数据流频繁闭集的在线挖掘

刘 春 郑 征 蔡开元 张师超

(北京航空航天大学 自动化科学与电气工程学院, 北京 100191)

摘 要: 基于算法 LossyCounting,提出了数据流频繁闭集的在线挖掘算法 LC_Closed (LossyCounting_Closed). 设计了基于前缀树的频繁闭集压缩存储结构 CI-forest (Closed Itemsets-forest),利用该数据结构可以快速的插入和查询闭集模式,且在处理新的事务数据时能够快速定位相关的历史闭集模式. 该算法采用在线的处理方式,提高了算法的实时性. 实验的结果证明该算法是有效的.

关 键 词: 数据挖掘; 数据流; 频繁闭集; 在线

中图分类号: TP 182

文献标识码: A **文 章 编 号**: 1001-5965(2008)08-0969-04

Online mining frequent closed itemsets over data stream

Liu Chun Zheng Zheng Cai Kaiyuan Zhang Shichao

(School of Automation Science and Electrical Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: Based on the algorithm LossCounting, a novel approach called LossyCounting_Closed (LC_Closed) for mining closed frequent itemsets over data stream was proposed. A new summary data structure called Closed-Itemsets-forest (CI-forest) was developed for maintaining only closed frequent itemsets. The insertion and query of closed itemsets can be rapidly made based on the data structure CI-forest, and the location of the associated historical closed itemsets in the stage of dealing with the new transaction is also facilitated by CI-forest. Since the algorithm maintains closed itemsets online, the current closed frequent itemsets can be output in real time based on user's specified thresholds. The effectiveness of the proposed method is shown in the experimental results.

Key words: data mining; data stream; frequent closed itemsets; online

频繁闭集的挖掘可以提供完全的且无信息丢失的频繁模式. 传统的频繁闭集的挖掘方法[1],其对象都是静态数据集,且需要多次访问数据库. 但是,数据流是一个有序的事务数据序列. 不同于静态数据集,它具有如下的特点[2]: ①往往是无边界和快速的; ②数据量往往是巨大的,而可利用的存储空间有限; ③要求处理的算法必须具有实时性. 在数据流的环境下,算法往往只能单遍(one-pass)扫描数据. 因此,传统的方法都不能应用于数据流环境下的频繁闭集挖掘.

文献[3]第1次提出了数据流频繁集挖掘的著名算法 LossyCounting. 本文基于该算法,提出了数据流频繁闭集挖掘算法 LC_Closed (LossyCounting_Closed). 设计了一种新的数据结构 CI-forest (Closed Itemsets-forest) 来压缩存储数据流中的闭集.

1 基本概念

算法 LossyCounting[3]需要用户输入两个参数: (s, ε),这里 s 为最小支持度, ε 为误差上限. 数据流可以如下定义.

I 代表 n 个项目的集合, $I = \{i_1, i_2, \cdots, i_n\}$. 对任意一非空集合 $X \subseteq I$, 称 X 为一项目集. X 中的项目以由小到大的字典(lexicographic)顺序排列. 一个事务数据 $T = <k, X>$, k 为该事务数据的标识, X 为该事务数据所包含的项目集. 事务数据流

收稿日期: 2007-07-20

基金项目: 国家自然科学基金资助项目(60633010); 中国博士后基金资助项目(20070410453)

作者简介: 刘 春(1982-),男,河南信阳人,硕士生, liuchun@asee.buaa.edu.cn.

DS是一个无限的数据流片段的序列，$DS = B_1, B_2, \cdots, B_k$，每个片段携带一个唯一的标识，$k$为当前片段 B_k的标识. $B_i = T_1, T_2, \cdots, T_n$，$n$为每个片段的大小，其值由 ε决定 $(n = 1/\varepsilon)$.

由于数据流的特性，任何单遍算法都不可能获取准确的频繁模式信息，得到的只是近似的结果. 因此，本文用 $E\sup(X)$来表示概要数据结构中所记录的项目集 X的支持度.

