计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
42(12):2192~2198, 2005

挖掘数据流中的频繁模式

刘学军[1,2] 徐宏炳[1] 董逸生[1] 王永利[1] 钱江波[1]
[1](东南大学计算机科学与技术系 南京 210096)
[2](南京工业大学信息科学与工程学院 南京 210009)
(lxj-gd@vip.sina.com)

Mining Frequent Patterns in Data Streams

Liu Xuejun[1,2], Xu Hongbing[1], Dong Yisheng[1], Wang Yongli[1], and Qian Jiangbo[1]
[1](*Department of Computer Science and Technology, Southeast University, Nanjing* 210096)
[2](*College of Information Science and Engineering, Nanjing University of Technology, Nanjing* 210009)

Abstract Finding frequent items is one of the most basic problems in the data streams. The limitless and mobility of data streams make the traditional frequent-pattern algorithm difficult to extend to data streams. According to data streams characteristic, inspired by the fact that the FP-growth provides an effective algorithm for frequent pattern mining, a new FP-DS algorithm for mining frequent patterns from data streams is proposed. In addition, the method, in which data streams are partitioned and frequent items are mined step by step, is adopted in the algorithm. So users may continuously get present frequent items online and any length frequent patterns for data streams can effectively be mined. Through introducing error ε, a large number of non- frequent items will be cut down and the storage space of the data streams can be reduced. Based on this algorithm, the error of support is guaranteed not to exceed ε. The analysis and experiments show that this algorithm has good performance.

Key words data streams; frequent patterns; FP-DS algorithm; stream data mining

摘 要 发现数据流中的频繁项是数据流挖掘中最基本的问题之一.数据流的无限性和流动性使得传统的频繁模式挖掘算法难以适用.针对数据流的特点,在借鉴 FP-growth 算法的基础上,提出了一种数据流频繁模式挖掘的新方法:FP-DS 算法.算法采用数据分段的思想,逐段挖掘频繁项集,用户可以连续在线获得当前的频繁项集,可以有效地挖掘所有的频繁项集,算法尤其适合长频繁项集的挖掘.通过引入误差 ε,裁减了大量的非频繁项集,减少了数据的存储量,也能保证整个数据集中项目集支持度误差不超过 ε.分析和实验表明算法有较好的性能.

关键词 数据流;频繁模式;FP-DS 算法;流数据挖掘

中图法分类号 TP311

1 引 言

发现数据流中的频繁模式是当前数据挖掘领域的一个新的研究热点.数据流的无限性和流动性使得传统的频繁模式挖掘算法难以适用,基于数据流的频繁模式挖掘面临着十分艰巨的挑战.

根据数据流的特点,文献[1]提出了 FP-stream 数据结构,用于解决数据流频繁项集的挖掘,当事务的平均长度和频繁模式都很短时,算法是有效的.

收稿日期:2004-12-13;修回日期:2005-09-05
基金项目:江苏省高技术基金项目(BG2004034);江苏省 2004 年度研究生创新计划基金项目(xm04-36)

反之,算法效率迅速下降. 文献[2]采用数据分段的思想,给出了求解单个频繁项的有效算法,同时也对频繁项集的求解给出了实践的方法. 文献[3]根据 Hoeffding Bound 理论对数据流分段,用这个小的数据流分段代替整个数据所有长度对项集计数进行估计. 文献[4,5]求解的重点只是单个模式的情况. 文献[6]研究的重点是发现最近的频繁模式. 文献[7]研究了序列模式的挖掘问题,发现滑动窗口中的频繁项集. 文献[8]和文献[9]则分别研究了数据流的多层频繁模式和半结构化数据流的频繁模式发现. 文献[10,11]的方法比较适用于单个模式或模式事先固定且不发生改变的情况.

