

Fast algorithm for mining association rules based on relationship matrix

HU Hui-rong[1], WANG Zhou-jing[2]
(1. *Information Science and Technology School, Guangdong University of Business Studies, Guangzhou Guangdong* 510320, *China*; 2. *Department of Automation, Xiamen University, Xiamen Fujian* 361005, *China*)

Abstract After the introduction of famous Apriori algorithms, an efficient algorithm SLIG (Single-level Large Itemsets Generation) learning from relation theory and "AND" operation on recognizable vectors was proposed. SLIG transformed the production process of frequent itemset to the vector calculation process in relationship matrix and only needed to scan the database once. Empirical evaluation and experiments show that SLIG is more efficient than the algorithms that need to pass the large database many times.

Keywords association rule; frequent itemset; recognizable vectors; recognizable matrix

0 引言

关联规则[1]挖掘作为数据挖掘的一个重要研究分支, 主要研究从大型数据集中发现隐藏的、有趣的、属性间存在的规律。由于关联规则挖掘的对象是包含有海量原始数据和大量项目的事务数据库, 因此如何提高从大型数据库中挖掘关联规则的效率和伸缩性, 以便有效降低计算的复杂性、提高算法的运行速度, 便成为关联规则挖掘研究中的核心问题。目前, 该课题的研究已得到了国际众多研究者的广泛关注, 并提出了许多有效算法[2~6]。但绝大多数挖掘算法都是基于 Apriori 算法所提出的"自底向上"的思想, 其不足之处在于需要对数据库进行多次遍历, 即使进行了优化, Apriori 算法的固有的缺陷还是无法克服。

本文应用关系理论思想, 引入了项目可辨识向量及其运算, 提出了高效算法 SLIG (Single-level Large Itemsets Generation), 将频繁项目集的产生过程转化为项目集的关系矩阵中向量运算过程, 算法只需扫描数据库一遍, 克服了 Apriori 及其相关算法产生大量候选集和需多次扫描数据库的缺点。首先, 算法只对数据库遍历一次, 削减了 I/O 操作时间; 其次, 研究表明在生成和发现频繁 2-项集上耗费时间十分可观[7]。算法 SLIG 引入了项目可辨识向量的"与"运算, 对于 m 个项目, 只须进行 C_m^2 次"与"运算, 即可得到满足最小支持度的频繁 2-项集, 缩短了产生频繁 2-项集所耗费的时间; 最后, 只需要对 k 个项目的向量进行"与"运算, 就可以容易地求得频繁 k-项集的支持度, 不需要扫描整个事务数据库, 只需扫描小得多的向量。

1 关联规则描述

设 $I = \{i_1, i_2, \cdots, i_n\}$ 是项集, 其中的元素称为项(item)。记 DB 为交易(transaction) T 的集合, 这里交易 T 是项的集合, 并且 $T \subseteq I$。对应每一个交易有唯一的标识, 如交易号, 记作 TID。设 X 是一个 I 中项的集合, 如果 $X \subseteq T$, 那么称交易 T 包含 X。

一个关联规则是形如 $X \Rightarrow Y$ 的蕴含式, 这里 $X \subset I$, $Y \subset I$, 并且 $X \cap Y = \varnothing$。

规则 $X \Rightarrow Y$ 在交易数据库 DB 中的支持度(Support)是交易集中包含 X 和 Y 的交易数与所有交易数之比, 记为 $Support(X \Rightarrow Y) = P(X \cup Y)$;

规则 $X \Rightarrow Y$ 在交易集中的可信度(Confidence)是指包含 X 和 Y 的交易数与包含 X 的交易数之比, 记为 $Confidence(X \Rightarrow Y) = P(Y \mid X)$。

给定一个交易集 DB, 挖掘关联规则问题就是产生支持度和可信度分别大于用户给定的最小支持度(min_sup)和最小可信度(min_conf)的关联规则。

多数算法把挖掘关联规则的过程分为两个阶段:

1)获取频繁集。这些项集出现的频繁度至少和预定义的最小支持度一样。

2) 由频繁集产生关联规则。这些规则必须满足最小可信度。

第一阶段的任务是迅速高效地找出 DB 中全部频繁项目

收稿日期: 2004- 12- 14

作者简介: 胡慧蓉(1977-), 女, 江西吉安人, 助教, 硕士, 主要研究方向: 数据仓库、数据挖掘; 王周敬(1964-), 男, 福建尤溪人, 副教授, 主要研究方向: 人工智能、软件工程。

