高效的基于动态数据更新的关联规则挖掘算法

High – efficient algorithms of mining association rules based on incremental updates

刘 松　林海萍　(广州 广东商学院信息学院 510320)

摘要:在本文中,我们针对动态关联规则挖掘问题提出两个有效的处理算法,即 EIM – A 和 EIM – G 算法。它们能根据数据库的动态变化,高效地进行关联规则的更新。通过知识数据库的维护,最多只需要扫描原始数据库一次,就能得到所需的频繁项目集,能有效地降低更新关联规则所需的成本。

关键词:关联规则　动态关联规则挖掘　变动数据　原始数据库

1　引言

关联规则是目前应用最广泛的数据挖掘技术,在以往的研究中,已经提出了不少算法来改善关联规则的性能,例如 Apriori、Partition、DHP、DIC 等,但这些算法一般都只适用于静态数据库的挖掘,如果数据库的内容发生变动时,则必须重新执行一次该算法才能更新有关规则。显然这种做法是相当费时的。为了解决这种动态关联规则维护更新的问题,已经有一些研究提出了解决方法,如 FUP 、SWF 算法等,虽然这些算法比静态挖掘关联规则节省了许多的成本并提高了效率,但这些方法仍有不足之处,如有的需要多次的扫描原始数据库以得到某些候选项目集,有的需要重新产生大量的候选 k – 项目集来更新频繁项目集,使得性能降低。

在本文中,我们针对关联规则的动态维护问题提出两个新的算法 EIM – A 和 EIM – G。这两个算法都是通过一个知识数据库的建立来维护更新关联规则所需要的信息。EIM – A 算法专门针对数据库中有新增数据的情况,EIM – G 算法可同时处理新增和删除数据的情况,同时为了避免产生的候选项目集过大,在算法中我们还采用了哈希表技术来压缩候选项目集的大小。通过上述处理方式可将扫描原始数据库的次数缩减到最多只要利用数据集中所找出的频繁项目集重新扫描原始数据库一次,就可以得到更新后的全部频繁项目集。

2　名词和符号的定义

在这一部分,将给出我们算法中所涉及到的名词、符号及其定义说明。

名词、符号	定义说明
KDB	知识数据库
DB	原始数据库中所有数据集合
$\Delta+$	新增数据库中所有数据集合
$\Delta-$	删除数据库中所有数据集合
UD	更新后数据库中所有数据集合
ADB	交易数据合并数据库
MII	知识数据库中的最小非频繁项目集
L^k_{DB}	原始数据库 DB 中所保留已挖掘出的频繁 k – 项集集合
L^k_{db}	新增数据库 $\Delta+$ 中所挖掘出的频繁 k – 项集集合
L^k_{UD}	更新后数据库 UD 中挖掘出的频繁 k – 项集集合
C^k_{db}	新增数据库 $\Delta+$ 中的侯选 k – 项集集合
minsup	最小支持度阀值(%)
δ_X	项目集出现在 DB 中的次数
δ^+_X	项目集出现在 $\Delta+$ 中的次数
δ^-_X	项目集出现在 $\Delta-$ 中的次数
σ_X	项目集出现在 UD 中的次数
|DB|	原始数据库中的交易数据量
$\|\Delta+\|$	新增数据库中的交易数据量
$\|\Delta-\|$	删除数据库中的交易数据量
|UD|	更新后数据库中的交易数据量
X. supportDB	X 项目集在 DB 中的支持度

定义1 更新后的数据库UD为原始数据库DB加上新增数据库Δ+再减去删除数据库Δ−。

$$UD = DB + \Delta + - \Delta -$$

定义2 项目集X在更新后数据库中出现的次数σ_X为在原始数据库中出现次数δ_X与新增数据库中出现次数δ_X^+的和再减去在删除数据库中出现的次数δ_X^-。

$$\sigma_X = \delta_X + \delta_X^+ - \delta_X^-$$

3 定理及其证明

在这一部分，将针对我们所提出的算法给出相关的理论及其证明，以保证我们算法执行的正确性。

定理1：假设X为一长度为k的候选项目集，则当X为非频繁项目集时，包含X的所有父集也一定是非频繁项目集。

证明：假设Y为X的父集，则可知Y在交易中出现的次数一定小于或等于X出现的次数，所以$\delta_X < minsup$而且$Y < X$，所以$\delta_Y < minsup$。所以当X为非频繁项目集时，其父集也必为非频繁项目集。

