数据流中频繁闭项集的近似挖掘算法

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

摘 要: 在数据流中挖掘频繁项集得到了广泛的研究, 传统的研究方法大多关注于在数据流中挖掘全部频繁项集. 由于挖掘全部频繁项集存在数据和模式冗余问题, 所以对算法的时间和空间效率都具有更大的挑战性. 因此, 近年来人们开始关注在数据流中挖掘频繁闭项集, 其中一个典型的工作就是Moment算法. 本文提出了一种数据流中频繁闭项集的近似挖掘算法A-Moment. 它采用衰减窗口机制、近似计数估计方法和分布式更新信息策略来解决Moment算法中过度依赖于窗口和执行效率低等问题. 实验表明, 该算法在保证挖掘精度的前提下, 可以比Moment获得更好的效率.

关键词: 数据挖掘; 数据流; 频繁闭项集
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2007) 05-0900-06

An Algorithm to Approximately Mine Frequent Closed Itemsets from Data Streams

LIU Xu, MAO Guo-Jun, SUN Yue, 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: Mining frequent itemsets from data streams has extensively been studied, and most of them focus on finding complete set of frequent itemsets in a data stream. Because of numerous redundant data and patterns in main memory, they cannot get very good performance in time and space. Therefore, mining frequent closed itemsets in data streams becomes a new important problem in recent years, where algorithm Moment was regarded as a typical method of them. This paper presents an algorithm, called A-Moment, which uses the damped window technique, approximate count method and distributed updating strategy to get higher mining efficiency. Experimental results show that our algorithm performs much better than the previous approaches.

Key words: data mining; data stream; frequent closed itemset

1 引言

在关联规则、序列模式挖掘等研究领域, 挖掘频繁项集是最基础和最关键的步骤. 挖掘频繁项集是数据挖掘中一个活跃的研究领域. 1994年, R. Agrawal提出了Apriori算法[1], 这是一个最有影响的挖掘布尔关联规则频繁项集的算法. 但是, 由于Apriori算法需要产生庞大的候选项集和多次扫描数据库[2,3], 因此导致较差的时空效率. 2000年, Han等提出了FP-Tree算法[2], 它不使用候选项集而是直接将数据库的扫描结果存放到紧缩的频繁模式树中, 是一个两次数据库扫描的频繁项集挖掘算法. 另一个有代表性的工作是频繁闭项集(Frequent Closed Itemset)的挖掘方法. 1999年, Pasquierti首先提出了闭项集概念[3], 随后挖掘频繁闭项集的问题被广泛研究[4~6]. 由于频繁闭项集只关注挖掘一些特殊的频繁项集, 并且不丢失项集支持度的信息, 因而可以减少项集的存储空间与处理时间, 进而提高挖掘的效率.

数据流(Data Stream)是指可能无限的、持续而快速到达的数据序列. 事实上, 数据流的概念无处不在的. 比如, 一个大型超市每天产生数百万甚至千万条购买记录、一个地球探测卫星每天要产生十亿字节的数据, 并且这些数据随着时间还在不断地增长. 尽管挖掘频繁项集仍然是数据流中知识发现的一个基础性工作, 但是在这类大容量的动态变化的数据流中进行频繁项集挖掘出现了新的具有挑战性的问题, 因此近年来得到广泛关注[7,9,11]. 挖掘数据流算法必须能在有限的内存空间和限定的时间内快速形成模式的归纳, 因此对时间和空间效率的要求要比静态的数据库挖掘要高. 本文提出了一种在数据流中近似地挖掘频繁项集的算法A-Moment. 它以Moment[11]算法的主要数据结构CET为基础, 通过引进衰减窗口机制和近似估计方法等来减少内存使用规模和提高挖掘效率.