本文结合数据流和频繁模式挖掘本身的特性,在借鉴 FP-growth 算法[12]的基础上,提出了一种新的数据流频繁模式挖掘算法:FP-DS 算法. 实验表明,FP-DS 算法具有较好的时间和空间效率,尤其适合长频繁项集的挖掘. 全文分为 5 节,其中,第 2 节对基本问题进行了描述;第 3 节给出了我们的 FP-DS 算法;第 4 节是算法的实验结果和性能分析;第 5 节进行了全文总结.

2 问题定义和描述

定义 1. 设 DS 是数据流序列,其中包含项目集 X 的事务数称为项目集 X 的支持数,记为 $f_{DS}(X)$. 项目集 X 的支持度记为 $X.sup$,$X.sup = f_{DS}(X)/|DS|$,其中 $|DS|$ 是数据流 DS 中的事务数.

定义 2. 设给定的支持度 S 和允许偏差 ε,$|N|$ 表示到目前为止,所见到的数据流 DS 的事务数,对于项目集 X,如果有 $f_{DS}(X) \geqslant (S-\varepsilon)|N|$(即 $X.sup \geqslant S-\varepsilon$),则称 X 为 DS 中的频繁项目集,简称频繁项集;如果 $f_{DS}(X) > \varepsilon|N|$,则称 X 为 DS 中的临界频繁项目集,简称临界频繁项集;如果 $f_{DS}(X) \leqslant \varepsilon|N|$,则称 X 为 DS 中的非频繁项目集,简称非频繁项集.

定义 3. FP-DS 树.

① 它由树根(记做 null)、临界频繁项集构成的前缀子树和临界频繁项头表组成;

② 除根结点之外的每一个结点由 7 个域组成($data$,f,$reval$,$pnode$,par,$leftchild$,$rightchild$),$data$ 为该结点代表的项目名,f 表示包含从根结点(不含根结点)开始到该结点所构成项集的所有项集计数和,$reval$ 是重构 FP-DS 树时为避免项集的重复插入而设的一个计数值,$pnode$ 为指向具有相同项目名的下一个结点的指针,par 是指向父结点的指针,$leftchild$ 是指向第 1 个孩子结点的指针,$rightchild$ 是指向兄弟结点的指针;

③ 头表中的每个元组由两部分组成:项目名和指向树中有相同项目名的第 1 个结点.

满足上述特征的树称为 FP-DS 树. 一个 FP-DS 树结构如图 1 所示. 由此可见,FP-DS 树类似 FP-tree,它的每个结点与 FP-tree 相比多了一个 $reval$ 域,在生成 FP-DS 树时,每个结点的 $reval$ 值等于 f 的值.

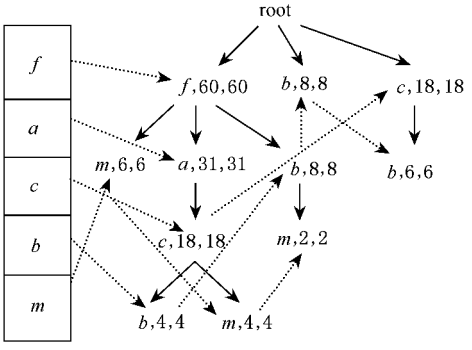

Fig. 1 FP-DS tree structure.

图 1 FP-DS 树结构

定理 1. 将数据流 DS 分段 $N_1, N_2, \cdots, N_i, \cdots$,令 $N = N_1 \cup N_2 \cup \cdots \cup N_k$,$1 \leqslant i < k$,任意项集 X,对于给定的支持度 S 和允许偏差 ε,若 $f_{N_i}(X) \leqslant \varepsilon|N_i|$,$f_N(X) \geqslant S|N|$,则 $f_{N_k}(X) \geqslant (S-\varepsilon)|N|$. 其中 $|N_i|$ 表示第 i 段的事务数.

证明. 因为 $f_N(X) = \sum_{i=1}^{k-1} f_{N_i}(X) + f_{N_k}(X)$,所以 $f_{N_k}(X) = f_N(X) - \sum_{i=1}^{k-1} f_{N_i}(X) \geqslant S|N| - \sum_{i=1}^{k-1} \varepsilon|N_i| \geqslant S|N| - \varepsilon|N| = (S-\varepsilon)|N|$,即 $f_{N_k}(X) \geqslant (S-\varepsilon)|N|$. 证毕.

