时间敏感数据流上的频繁项集挖掘算法

李海峰 章 宁 朱建明 曹怀虎

(中央财经大学信息学院 北京 100081)

摘 要 数据流中的数据分布随着时间动态变化,但传统基于事务的滑动窗口模型难以体现该特征,因此挖掘结果并不精确.首先提出时间敏感数据流处理中存在的问题,然后建立基于时间戳的滑动窗口模型,并转换为基于事务的可变滑动窗口进行处理,提出了频繁项集的挖掘算法FIMoTS.该算法引入了类型变化界限的概念,将项集进行动态分类,根据滑动窗口大小的变化对项集进行延迟处理,仅当项集的类型变化界限超出一定阈值的时候才进行支持度的重新计算,能够达到剪枝的目的.在4种不同密度的数据集上完成的实验结果显示,该算法能够在保证内存开销基本不变的情况下显著提高计算效率.

关键词 频繁项集;数据流;时间敏感;滑动窗口;数据挖掘

中图法分类号 TP311 **DOI号**: 10.3724/SP.J.1016.2012.02283

Frequent Itemset Mining over Time-Sensitive Streams

LI Hai-Feng ZHANG Ning ZHU Jian-Ming CAO Huai-Hu

(*School of Information, Central University of Finance and Economics, Beijing* 100081)

Abstract Stream data arrives dynamically when stream continues, which cannot be reflected by the traditional transaction-based sliding window, thus the results are not accurate. This paper focuses on this problem and builds a timestamp-based sliding window model, which is afterwards converted into a transaction-based variable sliding window; based on this model, a frequent itemset mining algorithm named FIMoTS is proposed. In this algorithm, we introduce the type transforming bound to dynamically classify the itemsets into categories; as a result, these itemsets can be deferred processed with regard to the window size, that is, an itemset will not be processed unless its type transforming bounds reach to a threshold. Consequently, the computational pruning can be conducted. The experimental results over four different datasets show that our algorithm significantly outperform the Naïve method.

Keywords frequent itemsets; data stream; time-sensitive; sliding window; data mining

1 引 言

频繁项集是Agrawal在1994年提出来的[1],其动机是为了寻找超市事务数据的频繁集,用以分析客户的购买行为.近年来随着在众多行业中广泛的应用,频繁项集挖掘的研究受到了广泛关注,逐渐发展成为数据挖掘中的重要分支,其目标是在数据集中寻找用户感兴趣的模式,例如关联规则[2]、序列模式[3]、数据相互关系[4]等.

网络及大规模商业集成等应用的普及产生了大量的流数据.数据流是动态数据的一种典型代表,具

收稿日期:2012-06-05;最终修改稿收到日期:2012-08-24.本课题得到国家自然科学基金(61100112)、教育部人文社会科学研究青年基金(11YJCZH006)、北京市自然科学基金(9092014,4112053)、中央财经大学科研创新团队支持计划资助.李海峰,男,1979年生,博士,讲师,主要研究方向为数据管理和数据挖掘.E-mail:mydlhf@126.com.章 宁,女,1975年生,博士,教授,主要研究领域为数据外包和信息外包.E-mail:zhangning75@gmail.com.朱建明,男,1965年生,博士,教授,主要研究领域为无线网络和网络安全.曹怀虎,男,1977年生,博士,副教授,主要研究方向为网络体系结构和移动互联网技术.

有高速、无限、连续且动态变化的特点，在数据流的环境下，频繁项集的挖掘算法必须用有限的内存空间去保存和处理数据，而且需要实现对数据的增量分析处理. 这些挑战成为研究数据流管理和挖掘的人员关注的焦点.

通常来说，用户关注的是数据流最近到达数据的分析结果，因此研究人员提出了衰减模型[5-6]和滑动窗口模型[7-19]以降低存储和计算代价.

文献[5]为了反映数据流的变化趋势，采用衰减因子来降低历史数据的影响. 算法 estDec 采用格来保存显著项集，并通过其子集进行支持度的估计，其过程分为 4 个阶段：参数更新、计数更新、延迟插入和频繁项集选择. 采用衰减因子虽然可以有效反映数据流变化趋势，但无法记忆过去某段时间的内容，因此无法满足用户对历史项集支持度的查询，文献[6]则维护了比前缀树压缩率更高的基于后缀的频繁项集森林(IsFI-forest)，提出的算法 DSM-FI 采用了宽松的最小支持度作为阈值来得到潜在的频繁项集.

文献[7]结合了滑动窗口的优点，在算法 FP-stream 中采用倾斜窗口来保存每个频繁项集的支持度，为了节省空间，倾斜窗口将时间段进行不同粒度的划分，越靠近当前时间点的时间段粒度越小，这样就可以在满足用户时间相关查询的情况下降低空间代价. 文献[8]采用了以事务作为基本单位的滑动窗口作为挖掘模型，提出的算法 estWin 将潜在频繁项集定义为显著项集，而忽略非显著项集的处理，通过数据流滑动窗口的长度来估计项集是否显著，确定监控项集的内容. 文献[9]提出的算法 StreamMining 采用了两次扫描来得到结果，第一次扫描获取候选集，对只包含两个项目的项集进行挖掘，而对其它长度的项集则根据 apriori 性质进行挖掘，其精确度由给定的参数来确定. 为了有效压缩数据并且便于项集操作，StreamMining 采用了名为 treehash 的数据结构，通过项目的统计减少子集的检测，并通过事务的在线监测来减少子集的插入. 文献[10]为了处理数据流的大型滑动窗口，提出了算法 SWIM. SWIM 将窗口分为多个块，每次更新一块数据来提高效率，并且根据 FP-tree 条件树的概念，用模式树 PT 直接保存频繁模式. 据此还提出了两种算法 DTV 和 DFV 以及二者的混合算法，通过条件计数来完成频繁模式的验证，可以得到精确的计算结果. 文献[11]完成了同样问题的改进算法. 文献[13]为了检测项目出现的最大频率，以减少用户对参数的设置，重新定义了频繁项目的频繁度，并提出了一种高效的针对新型频繁项目挖掘的方法. 文献[12]扩展了这种定义方法，挖掘在不同滑动窗口中具有最大频率的频繁项集. 文献[14]则利用项目的信息来提高挖掘频繁项集的效率.