2 相关工作

2.1 数据流的窗口模型及其相关算法

根据数据流挖掘的窗口模型[7], 目前出现三种主要技术: 里程碑(Landmark Window)窗口、滑动窗口(Sliding Window)和衰减窗口(Damped Window). 在里程碑窗口处

收稿日期: 2006-03-27; 修回日期: 2006-12-04
基金项目: 国家自然科学基金重大项目(No. 60496322, 60496327)

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

理模型中, 它们总是关注整个数据流中的数据, 并通过对整个历史数据的分析得到全局性的频繁模式. 的确, 全局性的知识模式是许多数据流挖掘中的期望结果. 但是, 由于数据流的大容量和不可预测的数据到达速度, 近期的研究表明里程碑窗口处理模型必须结合快速的近似归纳技术或合适的数据淘汰技术才能真正适合数据流的挖掘[8]. 最有代表性的提出基于里程碑窗口的数据流挖掘算法是方法是 Lossy Counting[9], 它是基于 Apriori 算法思想, 但是利用近似归纳技术实现数据一次扫描.

在滑动窗口处理模型中, 关注点总是被放在最近发生的若干事务上, 因此, 它们的挖掘结果是某段时间内的局部频繁模式. 在大多数的数据流环境中, 这种局部模式是不适合的, 这应该说是滑动窗口的固有缺陷. 但是, 滑动窗口处理模型具有易于理解、设计简单等优点, 因此在数据流挖掘中也得到广泛的研究和应用. 2003 年, Teng 等提出了一种称为 FTP-DS 算法[10], 它是在滑动窗口中使用统计回归技术来挖掘频繁项集. 2004 年, Chi 等给出了 Moment 算法[11], 它也是基于滑动窗口技术的, 但是 Moment 仅关注在数据流中挖掘频繁闭项集, 因此它可以被期望来减少内存数据结构的规模和获得较高的挖掘效率.

在衰减窗口处理模型中, 每个事务都对应一个权值, 而且这种权值随时间的增加而减少. 因此, 它能在这些权值的控制下考虑历史数据相关信息的保存以及裁减等工作. 这种模型在数据流挖掘上具有可以期望的应用前景. 在衰减窗口处理模型中, 比较有代表性的方法是 2003 年 Chang 等提出的 estDec 算法[12]. 它通过定义一个称为衰减因子的参数, 使得较早到达数据流的事务的影响逐渐减弱. 2003 年, Giannell 等提出了一种传统的 FP-Tree[3] 改造的处理数据流的算法 FP-stream[8], 该方法利用不同时间粒度来实现不同时间段的频繁项集的生成工作.

2.2　频繁闭项集

挖掘频繁闭项集是以广泛讨论的挖掘频繁项集为基础提出和发展起来的. 给定一个被讨论的项目全集 $I=\{i_1, i_2, \cdots, i_m\}$ 和定义在 I 的一个事务数据库 $D=\{(tid_1, t_1), \cdots, (tid_n, t_n)\}$, 其中 $t_k \subseteq I$ ($k=1, 2, \cdots, n$) 被称为一个项目集 (Itemset), 表示数据库的一个事务所包含的所有项目, 并且 tid_k 是这个事务的标识符. 设 $I_1 \subseteq I$, 项目序列 I_1 在 D 上的支持度是指 D 中包含 I_1 的事务在 D 中所占的百分比, 即 $\text{support}(I_1)=\|\{t \in D \mid I_1 \subseteq t\}\| / \|D\|$. 给定一个最小支持度 $minsup$, $I_1 \subseteq I$ 被称为是一个频繁项集当且仅当 I_1 的支持度 $\text{Support}(I_1) \geqslant minsup$. 因此, 在 D 上挖掘频繁项集的任务就是寻找 I 的所有支持度不小于 $minsup$ 的所有频繁子集.

如前所述, 由于挖掘一般性的频繁项集存在相当多的重复信息 (例如, 如果挖掘出的频繁集是 {ABC, AB, A}, 那么根据 Apriori 属性[1], AB 和 A 是冗余的). 因此, 自从 1999 年, Pasquietti 提出挖掘频繁闭项集的思想后, 许多文献讨论了这个问题[4, 6, 11, 13], 其中文献 [13] 对闭项集、频繁闭项集以及相应的概念给出了严格地定义, 这里不再赘述. 简单地说, 定义在项目全集 I 和事务数据库 D 上的一个频繁闭项集 FCI 是 D 上的频繁项集 FI 的一个特殊子集: FCI 中的任何项集的支持度都不能小于或等于它的超集的支持度.