集, 这是关联规则挖掘的中心问题, 也是衡量关联规则挖掘算法的标准, 一旦找到了频繁集, 就可以用文献[2]提供的算法快速生成相应的关联规则。

2 SLIG算法

2.1 相关定义

定义1 设 $I=\{1, 2, \cdots, m\}$, 项目 i的可辨识向量 $\boldsymbol{D}_i$ 定义为: $\boldsymbol{D}_i=(d_{1i}, d_{2i}, \cdots, d_{ji}, \cdots, d_{mi})^{\mathrm{T}}$,

其中 $d_{ji}=\begin{cases}0, \text{如果项目 } i \notin T_j \text{ 事务}\\ 1, \text{如果项目 } i \in T_j \text{ 事务}\end{cases}$

定义2 项目 i的支持度 $Sup(\boldsymbol{D}_i)=\sum_{j=1}^{m} d_{ji}$, 即 $\boldsymbol{D}_i$中1的个数。

很明显, 向量 $\boldsymbol{D}_{i1}\wedge\boldsymbol{D}_{i2}\wedge\cdots\wedge\boldsymbol{D}_{ik}$仍然是一向量, 其中"∧"是向量的"与"操作, 假设 $\boldsymbol{D}_s=(d_{1s}, d_{2s}, \cdots, d_{js}, \cdots, d_{ms})^{\mathrm{T}}$, 则向量 $\boldsymbol{D}_s$的支持度 $Sup(\boldsymbol{D}_s)=\sum_{j=1}^{m} d_{js}$, 即 $\boldsymbol{D}_s$中1的个数。

定义3 项目集 $\{i_1, i_2, \cdots, i_k\}$的支持度是向量 $\boldsymbol{D}_{i1}\wedge\boldsymbol{D}_{i2}\wedge\cdots\wedge\boldsymbol{D}_{ik}$的支持度 $Sup(\boldsymbol{D}_{i1}\wedge\boldsymbol{D}_{i2}\wedge\cdots\wedge\boldsymbol{D}_{ik})$, 表示项目 $i_1, i_2, \cdots, i_k$一起出现的支持度, 表征的是项目 $i_1, i_2, \cdots, i_k$是否同时频繁出现。

2.2 SLIG算法思路

算法SLIG分为三个阶段:

1) 向量集的产生, 频繁1-项集产生阶段

按照定义1, 扫描事务数据库一次为每个项目建立一个向量。同时按照定义2计算每个项目的支持度, 定义如果 $Sup(\boldsymbol{D}_i)\geqslant$用户自定义的最小支持度阈值 min_sup(为最小支持度计数), 则 $\{i\}$为频繁1-项集, 依此可确定所有的频繁1-项集。

2) 频繁2-项集产生阶段

定义4 第1阶段确定频繁1-项集后, 设频繁1-项集集合 L_1为 $\{\{s\}, \cdots, \{t\}, \cdots, \{p\}\}$, 当中频繁1-项集的总数目为 q, 则项目集的关系矩阵 $\boldsymbol{R}_{q\times q}$定义为:

$$\boldsymbol{R}_{q\times q}=\begin{bmatrix} r_{ss} & \cdots & r_{st} & \cdots & r_{sp}\\ & \ddots & \vdots & \vdots & \vdots\\ & & r_{tt} & \cdots & r_{tp}\\ & & & \ddots & \vdots\\ & & & & r_{pp}\end{bmatrix}=(R_s, \cdots, R_t, \cdots, R_p)$$

其中, $r_{st}=\begin{cases}1, \text{当 } s\neq t \text{且 } Sup(\boldsymbol{D}_s\wedge\boldsymbol{D}_t)\geqslant min\_sup \text{时}\\ 0, \text{当 } s\neq t \text{且 } Sup(\boldsymbol{D}_s\wedge\boldsymbol{D}_t)<min\_sup \text{时}\\ 0, \text{当 } s=t \text{时}\end{cases}$

r_{st}表示某个项目 s和项目 t一起出现的支持度, 因此 r_{ss}, r_{tt}, r_{pp}不具有任何意义, 全部设为0, 很明显的, $\boldsymbol{R}_{q\times q}$是对称矩阵。