尽管以上方法能够高效地完成对滑动窗口中的数据挖掘，但均是基于事务作为基本单元的挖掘方法，也就是说，滑动窗口的大小根据事务数量来决定，这种方法存在一定的弊端. 文献[15]假定每次进入滑动窗口的事务数量不等，首次提出了时间相关的数据流模型[16]，并给出了基于块的频繁项集的挖掘算法，为了从历史数据流中获取项集支持度，该算法采用衰减计数表来存储和估计数据信息，并通过数据合并以减少内存使用. 其主要问题在于，尽管使用了错误报警机制，该算法挖掘的结果仍然存在一定的误差. 文献[17]为了降低删除滑动窗口中的事务对挖掘结果的影响，提出了两种算法：第 1 种算法 ATS 是基于平均时间戳的，根据平均时间值与当前时间值的比较，来判断项集是否符合一致性分布，决定是否剪枝该项集. 第 2 种算法 FCP 是基于支持度的变更点的，通过比较项集发生的当前时间与最后一次发生的时间来决定当前时间是否为支持度的变更点，以改变项集第一次出现的时间，从而达到缩小滑动窗口的目的. 文献[18]扩展了 FCP 算法来解决文献[15]中提出的问题，即挖掘以时间戳作为基本单位的滑动窗口，用 PS-tree 来保存频繁项集，用局部贪婪算法来处理每个时间单元到达大量数据的情况，但该算法同样没有解决结果存在误差的问题. 文献[19]也提出了一种时间敏感的数据流挖掘算法，但该算法并没有针对该模型的特点进行优化，因此算法的效率不高.

本文针对以上问题，结合时间敏感数据流的特性，提出了一种高效而且精确的频繁项集挖掘算法. 本文的主要贡献如下：

(1) 提出了基于时间区间的滑动窗口模型，该模型与传统滑动窗口模型中以事务数量作为衡量滑动窗口大小的方式不同，更能够体现现实中数据流的时间特性.

(2) 将提出的滑动窗口模型转换为传统的基于事务的可变滑动窗口模型，并根据该模型定义了基于相对支持度的频繁项集的挖掘问题.

(3) 引入了类型变化界限的定义，将项集进行动态分类，通过延迟处理来裁剪冗余数据计算，进而提出了高效的频繁项集挖掘方法，该方法能够在保

证节省空间代价的同时，提高挖掘的效率.

(4) 通过在两种真实数据集和两种模拟数据集上完成了算法的实验验证，并与该问题的 Naïve 算法进行了对比，实验表明了本文算法的可行性和高效性.

本文第 2 节介绍频繁项集的相关知识；第 3 节提出时间敏感的滑动窗口模型以及转化后的可变滑动窗口模型，并据此提出本文需要解决的问题；第 4 节提出基于类型变化界限的频繁项集挖掘算法 FIMoTS；第 5 节通过实验进行算法验证；第 6 节是本文的总结.

2　预备知识

频繁项集是数据集中发生次数不小于用户定义阈值的项集. 用 $\Gamma=\{i_1,i_2,\cdots,i_n\}$ 来表示所有不同项目的集合，其中 $|\Gamma|=n$ 表示 Γ 的长度，称长度为 k 的 Γ 的子项集 X 为 k-项集. 为了简单起见，可以把项集 $X=\{x_1,x_2,\cdots,x_m\}$ 表示为 $x_1x_2\cdots x_m$. 给定数据集 $D=\{T_1,T_2,\cdots,T_v\}$，每一个 $T_i(i=1,\cdots,v)$ 表示一个基于 Γ 的事务，包含事务 id 和对应的项集 X. 对于事务 $T=(tid,X)$ 来说，如果存在项集 Y 使得 $Y\subseteq X$，则称事务 T 支持项集 Y，所有 D 中支持项集 Y 的事务集合称为 Y 的覆盖，表示为 $cover(Y,D)=\{T|T\subseteq D\wedge Y\subseteq X\wedge X\in T\}$. Y 的绝对支持度是覆盖其事务的集合大小，表示为 $\Lambda(X,D)=|cover(X,D)|$，其相对支持度是 Y 在 D 中发生的概率，即 Y 的绝对支持度与数据集 D 大小的比值，表示为 $\Lambda_r(Y,D)=\frac{\Lambda(Y,D)}{|D|}$. 给定相对最小支持度阈值 λ_r，如果 $\Lambda_r(Y,D)\geq\lambda_r$，则称项集 Y 为频繁项集.

3　基于时间戳的滑动窗口模型

3.1　研究动机

数据流具有动态性，其数据受到实际应用和网络环境的影响，到达数量的分布是不均匀的，也就是说，数据到达具有时间相关性，而过期数据往往不受关注. 例如，在超市中，由于客流量在不同时间段的不同，交易在客流高峰期会集中出现，而在其它时段出现的较为稀疏. 传统的滑动窗口模型通常采用事务数量来决定窗口大小，这存在一定的缺陷. 图 1 描述了这一数据流到达情况，并展示了采用基于事务的处理模型和基于时间戳的处理模型时滑动窗口中数据的变化情况，同时，将基于时间戳的模型转换为基于事务的处理模型，并与前者进行了对比. 如图 1 (a)所示，假如每个时间段 Ti 是超市中的半天时间，