根据定理 1,可以得出如下结论:删除当前的非频繁项集,即使它将来成为频繁项集,其支持度误差至多不会超过 ε,这种删除不会影响频繁项集的正确输出. 因此,我们只需保存临界频繁项集.

3 FP-DS 算法

3.1 全局临界频繁 1 项集的生成和更新

数据流分段 $N_i(i = 1,2,\cdots)$,每段长度 $|N_i| = \lceil 1/\varepsilon \rceil$ 或 $\lceil 1/\varepsilon \rceil$ 的整数倍. 为方便算法的描述,取 $|N_i| = \lceil 1/\varepsilon \rceil$. 全局 1 项集的数据集合 f_list 是

一个数组,每个 1 项集是数组中的一个元素,数据类型是一个有 3 个数据域的结构体($data$,f,del),其中,$data$ 是 1 项集的取值,f 是 1 项集的计数,del 是删除该 1 项集的条件变量. 下面给出算法描述.

算法 1. scands_DB 算法. 扫描 N_i 段数据,收集临界频繁项的集合和它们的支持数,生成和更新全局临界频繁 1 项集集合. 在 N_1 段,初始 f_list 为空.

输入:N_i 段数据;N_{i-1} 段的 f_list($i>1$ 时);

输出:N_i 段的 f_list.

① 对每一个新到的数据流元素 e_i,if $e_i \in f\_list$ then

② $e_i.f = e_i.f + 1$,$e_i.del = e_i.del + 1$;

③ else 插入 e_i,令 $e_i.f = 1$,$e_i.del = 1$;

④ f_list 按支持数降序排列;

⑤ for each $e_i \in f\_list$ {

⑥ $e_i.del = e_i.del - 1$;

⑦ if $e_i.del = 0$ then 删除 e_i.}

定理 2. 按上述方法从 f_list 中删除的 1 项集一定是非频繁项集,f_list 是一个临界频繁 1 项集的数据集合.

证明. 任意 1 项集 $a \in f\_list$,由于每段内 del 值减 1,即 $a.del = a.del - 1$,设到第 N_i 段,总共被减去 k 次,则 $k \leqslant i$,而 $a.f \leqslant a.del + k$,所以,$a.f \leqslant a.del + i$,在第 N_i 段内被删除的条件是 $a.del = 0$,此时,$a.f \leqslant i$,又因为 $\varepsilon|N| = \varepsilon \times i \times |N_i| = \varepsilon \times i \times \lceil 1/\varepsilon \rceil \geqslant i$,所以 $a.f \leqslant \varepsilon|N|$,即 a 是非频繁项集,所以,f_list 是一个临界频繁 1 项集的数据集合.

如果 1 项集是非频繁项集,那么它的超集一定是非频繁项集,再由定理 2 可知:FP-DS 算法中从 f_list 中删除 1 项集不会导致频繁多项集的丢失.

定理 3. f_list 的最大存储空间为 $O(L/\varepsilon)$. 其中 L 为事务的平均长度.

要证明定理 3,需要用到定理 4. 下面给出定理 4.

定理 4. 设 a 是任意 1 项集,$I(DS,\varepsilon) = \{a \in I$ 且 $f_{DS}(a) > \varepsilon|DS|\}$,则 $|I(DS,\varepsilon)| \leqslant L \times \lceil 1/\varepsilon \rceil$.

证明. 用反证法,因为 $I(DS,\varepsilon)$ 中数据项的支持数都大于 $\varepsilon|DS|$,所以 $|I(DS,\varepsilon)| \times \varepsilon|DS| \leqslant L \times |DS|$,假设 $|I(DS,\varepsilon)| > L \times \lceil 1/\varepsilon \rceil$ 成立,则 $|I(DS,\varepsilon)| \times \varepsilon|DS| > L \times \lceil 1/\varepsilon \rceil \times \varepsilon|DS| \geqslant L \times 1/\varepsilon \times \varepsilon|DS| = L \times |DS|$,即 $|I(DS,\varepsilon)| \times \varepsilon|DS| > L \times |DS|$,这与已得到的结论 $|I(DS,\varepsilon)| \times \varepsilon|DS| \leqslant L \times |DS|$ 相矛盾,命题得证. 证毕.