例子 **1**　假设 $I=\{A, B, C, D\}$, $D=\{AB, ABC, CD, ABC\}$, $minsup=50\%$ (或者说最小支持数为 2), 则频繁项集 FI = {A3, B3, C3, AB3, AC2, BC2, ABC2} (该表示中后面数字为前面的项集的支持数). 但是, 它的频繁闭项集 FCI = {C3, AB3, ABC2}. 这是因为 A3 和 B3 与它的超集 AB3 具有相同支持度; AC2 和 BC2 与它的超集 ABC2 具有相同支持度; 但是 C3、AB3 和 ABC2 满足闭合的条件.

2.3　Moment 算法及其存在的问题

Moment 算法的核心是在内存中维护一种称为 CET (Closed Enumeration Tree) 的数据结构. CET 的任何节点的儿子节点一定是这个节点的超集, 并且是该节点和其右面兄弟节点的并集. 实际上, Moment 算法的 CET 的节点被划分成四种类型[11], 并根据这四种类型的节点算法进行不同的处理. 为了更严格地表述下面的问题, 定义 1 给出了对应的定义.

定义 **1**　(Moment 的 CET 中的节点类型) 按节点的项目集频度和关联, CET 的节点被划分成 4 类: (1) 不频繁 (Infrequent) 节点: 该类节点是不频繁的, 但是其父节点是频繁的; (2) 没有前途 (Unpromising) 节点: 该类节点是频繁的, 但是有一个与该节点支持度相同的闭项集包含该节点并且它们不构成直接父子关系; (3) 中间 (Intermediate) 节点: 该类节点是频繁的, 并且存在一个与该节点支持度相同的儿子节点, 该节点不是 Unpromising 节点; (4) 闭 (Closed) 节点: 这些节点是频繁闭项集, 它们可以是叶子节点也可以是内部节点.

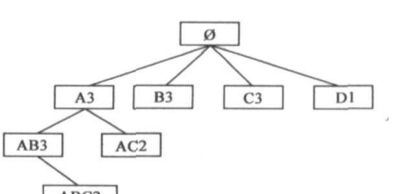


图 1　节点类型示意图

例子 **2**　假如一颗 CET 树如图 1 所示, 并且 $minsup=50\%$, 则 D1 是 Infrequent 节点; AC2 和 B3 是 Unpromising 节点; A3 是 Intermediate; ABC2、AB3 和 C3 是 Closed 节点.

由于事务的到来离去, 滑动窗口的数据会发生变

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

化,所以相应的 CET 节点的性质和支持度等信息可能发生变化.因此,Moment 有两个主要的操作来支持这种变化[11]:(1)当一个新事务 I_{new},加入滑动窗口时,遍历 CET 并且对于它的所有节点进行支持度和节点性质的维护.它提供一个称为 Addition(n_1, I_{new}, D, *minsup*)的操作来实现.(2)当一个旧事务 I_{old},离开滑动窗口时,Moment 调用一个称为 Deletion(n_1, I_{old}, D, *minsup*)的操作来实现相关节点的信息维护.

Moment 算法存在两个主要问题:

(1)Moment 算法采用了滑动窗口机制,只能发现滑动窗口内的局部频集.仅挖掘局部频集对一些应用而言是不够的.尽管通过窗口大小的调整,可以减轻这种影响[11],但是这种调整能力是有限的.因此,这样的方法很难用来实时挖掘数据流的全局变化.

(2)Moment 算法使用的是一种精确的算法模型,要频繁地维护和更新信息,影响其效率.另外,新旧事务在窗口的替换通过两个独立的操作(Addition 和 Deletion)的迭代调用来完成.这样的更新策略很明显的一个缺点就是可能引起数据颠簸问题,即一个旧事务离开窗口或者一个新事务进入窗口都可能引起内存数据结构的大幅度更新.