定理2：如果X为一在原始数据库DB中长度为k的频繁项目集，则当X在新增的数据库Δ+中为频繁项目集时，则X必然在更新后的数据库UD中为一频繁项目集。

证明：X在原始数据库中为一频繁项目集$=> \delta_X \geqslant |DB| \times minsup$

又X在新增数据库中也为一频繁项目集$=> \delta_X^+ \geqslant |\Delta +| \times minsup$

$\therefore \delta_X + \delta_X^+ \geqslant (|DB| + |\Delta +|) \times minsup$

定理3：如果一个项目集X是L_{UD}中的一个成员，则X一定是一个属于L_{DB}或L_{db}，或是同时属于L_{DB}和L_{db}的成员。

证明：假设X不属于L_{DB}或L_{db}，则有$\delta_X < |DB| \times minsup$而且$\delta_X^+ < |\Delta +| \times minsup$

$\therefore \delta_X + \delta_X^+ < (|DB| + |\Delta +|) \times minsup = |UD| \times minsup$，则X必不会出现在$L_{UD}$中。

所以当X出现在L_{UD}中时，则必至少出现于L_{DB}或L_{db}中的一个。

定理4：如果一个长度为k的项目集X属于L_{DB}^k，但不属于L_{UD}^k，则其一定不属于L_{db}^k。

证明：X属于L_{DB}^k，有$\delta_X \geqslant |DB| \times minsup$，而X不属于$L_{UD}^k$，有$\sigma_X < |UD| \times minsup$

假设$\delta_X^+ \geqslant |\Delta +| \times minsup$，则$\delta_X + \delta_X^+ \geqslant (|DB| + |\Delta +|) \times minsup = |UD| \times minsup$

但已知$\sigma_X < |UD| \times minsup$，所以假设不成立，即X一定不属于$L_{db}^k$。

定理5：利用L_{DB}^k中所有项目集扫描Δ+，则一定可以找出所有$L_{DB}^k \cap L_{db}^k$的频繁项目集。

证明：当$\delta_X^+ > |\Delta +| \times minsup$时，X为Δ+中的频繁项目集。利用$L_{DB}^k$中所有的项目集X扫描Δ+，则能找出这些项目集在Δ+中出现的次数，从而能判断其是否为Δ+中的频繁项目集。所以可以找出$L_{DB}^k \cap L_{db}^k$中的所有频繁项目集。

4 知识数据库

在算法中，我们采用了知识数据库来存储更新关联规则过程中所需要的信息，在其中保留的维护信息包括如下四项：

(1) Frequent Itemsets in DB

在原始数据库DB中通过某种静态挖掘算法找出的频繁项目集。

(2) Minimal Infrequent Itemsets

最小非频繁项目集是非频繁项目集且其没有子集或其子集都为频繁项目集的集合。在算法中用来生成我们需要的候选项目集。

(3) All 1− item counts

所有1−项集出现的次数。在挖掘过程中，可根据知识数据库中的1−项集来缩减DB数据库交易数据的长度。

(4) Hashing Table

哈希表用来判断知识数据库中的非频繁项目集，在删除数据的时候将显著地提高效率。

5 交易数据合并数据库ADB

为了进一步减少对原始数据库DB的I/O成本，我们在此利用了将相同交易数据做合并的处理，当交易数据相同时，则将其count做累加的计算，形成一个合并数据库ADB，这样可以大量减少交易数据的笔数。