定义 1 对一个集合 X，如果其支持度 $E\sup(X) \geq s \times N$，则称 X为一个频繁项目集，也简称频繁集 $(N$为事务数据总的个数$)$.

定义 2 对项目集 X，如果不存在项目集 X'满足：①$X \subset X'$；②$E\sup(X) = E\sup(X')$，则称 X为闭合的项目集，也称项目闭集.

定义 3 对一个事务数据 $<k, X>$，$X = \{x_1, x_2, \cdots, x_k\}$，则其后缀项目集为：$\{x_1, x_2, \cdots, x_k\}$，$\{x_2, \cdots, x_k\}$，$\{x_3, \cdots, x_k\}$，$\cdots$，$\{x_k\}$. 以 x_i为首项目的后缀项目集就是 X中去掉其前缀之后余下的项目组成的集合.

由项目闭集的定义 2，本文可以推导出如下的 4个引理.

引理 1 对一项目集 X，如果存在两个项目集 Y_1和 Y_2满足：$X \subset Y_1$，$X \subset Y_2$，$Y_2 \cap Y_1 = X$，$Y_1 \not\subset Y_2$，$Y_2 \not\subset Y_1$，则项目集 X一定是个项目闭集.

证明 假设 X不是项目闭集，则 $\exists Y, X \subset Y$，$E\sup(X) = E\sup(Y)$. 由 $X \subset Y_1 \Rightarrow E\sup(X) \geq E\sup(Y_1)$. $X \subset Y_2$，$Y_2 \not\subset Y_1 \Rightarrow \exists T_k$，$Y_1 \not\subset T_k$，$X \subseteq T_k$. 因此，$E\sup(X) \geq E\sup(Y_1) + 1$. 同理，可得 $E\sup(X) \geq E\sup(Y_2) + 1$. 又由 $E\sup(X) = E\sup(Y)$，可得 $E\sup(Y) > E\sup(Y_1)$，$E\sup(Y) > E\sup(Y_2)$. 由 $E\sup(Y) > E\sup(Y_1)$，$X \subset Y$，$X \subset Y_1$ 可得 $Y \subset Y_1$ (否则 $E\sup(X) > E\sup(Y)$). 同理，$Y \subset Y_2$. 因此可知 $Y \subseteq X$，这与假设矛盾，即可得 X一定是个项目闭集.

引理 2 给定项目集 X，且 X不是项目闭集，如果在给定的项目闭集的集合中，存在两个项目集 Y_1和 Y_2，且 $X \subset Y_1$，$X \subset Y_2$，则 $Y_1 \subset Y_2$或 $Y_2 \subset Y_1$.

证明 如果 $Y_1 \not\subset Y_2$，$Y_2 \not\subset Y_1$成立，则由引理 1可知项目 X一定是项目闭集. 这与已知矛盾，引理得证.

由引理 2，当 X不是项目闭集，且存在两个不同的项目集包含 X时，其中支持度大的项目集必然包含了由这两者所蕴含的 X的支持度信息. 这一结果可以扩展至当有 n个不同的项目集包含 X的情况. 下面依此给出引理 3.

引理 3 给定项目集 X，且 X不是项目闭集，如果在给定的项目闭集的集合中，存在 $n(n \geq 1)$个不同的项目集包含 X，则其中支持度最大的项目集必然包含了 X的所有支持度信息 (证明略).

引理 4 给定项目集 X和项目闭集的集合 U，如果 $X \in U$，则将 X加入 U时不会有新的闭集产生.

证明 假设会有新的项目闭集 V产生，由引理 3，在 U中必然存在项目闭集 T，T包含了项目集 V的全部支持度信息，且满足：$V \subset X$，$T \subseteq X$，$V \subset T$，$X \cap T = V$. 因此，$E\sup(V) = E\sup(T)$. 当 $<k, X>$到来之后，$E\sup'(T) = E\sup(T) + 1$，$E\sup'(V) = E\sup(V) + 1$ $(V \subset X, T \subseteq X)$. 显然，$E\sup'(V) = E\sup'(T)$，这也即意味着项目集 V仍然不是项目闭集.