3 频繁闭项集近似挖掘算法 A-Moment

简短地说,当一个事务在数据流中出现时,A-Moment 算法将按照 4 个步骤来处理它:(1)当前窗口评估阶段;(2)计数更新阶段;(3)CET 维护阶段;(4)频繁闭项集选择阶段.一般地说,前 3 个阶段对每个事务都要进行,但是最后一个阶段是周期性进行或者是在用户需要的时候进行.主函数的伪代码描述在下面的图 2 中.

```
A-Moment(CET, D)
输入: 数据流 D.
输出: 被 D 所更新后的 CET 树; 挖掘出的所有频繁闭项集 FC.
(1)  CET 的根节点←null; m←CET 的根节点;
(2)  产生数据库 D 的公共项目集 f(D), 并为每个项目建立节点
     连接到根节点上;
(3)  FOR each transaction k∈D DO
(4)    Window_Evaluate(D, window_size, k); // 当前窗口评估阶段
(5)    Count_Maintain(CET, k); // 计数更新阶段;
(6)    CET_Maintain(CET, FC, window_size, m, k); // CET 维护阶
       段
(7)    IF(a request is given by users)// 频繁闭项集选择阶段
(8)    Periodically_Maintain(CET, window_size, D, k);
(9)    Pattern_Output(FC);
(10)   ENDIF
(11) ENDFOR
```

图 2 A-Moment 算法描述

3.1 当前窗口评估阶段

这个阶段的主要任务是计算出当前窗口的大小.在衰减窗口处理模型[12]中,所谓的当前窗口是虚拟的.事实上,在不同时间内,一个节点的支持计数的权重是不同的,越老的事务对支持计数的贡献越小,因此需要动态地评估窗口的理论期望值以计算节点的支持度.这样做的意义在于,既保证了老事务的影响逐渐减少(但不是突然消失),又解决了 Moment 算法的过于依赖窗口而发生的数据颠簸问题.当前窗口大小的估算通过下面的定义 2 和 3 给出.

定义 **2**(衰减率) 给定一个衰减基 $b>1$ 和一个衰减基生命 $h\geqslant 1$,则衰减率 $d=b^{-(1/h)}$.

实际上,一个衰减率反映了历史数据相对于当前到达数据应该衰减的比率.

定义 **3**(窗口衰减计算) 给定衰减率 d,假如一个事务到达前的窗口大小为 $|D_{k-1}|$ 的话,则这个事务到达后的窗口大小 $|D_k|$ 可以通过下面的公式来估计:

$$|D_k|=|D_{k-1}|\times d+1$$

定理 给定 $b>1$ 和 $h\geqslant 1$,当事务数 $n\to\infty$,则窗口的大小会收敛于 $1/(1-d)$.

证明 显然,当第一个事务到达时,窗口大小 $|D_1|$ 是 1.依据定义 3,对于 $i=1,2,\cdots,k$,有:

$$|D_2|=|D_1|*d+1=d+1,$$

$$|D_3|=|D_2|*d+1=d^2+d+1,$$

$$\cdots,$$

$$|D_k|=|D_{k-1}|*d+1=d^{k-1}+d^{k-2}+\cdots+d+1$$

$$=(1-d^k)/(1-d).$$

根据定义 2 有:$b^{-1}\leqslant d<1$,所以,当 k 趋于无穷大的时候,$|D_k|$ 最终会收敛到 $1/(1-d)$.

该定理表明,通过调整衰减率 d,图 2 中的 Window_Evaluate(D, *window_size*, k) 函数能保证当前窗口被控制在所期望的范围内.

3.2 计数更新阶段

这个阶段的主要任务就是更新 CET 中已有节点的支持计数.由于在每个事务出现时,都需要重新评估当前窗口的大小,所以理论上说,CET 的每个节点的支持计数都可能发生变化.假如在每个事务到来的时候对所有的节点信息都进行更新,其维护开销将会是很大的.因此,A-Moment 算法采用分布式更新策略,即仅对和当前到达事务相关的节点进行批量衰减,而其他节点等待以后再更新.