假设交易数据同时发生的平均重复笔数为N笔，

则将交易数据库做合并后的交易数据量将会变成(|DB| ×1/N)。同样地,由于交易数据量的减少,则 I/O 成本的花费也将减少变成(|DB| ×1/N)。若平均重复交易数据量的 N 值越大,则合并后的交易数据库 ADB 的笔数也会减少越多,进行数据挖掘的速度则会相应的提高。

6 EIM – A 算法

EIM – A 算法首先对知识数据库中所保留的利用其他静态关联规则算法所找出的 DB 中的频繁项目集做更新的处理,以确定其是否是频繁项目集,接着再找出所有 Δ + 和 DB 中已更新过但不重复的频繁项目集,将这些项目集当成要扫描 DB 的候选项目集。由于这些项目集可能并不包含于知识数据库中,所以还必须以这些候选项目集扫描一次 DB 以得到正确更新后的出现次数。

在我们的算法中,为了减少扫描 Δ + 数据库的次数,采用了 DIC 算法来挖掘 Δ + 中的频繁项目集。DIC (Dynamic Itemset Counting) 算法是 Brin et al. 在 1997 年提出的,其主要的目的在于减少 I/O 存取所花费的时间。DIC 算法将数据库分割成许多个区段,搜寻数据库时是以区段为单位。由于在搜寻的过程中,有些候选项目集在某些区段中的支持度计算便已大于或等于使用者所定的最小支持度阀值,因此在搜寻完这些区段之后不必等到整个数据库都搜寻完毕便可马上过滤出频繁项目集并直接联结产生下一层次的候选项目集。利用这个特性,一旦新的候选项目集产生之后,便可直接在下一个区段开始计算其支持度。经过这样在每个区段中反复的过程,直到所有的频繁项目集都找出为止,可大大减少搜寻数据库的次数进而提高其性能。

6.1 执行流程

EIM – A 算法处理新增数据的流程如下:

步骤 1:由知识数据库中取得原始数据库 DB 中的各长度为 k 的频繁项目集集合 L^k_{DB},以及各频繁项目集出现在交易数据合并数据库 ADB 中的次数。

步骤 2:由于 L^k_{DB} 中各个频繁项目集在更新后的数据库中可能仍保持频繁,也可能变成非频繁项目集。所以在这个步骤,我们用全部 L^k_{DB} 中的频繁项目集扫描一次新增数据库 Δ +,以确定这些项目集在新增数据库 Δ + 中出现的次数。

步骤 3:为了减少读取数据库所需付出的 I/O 成本,在扫描新增数据库 Δ + 的同时,可开始进行搜寻 Δ + 中的频繁项目集的处理,即根据 DIC 算法,在第一个区段开始时,除了搜寻 L^k_{DB} 中这些频繁项目集在 Δ + 中出现的次数,也同时开始搜寻 Δ + 中 L^1_{db} 的次数。

步骤 4:在扫描完一次 Δ + 后,可以得到所有 L^k_{DB} 项目集在 Δ + 中出现的次数,和原来在 DB 中的次数相加后,则可以判断这些项目集是否为更新后数据库 UD 中的频繁项目集。假设 $X \in L^k_{DB}$,当 $\delta_X + \delta^+_X \geq |UD| \times minsup$,则将这些 X 放到知识数据库中成为 L^k_{UD};如果 $\delta_X + \delta^+_X < |UD| \times minsup$,即在 UD 中,X 已变成非频繁项目集,则将这些 X 放到 Prune_Set 中。根据定理 2,在 DB 与 Δ + 中都为频繁项目集的,则在 UD 中也一定为频繁项目集,这些项目集也会在此步骤被处理。