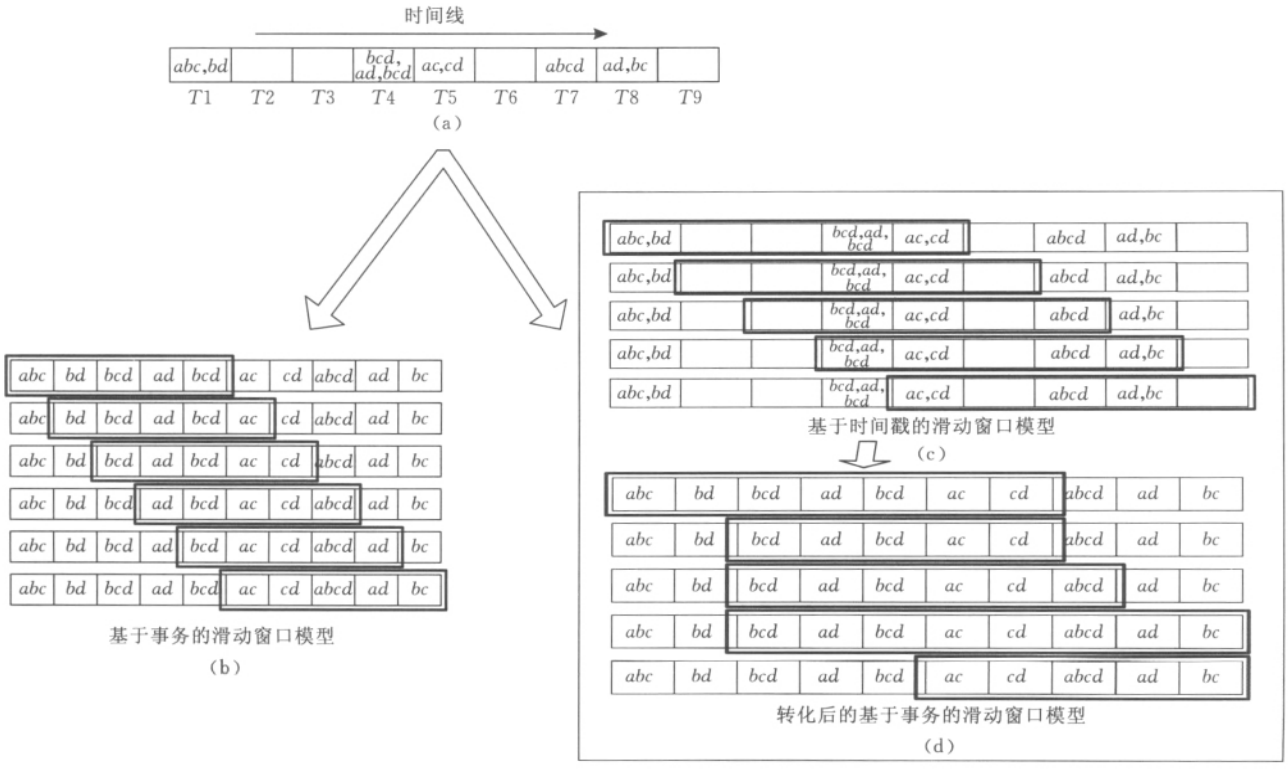

图 1　基于事务的滑动窗口与基于时间戳的滑动窗口

而管理员在 $T6$ 的结束时刻希望看到的是当天销售情况的分析，即 $T5$ 到 $T6$ 期间的销售分析，那么待分析的数据应该为$\{ac,cd\}$，而如果用基于事务数量的滑动窗口模型(图 1(b)，假定滑动窗口大小为 5)，实际分析的数据为$\{bcd,ad,bcd,ac,cd\}$，也就是说，实际分析了两天的数据，这样的结果与管理员希望得到的结果是有误差的. 因此，需要基于时间戳来决定滑动窗口的大小，精确的定位用户需求，图 1(c)显示了基于时间戳的滑动窗口模型，该模型能够体现用户对近期数据分析的需求.

图 1(c)中的模型可以转化为基于事务的可变滑动窗口模型，如图 1(d)所示. 在该图中，滑动窗口的大小会随着时间的变化而变化. 在这样的滑动窗口中，以绝对最小支持度作为阈值的方法不再适用，这是因为在动态变化的滑动窗口中，不同时间段的事务数量分布不均匀，从而导致挖掘的结果不准确. 例如，在图 1(d)中，假定一个项集在滑动窗口中的一半以上事务中出现则认为该项集为频繁项集，那么在$[T1,T5]$范围内，共有 7 个事务，最小支持度为 4，其频繁项集为$\{b:4,c:5,d:5\}$；但是在$[T2,T6]$范围内，如果最小支持度依然保持为 4，其频繁项集变为$\{c:4,\ d:4\}$. 但该范围内只有 5 个事务，其最小支持度应变为 3，也就是说，$\{cd:3\}$应该也是$[T2,T6]$范围内的频繁项集，但该项集由于使用固定的绝对最小支持度阈值而被忽略了，显然这是不合理的.

3.2 问题提出

根据上文所述，本文需要解决的问题是采用相对最小支持度来挖掘以时间戳为基本单位的滑动窗口模型上的频繁项集. 如图 1(c)所示，当新时间段的事务到达时，滑动窗口中最早时间段的事务将被删除. 尽管也有一些研究以时间作为单位进行研究，但通常是通过批处理的方式[20]来完成挖掘，这种方式无法实时获取数据挖掘的结果，因此不属于时间敏感性挖掘方法. 从已有的文献来看，近两年针对该问题的研究没有形成较好的解决思路，其方法有待于进一步改进.

3.3 Naïve 算法

该问题的简单解决方法是：扫描初始滑动窗口，获取所有的频繁项集和部分非频繁项集；随着滑动窗口的移动，每次事务插入和删除，均全部扫描已得到的项集并计算其支持度，根据支持度的类型变化剪枝或生成新的项集. 这也是文献[19]的解决思路. 该解决方法的缺点在于，每次的滑动窗口均需要大量的数据扫描、比较和计算，其计算开销较大.

4 FIMoTS 算法

在数据流中，当给定了阈值后，用户通常只关心频繁项集是什么，而很少关心频繁项集的支持度具体是多少，最大频繁项集的挖掘就是源于这种假设. 基于这个观察，本文将项集按照其支持度与最小支持度阈值的差值进行了分类，采用枚举树作为数据结构保存数据大纲，并提出了 FIMoTS 算法来解决第 3 节中提出的问题，该算法分别对事务到达和离开滑动窗口的情况进行了处理，利用项集的时间属性进行计算剪枝.

4.1 数据结构

在动态变化的滑动窗口模型中，项集具有以下的性质.

性质 1. 给定项集 I，如果 I 是频繁项集，对于新到达事务 T 来说，若 $I\subseteq T$，I 仍然是频繁项集，否则 I 可能变为非频繁项集.

性质 2. 如果 I 是非频繁项集，对于新到达事务 T 来说，若 $I\not\subset T$，I 仍然是非频繁项集，否则 I 可能变为频繁项集.

根据以上性质可知，随着滑动窗口大小的变化，绝对最小支持度阈值不断调整，使得没有支持度变化的项集也会发生类型更改，也就是说，一旦滑动窗口中的内容发生变化，就必须检测所有的项集，这样会产生大量不必要的计算开销. 为了节省计算开销，需要对项集按照待处理时间进行划分. 定义了项集的类型变化界限.

定义 1. 类型变化上界. 对于项集 I 来说，如果新到达 $ub-1$ 个事务一定不能使得 I 发生类型变化(从频繁变为非频繁，或从非频繁变为频繁)，而新到达 ub 个事务可能使得 I 发生类型变化，则称 ub 为 I 的类型变化上界，表示为 $ub(I)$.

定义 2. 类型变化下界. 对于项集 I 来说，如果删除 $lb-1$ 个事务一定不能使得 I 发生类型变化，而删除 lb 个事务可能使得 I 发生类型变化，则称 lb 为 I 的类型变化下界，表示为 $lb(I)$.

定理 1. 对于当前滑动窗口 $D(|D|=z)$来说，给定项集 I，采用分数的形式表示最小支持度和 I 的支持度，其相对支持度为 $\Lambda_r(I)=\frac{a}{b}(b\geq a)$，相对最小支持度为 $\lambda_r=\frac{c}{d}(d\geq c)$. 如果 I 为频繁项集，那么当满足 $\frac{d}{b}\times\frac{b-a}{d-c}z\leq|D|\leq\frac{ad}{bc}z$ 时，I 一定保持其

类型不变；如果 I 为非频繁项集，那么当满足 $\frac{ad}{bc}z<|D|<\frac{d}{b}\times\frac{b-a}{d-c}z$ 时，I 一定保持其类型不变.

证明.

(1) 当 I 为频繁项集，也就是 $\frac{a}{b}\geq\frac{c}{d}$，分两方面讨论：当滑动窗口中增加了 n 个事务时，在极端情况下，n 个事务都不包含 I，则 I 的支持度变为 $\frac{az/b}{z+n}$，若 I 仍然是频繁项集，则有 $\frac{az/b}{z+n}\geq\frac{c}{d}$，即 $|D|=z+n\leq\frac{ad}{bc}z$；当滑动窗口中删除了 m 个事务时，在极端情况下，m 个事务都包含 I，则 I 的支持度变为 $\frac{az/b-m}{z-m}$，若 I 仍然是频繁项集，则有 $\frac{az/b-m}{z-m}\geq\frac{c}{d}$，即 $m\leq\frac{ad-bc}{bd-bc}z$，也就是说，$|D|=z-m\geq\frac{d}{b}\times\frac{b-a}{d-c}z$.

