

关联规则 数据库 数据挖掘 周期性

50-53

经常性周期关联规则的研究

A Study of Frequent Cyclic Association Rule

黄益民

(西南师范大学计算机科学系 重庆 400715)

TP311.13

Abstract One of the most important data mining problems is mining association rules. In this paper, we considered the problem of founding frequent cyclic association rules. By exploiting the relationship between cycles and large itemsets, we identified optimization techniques that allow us to minimize the unnecessary amount of work performed during the data mining process. Furthermore, we demonstrated the effectiveness of these methods through a series of experiments.

Keywords Data mining, Association rule, Transaction database

近年来,数据挖掘问题引起了人工智能、数据库、统计学等各方面的专家的广泛兴趣,文[1]首先提出了关联规则的发现,随后如文[2~4]等进行了广泛的研究,但以上的工作都是将数据库看成是一个整体,没有考虑时间段的问题。本文考虑了发现经常性周期关联规则的问题。假如在一系列有一定周期间隔的时间单元内,一条关联规则的发生次数(即支持度和信度同时超过指定值的次数)达到用户指定的比率(最小经常性信度),就称此规则是一条“经常性周期关联规则”。本文提出了发现这种规则的多种不同方法并通过实验比较了它们的性能。相信这些技术将有助于确认销售趋势与用户需求。下面首先给出了经常性周期关联规则的数学定义,然后考虑直接扩展已有的关联规则发现算法来解决这个问题。

1 问题定义

设 $I = \{i_1, i_2, i_3, \dots, i_m\}$ 是一个项目的集合。 T 是一个事务的集合,每个 T 中的事务 S 都是一个项目子集,即 $S \subset I$ 。在这里的每一个事务 S 中,每个项目的购买数量未被考虑,这使得每个项目都可以用一个二进制变量来表示。

假设 X 是一个项目的集合 I 的一个子集,称 S 包含 X 当且仅当 $X \subset S$ 。包含 k 个项目的项目集(itemset)被称为“长度为 k 的项目集”。

为了能处理经常性周期关联规则,假定数据库中的数据可以被划分成 n 个有时间标记的数据子集。每一个数据子集对应于某一时间单元内的事务。标记第 i 个时间单元为 $t_i, 0 \leq i < n+1$, 相当于时间间隔 $((i-1) \times t, i \times t)$, 这里 t 是时间的单位。对每个时间单元

t_i , 用 d_i 标记在 t_i 内执行的事务的集合,称 d_i 为时间单元 t_i 对应的数据子集。因此,数据库 D 可以被表示为 d_i 的序列: $D = d_1, d_2, d_3, \dots, d_n$, 这里 d_i 中的一条关联规则具有如下形式: $X \rightarrow Y, X \subset I, Y \subset I$, 并且 $X \cap Y = \emptyset$ 。如果在 d_i 中同时包含 X 和 Y 的事务的比率为 s , 称规则 $X \rightarrow Y$ 在 d_i 中的支持度为 $s\%$ 。如果 d_i 中包含 X 的事务同时包含 Y 的比率为 c , 称规则 $X \rightarrow Y$ 在 d_i 中的信度为 $c\%$ 。

对于给定最小信度域值 min-conf 和最小支持度域值 min-sup , 称 d_i 中的一个项目集 X 是“大”(large)的, 当且仅当 d_i 中包含 X 的事务不少于 min-sup , 如果规则 $X \rightarrow Y$ 在 d_i 中的支持度和信度不小于 min-sup 和 min-conf , 称 $X \rightarrow Y$ 是 d_i 中的一条关联规则。

本文用数对 (l, o) 来表示一个时间周期, l 是时间间隔的长度, o 是偏移量, 即循环发生的第一个时间单元, $0 \leq o < l$ 。对于给定的时间周期 (l, o) , 如果对时间单元 t_i , 有 $i \bmod l = o$, 则称时间单元 t_i 在周期 (l, o) 中。将所有属于周期 (l, o) 的时间单元记为 $T(l, o)$, 即:

$$T(l, o) = \{t_{k \cdot l + o} \mid k \in \mathbb{Z}, 0 \leq k < n/l\}$$

当时间单元 t_i 在周期 (l, o) 中时, 其对应的数据子集 d_i 也被称为是在周期 (l, o) 中的。将所有在周期 (l, o) 中的数据子集记为 $D(l, o)$, 于是有:

$$D(l, o) = \{d_{k \cdot l + o} \mid k \in \mathbb{Z}, 0 \leq k < n/l\}$$

在本文中, 记周期 (l, o) 中的第 k 个时间单元为 $d_{k \cdot l + o}^i$, 在没有二义性的情况下, 它将被简单地记为 d^k 。

一条规则 $X \rightarrow Y$ 在周期 (l, o) 中的经常性计数(frequency-count)和经常性信度(frequency-confidence)被定义为:

$$\text{frequency-count}(X \rightarrow Y) = |\{k \mid 0 \leq k < n/l\}|$$

且 $X \rightarrow Y$ 是 $d_{(l,o)}^k$ 的一条关联规则; $\text{frequency_confidence}(X \rightarrow Y) = \text{frequency_count}(X \rightarrow Y) / |I|$; $0 \leq z < n$ 且 $\text{mod } l = 0$ |

如果一条关联规则在周期 (l, o) 中的经常性信度不小于一个指定的阈值——最小经常性信度 (min_freq_conf), 则称此关联规则在周期 (l, o) 中是经常性的。这里, 为了使发现的规则有意义, 最小经常性信度阈值应该是一个较大的值。在数据库中发现经常性关联规则就是要在给定周期或周期范围的情况下, 找出所有经常的关联规则。

与经常性关联规则相似, 还可以定义“大”项目集的“经常性”属性。在周期 (l, o) 中一个项目集被称为是“经常性”的, 当且仅当此项目集在 (l, o) 内不少于“最小经常性信度阈值 $\times (l, o)$ 中数据子集个数”的数据子集中是“大”的。

2 发现经常性周期关联规则的方法

在本节将探索在时间序事务数据库中发现经常性周期关联规则的方法。这里主要考虑的是在指定的周期内发现经常性关联规则的问题。对于周期在一定范围内变化时的经常性关联规则发现问题, 将另行讨论。

2.1 直接方法

在给定周期时发现经常性关联规则的最直接方法是: 使用某种已有的算法, 如文[2,3], 在每个时间单元中发现所有的关联规则, 然后用统计方法发现经常性规则。

在目前所提出的各种算法中, Apriori^[2] 算法及其各种改进形式是最成功的。Apriori 算法从长度为1的项目集开始, 不断循环, 每次产生长度更大的“大”项目集。在每一次循环中, 首先构造一个由候选项目集组成的集合, 然后扫描数据库, 计算项目集的支持度, 并在循环过程中根据子集是否“大”删去不可能“大”的候选项目集(称为支持剪枝, support-pruning)。这样在扫描数据库进行计数时需要检验的项目集数目可以大幅度减少, 从而提高程序运行的速度。在“大”项目集产生以后, 超过最小信度域值 min_conf 的规则可根据某种已有的算法^[3] 产生。在每个时间单元中的关联规则都被发现以后, 即可统计产生经常性周期关联规则。

经过仔细研究, 可以发现实际上统计关联规则在 (l, o) 中的所有数据子集里出现的次数是不必要的。例如, 如果知道某规则不在 (l, o) 中的头 $|D(l, o)| \times (1 - \text{min_freq_conf}) + 1$ 个数据子集里出现, 则可以得出结论: 这条规则不是周期 (l, o) 中的经常性规则。一般地, 对于经常性周期关联规则, 有如下的性质:

在统计了 $D(l, o)$ 中的 m 个数据子集后, 这里 $m > |D(l, o)| \times (1 - \text{min_freq_conf}) + 1$, 如果一条关联规则

出现的次数小于 $m - n \times (1 - \text{min_freq_conf})$, 则此规则在 (l, o) 中不可能是经常性的。

因此, 可以用方法1在 $D(l, o)$ 中发现经常性关联规则:

方法1

1. 用 Apriori (或其他) 算法发现所有在 $D(l, o)$ 中的数据子集里的关联规则。
2. 置 (l, o) 的候选经常性关联规则集为空。对于开始的 $|D(l, o)| \times (1 - \text{min_freq_conf}) + 1$ 个数据子集, 将它们中的关联规则插入到候选集中, 如果候选集中当前还没有此规则, 则将其放入并置其计数为1, 如此规则已在候选集中, 其计数增1。
3. 对剩下的每个数据子集, 检查第2步得到的候选集中的规则在其中是否成立, 如果是, 候选集中规则的计数加1; 否则, 检查此规则的计数, 如果计数值小于这一步已查看的数据子集个数, 将其从候选集中删去(此方法被称为经常性剪枝)。