如果 $r_{st}=1$, 则项目集 $\{s, t\}$为频繁2-项集; 反之, 如果 $r_{st}=0$, 则项目集 $\{s, t\}$肯定不是频繁2-项集。

3) 频繁(k+1)-项集 $(k\geqslant 2)$产生阶段

得到频繁2-项集后, 对于每一个频繁k-项集 $(k\geqslant 2)$, 其最后的一个项目为 i_k, 对于任意一个 u从 i_k+1到 p, 如果 $r_{i_k u}=1$, $u\in(i_k+1, \cdots, p)$, 而且 $Sup(\boldsymbol{D}_{i_1}\wedge\cdots\wedge\boldsymbol{D}_{i_k}\wedge\boldsymbol{D}_u)\geqslant min\_sup$, 则 $\{i_1, \cdots, i_k, u\}$为频繁 $(k+1)$-项集。直到没有更长的频繁 $(k+1)$-项集产生, 算法结束。

```
k = 2;
while L_k ≠ ∅
{ L_{k+1} = ∅;
  for all itemsets { i_1, i_2, ..., i_k } ∈ L_k
    for ∀ u ∈ ( i_k + 1, ..., p )
      if r_{i_k u} = 1 then
        if( Sup ( D_{i_1} ∧ ... ∧ D_{i_k} ∧ D_u )) >= min_sup then
          L_{k+1} = L_{k+1} ∪ {{ i_1, ..., i_k, u }}
  k = k + 1; }
```

2.3 SLIG算法具体示例

以表1的交易数据库为例, 假设 $min\_sup=2$。

表1 给定的交易数据库 DB

TID	Items in transaction
T_1	1, 3, 4, 5, 6
T_2	1, 2, 5
T_3	3, 6
T_4	1, 3, 4, 6
T_5	3, 6

$\boldsymbol{D}_1=(11010)^{\mathrm{T}}$, $\boldsymbol{D}_2=(01000)^{\mathrm{T}}$, $\boldsymbol{D}_3=(10111)^{\mathrm{T}}$, $\boldsymbol{D}_4=(10010)^{\mathrm{T}}$, $\boldsymbol{D}_5=(11000)^{\mathrm{T}}$, $\boldsymbol{D}_6=(10111)^{\mathrm{T}}$; $sup(\boldsymbol{D}_1)=3$, $sup(\boldsymbol{D}_2)=1$, $sup(\boldsymbol{D}_3)=4$, $sup(\boldsymbol{D}_4)=2$, $sup(\boldsymbol{D}_5)=2$, $sup(\boldsymbol{D}_6)=4$; 得到频繁1-项集集合 L_1: {{1}, {3}, {4}, {5}, {6}}。

项目: $\boldsymbol{R}_{5\times 5}=\begin{bmatrix} 0 & 1 & 1 & 1 & 1\\ & 0 & 1 & 0 & 1\\ & & 0 & 0 & 1\\ & & & 0 & 0\\ & & & & 0\end{bmatrix}$

得到频繁2-项集集合 L_2: {{1, 3}, {1, 4}, {1, 5}, {1, 6}, {3, 4}, {3, 6}, {4, 6}}。

对于频繁2-项集{1, 3}, 其最后的项目为3, 因为存在 $r_{34}=1$, $\boldsymbol{D}_1\wedge\boldsymbol{D}_3\wedge\boldsymbol{D}_4=(10010)^{\mathrm{T}}$, $Sup(\boldsymbol{D}_1\wedge\boldsymbol{D}_3\wedge\boldsymbol{D}_4)$为 $2\geqslant min\_sup$, 所以得到频繁3-项集{1, 3, 4}。