(2) 当 I 为非频繁项集，也就是 $\frac{a}{b}<\frac{c}{d}$，分两方面讨论：当滑动窗口中增加了 n 个事务时，在极端情况下，n 个事务都包含 I，则 I 的支持度变为 $\frac{az/b+n}{z+n}$，若 I 仍然是非频繁项集，则有 $\frac{az/b+n}{z+n}<\frac{c}{d}$，即 $n<\frac{cb-ad}{bd-bc}z$，也就是说，$|D|=z+n<\frac{d}{b}\times\frac{b-a}{d-c}z$；当滑动窗口中删除了 m 个事务时，在极端情况下，m 个事务都不包含 I，则 I 的支持度变为 $\frac{az/b}{z-m}$，若 I 仍然是非频繁项集，则有 $\frac{az/b}{z-m}<\frac{c}{d}$，即 $|D|=z-m>\frac{ad}{bc}z$.

证毕.

引理 1.　若 I 为频繁项集，其类型变化上界为 $ub(I)=\left\lceil\frac{ad}{bc}z-z+1\right\rceil$，类型变化下界为 $lb(I)=\left\lceil z-\frac{d}{b}\times\frac{b-a}{d-c}z+1\right\rceil$；若 I 为非频繁项集，其类型变化上界为 $ub(I)=\left\lceil\frac{d}{b}\times\frac{b-a}{d-c}z-z+1\right\rceil$，类型变化下界为 $lb(I)=\left\lceil z-\frac{ad}{bc}z+1\right\rceil$.

证明.　从定理 1 可以看出，根据 I 的不同类型，其类型在一定滑动窗口范围内是不发生变化的，因此超出了这个范围的临界值就是 I 的类型变化界限.　　证毕.

性质 3.　对于频繁项集来说，其类型变化界限随着支持度的增大而增大.

性质 4.　对于非频繁项集来说，其类型变化界限随着支持度的增大而减小.

为了实现快速的项集匹配和增量处理，在内存中采用枚举树结构保存项集信息. 在枚举树中，父结点项集是子结点项集的子集；根结点和中间结点均为频繁项集，而叶结点代表非频繁项集或没有兄弟结点的频繁项集. 采用三元组 $\langle it,sup,ts\rangle$ 来表示每个结点，其中，it 代表一个项集，并具有唯一的词典顺序(用 $\prec$ 表示，例如，$a\prec b\prec c$ 表示 a 的词典序小于 b 的词典序，且 b 的词典序小于 c 的词典序，那么，由 a、b 和 c 组成的项集可以按照唯一的顺序进行排列，即 $a\prec b\prec c\prec ab\prec ac\prec bc\prec abc$，同一层中的结点按照词典顺序的升序排列)；$sup$ 表示项集的相对支持度；ts 表示项集最近更新的时间戳，记录时间戳的目的是为了防止在树的遍历过程中重复计算. 另外，在枚举树中，用 ib 数组保存非频繁项集的类型变化界限值，用 fb 数组保存频繁项集的类型变化界限值；两个数组中的每个元素分别由二元组 $\langle iub,ilb\rangle$ 和 $\langle fub,flb\rangle$ 组成，分别保存类型变化上界值、类型变化下界值，每个元组通过指针指向对应结点，该指针称为界限指针. 假定最小支持度为 1/3，图 2 显示了图 1(c)中初始滑动窗口所对应的枚举树.

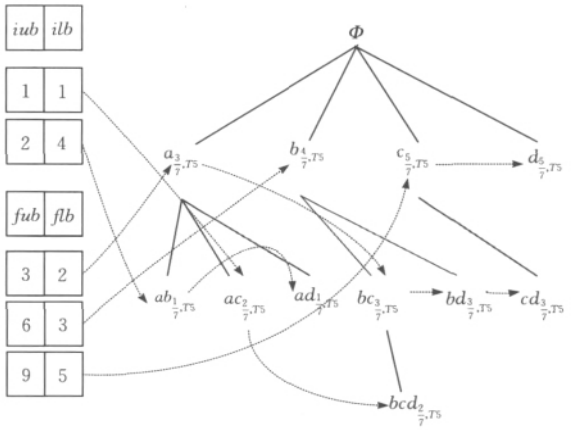

图 2　初始滑动窗口对应的枚举树

定理 2.　给定相对最小支持度，每个相对支持度对应唯一的类型变化界限.

证明.　令相对最小支持度为 $\lambda_r=\frac{c}{d}$，任给项集 I 的相对支持度 $\Lambda_r(I)=\frac{a}{b}$，不失一般性，假设 $\frac{a}{b}\geq\frac{c}{d}$，如果存在另外一个项集 J 的相对支持度 $\Lambda_r(J)=\frac{a'}{b'}\neq\frac{a}{b}$，使得 $ub(I)=ub(J)$，即 $\left\lfloor\frac{ad}{bc}z\right\rfloor-$

$z+1=\left\lfloor\frac{a'd}{b'c}z\right\rfloor-z+1$，即 $\left|\left(\frac{a}{b}-\frac{a'}{b'}\right)\frac{d}{c}z\right|<1$，也即 $\left|\frac{a}{b}-\frac{a'}{b'}\right|z<\frac{c}{d}<1$，而根据相对支持度的定义，$\frac{a}{b}$ 和 $\frac{a'}{b'}$ 必然大于等于 $\frac{1}{z}$，所以 $\left|\frac{a}{b}-\frac{a'}{b'}\right|z$ 必然大于等于 1，显然是矛盾的；同理，也不可能存在 $lb(I)=lb(J)$. 因此命题得证. 证毕.

为了创建枚举树，首先创建代表空项集的根结点，然后在根结点下创建 $|\Gamma|$ 个不同项目 i 对应的结点 n_i，并计算相应的类型变化界限；最后，针对每个结点进行递归处理，生成其子孙结点，直到生成的结点对应的项集为非频繁项集. 该算法如下所示.

算法 1. 频繁项集挖掘的初始化算法 INITIAL.

输入：滑动窗口 D，项目结点相对最小支持度 λ_r，枚举树结点 n_I

输出：枚举树 ET

方法：

```
1.  if n_I.sup<λ_r then
2.      计算 I 的类型变化界限，生成类型变化界限数组值，将相应的界限指针指向 n_I；
3.  else
4.      for each n_I 的兄弟结点 n_J do
5.          if n_J.sup≥λ_r then
6.              生成结点 n_{I∪J}；
7.              计算其支持度和类型变化界限，生成类型变化界限数组值，将相应的界限指针指向 n_{I∪J}；
8.          end if
9.      end for
10.     for each 新生成结点 n_{I∪J} do
11.         调用 INITIAL(D,λ_r,n_{I∪J})；
12.     end for
13. end if
```

数据流进入滑动窗口的最初阶段，可以看作是对静态数据的挖掘. 算法 1 是一个深度优先算法，描述了初始阶段的频繁项集挖掘过程，并生成枚举树.