假如 CET 中的一个节点 n 被当前到达的事务 k 所包含的话,则此节点的计数需要更新,其的更新方法定义如下:

$$\mathrm{cnt}_k(n)=\mathrm{cnt}_{\mathrm{pre}}(n)\times d^{(k-MRtid_{\mathrm{pre}})}+1$$

其中 $\mathrm{cnt}_k(n)$代表事务 k 处理后的节点 n 的支持数,而 $\mathrm{cnt}_{\mathrm{pre}}(n)$和 $MRtid_{\mathrm{pre}}$代表最近一次节点 n 的支持数和最

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

近一次更新节点 n 的事务编号.图 2 中的函数 Count_Maintain(CET, k) 就是依据上面的公式,在一个事务 k 从数据流中到达时,来完成 CET 相关节点的支持数更新的.

在大多数情况下,A-Moment 算法的分布式更新策略不会导致信息的过多丢失.但是,假如一些节点的更新时间间隔过长进而可能影响系统的挖掘精度的话,那么就应该考虑调用相应的后台进程来进行强制更新,这一工作可以结合后面的频繁闭项集选择阶段进行.

3.3 CET 维护阶段

这个阶段的主要任务是完成 CET 节点的维护,主要包括可能的新节点的增加、旧节点删除以及可能的节点类型的修正.在 Moment 算法中 CET 节点被划分成四类(见定义 1),这样的划分为 CET 信息的维护和裁剪提供了方便.但是,仔细分析这种划分,可以进一步划分成两个大类:Non_Extending 和 Extending 节点.正如它们的名字所暗示的那样,对于 Non_Extending 类节点,我们只需要维护它本身的信息,而不需要在 CET 中去保留它的后代节点.但是对于 Extending 类节点,不仅需要维护它本身的信息,而且需要在 CET 中保留它的后代节点及其相关信息.

定义 4(A-Moment 的 CET 中的节点类型)　A-Moment 算法在原来 Moment 的四节点划分基础上进一步将 CET 中的节点划分成两大类 Non_Extending 和 Extending 节点:(1)Infrequent 和 Unpromising 总称为 Non_Extending 节点;(2)Intermediate 和 Closed 总称为 Extending 节点.

总的来说,维护过程是从根节点开始递归进行的,它需要检验节点的性质是否发生变化.如果发生了变化,就需要进行一系列操作来更新节点的类型.A-Moment 的节点增加和删除是依据上面定义 1 和 4 所定义的四小类和两大类节点类型来完成的.下面的性质保证了节点维护的高效性和有效性.

性质 1　如果 i 是 Non_Extending 类节点,则:(1)i 不是 Extending 类节点;(2)i 的所有儿子节点也不可能是 Closed 节点.

证明　(1)根据定义 4,如果 i 是 Non_Extending 类节点,显然 i 不是 Extending 类节点.

(2)根据 Non_Extending 类节点的定义,如果 i 是 Non_Extending 类节点,那么根据定义 4,i 是 Infrequent 或者是 Unpromising 节点.假如 i 是 Infrequent 节点,则 i 不频繁,按照 Apriori 属性[1],它的任何超集不可能是频繁的,因此 i 的所有后代节点也不可能是定义 1 定义的 Closed 节点.假如 i 是 Unpromising 节点,则依据定义 1,i 虽然可能有超集是 Closed 节点,但是它们不能构成直接父子关系,所以它的所有儿子节点都不是 Closed 节点.

性质 2　如果 i 是 Intermediate 节点,则:(1)i 不是 Closed 节点;(2)i 的儿子节点中可能存在 Closed 节点.

证明　根据定义 1,很容易得出这个性质.

性质 1 和 2 告诉我们,可以在保证一定的精度下,在 CET 中动态地增加 Extending 类节点和删除 Non_Extending 类节点,从而获得较好的存储和执行效率.