同理, 可得到频繁3-项集: {1, 3, 6}, {1, 4, 6}, {3, 4, 6}。

对于频繁2-项集{1, 6}, 其最后的项目为6, 因为并无以项目6开始的频繁项目集, 这一步到此为止, 频繁2-项集{3, 6}同理。

综上, 得到频繁3-项集集合 L_3: {{1, 3, 4}, {1, 3, 6}, {1, 4, 6}, {3, 4, 6}}。

对于频繁3-项集{1, 3, 4}, 其最后的项目为4, 因为存在 $r_{46}=1$, $\boldsymbol{D}_1\wedge\boldsymbol{D}_3\wedge\boldsymbol{D}_4\wedge\boldsymbol{D}_6=(10010)^{\mathrm{T}}$, $Sup(\boldsymbol{D}_1\wedge\boldsymbol{D}_3\wedge\boldsymbol{D}_4\wedge\boldsymbol{D}_6)$为 $2\geqslant min\_sup$, 所以得到频繁4-项集{1, 3, 4, 6}。

而对于频繁3-项集{1, 3, 6}, {1, 4, 6}, {3, 4, 6}, 其最后的项目为6, 因为并无以项目6开始的频繁项目集, 这一步到此为止。

综上, 得到频繁4-项集集合 L_4: {{1, 3, 4, 6}}。

对于频繁4-项集{1, 3, 4, 6}, 其最后的项目为6, 因为并无以项目6开始的频繁项目集, 这一步到此为止。

算法结束。

3 算法仿真运行结果与性能分析

本文测试使用的数据文件是模拟的数据库文件, 生成数据库文件的程序的源代码由IBM的Almaden研究中心提供。在使用程序生成数据库文件的过程中, 使用的一些主要参数如同文献[2]所描述。本文性能测试的硬件平台是一台普通的个人电脑(P4 1.1GHz/SDRAM 256M)。

3.1 算法 SLIG和 Apriori的性能对比分析

假定事务数据库 DB中有 $|D|$ 个事务, 每个事务平均有 $|T|$ 个项目。

在第 1趟中, Apriori和 SLIG都需要对事务数据库扫描一遍以计算每个项目的支持度, Apriori和 SLIG花费的时间大致相同。

在第 2趟中, Apriori算法需要产生 $\frac{|L_1|(|L_1|-1)}{2}$ 个候选频繁 2-项集, 而且需要扫描事务数据库 DB一遍以产生频繁 2-项集; 而 SLIG 对向量执行 $\frac{|L_1|(|L_1|-1)}{2}$ 个逻辑"与"操作来创建关系矩阵和产生频繁 2-项集。这一趟中, Apriori算法花费更多的时间。

假设在第 $k(k \geqslant 2)$ 趟中产生 $|L_k|$ 个频繁项目集。在第 k趟中, Apriori由频繁 (k-1)-项集 L_{k-1} 自身连接产生候选 k-项集 C_k, 生成 C_k 之后, 扫描事务数据库中的每个事务来计算这些候选 k-项集 的支持度。因此, Apriori算法的执行时间主要依赖于产生的候选项目集的数量和数据库中数据量的大小; 而 SLIG 根据关系矩阵由 (k-1)-项集延伸并且执行逻辑"与"操作得到 k-项集, 假设平均来说, 在关系矩阵中, 每一行有 q个 1, SLIG执行 $(k-1)\times|L_{k-1}|\times q$个逻辑"与"操作来发现所有的频繁 k-项集。每趟中, SLIG中逻辑"与"操作的数量远小于 Apriori中产生的候选项目集的数量和事务数据库的大小。

因此, 在大数据集下, SLIG算法产生频繁项集所使用的时间大大少于 Apriori算法。

3.2 测试过程及结果评估

设定 N = 1 000, $|L|$ = 2 000, $|D|$ = 100K, 选择 $|T|$ 的两个值分别为 5和 10, 而对应的 $|MT|$ 分别为 10和 20, $|I|$ 的值选择为: 3, 5 对应的 $|MI|$ 为: 5, 10, 我们用 TaMTxIbMIy来表示数据集, 其中 $a=|T|$, $x=|MT|$, $b=|I|$, $y=|MI|$。产生如下 3个数据集: T5MT10I3MI5, T10MT20I3MI5和 T10MT20I5MI10。

图 1显示了在不同的最小支持度阈值条件下算法 Apriori和算法 SLIG的时间对比情况, 比例大致从 4.5到 21.1, 同样条件下最小支持度阈值越低, 则比例越大。随着最小支持度阈值的降低, 算法 Apriori和算法 SLIG的时间性能差距增加, 因为 Apriori中候选项目集的数量和扫描数据库的耗费增加很多。