4.2 事务删除处理

当一组事务 Φ 从滑动窗口中删除的时候，假设被删除事务的数量为 n，根据类型变化下界和支持度的不同，分别进行处理：

所有类型变化下界小于等于 n 的项集 I 均有可能发生类型变化，需要通过重新计算来确定类型变化界限. 若 I 为频繁项集，那么只有当 I 被该组中的每个事务均包含时，I 的类型才可能发生变化；若 I 是非频繁项集，那么只有当 I 不被该组中的所有事务包含时，I 的类型才可能发生变化.

所有类型变化下界大于 n 的项集均不可能发生类型变化. 此时，一方面需要更新类型变化下界；另一方面，为了保证在删除事务后所有类型变化上界正确，对于每个未处理的项集 I 来说，若 I 是频繁的，则假设每个被删除的事务均包含 I；若 I 是非频繁的，则假设每个被删除的事务均不包含 I，用来更新类型变化上界.

定理 3. 令相对最小支持度为 $\lambda_r=\frac{c}{d}(d\geq c)$，假设被删除事务数量是 n，则项集 I 的类型变化下界 $lb(I)$ 变为 $lb(I)-n$. 若 I 为频繁项集，其类型变化上界 $ub(I)$ 变为 $ub(I)-\left\lceil\frac{d-c}{c}n\right\rceil$；若 I 为非频繁项集，其类型变化上界 $ub(I)$ 变为 $ub(I)-\left\lceil\frac{c}{d-c}n\right\rceil$.

证明. 当被删除事务数量为 n 时，对于类型变化下界大于 n 的项集 I（支持度为 $\frac{a}{b}$）不需要处理，但为了保证项集在极限情况下的类型正确，需要假设每个被删除事务均会影响项集的类型变化，若 I 为频繁项集，那么其新支持度变为 $\frac{az/b-n}{z-n}$，假设新类型变化下界为 $lb'(I)$，则有 $\frac{az/b-n-lb'(I)}{z-n-lb'(I)}<\frac{c}{d}$，从定理 2 可知 $lb'(I)=\left\lceil\frac{ad-bc}{bd-bc}z+1\right\rceil-n=lb(I)-n$，假设新类型变化上界为 $ub'(I)$，则有 $\frac{az/b-n}{z-n+ub'(I)}<\frac{c}{d}$，从定理 2 可知 $ub'(I)=\left\lceil\frac{ad-bc}{bc}z+1\right\rceil-\left\lceil\frac{d-c}{c}n\right\rceil=ub(I)-\left\lceil\frac{d-c}{c}n\right\rceil$；若 I 为非频繁项集，那么其新支持度变为 $\frac{az/b}{z-n}$，假设新类型变化下界为 $lb'(I)$，则有 $\frac{az/b}{z-n-lb'(I)}\geq\frac{c}{d}$，从定理 2 可知 $lb'(I)=\left\lceil z-\frac{ad}{bc}z+1\right\rceil-n=lb(I)-n$，假设新类型变化上界为 $ub'(I)$，则有 $\frac{az/b+ub'(I)}{z-n+ub'(I)}\geq\frac{c}{d}$，可知 $ub'(I)=\left\lceil\frac{bc-ad}{bd-bc}z+1\right\rceil-\left\lceil\frac{c}{d-c}n\right\rceil=ub(I)-\left\lceil\frac{c}{d-c}n\right\rceil$. 证毕.

例 1. 图 3 是滑动窗口中删除了事务 abc 和 bd 后的枚举树，最小相对支持度为 1/3，其过程可描述如下：

对频繁项集的类型变化界限，首先根据定理 3

进行更新，得到{−1:0}①、{2:1}和{5:3}三组值. 对于{−1:0}对应的项集 a、bc、bd 和 cd 分别进行支持度和类型变化界限计算，其中 a、bc 和 bd 对应的类型变化界限为{2:1}，而 cd 对应的类型变化界限为{5:3}.

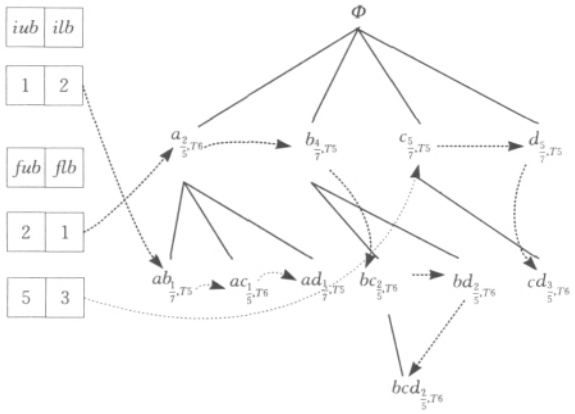

图 3　删除事务 abc 和 bd 后对应的枚举树

对非频繁项集的类型变化界限，首先根据定理 3 进行更新，得到{1:2}和{0:−1}两组值. 对于{0:−1}对应的项集 ac 和 bcd 分别进行支持度和类型变化界限计算，其中 ac 对应的类型变化界限为{1,2}，而 bcd 变为频繁项集，对应的类型变化界限为{2:1}.

4.3　事务插入处理

当一组事务 Φ 进入滑动窗口的时候，假设被插入事务的数量为 n，根据类型变化上界和支持度的不同，分别进行处理：

所有类型变化上界小于等于 n 的项集 I 均有可能发生类型变化. 若 I 为频繁项集，那么只有当 I 不被该组事务包含时，I 的类型才可能发生变化；若 I 是非频繁项集，那么只有当 I 被该组中的所有事务所包含时，I 的类型才可能发生变化.

所有类型变化上界大于 n 的项集 I 均不可能发生类型变化. 对于这样的项集来说，一方面要更新类型变化上界. 另一方面，为了保证在插入事务后所有类型变化下界正确，对于每个未处理的项集 I 来说，若 I 是频繁的，则假设每个插入事务均不包含 I；若 I 是非频繁的，则假设每个插入事务均包含 I，用来更新类型变化下界.

定理 4.　假设被插入事务数量是 n，则项集 I 的类型变化上界 $ub(I)$ 变为 $ub(I)-n$. 若 I 为频繁项集，其类型变化下界 $lb(I)$ 变为 $lb(I)-\left\lceil\frac{c}{d-c}n\right\rceil$；若 I 为非频繁项集，其类型变化下界 $lb(I)$ 变为 $lb(I)-\left\lceil\frac{d-c}{c}n\right\rceil$.

证明.　同定理 3.

例 2.　图 4 是滑动窗口中插入了事务 $abcd$ 后的枚举树，最小支持度为 1/3. 其过程可描述如下：

对频繁项集的类型变化界限，首先根据定理 4 进行更新，得到{0:0}和{1:2}和两组值. 对于{0:0}对应的项集 a、b、bc、bd 和 bcd 分别进行支持度和类型变化界限计算，得到支持度均为 3/6，类型变化界限为{4:2}.