由于 A-Moment 算法采用了窗口衰减的方法,在新事务到达节点的支持度有可能增加也可能减少,所以在算法中使用了一个称为 CET_Maintain 的方法对 CET 进行维护(不像 Moment 算法那样使用独立的 Addition 和 Deletion 方法).图 3 给出了 CET_Maintain 函数的伪代码描述.

```
CET_Maintain(CET, FC, window_size, m, k)
输入: k 到达后的窗口大小评估值 window_size; 当前到达事务 k; 开始更新的节点 m.
输出: 被 k 所更新后的 CET 树; 被 k 所更新后的所有频繁闭项集 FC.
(1)  IF(m 是不被 k 包含的非根节点) return;
(2)  FOR each m's child p DO
(3)    IF(p 保持为 Non_Extending 节点)p 加入 F1;
(4)    IF(p 从 Non_Extending 节点变成 Extending 节点)p 加入 F2;
(5)    IF (p 保持为 Extending 节点) p 加入 F3;
(6)    IF (p 从 Extending 节点变为 Non_Extending) p 加入 F4;
(7)  ENDFOR
(8)  FOR each m's child p DO
(9)    IF (p∈F1) 将 p 标记为正确的节点类型;
(10)   IF (p∈F2 or p∈F3)
(11)     Children_Generate (CET, p); //扩展 p 所有频繁儿子节点
(12)     CET_Maintain(CET, FC, window_size, p, k); //递归调用 CET_Maintain 函数
(13)     将 p 标记为正确的节点类型;
(14)     IF (p is a Closed node) FC= FC∪{p}; ENDIF
(15)     ENDIF
(16)   IF (p∈F4)
(17)     剪掉 p 的所有儿子节点; //对 Non_Extending 节点不再保留他们的儿子
(18)     将 p 标记为正确的节点类型;
(19)     IF(p is a Closed node) FC= FC∪{p}; ENDIF
(20)     ENDIF
(21)   ENDFOR
```

图 3　CET_Maintain 函数描述

CET_Maintain 函数里调用了函数 Children_Generate (CET, p),该函数用来为一个 CET 中的节点 p 扩充儿子节点.按照 A-Moment 的基本思想,CET 中仅保留 Extending 节点的儿子节点信息,因此 Children_Generate(CET, p)将对 p 的所有频繁超集进行测试和在 CET 中进行必要的扩展:把 p 与其每个频繁右兄弟节点的并集作为 p

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

的新的儿子节点加入到 CET 中或者更新 p 的已有儿子节点的信息.

3.4 频繁闭项集选择阶段

本阶段的主要任务是对当前已经生成的频繁闭项集进行输出,图 2 中的调用的 Pattern_Output(FC)函数正是为了完成这个任务.这个阶段是根据用户要求周期性进行的.像前面已经声明的那样,所有生成的频繁闭项集被实时地放入到数据结构 FC 中.为了提高检索效率,FC 采用按项目集的 hash 值方法进行组织.

另外,为了解决分布式更新 CET 节点的时间间隔过长而导致一些节点的信息陈旧的问题,可以考虑在此阶段调用一个后台进程 Periodically_Maintain(CET, $window\_size$, D, k)来对 CET 中的所有节点的支持数等信息进行强制更新:从根节点开始,按深度优先遍历 CET,对每个节点 n 的支持数按照公式 $cnt_k(n)=cnt_{pre}(n)\times d^{(k-MRid_{pre})}$ 进行估计,然后根据它的节点类型进行节点维护.基本方法和前面介绍的 Count_Maintain 和 CET_Maintain 函数类似,这里不再赘述.

4 实验研究

为了评估算法的性能,我们比较了 Moment 和 A-Moment 算法处理每个事务的平均时间,挖掘出的频繁闭项集数目,以及算法对于不同大小数据集的伸缩性.本节的所有实验环境是:2.8GHz CPU 和 1G RAM 的 PC 机;操作系统为 WindowsXP;开发环境为 Visual studio C++6.0.