根据定理 4,需要保存的 1 项集元素个数不超过 $L \times \lceil 1/\varepsilon \rceil$ 个,也就是说,当允许误差为 ε 时,按上述方法计算所有的 1 项集,至多仅需要空间 $O(L/\varepsilon)$,定理 3 得证. 定理 3 给出了全局 1 项集 f_list 的最大耗费空间. f_list 的平均长度与数据流的事务数和项目的数量没有任何关系,只与 ε 和事务平均长度有关. 一般来说,事务的平均长度是固定的(通常 30 以下),因此,即使 ε 较小(如 ε 取 0.0001),保存全局 1 项集 f_list 也不需要很大的存储空间.

3.2 FP-DS 树的生成和更新

定义 4. 由 FP-DS 树的某一结点 e 出发到树根所形成的路径上各个结点所组成的项集称为 e 的一个模式,简称为模式 e. 所有的模式 e 组成的集合称为 e 的模式集.

FP-DS 树压缩存储全局临界频繁集. 每个分段新生成一棵 FP-DS 树,同时删除上一个分段的 FP-DS树. FP-DS 树的形成与 FP-tree 类似. 创建树的根部(root)和头表,将所有临界频繁项集插入到树中,插入第 1 个临界频繁项集可以得到树的第 1 个分枝. 只有那些在 f_list 中的项才会被选中. 结点的排列顺序按照 f_list 的顺序进行排列,对于第 2 个临界频繁项集,若与已有的分枝有共同的前缀,将它们的计数 f 分别加该临界频繁项集的计数,并将没有的项目扩展到新的分枝,如此反复,直到插入了所有临界频繁项集. 为了便于对树的遍历,建立树的头表,使得头表中的每个项通过结点链指向它在树中的结点.

在生成第 i 段的 FP-DS 树时,临界频繁项集即包括本段内由改进的 FP-growth 算法生成的临界频繁闭合项集,也包括由第 $i-1$ 段的 FP-DS 树保存的临界频繁项集.

算法 2. reconstruct 算法. 生成 N_i 段的 FP-DS 树.

输入:N_i 段数据,N_{i-1} 段的 FP-DS 树($i>1$ 时),允许误差 ε;

输出:N_i 段的 FP-DS 树.

① 生成 N_i 段的初始 FP-DS 树,包括头表和一个根结点,头表根据 N_i 段的 f_list 生成;

② if($i>1$) {

③ 调用 $insertconstruct$ 函数;/* 将第 $i-1$ 段

的 FP-DS 树保存的临界频繁项集插入到第 i 段的 FP-DS 树中 */

④ 删除第 $i-1$ 段的 FP-DS 树;}

⑤ 以 ε 为支持度,调用改进的 FP-growth 算法生成本段的临界频繁闭合项集,对每个项集都按照 f_list 的顺序进行排序,去除那些在 f_list 中不存在的项,然后,插入到 N_i 段的 FP-DS 树中;

⑥ 释放 FP-growth 算法占用的存储空间.

insertconstruct 函数描述如下:

① 按 N_{i-1} 段 FP-DS 树头表的逆序取项目 e_i;

② for each e_i {

③ 取得 FP-DS 树中 e_i 的模式集,其中的模式分别是模式 e_{i1},模式 e_{i2},模式 e_{i3},…;

④ for each e_{ij},if ($e_{ij}.reval \neq 0$) then {

⑤ if ($e_i \in N_i$ 段 f_list)

⑥ 按 N_i 段的 f_list 排序 e_{ij} 模式中的各项目,去除不包含在 f_list 中的项目,插入到 N_i 段的 FP-DS树中;}

⑦ e_{ij} 模式中各结点的 *reval* 值减去 $e_{ij}.reval$.}}

从算法的实现过程可以看出,每个分段内只保留了临界频繁项集,根据定理 1,可以输出所有满足定义 2 的频繁项集.