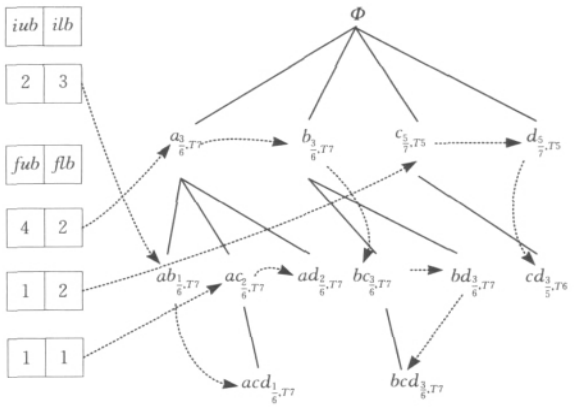

图 4　插入事务 $abcd$ 后对应的枚举树

对非频繁项集的类型变化界限，首先根据定理 4 进行更新，得到{0:0}. 对对应的项集 ab、ac 和 ad 分别进行支持度和类型变化界限计算，其中 ab 对应的类型变化界限为{2,3}，而 ac 和 ad 变为频繁项集，对应的类型变化界限为{1:1}；同时，生成新的项集 acd，其支持度为 1/6，类型变化界限为{2,3}.

4.4　算法设计与优化讨论

根据定理 3 和定理 4 可以将事务的插入和删除操作合并在一起，算法 2 是 FIMoTS 的总体思路. 该算法会在新的数据到达时，分别扫描类型变化界限数组，更新类型变化界限值，并将其中变为 0 或者负值的界限值所指向的结点进行支持度的重计算，并根据其新的类型变化界限重新归类.

算法 2.　FIMoTS 算法.

输入：滑动窗口 D，项目结点相对最小支持度 λ_r，被删除事务组 Φ，新到达的事务组 Φ'，类型变化界限数组 fb 和 ib，当前滑动窗口的时间戳 TS

输出：枚举树 ET

方法：

1.　for each $fb[i]$ do

2.　　$fb[i].flb=fb[i].flb-|\Phi|-\left\lceil\frac{c}{d-c}|\Phi'|\right\rceil$;

① {−1:0}分别对应类型变化上界和类型变化下界.

```
3.    fb[i].fub=fb[i].fub-⌊(d-c)/c|Φ|⌋-|Φ'|;
4.  end for
5.  for each fb[i] do
6.    if fb[i].flb<=0 or fb[i].fub<=0 then
7.      for each fb[i]指向的结点 n_l do
8.        计算 I 的新支持度和新类型变化界限值,
          并调整界限指针的指向;
9.        if n_l.sup<λ_r then
10.         剪枝 n_l 的所有子结点;
11.       end if
12.     end for
13.   end if
14. end for
15. for each ib[i] do
16.   ib[i].ilb=ib[i].ilb-|Φ|-⌊(d-c)/c|Φ'|⌋;
17.   ib[i].iub=ib[i].iub-⌊c/(d-c)|Φ|⌋-|Φ'|;
18. end for
19. for each ib[i] do
20.   if ib[i].ilb<=0 or ib[i].iub<=0 then
21.     for each fb[i] 指向的结点 n_l do
22.       计算 I 的新支持度和新类型变化界限值,
          并调整界限指针的指向;
23.     end for
24.     for each 新频繁结点 n_l do
25.       调用 INITIAL(D,λ_r,n_l);
26.     end for
27.   end if
28. end for
```

在流数据挖掘过程中,在重新计算某个项集的支持度时,增量计算能够提高挖掘效率,用在枚举树中结点的时间戳可以实现该计算方式.假设滑动窗口当前到达 T_i 处,而目前需要处理的结点 I 保存的时间戳为 T_j,也就是说,I 在时间戳 T_j 后就没有更新过,其支持度为 $\frac{a}{b}$,因此需要根据这段时间到达和离开的事务重新计算支持度:若滑动窗口包含了 n 个时间戳的项集,T_{i-n} 到 T_{j-n} 期间共有 x 个事务,包含 I 的有 y 个事务;T_i 到 T_j 期间共有 t 个事务,包含 I 的有 s 个事务,则 I 的新支持度为 $\frac{a-y+s}{b-x+t}$.

定理 5. 给定最小相对支持度 $\lambda_r=\frac{c}{d}$ 和滑动窗口 D,为了保证项集 I 支持度的增量更新,需要保存 $2|D|$ 个事务.

证明. 不失一般性,假设项集 I 在 T_i 时刻为频繁项集,且在 T_j 时刻需要更新,也就是说,此时 I 的类型变化界限是小于或者等于 0 的.由于在滑动窗口移动过程中,插入和删除事务均假设对 I 的支持度产生影响,那么从 T_{i-n} 到 T_{j-n} 之间事务数量的最大值就是能够保证 I 在事务删除时能够进行增量更新需要保存的事务数量.为了保证该最大值,则需要将从 T_i 到 T_j 之间事务数量的最小化,以及 I 支持度的最大化,即在 $\frac{a-y+s}{b-x+t}$ 中,以 $a=b=|D|$ 和 $t=s=0$ 为前提求得 x 的最大值.由于只有每个删除事务均包含 I,才能使得 I 的支持度降低,所以应满足 $x=y$,给定最小相对支持度 $\lambda_r=\frac{c}{d}$,只有 $y=b=|D|$ 的情况下才可能使得 I 变为非频繁项集,即最多需要保存包含滑动窗口在内的 $2|D|$ 个事务. 证毕.

如图 5 所示,采用了扩展的 FP-tree[21] 来维护数据流中的原始数据,不但可以降低内存使用代价,还可以满足事务删除处理的需要.扩展的 FP-tree 不是仅仅保存最小支持度以上的项目集合,而是保存了所有滑动窗口中事务的项目集合,这样可以迅速计算支持度.另外,由于保存了事务的完整信息,扩展的 FP-tree 可以实现增量更新.

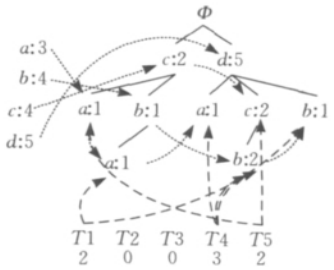

图 5 初始滑动窗口对应的 FP-tree

根据定理 5,扩展的 FP-tree 维护了 $2|D|$ 个事务,为了便于识别,在实现中增加了时间戳集合,用来标识每个时间段到达的事务.

定理 6. 给定最小相对支持度 $\lambda_r=\frac{c}{d}$ 和滑动窗口 $D(|D|=z)$,如果在一个时间戳内有 m 个事务到达,同时有 n 个事务离开,那么有以下结论成立:

(1) 对于一个频繁项集 $I\left(\Lambda_r(I)=\frac{a}{b}\right)$ 来说,其类型变化界限可以持续在 $\left\lceil\frac{(ad-bc)z}{(bd-bc)n+bcm}\right\rceil$ 个时间戳内不需要重新计算.

(2) 对于一个非频繁项集 I 来说,其类型变化界限可以持续在 $\left\lceil\frac{(bc-ad)z}{(bd-bc)m+bcn}\right\rceil$ 个时间戳内不需

要重新计算.