在下面的讨论中，字母 U将被用来表示由 CI-forest所代表的项目闭集的集合.

2 *LC_Closed* 算法

2.1 数据结构 *CI-forest*

定义 4 CI-forest是一个扩展的前缀树，当有 3个事务数据 $(<1, ABCD>$，$<2, ACD>$，$<3, CD>)$到来时的数据结构图如图 1.

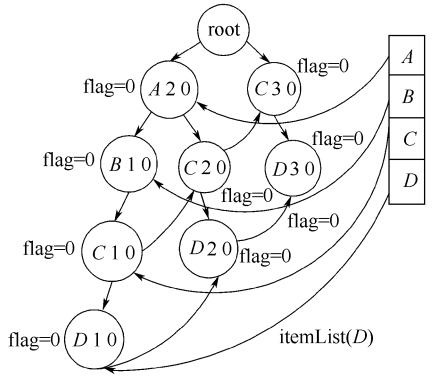


图 1 当有 3个事务到来时的 *CI-forest*

树中的每个结点包含的信息为 item，f，Δ，flag，这里 item为项目，f为该结点代表的集合的支持度，即 $E\sup(X)$，Δ为该结点代表的闭集的估计信息，记录的是该结点代表的闭集产生时的片段标识，flag为一个标志位，以标记一个结点在遍历的过程中是否被访问. 图 1中，结点 A携带的信息为 $(A\ 2\ 0)$，这里 item为 A，f为 2，Δ为 0，图中所有结点的标识位 flag均为 0.

树中有着相同项目的结点由指针连接起来，且每个项目都有一个指针指向该结点链 (itemList)，图 1标出了项目 D的结点链 itmeList(D). 以 root结点的孩子结点为根结点的子树，称为该

根结点项目的项目子树(itemTree).

2.2　算法分析

本文在线的处理到来的事务数据,假设当前到来的事务数据为 $<k, X>$,则项目集 X 与项目闭集的集合 U 之间的可能的关系可以用图 2所示的关系树来表示.

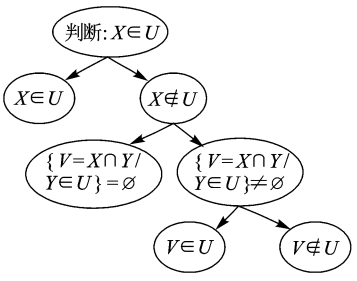


图 2　关系树

下面将讨论树中各个结点对应的情况.

1) $X \in U$. 这表明项目集 X 是 U 中已经存在的项目闭集. 从引理 4可知,此时不会有新的闭集出现. 在这种情况下,只须更新项目闭集 X 和 U 中包含的 X 的所有子集的支持度.

2) $X \notin U$. 这表明项目集 X 是一个新的集合,因此其一定是新的闭集,应该插入到 U 中. 但是,此时 $X \cap Y(Y \in U)$ 又存在两种情况: ①$\{V = X \cap Y / Y \in U\} = \varnothing$. 显然,在这种情况下只须将 X 插入到 U 中去. ②$\{V = X \cap Y / \in U\} \neq \varnothing$. 此时,根据交集 V 的状态,又有两种情况: $V \in U$,这表明 V 已经是 U 中的闭集,此时只需更新 V 的支持度; $V \notin U$,在这种情况下,由引理 1,可知 V 是新的项目闭集,因此,此时应该将 V 插入到 U 中.

由以上的分析可知,当一个新的事务数据到来时,主要的操作就是判断一个项目集是否属于当前集合 U,以及判断其与 U 中的闭集之间是否存在交集.

2.3　算法实现

这一部分将给出基于 CI-forest的算法 LC_Closed,其详细步骤如下,其中 x_i 代表项目, x_i' 代表其对应的后缀项目集.