3.3 FP-DS 树的剪枝

FP-DS 树剪枝的目的是在允许的误差范围内删除尽可能多的结点,以减少数据的存储量. 剪枝沿着从树叶到树根的次序. 根据头表的逆序,在结点链的帮助下,从 FP-DS 树中找到对应的所有叶结点,按照支持数从小到大的次序删除叶结点,删除的累计支持数小于该 1 项集的 f 减去 del 的值. 从支持数小的结点开始删除,可以删掉更多的结点.

定理 5. 按上述方法对 FP-DS 树剪枝,所有频繁项集支持度与其真实支持度误差小于或等于 ε.

证明. 设任意 1 项集 a,设到目前为止数据流的段数为 i,a 的支持数总共减少了 k,则有 $a.f + k = a.del + i$,又因为 $a.f \geqslant a.del$,所以,$k \leqslant i$. 设 a 所对应的超集为 X,它的真实支持数为 $f(X)$,到目前为止,它的支持数和减少的支持数分别为 $f_e(X)$ 和 $k1$,则有 $f(X) = f_e(X) + k1$. 因为 $a \in X$,所以 $k1 \leqslant k \leqslant i$,故 $f(X) - f_e(X) = k1 \leqslant i \leqslant \varepsilon \times i \times \lceil 1/\varepsilon \rceil = \varepsilon |N|$,即 $f(X) - f_e(X) \leqslant \varepsilon |N|$,命题得证. 证毕.

根据定理 5,可以得出结论:我们提出的剪枝算法是正确的.

3.4 频繁项集的生成

频繁项集的生成算法与 *insertconstruct* 函数类似,前者处理的是项目的计数 f,后者处理的是项目的 *reval* 值. 具体描述如下.

算法 3. dsgrowth 算法.

输入: N_i 段 FP-DS 树,支持度 S(设对应的支持数为 SF);

输出: 频繁项集.

① 按 N_i 段 FP-DS 树头表的逆序取项目 e_i;

② for each e_i {

③ 取得 FP-DS 树中 e_i 的模式集,其中的模式分别是 e_{i1} 模式,e_{i2} 模式,e_{i3} 模式,…

④ for each e_{ij} {

⑤ if ($e_{ij}.f > SF$)

⑥ 输出 e_{ij} 模式和它的计数;

⑦ if ($e_{ij}.f \neq 0$) then

⑧ e_{ij} 模式中各结点的 f 值减去 $e_{ij}.f$.} }

3.5 完整的 FP-DS 算法

完整的 FP-DS 算法描述如下:

算法 4. FP-DS 算法. 使用 FP-DS 树,挖掘频繁模式.

输入: 数据流 DS,最小支持度阈值 S,允许误差 ε;

输出: 频繁模式的完全集.

① 数据流分段. 数据流分段 N_i($i = 1, 2, \cdots$),每段长度 $|N_i| = \lceil 1/\varepsilon \rceil$ 或 $\lceil 1/\varepsilon \rceil$ 的整数倍. 下面的算法描述中,取 $|N_i| = \lceil 1/\varepsilon \rceil$.

②for($i = 1$;$i \leqslant$数据流的段数;i ++) {

③ 调用 scands_DB,生成 N_i 段的 f_list.

④ 调用 reconstruct,生成 N_i 段的 FP-DS 树.

⑤ 每隔若干段,调用剪枝算法,降低数据的存储量.