步骤 5:利用前面步骤所计算的原始数据库 DB 中这些频繁项目集所得到的信息,可进一步减少 Δ + 中产生的候选项目集。根据定理 4,如果一个长度为 k 的项目集 X 属于 L^k_{DB},而不属于 L^k_{UD},则表示 X 在 Δ + 中为非频繁项目集,即在步骤 4 中我们放到 Prune_Set 中的那些项目集。由于在 Prune_Set 中的项目集在 Δ + 中都为非频繁项目集,根据定理 1,包含在 Prune_Set 中的项目的父项目集也一定为非频繁项目集,所以在计算完原始数据库中的频繁项目集后,便可以开始利用 Prune_Set 做过滤的处理,只要在 Δ + 中所产生的候选项目集为存在于 Prune_Set 中项目集的父集合,则不必再计算这些候选项目集出现的次数。

步骤 6:在找出所有 Δ + 中的频繁项目集 L^k_{db} 后,还必须找出这些频繁项目集在原始数据库 DB 中的次数,以确定其是否在更新后数据库 UD 中为频繁项目集。由于在步骤 4 时,已经完整地更新过所有 L^k_{DB} 中的项目集。根据定理 5,也同时已经处理过 $L^k_{DB} \cap L^k_{db}$ 中的项目集,并且这些项目集根据定理 2 也已经存在于 L^k_{UD} 中。所以在这个步骤,只需把 L^k_{db} 中没有出现在 L^k_{DB} 中的项目集当成更新后数据库的候选项目集 L^k_{UD},重新再扫描一次相同交易数据合并过的数据库 ADB。

步骤 7:根据步骤 6 我们可以得到 L^k_{UD} 中的项目集在 DB 中的次数,并且这些项目集在 Δ + 中的次数也都被记录在 L^k_{db} 中。假设 X 为 L^k_{UD} 中的项目集,则 $\delta_X + \delta^+_X \geq |UD| \times minsup \Rightarrow X \in L^k_{UD}$。

6.2 时间复杂度分析

假设知识数据库中 L^k_{DB} 的项目集有 $|L^k_{DB}|$ 个，而新增数据库 $\Delta+$ 所产生的频繁项目集 L^k_{db} 有 $|L^k_{db}|$ 个，而每个项目集和一笔交易数据分配的时间为 t，花在找 $\Delta+$ 中频繁项目集的时间为 TI，总共花的时间为 T。则我们可以对算法的时间复杂度做如下分析：

(1) L^k_{DB} 和 $\Delta+$ 所产生的频繁项目集没有交集时：

$$T = (|L^k_{DB}| \times |\Delta+| + |L^k_{db}| \times |ADB|) \times t + TI$$

(2) L^k_{DB} 和 $\Delta+$ 所产生的频繁项目集有交集时：

$$T = (|L^k_{DB}| \times |\Delta+| + |L^k_{db}| - |L^k_{db} \quad L^k_{DB}| \times |ADB|) \times t + TI$$

而当 L^k_{db} 全部都包含于 L^k_{DB} 中时，可得 $|L^k_{db}| - |L^k_{db} \subset L^k_{DB}| = 0$，所以可以将(2)的公式简化如下：

$$T = |L^k_{DB}| \times |\Delta+| \times t + TI$$

由以上的式子可以发现，在我们的算法中，当 L^k_{db} 和 L^k_{DB} 有交集时，则只需要以 $|L^k_{db}| - |L^k_{db} \subset L^k_{DB}|$ 的量对 ADB 扫描一次，没有交集时也只需要利用 $|L^k_{db}|$ 扫描一次 ADB 则可以得到更新后的关联规则。

7 EIM-G 算法

7.1 执行流程

步骤 1：由知识数据库中得到原始数据库 DB 中的各长度为 k 的频繁项目集集合 L^k_{DB}，以及保留的最小非频繁项目集。

步骤 2：由于 L^k_{DB} 中的频繁项目集和最小非频繁项目集可能在更新后的数据库仍保持频繁，也可能变成非频繁项目集。所以在这个步骤，我们更新这些项目集在 $\Delta-$ 和 $\Delta+$ 中出现的次数。