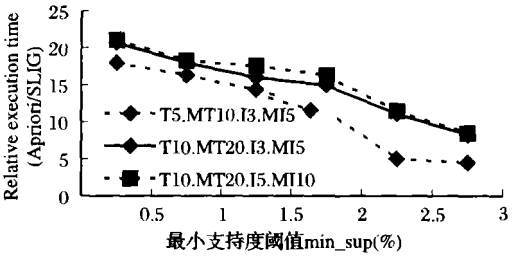

图 1　算法 Apriori和 SLIG的时间对比

从图 1可以看出, SLIG算法产生频繁项集所使用的时间大大少于 Apriori算法, 因为为了产生频繁项目集, 算法 SLIG只需扫描一遍数据库, 而 Apriori算法则需要扫描多遍数据库, SLIG算法访问数据库文件的次数比 Apriori算法要少得多。

参考文献:

[1] AGRAWAL R, MIELINSKI T, SWAMI A. Mining Association Rules between Sets of Items in Large Database[A]. Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data[C]. Washington DC, USA, 1993. 207-216.

[2] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules[A]. Proceedings of the 20th Int'l Conference on Very Large Databases[C]. Santiago, Chile, 1994. 487-499.

[3] 欧阳为民, 郑诚, 蔡庆生. 国际关联规则研究评述[J]. 计算机科学, 1999, 26(3): 41-44.

[4] SRIKANT R, AGRAWAL R. Mining Generalized Association Rules [A]. Proceedings of the 21st VLDB Conference[C], 1995. 407-419.

[5] HAN J, FU Y. Discovery of Multiple-Level Association Rules from Large Databases[A]. Proceedings of the 21st VLDB Conference [C], 1995. 420-431.

[6] SAVASERE A, OMIECINSKI E, NAVATHE SB. An Efficient Algorithm for Mining Association Rules in Large Data Bases[A]. Proceedings of the 21st VLDB Conference[C], 1995. 432-444.

[7] 向阳, 张巍. 基于事务数据库的关联规则采掘算法研究[J]. 山东科技大学学报(自然科学版), 2001, 20(2): 55-59.

[8] PARK JS. Using a hash-based method with transaction trimming for mining association rules[J]. IEEE Transaction on Knowledge and Data Engineering, 1997, 9(5): 813-825.

(上接第 1576页)

了闭合运算的计算量; 对于复杂的地形, 由于目标在 z方向分布相对平均, 算法优化部分抛弃的数据较少, 闭合运算计算量仍然较大, 所以算法效率较低。考虑到以下因素: 可以在 x-y平面进一步减小待处理数据以优化算法, 算法改进空间较大; 基于数学形态学的聚类算法易于改进为快速并行算法, 所以算法性能还比较理想。

4 结语

本文提出了一种基于数学形态学的聚类算法。该算法通过闭合运算将空间对象聚成类, 一次完成三维空间聚类; 可以快速处理非凸的、复杂的聚类形状; 由于该算法基于数学形态学, 所以易于实现其高性能并行算法, 具有较大的改进空间, 可以进一步提高算法性能。本文通过实验对算法有效性进行了验证, 该优化算法已成功地应用于炮兵作战阵地地形分析与选取的计算机辅助阵地选取系统。此外, 算法稍加改进就能够处理人口、产量空间分布等问题, 实现时可以将量化的人口、产量作为第三维。

参考文献:

[1] HAN JW, KAMBER M. Data Mining: Concepts and Techniques [M]. Simon Fraser University, 2000. 405-410.

[2] 周海燕, 王家耀, 吴升. 空间数据挖掘技术及其应用[J]. 测绘通报, 2002(2): 11-13.

[3] 阮秋琦. 数字图像处理学[M]. 北京: 电子工业出版社, 2001. 431-439.

[4] 唐常青, 吕宏伯, 黄铮, 等. 数学形态学方法及其应用[M]. 北京: 科学出版社, 1990. 145-160.

[5] HATHAWAY RJ, BEZDEK JC. Fuzzy c-Means Clustering of Incomplete Data[J]. IEEE Transactions on Systems, Man, and Cybernetics, 2001, 31(5): 80-88.

[6] 邸凯昌. 空间数据发掘与知识发现[M]. 武汉: 武汉大学出版社, 2003. 158-172.