⑥ 调用 dsgrowth,输出频繁项集. }

当数据流的分段是 $\lceil 1/\varepsilon \rceil$ 的整数倍时(设 n 倍),则第 1 遍扫描 N_i 后,f_list 中每个 1 项集 $e_j.del = e_j.del - n$,当 $e_j.del = 0$ 时,删除 e_j. 其他部分与上述算法描述相同. 实际上,第 $i-1$ 分段输出频繁项集和向第 i 分段的 FP-DS 树中插入临界频繁项集可以同时进行,这样第 $i-1$ 分段的 FP-DS 树只需遍历 1 次,进一步提高了算法的效率.

4 性能分析和实验研究

本文采用的数据流是由 IBM 合成数据发生器产生的顾客购物数据. 实验采用了 3 套数据 T15I10D1000K,T15I6D1000K 和 T7I4D3000K,其中 $|T|$ 表示数据集中事务的平均长度,$|I|$ 表示潜在频繁项集的平均长度,$|D|$ 表示总的事务数目,T7I43000K 有 1K 个不同项目,其他两套数据有 10K 个不同项目,数据发生器的其他参数采用缺省值. 数据流分段,每段包含 50000 条事务. 设支持度为 S,取允许误差 $\varepsilon = 0.1 \times S$. 实验主要研究了算法的时间效率和空间效率,并和其他算法做了实验对比.

4.1 算法性能分析

实验环境为 Intel 赛扬 1GHz CPU、内存 128MB、Redhat 9.0 操作系统. 采用数据集 T15I10D1000K,支持度 S 分别取 0.007,0.005 和 0.004,观察算法的时间效率和空间效率. 图中的最大存储空间是指处理每个数据分段时算法需要的最大存储空间(也包括辅助内存和临时内存),该数据由 Linux 的系统监控器得到 运行时间是指每个数据分段的运行时间(不含从外部设备读入数据的时间).

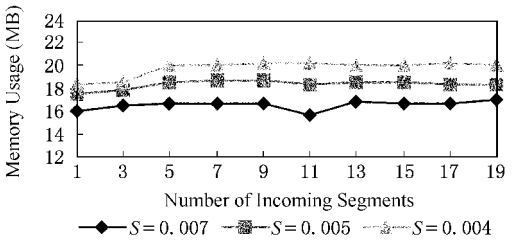

Fig. 2 Maximal of incoming segment.

图 2 每个数据分段最大存储空间

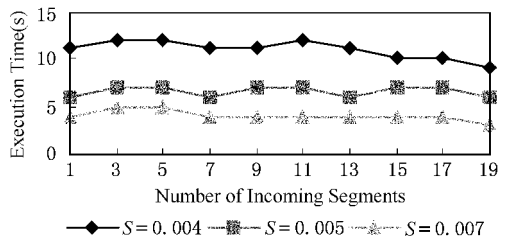

Fig. 3 Execution time in each segment.

图 3 每个数据分段运行时间

从图 2 和图 3 可以看出,随着支持度 S 的下降,最大存储空间和每段运行时间都逐渐增加. 但是,当支持度一定时,最大存储空间和每段运行时间都逐渐趋于稳定,并不会随着段数的增加而不断增加,也就是说与数据集的大小无关,因此,算法可以应用于数据流中. 当支持度 S 较小时,支持度的改变对运行时间的影响比较显著,主要原因是算法的最大时间消耗是每个数据分段调用改进的 FP-growth 算法生成临界频繁闭合项集,使用 FP-growth 算法挖掘频繁闭合模式时,需要递归地生成条件 FP 树,并且每产生一个频繁模式就要生成一个条件 FP 树,当支持度 S 较小时,允许误差 ε 更小(本文取 $\varepsilon = 0.1S$),FP-growth 算法以很小的 ε 为支持度生成临界频繁闭合项集,将产生大量的频繁模式,动态的生成和释放这些条件 FP 树将耗费大量时间.