步骤 3：检查 L^k_{DB} 中的频繁项目集是否仍为频繁项目集，并且更新维护知识数据库中的哈希表。

步骤 4：由于最小非频繁项目集在 $\Delta-$ 中出现的次数较少，而 |UD| 又会因为数据量的减少而使得其支持度降低，所以非频繁项目集有可能变成频繁项目集，所以在这个步骤检查最小非频繁项目集，找出由非频繁转为频繁的项目集，再利用这些项目集与步骤 3 已更新过的频繁项目集结合，产生要扫描 UD 中的新的候选项目集，而候选项目集的最大长度则可以由知识数据库中的 1-项集得到，并利用知识数据库中所维护的哈希表过滤候选项目集。

步骤 5：利用这些候选项目集扫描一次更新后的数据库 $(ADB-\Delta-)+\Delta+$，找出所有候选项目集在 UD 中出现的次数，如果候选项目集 X 出现的次数为

$\sigma_X \geq |UD| \times minsup$，则这些项目集则成为更新数据库中的频繁项目集。

7.2 时间复杂度分析

假设知识数据库中 L^k_{DB} 的项目集有 $|L^k_{DB}|$ 个，最小非频繁项目集有 |MII| 个，由非频繁项目集变成频繁项目集所产生的候选项目集为 |C| 个，而每个项目集和一笔交易数据分配的时间为 t，而新增数据和删除数据集分别为 $\Delta+$ 和 $\Delta-$，总共花的时间为 T。则我们可以对算法的时间复杂度做如下分析：

(1) 有最小非频繁项目集 MII 变成频繁项目集时：

$$T = ((|L^k_{DB}| + |MII|) \times (|\Delta+| + |\Delta-|) + |C| \times |ADB|) \times t$$

(2) 没有最非频繁项目集 MII 变成频繁项目集时：

$$T = ((|L^k_{DB}| + |MII|) \times (|\Delta+| + |\Delta-|)) \times t$$

由以上式子可以发现，当没有任何最小非频繁项目集 MII 变成频繁项目集时，则不需要扫描原始合并数据库 ADB。当有最小非频繁项目集由于数据的变动而变成频繁项目集时，才会产生新的候选项目集，并且以这些候选项目集扫描一次 ADB，进一步得到候选项目集的出现次数，判断其是否为频繁项目集。所以当原始数据库越大时，执行算法的效率将越高。

8 结束语

本文所提出的算法和以往的动态挖掘算法相比，主要是减少了 I/O 成本上的花费，可将扫描原始数据库的次数控制在一次以内，能有效地降低更新关联规则所需要付出的成本，同时在挖掘的过程中还采用了哈希表技术来压缩候选项目集的规模，总的执行效率比以往算法有较大地提高。

当然，在数据挖掘的研究中，除了本文所探讨的关联规则外，还有分类、聚类等各种其他类型的挖掘方法，各方法由于其本身的流程和目标的不同，各有不同

(下转第 58 页)

(上接第 54 页)

的应用范围。所以如何将本文中的方法与其他类型的挖掘算法结合,以期挖掘出更多元化的隐藏信息或更具有实际意义的信息,是我们今后进一步研究的方向。

参考文献

1 S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic Itemset Counting andImplication Rules for Market Basket Data", 1997 ACM SIGMOD Conference onManagement of Data, pp. 255 – 264, 1997.

2 J. Tang, "Using Incremental Pruning to Increase the Efficiency of Dynamic ItemsetCounting for Mining Association Rules", Proceedings of the 1998 ACM 7th international conference on Information and knowledge management, pp. 273 – 280, 1998.

3 J. Hanand J. Pei. Mining "Frequent Patterns by Pattern – Growth: Methodology andImplications". ACM SIGKDD Explorations (Special Issue on Scaleble Data MiningAlgorithms), December 2000.

4 Chang – Hung Lee, Cheng – Ru Lin, and Ming – Syan Chen "Sliding – Window Filtering: An EfficientAlgorithm forIncremental Mining" SIGKDD 2001.