证明.

(1) 从引理 1 可以看到一个频繁项集 I 的类型变化上界 $ub(I)\approx\frac{ad-bc}{bc}z$，当一个时间戳的事务到达时，根据定理 3，$ub(I)$ 应减去 $\frac{d-c}{c}n$，根据定理 4，$ub(I)$ 应减去 m，平均看来，$ub(I)$ 在经过 $t=\left\lceil\frac{\frac{ad-bc}{bc}z}{\frac{d-c}{c}n+m}\right\rceil=\left\lceil\frac{(ad-bc)z}{(bd-bc)n+bcm}\right\rceil$ 个时间戳后变为负值，同样，其类型变化下界 $lb(I)\approx\frac{ad-bc}{bd-bc}z$，根据定理 3 和定理 4 可知，它将在 $t=\left\lceil\frac{\frac{ad-bc}{bd-ac}z}{n+\frac{c}{d-c}m}\right\rceil=\left\lceil\frac{(ad-bc)z}{(bd-bc)n+bcm}\right\rceil$ 个时间戳后变为负值.

(2) 证法同(1).　　证毕.

5　实　验

用 Visual C# 在内存为 2 GB，CPU 为 Petium Dual 1.6 GHz，操作系统为 Windows XP 的机器上实现了 FIMoTS 算法，并与 Naïve 算法进行比较. 本文没有与文献[18]的研究结果进行对比，这是因为尽管与文献[18]中研究的问题类似，但由于文献[18]的挖掘结果存在误差，而本文则是对结果的精确挖掘，二者不具有可比性. 采用了 4 种标准的数据集作为实验的测试数据集，包括 2 种由 IBM 的数据生成器生成的模拟数据集 T10I4D100K 和 T40I10D100K 以及 2 种真实数据集 KOSARAK 和 MUSHROOM，其中 KOSARAK 是匈牙利一家在线新闻门户网站的点击流数据，MUSHROOM 记录了不同品种的蘑菇特性. 表 1 列举了 4 种数据集的主要数据特征.

表 1　数据集的主要特征

数据集	事务数量	平均事务长度	最小事务长度	最大事务长度	项目数量
T10I4D100K	100 000	10.1	1	29	870
T40I10D100K	100 000	39.6	4	77	942
KOSARAK	990 002	7.1	1	2497	36 841
MUSHROOM	8129	23.0	23	23	119

选取了每个数据集的前 n 个事务作为初始滑动窗口的数据内容，剩余的事务作为新增数据. 在实验过程中为了体现数据流的时间特性，将数据随机划分为块，每个块对应一个时间戳到达的事务，其中包含 θ 个事务(θ 是随机生成的数值，该数值在 $[0,0.01n]$ 内变化). 以 KOSARAK 数据集为例，如果选择前 20 000 个事务作为初始滑动窗口的内容，则将在后续的事务中每次随机选择 $[0,200]$ 之间的事务数量作为一次到达的数据量.

5.1　运行时间代价比较

图 6 显示了 Naïve 算法与 FIMoTS 算法在不同

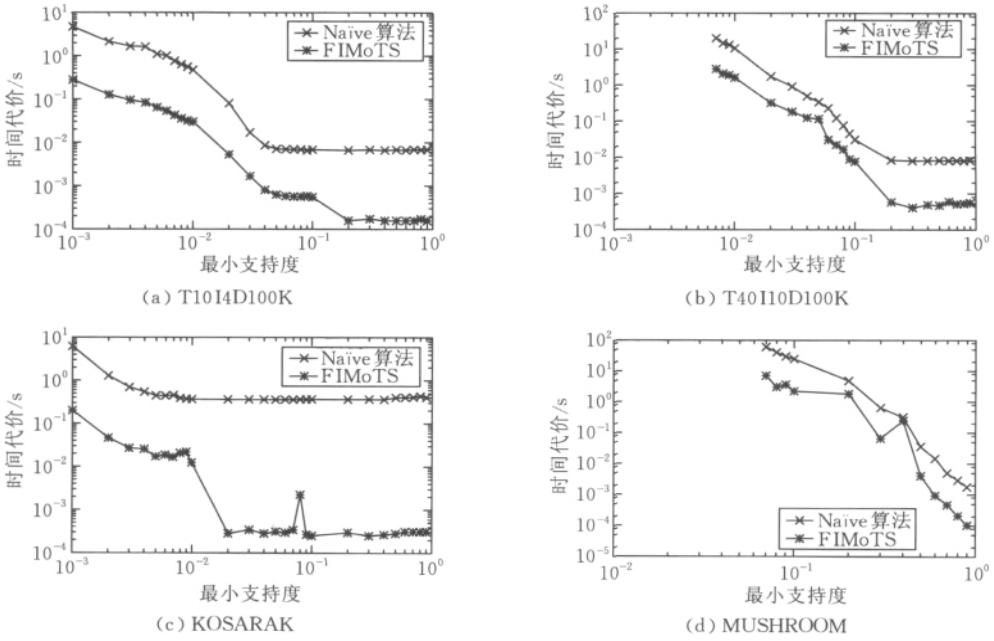

图 6　不同最小支持度情况下的运行时间代价比较

最小支持度阈值 λ_r 情况下每个事务的平均运行时间代价. 从 4 种不同密度数据集上的运行情况来看，Naïve 算法的运行时间普遍达到 FIMoTS 算法运行时间的十倍左右，在最小支持度比较高的情况下能够达到将近百倍甚至上千倍，说明 FIMoTS 通过剪枝获得了较高的计算效率. 但是，其计算效率的提高并不是随着最小支持度呈现线性变化，如在 KOSARAK 中最小支持度为 0.08 时，其运行代价反而要高于最小支持度为 0.07 和 0.09 的运行代价，说明该算法在剪枝的过程中具有随机性，与具体数据内容有关.

另外，比较在同一阈值下的运行时间，可以看到，MUSHROOM 的数据密度比较大，因此其运行时需要的计算代价也较大，且 FIMoTS 算法的优化效果没有十分明显，而在 KOSARAK 上，尽管其数据密度较小，但某些事务很长，最长达到了 2497，这也影响了 Naïve 算法在其上的运行效率，使得 FIMoTS 算法优化的效果在最小支持度较高的情况下较为明显.

5.2 项集数量比较

表 2 是滑动窗口到达某一时刻 Naïve 算法与 FIMoTS 算法的枚举树更新情况，分别在 4 种数据集上选取了一个最小支持度阈值，显示了该时刻的结点增加、删除和最终数量，可以看到，FIMoTS 的最终结点数量是与 Naïve 算法相同的，表明了该算法是一个能够进行精确挖掘的方法. 同时从增加和删除的结点数量来看，FIMoTS 算法均少于 Naïve 算法，表明在挖掘过程中其内存使用代价要略低于或与 Naïve 算法基本持平. 这是因为该算法能够忽略一些临时结点的处理，比如当一个结点刚生成即被删除掉，或一个结点刚被删除又需要重新生成的情况. 这也是该算法能够在计算代价上更加有效的原因之一.