4.2 实验对比分析

文献[2]的算法是一种典型的数据流频繁模式挖掘方法. 我们采用与文献[2]相同的实验环境和实验数据集对算法进行测试,并将运行结果与文献[2]的运行结果进行对比分析. 实验数据集是 T15I6D1000K,在支持度从 0.004～0.01 之间变化时,FP-DS 算法的最大存储空间是从 17.8MB～21.7MB(含辅助内存和临时内存). 文献[2]算法的运行时间会随着存储空间的增大而降低,我们选择内存消耗为 28MB 的运行时间与我们的算法进行比较,如图 4 所示. 图 4 中 Algorithm Ⅰ 和 Algorithm Ⅱ 分别代表 FP-DS 算法和文献[2]提出的算法. 可以看出,FP-DS 算法即使在消耗较小的空间的情况下,其时间效率也高于文献[2]的算法. FP-DS 算法的主要时间和空间开销是每段调用 FP-growth 算法的时间开销和临时空间开销,如果采用更有效的改进 FP-growth 算法可以进一步提高 FP-DS 算法的时间和空间效率.

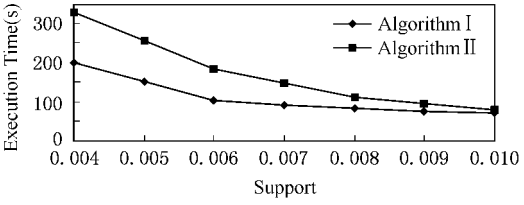

Fig. 4 Comparison of execution time.

图 4 运行时间的比较

FP-DS 算法和文献[1]提出的算法都是以 FP-growth 算法为基础的,所以,我们也采用了与文献[1]相近的实验环境和相同的实验数据集进行算法的对比分析. 以 T7I4D3000K 作为测试集,支持度为 0.005. 文献[1]的方法比较适合很短的项集,采

用上述测试集,每秒钟只能处理约 180 条交易(见文献[1]实验部分),显然,在高速数据流环境下,这样的处理速度是远远不够的,而 FP-DS 算法平均每秒可以处理 15000 条交易. 我们也比较了每段 FP-stream 树和 FP-DS 树的最大存储空间. 在每个数据分段,FP-stream 树的最大存储空间约为 2.5MB,FP-DS 树的最大存储空间约为 0.9MB,原因是 FP-stream 树需要存储每一个临界频繁项集和它的各个子集,而 FP-DS 树则不必存储临界频繁项集的各个子集,当频繁项集的平均长度更大时,FP-DS 树将比 FP-stream 节省更多的存储空间. 由上述分析可以看出,与文献[1]的方法相比,FP-DS 算法有更好的时间和空间效率.

5 结束语

本文提出了一种有效的数据流频繁模式挖掘算法:FP-DS 算法,用户可以连续在线获得当前所有的频繁项集,不存在模式延迟现象,与已有的相关算法相比,FP-DS 算法尤其适合长频繁项集的挖掘. FP-DS 算法即不需要列举交易的每个子集,也不需要产生大量的频繁候选项,大大提高了算法的效率,特别是存在大量长频繁项集时,算法效率的提高更为明显. 在 FP-DS 算法中,我们提出了一种 FP-DS 树数据结构来存储潜在的频繁项集,每个数据分段只存储了潜在频繁闭合项集,不需要独立存储项集的各个子集,极大地减少了项集的存储量,而且,项集按全局 1 项集的支持数降序排列,使得出现频率越高的项目越靠近树根,这样的压缩树有更高的压缩率. 实验表明,FP-DS 算法具有较好的时间和空间效率.

参 考 文 献

1 C. Giannella, J. Han, J. Pei, *et al*. Mining frequent patterns in data streams at multiple time granularities. In: H. Kargupta, A. Joshi, K. Sivakumar, eds. Next Generation Data Mining. Cambridge, Massachusetts: MIT Press, 2003. 191~212

2 G. S. Manku, R. Motwani. Approximate frequency counts over streaming data. The 28th Int'l Conf. Very Large Data Bases (VLDB 2002), Hong Kong, 2002

3 Song Guojie, Wang Tengjiao, Tang Shiwei, *et al*. Estimation and maintenance of frequent pattern in data streams. National Data Base Conf. 2003, Changsha, 2003 (in Chinese)
(宋国杰,王腾蛟,唐世渭,等. 数据流中频繁模式的评估与维护. 第 20 届全国数据库学术会议,长沙,2003)