4.1 数据集

本节实验所使用的数据集是由 IBM 的合成数据产生器[11]生成的.我们在生成数据集时使用了产生器里的 4 个常用参数:事务的最大长度 T;事务的平均长度 I;平均最大模式的长度 P;数据集中的事务数量 D.因而,T20.I5.P4.D20k 表示事务最大长度为 20,事务平均长度为 5,平均最大模式长度为 4,事务数量为 20k 的数据集.

4.2 实验结果和分析

实验 **1**(A-Moment 与 Moment 算法处理一个事务的平均时间比较).在实验中,Moment 使用的窗口大小为 10K,A-Moment 的衰减率设为 0.9999(虚拟窗口的大小大约收敛在 10K),使用的数据集是 T20.I5.P4.D20k.图 4 给出了相应的实验结果.

从图 4 中可以看出,对所有的支持度而言,A-Moment 算法在单个事务的平均处理时间上都比 Moment 要快(A-Moment 大约是 Moment 的一半).因此,A-Moment 算法比 Moment 能更及时地捕捉到数据流的模式变化.导致这一结果的主要原因是,A-Moment 算法采用了不精确估计支持度的方法和分布式更新 CET 的技术,因而保证 CET 节点的计数值的快速计算和减少了节点的频繁增删,达到了提高效率的目的.

实验 **2**(A-Moment 与 Moment 算法在不同支持度下挖掘出的频繁闭项集的数目比较).本实验使用的数据集和实验参数与实验 1 相同.表 1 给出了相应的实验结果.

表 **1** 频繁闭项集数目比较

支持度 / 算法	1%	2%	3%	4%	5%	6%	7%	8%	9%
A-Moment	2079	893	520	353	258	185	147	123	104
Moment	2086	887	518	353	255	186	145	123	103

从表 1 可以看出,在相同支持度下,A-Moment 和 Moment 产生的频繁闭项集数量基本相当.这一结果从某种意义上说明,A-Moment 的近似估计机制具有很高的精确度.因此,A-Moment 的近似计数估计方法和分布式更新策略对数据流挖掘是可行的.

实验 **3**(A-Moment 与 Moment 算法在不同数据集容量上的效率比较).在实验中,Moment 的窗口大小为 10K,A-Moment 算法的衰减率为 0.9999,两个算法的最小支持度都是 10%,实验使用的数据集是 T20.I5.P4.D30k~T20.I5.P4.D90k.图 5 给出了 Moment 算法和 A-Moment 算法在不同大小的数据集上伸缩性的比较.

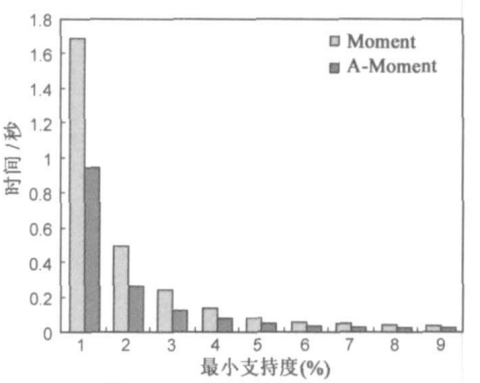


图 4 处理平均时间比较

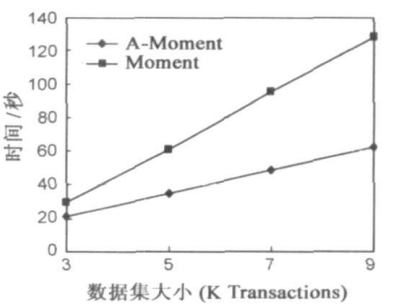


图 5 不同数据集的处理时间比较

如图 5 所示,相对于 Moment 算法,A-Moment 算法在所有数据集上都有更快的挖掘速度,同时,随着数据集容量的攀升,A-Moment 算法比 Moment 具有更小的增长

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

速度. 因此, A-Moment 对大型数据流的挖掘具有更好的适应性.