经常性剪枝方法可以与支持度剪枝相结合, 产生更为有效的经常性关联规则发现算法。

2.2 经常性剪枝

方法1的主要运行时间是花费在计算每一个数据子集中各项目集的支持度上, 有效利用经常性剪枝技术可以减少运行时间, 此技术的核心在于减少需要计算支持度的项目集的数目。这种技术被描述如下:

性质1 给定周期 (l, o) 中的所有数据子集和最小经常性域值 (min_freq_conf), 如果发现一条规则(或“大”项目集)在 (l, o) 中的数据子集里未出现的次数大于 $|D(l, o)| \times (1 - \text{min_freq_conf}) + 1$, 则可以得出结论, 此规则(或“大”项目集)在 (l, o) 中不是经常性的。

因此, 不需要对周期 (l, o) 中的数据子集产生所有的规则(或“大”项目集), 在大多数情况下, 最小经常性信度域值 (min_freq_conf) 较大, 因而 $|D(l, o)| \times (1 - \text{min_freq_conf}) + 1$ 的值较小, 所以能够有效地发现非经常性关联规则和“大”项目集。

为了有效地将经常性剪枝与支持度剪枝相结合, 可利用如下的性质:

性质2 如果关联规则 $X \rightarrow Y$ 在周期 (l, o) 中的经常性信度为 u , 则项目集 X 和 $X \cap Y$ 都是 (l, o) 中的“大”项目集, 且它们的经常性信度不小于 u 。

根据经常性关联规则和经常性“大”项目集的定义, 此性质是显然的, 因此如果 $X \rightarrow Y$ 是 $d_{(l,o)}^k$ 中的关联规则, 那 $X, X \cap Y$ 都是 $d_{(l,o)}^k$ 中的“大”项目集。

性质3 如果“大”项目集 X 在 (l, o) 中的经常性信度为 u , 则 X 的所有子集在 (l, o) 中的经常性信度不小于 u 。

如果项目集 X 在 $d_{(l,o)}^k$ 中是“大”的, 它的所有子集在 $d_{(l,o)}^k$ 中也是“大”的, 此结论显然成立。

2.3 对于“大”项目集的经常性剪枝

通过结合经常性剪枝和性质2,可以先发现所有的经常性“大”项目集,然后产生经常性规则。此方法被简要描述如下:

方法2

1. 首先在 (I, σ) 的头 $|D(I, \sigma)| > (1 - \text{min-freq-conf}) + 1$ 个数据子集中发现所有的“大”的项目集,将它们及其计数放入一个经常性“大”项目集的候选集中;
2. 然后,对于 (I, σ) 中剩下的每一个数据子集,只需检验候选集中所有的“大”项目集,看项目集的支持度是否大于最小支持度阈值(min-sup),如果是,候选集中此项目集的计数加1;在对每个数据子集扫描完成后对候选集进行经常性剪枝。
3. 在所有的经常性“大”项目集都被发现以后,可用它们计算每一个数据子集中的关联规则,再使用统计或经常性剪枝方法产生经常性关联规则。

需要说明的是,在第2步中,由于只是检查候选集中项目集是否是“大”的,被检查的数据子集只需要被扫描一次。由于 min-freq-conf 一般较大,所以在第1步中用于产生候选集的数据子集的数目较小,而对于算法Apriori,发现一个数据子集中所有关联规则所需的时间比扫描一遍数据子集所花的时间要多得多,因此此方法可以有效地减少计算时间。也注意到第3步中对每个数据子集产生关联规则时,需要知道此数据子集中所有“大”项目集的支持度。由于是事先产生所有经常性“大”项目集,为了避免再次扫描数据库,需要存储每个数据子集中候选“大”项目集的支持度。在数据子集的数目较大或候选“大”项目集的项目数较大时,需要一定的存储空间。为克服此不足,同时加大剪枝的力度,通过对此方法的改进,可以得到方法3。

根据性质1,如果在 (I, σ) 中一条规则是经常性的,它在前 $|D(I, \sigma)| \times (1 - \text{min-freq-conf}) + 1$ 个数据子集中至少出现一次。可以先找出这些规则,在以后的数据子集里只进一步确认它们是经常性的。