算法 LC-Closed如下.

输入:数据流 DS,最小支持度 s,误差上限 ε

输出:频繁闭集

```
1:  foreach transaction T_k of DS
2:      Addition(T_k, CI-forest);
3:      n++; //计数器
4:      if n ×ε = 0 //到达片尾
5:          pruning; //剪枝
6:      if user requests
7:          output;
Addition(T<K, X>, CI-forest)
8:  Generate all suffix-itemsets <x_1', x_2', …, x_i', …, x_l'> of X;
9:  If x_1' ⊑ CI-forest
10:     For each itemTree(x_i) from x_1 to x_l
11:         If itemTree(x_i) exists
12:             Then update the subsets of X in itemTree(x_i);
13:     Else
14:         For each item x_i from x_1 to x_l
15:             Travel itemList(x_i) using suffix-itemset x_i'
                to obtain intersections;
16:             Insert all intersections to itemTree(x_i);
17:         Insert the itemset x_i';
```

由于基于算法 LossyCounting[3],每个结点都携带一个估计信息 Δ,这个值在该闭集产生时得到,记录的是产生该闭集的片段标识,且在该闭集的存在期间不变. 算法每隔 $1/\varepsilon$ 片段大小就执行 1次删除操作,删除那些 $(f + \Delta) < \varepsilon \times N$ (N 为到来的事务数据的个数)的所有结点.

在 CI-forest中,结点都是以递增的字典顺序向下生长的,因此采用深度优先的策略遍历结点链 itemList(x_j) $(i \leq j \leq n)$就能够得到所有以 x_j 为首项目的交集. 同时,以递增的字典顺序来遍历各个项目的 itemList,这样在遍历 itemList(x_j)时,不用考虑 itemList(x_j)以上的结点. 且为了避免结点的重复访问,在遍历 itemList(x_j) $(i \leq j \leq n)$时,如果结点 x_j 的标志位 flag没有被置位,则深度优先遍历以该结点为根的子树,并将子树中的被访问的结点都置位,如果结点 x_j 已经被置位,则跳过该结点并将该结点复位,这是设置标志位 flag的原因. 在这过程中,如果得到相同的交集携带不同的支持度,则由引理 3只保留其中支持度最大的.

在遍历 itemList(x_j)和 itemTree(x_j) $(i \leq j \leq n)$过程中,为了提高效率,项目 x_j 的后缀项目集 x_j' 被用来替代项目集 X 来判断要检查的项目是否属于 X.

下面用例 1来说明算法的过程.

例 1　当已经有两个事务数据 $<1, ABCD>$、$<2, ACD>$到来时 CI-forest的简略结构如图 3.

当事务数据 $<3, CD>$ $(X = \{CD\})$到来时,先生成所有的后缀项目集: $x_1' = CD$, $x_2' = D$. 然后以深度优先的策略,先后遍历 ItemList(C)和 ItemList(D). 遍历 ItemList(C)时得到交集: $(CD\ 1\ 0)$, $(CD\ 2\ 0)$,此时的 CI-forest的状态如图

4. 遍历 ItemList(D)时,结点 D都被访问过,所以均跳过这些结点,并将他们的标志置位. 然而,当遍历项目 C的结点链时,得到两个相同的交集:(CD 1 0),(CD 2 0). 根据引理 3,可以将(CD 1 0)丢弃.