5 结论

数据流中挖掘频繁项集是当前数据挖掘领域的一个研究热点, 而传统的精确挖掘频繁项集的算法不能很好地适应数据流的高速、无限、不可预测的特点. 本文提出了一种在数据流中近似挖掘频繁闭项集的算法 A-Moment. 该算法通过衰减窗口机制、近似计数估计方法和分布式更新信息策略来解决数据流挖掘中内存和时间瓶颈问题. 总的来说, A-Moment 算法的衰减窗口机制能有效地避免滑动窗口所带来的局部模式挖掘和数据颠簸问题; 分布式更新策略和计数估计方法提高了算法的时间效率. 实验表明, A-Moment 算法在保证相当高的精度的前提下, 比 Moment 算法能更快地形成频度模式.

参考文献:

[1] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases[A]. Proceedings of the 20th International Conference on Very Large Data Bases[C]. San Francisco: Morgan Kaufmann, 1994. 487—499.

[2] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[A]. 2000 ACM SIGMOD International Conference on Management of Data[C]. Dallas: ACM Press, 2000. 1—12.

[3] Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering frequent closed itemsets for association rules[A]. In Proceeding of the 7th International Conference on Database Theory[C]. Jerusalem, Israel: Springer, 1999. 398—416.

[4] Pei J, Han J, Mao R. CLOSET: an efficient algorithm for mining frequent closed itemsets[A]. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery[C]. Dallas: ACM Press, 2000. 21—30.

[5] Burdick D, Calimlim M, Gehrke J. MAFIA: a maximal frequent itemset algorithm for transactional databases[A]. Proceedings of the 17th International Conference on Data Engineering[C]. Heidelberg: IEEE Computer Society Press, 2001. 443—452.

[6] Zaki M, Hsiao C. Charm: an efficient algorithm for closed association rule mining[R]. New York: RPI, 1999.

[7] Zhu Y, Shasha D. StatStream: statistical monitoring of thousands of data streams in real time[A]. Proceedings of the 20th International Conference on Very Large Data Bases[C]. Hong Kong, China: Morgan Kaufmann, 2002. 358—369.

[8] Giannella C, Han J, Robertson E, Liu C. Mining frequent itemsets over arbitrary time intervals in data streams[R]. Bloomington: Indiana University, 2003.

[9] Manku G, Motwani R. Approximate frequency counts over data streams[A]. Proceedings of the 28th International Conference on Very Large Data Bases[C]. Hong Kong, China: Morgan Kaufmann, 2002. 346—357.

[10] Teng W-G, Chen M-S, Yu P S. A regression-based temporal pattern mining scheme for data streams[A]. Proceedings of the 29th International Conference on Very Large Data Bases[C]. Berlin, Germany: Morgan Kaufmann, 2003. 607—617.

[11] Chi Y, Wang H, Yu P, Muntz R. MOMENT: maintaining closed frequent itemsets over a stream sliding window[A]. Proceedings of the 2004 IEEE International Conference on Data Mining[C]. Brighton, UK: IEEE Computer Society Press, 2004. 59—66.

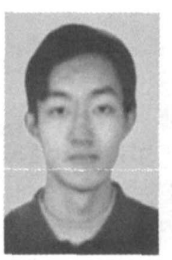
[12] Chang J, Lee W. Finding recent frequent itemsets adaptively over online data streams[A]. Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining[C]. Washington, USA: ACM Press, 2003, 487—492.

[13] 刘君强, 孙晓莹, 庄越挺, 潘云鹤. 挖掘闭合模式的高性能算法[J]. 软件学报, 2004, 15(1): 94—102.
Liu J Q, Sun X Y, Zhuang Y T, Pan Y H. Mining frequent closed patterns by adaptive pruning[J]. Journal of Software, 2004, 15(1): 94—102. (in Chinese)

作者简介:

刘　旭　男, 1982 年生于北京, 北京工业大学计算机学院硕士, 主要研究领域为数据挖掘与知识发现. E-mail: liuxu1210@emails.bjut.edu.cn

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



孙　岳　男, 1982 年生于北京, 北京工业大学计算机学院硕士. 主要研究领域为数据挖掘与知识发现.

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.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net