方法3

1. 首先找到 (I, σ) 里前 $|D(I, \sigma)| \times \text{min-freq-conf} + 1$ 个数据子集中的所有关联规则,将这些规则及它们出现的次数存入一个经常性规则的候选集。
2. 对于剩下的数据子集,检查候选集中的规则是否是在这些数据子集中,并进行经常性剪枝。

在第2步完成后,候选集中的所有关联规则在 (I, σ) 中都是经常性的。与方法2相比,此方法效率更高,因为它在剪枝时考虑了关联规则的信用,并将删去不属于任何规则的“大”项目集。

也可以将经常性剪枝与性质3相结合。显然,可以对于任何长度的“大”项目集使用经常性剪枝。大量的实验表明,在一个数据子集中产生所有的“大”项目集的时间主要花在发现长度为2的“大”项目集上。这是因为在大多数情况下,长度为2的“大”项目集的候选集是

所有候选集中最大的。如果能够减小长度为2的候选项目集,就可以显著节省时间。

我们的目标是发现所有经常性“大”项目集,所以不需要知道数据子集中的所有“大”项目集,只需找出那些可能是经常性的项目集即可。根据性质3,如果大小为 k 的“大”项目集是经常性的,那么它的所有长度为 $(k-1)$ 的“大”项目子集也是经常性的。因此,可以首先找出所有长度为 $k-1$ 的经常性“大”项目集;然后,用Apriori算法产生长度为 k 的经常性“大”项目集的候选集,再扫描所有数据子集产生长度为 k 的经常性“大”项目集。实验表明, (I, σ) 中长度为 $k-1$ 的经常性“大”项目集的数目一般小于单个数据子集中长度为 $k-1$ 的“大”项目集的数目。因此用周期 (I, σ) 中的长度为 $k-1$ 的经常性“大”项目集产生的长度为 k 的经常性“大”项目集的候选集比用单个数据子集中长度为 $k-1$ 的“大”项目集产生的候选集要小。在扫描数据库时使用这个候选集将减少程序运行的时间。将此想法与经常性剪枝相结合,可以得到方法4。

方法4

1. 在 (I, σ) 的前 $|D(I, \sigma)| > \text{min-freq-conf} + 1$ 个数据子集中发现所有长度为1的“大”项目集,并将它们及其计数放入长度为1的经常性“大”项目集的候选集;
2. 对于剩下的每一个数据子集,检查候选集中的项目集在它中间是否是“大”的,如果是,则此项目集在候选集中的计数增1;在一个数据子集扫描完成后,对候选集进行经常性剪枝。(此过程完成后,候选集中将包括所有长度为1的经常性“大”项目集。)
3. 假定已经得到了 (I, σ) 中所有长度为 $k-1$ 的经常性“大”项目集,根据Apriori算法产生所有长度为 k 的经常性“大”项目集的候选集。
4. 对于 (I, σ) 的前 $|D(I, \sigma)| \times \text{min-freq-conf} + 1$ 个数据子集,检查候选集中的项目集是否是“大”的,并对“大”的次数计数。本步骤完成后,删除候选集中所有计数为0的项目集。
5. 对于剩下的每一个数据子集,检查候选集中的项目集在它中间是否是“大”的,如果是,则此项目集在候选集中的计数增1;在一个数据子集扫描完成后,对候选集进行经常性剪枝。(此过程完成后,候选集中将包括所有长度为 k 的经常性“大”项目集。)
6. 重复步骤3到步骤5,直到某个经常性“大”项目集为空。
7. 到此,已经得到了 (I, σ) 中所有经常性“大”项目集,可以用它们产生每个数据子集中的关联规则,然后使用统计方法并结合经常性剪枝来获得所有的经常性关联规则。

正如前面所讨论的,在某一个数据子集中产生关联规则时,需要知道它中间“大”项目集的支持度。在实验中,发现时,将正被扫描的数据子集中经常性“大”项目集的支持度存放到一个磁盘文件里,这样可在产生关联规则(第7步)时用此文件作为数据输入,避免重新扫描一遍数据子集。

3 性能研究

3.1 测试数据集的产生

本文所用到的测试数据集是按照文[2]中所描述的数据产生算法得到的,为了提高“大”项目集的数目,算法按文[3]的方法进行了修改。为了产生经常性关联规则,还加入了一个新的参数 u 。所有参数的意义及缺省取值如下表:

符号	含义	缺省取值
D	每个数据子集中事务的数目	10000
T	事务的平均大小	10
I	可能的“大”项目集的平均大小	4
L	可能的“大”项目集的数目	500
N	项目数	1000
K	周期内数据子集的个数	20
u	数据子集的相似性参数	0.8

为了强调实际情况下,有周期性关联规则也有非周期性关联规则的事实,实验中不是用同一个 L 来产生所有的数据子集,而是为每一个数据子集产生一个 L ,并用相似性参数 u 来刻画一个周期中数据子集的相似性。 u 是一个百分比,我们首先产生一个有 L/u 个可能的“大”项目集组成的候选集。然后,每次从中选出 L 个项目集来产生数据子集。一个项目集被选中的可能性与其产生时得到的权值有关。这样,当 u 越大时,用于产生各个数据子集的项目集的集合就越相似,数据子集也就越相似。这也意味着发现的经常性关联规则更多。

3.2 性能比较的结果

在实验中使用的计算机 CPU 为 Pentium,主频 300MHz,有 64M 内存,操作系统为 Windows98/NT。程序用 Microsoft Visual C++ 6.0 编写。实验使用 Apriori 作为基本的关联规则发现算法。与文[2]中一样,用哈希树存储候选项目集以提高匹配速度。对方法 1,使用的是文[5]中的 Apriori 算法及规则生成算法。在统计规则及“大”项目集的经常性计数时,也使用了哈希表,其结构与 Apriori 算法中哈希表的结构类似。

3.2.1 数据子集数目变化时性能的变化 当周期中数据子集的数目从 5 变到了 30 时,方法 1,3,4 所用的时间都在线性增加。当最小经常性信度为 0.8 时,方法 1 的执行时间大约是算法 3 和 4 的执行时间的 4 到 5 倍。这显然是因为在方法 3 和 4 中只有 20% 的数据子集被用于产生“大”项目集,其余的数据子集只被扫描了一次。

扫描一个数据子集的时间比产生“大”项目集使用的时间短得多,所以整体时间花费大大减小。

在实验中,方法 4 的执行时间比 3 要稍微少一些。这是因为方法 4 可以对长度为 1 的经常性“大”项目集进行经常性剪枝,从而减少了长度为 2 的“大”项目集的数目。

3.2.2 最小经常性信度发生变化时性能的变化

最小经常性信度的变化(从 60% 到 90%)对于方法 1 无影响;而对方法 3 和 4,随着此阈值的增加,花费的时间将减少,这是因为用于产生经常性项目集的候选集的数据子集个数减少了。

3.2.3 相似性参数对性能的影响 相似性参数 u 在 90% 到 50% 之间变化时,方法 3 的执行时间变化不大而方法 4 的执行时间有明显减少。如上面讨论的那样,此参数减少时,经常性“大”项目集的数目也会随之减少。对方法 3,其第一部分的执行时间不受此参数的影响,第二部分的执行时间略有减少。对方法 4,由于长度为 2 的“大”经常性项目集的候选集的缩小,时间花费明显减少了。

结论 本文研究了在一系列具有周期性的时间单元中发现周期性关联规则的问题。通过仔细研究周期、数据子集及“大”项目集等的联系,本文提出了多种发现周期性关联规则的方法并在一系列实验中,比较了三种主要方法的性能差异。

目前对于周期性关联规则的问题还有许多方面值得进一步研究,如在周期可变的情况下如何发现规则,如何在周期可变时减少时间花费,如何更有效地将支持度剪枝与经常性剪枝结合使用等等。作者准备进一步对这些问题进行研究。

参考文献

- 1 Agrawal R, et al. Mining Association Rules between Sets of Items in Large Databases. In: Proc. of the 1993 ACM SIGMOD Intl. Conf. on Management of Data, Washington, DC, May 1993
- 2 Arimura R, Syriani R. Fast Algorithms for Mining Association Rules. In: Proc. of the 20th Intl. Conf on VLDB, Santiago, Chile, September 1994
- 3 Park J S, et al. An Effective Hash-based Algorithm for Mining Association Rules. In: Proc. of the 1995 ACM SIGMOD Intl. Conf. on Management of Data, May 1995
- 4 O'zden B, et al. Cyclic Association Rules. In: Proc. of the 1998 Intl. Conf. on Data Engineering (ICDE'98), Orlando, FL, Feb. 1998
- 5 Agrawal R, et al. Fast Discovery of Association Rules. In: Advances in Knowledge Discovery and Data Mining, AAAI Press, 1995