4 R. M. Karp, C. H. Papadimitriou, S. Shenker. A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Systems, 2003, 28(1): 51~55

5 M. Charikar, K. Chen, M. Farach-Colton. Finding frequent items in data streams. The 29th Int'l Colloquium on Automata, Languages and Programming, Malaga, Spain, 2002

6 Joong Hyuk Chang, Won Suk Lee. Finding recent frequent itemsets adaptively over online data streams. The 9th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD 03), Washington, D. C., 2003

7 Wei-Guang Teng, Ming-Syan Chen, Philip S. Yu. A regression-based temporal pattern mining scheme for data streams. The Int'l Conf. Very Large Data Bases, Berlin, Germany, 2003

8 Graham Cormode, Flip Korn, S. Muthukrishnan, *et al*. Finding hierarchical heavy hitters in data streams. The Int'l Conf. Very Large Data Bases (VLDB) 2003, Berlin, Germany, 2003

9 Tatsuya Asai, Hiroki Arimura, Kenji Abe, *et al*. Online algorithms for mining semi-structured data stream. The IEEE Int'l Conf. Data Mining (ICDM) 2002, Maebashi City, Japan, 2002

10 Graham Cormode, S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. The ACM Symposium on Principles of Database Systems (PODS) 2003, San Diego, CA, USA, 2003

11 Cheqing Jin, Weining Qian, Chaofeng Sha, *et al*. Dynamically maintaining frequent items over a data stream. The Conf. Information and Knowledge Management (CIKM) 2003, New Orleans, Louisiana, USA, 2003

12 Jiawei Han, Jian Pei, Yiwen Yin. Mining frequent patterns without candidate generation. The 2000 ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '00), Dallas, USA, 2000

Liu Xuejun, born in 1971. He is a Ph. D. candidate. His main research interests include data stream management and data mining.
刘学军,1971 年生,博士研究生,主要研究方向为数据流管理、数据挖掘等.

Xu Hongbing, born in 1947. He is a professor and master supervisor. His main research interests include database theory, semantic web and real-time data processing.
徐宏炳,1947 年生,教授,硕士生导师,主要研究方向为数据库、语义 Web 和实时数据采集处理等.

Dong Yisheng, born in 1940. He is a professor and Ph. D. supervisor. His main research interests include database theory, information system and software engineering.
董逸生,1940 年生,教授,博士生导师,主要研究方向为数据库、信息系统和软件工程等.

Wang Yongli, born in 1974. He is a Ph. D. candidate. His main research interests include data stream management, information system.

王永利,1974 年生,博士研究生,主要研究方向为数据流管理、信息系统等.

Qian Jiangbo, born in 1974. He is a Ph. D. candidate. His main research interest includes data stream management and hardware-software codesign.

钱江波,1974 年生,博士研究生,主要研究方向为数据流管理、软硬件协同设计等.

Research Background

In recent years, data stream mining has become a new research issue. To find the frequent items over data stream is one of the fundamental problems in this field, which has found application in many problem domains, such as network traffic measurements, data mining, web-server logs analysis, telecom call records analysis, sensor network and so on.

But the peculiarity of data stream makes it difficult to apply the traditional frequent items mining algorithm and the frequent pattern mining based on data stream is an arduous challenge. In this paper, inspired by the fact that the FP-growth provides an effective algorithm for frequent pattern mining, a new FP-DS algorithm for mining frequent patterns from data streams is proposed. Compared with the existing related algorithms, there is not a pattern-delay, and the algorithm efficiency doesn't descend dramatically with the growth of length of frequent pattern. Consequently, it is especially suitable for the mining of long frequent itemsets over data streams. Our work also widens the application field of FP-growth algorithm.

Our work is supported by the High Tech Project of Jiangsu Province of China (BG2004034), and the Graduate Student Innovation Project of Jiangsu Province of China (xm04-36).