表 2 结点更新数量比较

数据集	最小支持度	初始结点数量	Naïve 算法的更新结果			FIMoTS 算法的更新结果		
			增加结点数量	删除结点数量	最终结点数量	增加结点数量	删除结点数量	最终结点数量
KOSARAK	0.001	830129	5070	7339	827860	5036	7305	827860
MUSHROOM	0.1	2127694	245475	178600	2194569	217920	151045	2194569
T10I4D100K	0.001	543096	51891	60917	534070	51460	60486	534070
T40I10D100K	0.01	1054435	331555	311232	1074758	319455	299132	1074758

6 总 结

数据流中的数据具有时间相关性. 本文提出了一种能够体现数据流时间特性的频繁项集挖掘方法 FIMoTS. 该方法引入了类型变化界限的概念，将数据快速保存到枚举树中，利用类型变化界限的不同对项集进行动态分类，在数据到达和数据离开的时候，根据滑动窗口大小的变化进行计算裁剪，仅对超出类型变化界限的项集进行处理；另外，讨论了采用时间戳进行增量计算的优化策略. 实验结果表明，与 Naïve 方法相比，该方法能够在不增加内存开销的情况下显著提高计算效率，是可行的，有效的.

参 考 文 献

[1] Agrawal R, Imielinski T, Swami A N. Mining association rules between sets of items in large databases//Proceedings of the ACM SIGMOD the International Conference on Management of Data. Vienna, Austria, 1993: 207-216

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules//Proceedings of the VLDB the Very Large Databases. Santiago, Chile, 1994: 487-499

[3] Agrawal R, Srikant R. Mining sequential patterns//Proceedings of the ICDE the International Conference on Data Engineering. Taipei, China, 1995: 3-14

[4] Xiong H, Tan P-N, Kumar V. Hyperclique pattern discovery. DMKD the Data Mining and Knowledge Discovery, 2006, 13(2): 219-242

[5] Chang J H, Lee W S. Finding recent frequent itemsets adaptively over online data streams//Proceedings of the International Conference on Knowledge Discovery and Data Mining. Washington, DC, USA, 2003: 487-492

[6] Li H, Lee S, Shan M. An efficient algorithm for mining frequent itemsets over the entire history of data streams//Proceedings of the International Workshop Frequent Itemset Mining Implementations. Seattle, WA, USA, 2004: 20-24

[7] Giannella C, Han J, Pei J, Yan X, Yu P S. Mining frequent patterns in data streams at multiple time granularities//Kargupta H, Joshi A, Sivakumar K, Yesha Y eds. Next Generation Data Mining. AAAI/MIT, 2003: 191-210

[8] Chang J H, Lee W S. estWin: Adpatively monitoring the recent change of frequent itemsets over online data streams//Proceedings of the Conference on Information and Knowledge Management. New Orleans, Louisiana, USA, 2003: 536-539

[9] Jin R, Agrawa G. An algorithm for in-core frequent itemset mining on streaming data//Proceedings of the IEEE International Conference on Data Mining. Houston, Texas, USA, 2005: 210-217

[10] Mozafari B, Thakkar H, Zaniolo C. Verifying and mining frequent patterns from large windows over data streams// Proceedings of the International Conference on Data Engineering. Cancun, Mexico, 2008: 179-188

[11] Deypir M, Sadreddini M H. An efficient algorithm for mining frequent itemsets with large windows over data streams. International Journal of Data Engineering, 2011, 2(3): 119-125

[12] Calders T, Dexters N, Gillis J M, Goethals B. Mining frequent itemsets in a stream//Proceedings of the International Conference on Data Mining. Omaha, NE, USA, 2007: 83-92

[13] Calders T, Dexters N, Goethals B. Mining frequent items in a stream using flexible windows. Intelligent Data Analysis, 2008, 12(3): 293-304

[14] Li H F, Huang H Y, Lee S Y. Fast and memory efficient mining of high-utility itemsets from data streams: With and without negative item profits. Kowledge and Information Systems, 2011, 28(3): 495-522

[15] Lin C, Chiu D, Wu Y, Chen A L P. Mining frequent itemsets from data streams with a time-sensitive sliding window//Proceedings of the SIAM International Conference on Data Mining. Trondheim, Norway, 2005: 68-79

[16] Khan M S, Coenen F, Reid D, Patel R, Archer L. A sliding windows based dual support framework for discovering emerging trends from temporal data. Knowledge-Based Systems, 2010, 23(4): 316-322

[17] Koh J, Shin S. An approximate approach for mining recently frequent itemsets from data streams//Proceedings of the International Conference on Data Warehousing and Knowledge Discovery. Krakow, Poland, 2006: 352-362

[18] Koh J, Don Y. Approximate mining recently representative patterns on data streams//Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. Nanjing, China, 2007: 231-243

[19] Li H, Lee S Y. Mining frequent itemsets over data streams using efficient window sliding techniques. Expert System with Applications, 2009, 36(2): 1466-1477

[20] Yu J X, Chong Z, Lu H, Zhou A. False positive or false negative: Mining frequent itemsets from high speed transactional data streams//Proceedings of the VLDB the Very Large Databases. Toronto, Canada, 2004: 204-215

[21] Han J, Pei J, Yin Y, Mao R. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery, 2004, 8(1): 53-87

LI Hai-Feng, born in 1979, Ph. D., lecturer. His research interests include data mining and data management, stream mining.

ZHANG Ning, born in 1975, professor, Ph. D. supervisor. Her research interest is information management.

ZHU Jian-Ming, born in 1965, Ph. D. supervisor, professor. His research interests include wireless network, network security.

CAO Huai-Hu, born in 1977, Ph. D., associate professor. His research interests include computer network architecture and mobile internet.

Background

The concept of frequent itemset mining was firstly proposed in 1994 by Agrawal. In recent years, with the wide used applications in industry, frequent itemset mining has been developed into an important branch of data mining. An itemset is frequent if its support is more than the threshold specified by users. The goal of frequent itemset mining is looking for the information users are interested in, such as association rules, sequential patterns and so on. Stream frequent itemset mining is a new direction, which is aimed at achieving the real-time data results as soon as possible. Traditional frequent itemset mining algorithm can not meet the needs of users, mainly in two aspects: On the one hand, stream is fast, and dynamic, data cannot be stored in the secondary disc to conduct multiple scans, thus, it has to be maintained in memory incrementally; on the other hand, when handle the itemsets, traditional model is a transaction-based sliding window, which is simple but not adapted to the real streams, since the data in stream is always timestamp-related. So far, we focus on the second problem and propose an efficient algorithm, which can mine the time sensitive streams in real time. A Naïve concept, the type transforming bound, is proposed in this paper to dynamically classify the itemsets according to their support, which can result in a large amount computation being pruned. As shown in the experimental results, our algorithm significantly outperforms the previous one. This work is supported by our National Nature Science Foundation of China (No. 61100112), and it introduces a new consideration for us to address the uncertain stream mining problem.