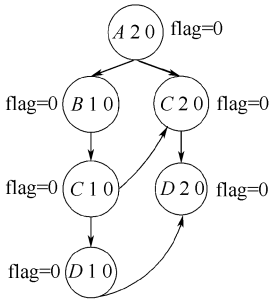


图 3 CI-forest结构图

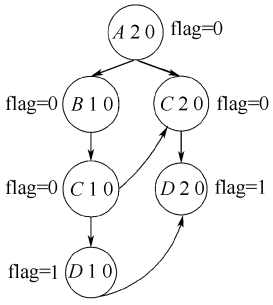
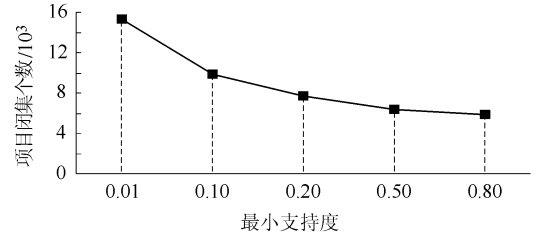
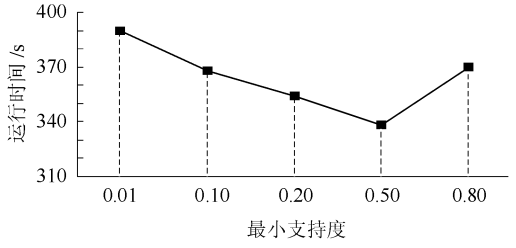


图 4 遍历完 itemList(C)

插入一个新的事务数据需要更新的结点减少,这样算法执行的时间自然减少. 但是,随着最小支持度的增加,数据流每片的大小也减小了,片的总数目也增多了,执行剪枝操作的次数也跟着增加了. 因此,在某一时刻,由剪枝操作增加所引起的消耗必然超过由最小支持度增加所减少的消耗. 从这一时刻起,算法执行的时间开始不断的增加.



a 项目闭集数目



b 运行时间

图 5 产生的项目闭集数目和运行时间

3 仿真实验

这一部分将对本文算法进行验证和评估. 实验数据由 IBM 数据生成器产生,如 T10I6D100k (方式与文献[1]相同),这里 T为事务数据的平均长度,I为模式的平均长度,D为数据集中事务数据的总数. 算法将一条一条的读入每个数据集中的数据,来模拟数据流的环境. 所有的实验都将在 cpu为 1.7G,内存为 512M的机器上进行,且所有的程序都是用 C++实现的,编译器用的是 VC6.0. 在实验中,设定参数 ε=0.1s

首先将 Addition算法同算法 FPclose[4]进行比较,验证二者产生的结果是否一致来确定分析的正确性. 实验中,FPclose[4]算法的支持度设为 1,即产生所有的闭集. 通过实验发现,二者产生的闭集是一致的. 这说明对算法 LC_Closed所做的分析和实现是有效的.

图 5反映算法作用于数据集 T10I6D100k的性能情况. 从图 5a中可以看出,生成的项目闭集的数目随着最小支持度的增大而减少. 但是,图 5b却显示出算法执行的时间随着最小支持度的增大先减小而后增大. 这是因为,在开始随着最小支持度的增大,CI-forest的规模逐渐减小,这样每

4 结束语

本文提出了挖掘数据流频繁闭集的算法 LC_Closed. 一个新的数据结构 CI-forest被用来存储闭集. 该算法在线的处理到来的事务数据,在每次添加新的事务数据时,只是更新相关的闭集结点,有效的缩减了搜索空间. 当有用户需求时,能立即输出用户所关心的频繁闭集,提高了实时性. 实验证明该算法是有效的.

参考文献 (References)

[1] Wang J, Han J, Pei J. CLOSET+: searching for the best strategies for mining frequent closed itemsets[C]// SIGKDD'03. WDC, USA: ACM, 2003: 236 - 245

[2] Babcock B, Babu S, Datar M, et al. Models and issues in data stream systems[C]//ACM PODS'02. Madison, USA: ACM, 2002: 1 - 16

[3] Manku G, Motw R. Approximate frequency counts over data streams[C]//Proc 28th Int Conf of VLDB. Hongkong, China: Morgan Kanfmann, 2002: 346 - 357

[4] Gosta G, Zhu J. Efficiently using prefix-trees in mining frequent itemsets[C]//Proc of IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FMI'03). 2003

(责任编辑:刘登敏)