窗口模式下在线数据流中频繁项集的挖掘 *

李 可, 任家东
(燕山大学 信息科学与工程学院, 河北 秦皇岛 066004)

摘 要: 拟采用一种基于滑动窗模式的单遍挖掘算法,专注于处理近期数据;为了减少处理时间和占用的内存,设计了一种新的事务表示方法。通过处理这个事务的表达式,频繁项集可以被高效输出,并解决了使用基于 Apriori 理论的算法时,由候选频繁 1-项集生成频繁 2-项集时数据项顺序判断不准确问题。该算法称为 MRFI-SW 算法。

关键词: 在线数据流; 频繁项集; 滑动窗
中图分类号: TP311.13 **文献标志码**: A **文章编号**: 1001-3695(2010)05-1711-03
doi:10.3969/j.issn.1001-3695.2010.05.029

Online data stream mining of recent frequent itemsets based on sliding window model

LI Ke, REN Jia-dong
(*College of Information Science & Engineering, Yanshan University, Qinhuangdao Hebei* 066004, *China*)

Abstract: This paper proposed a one-pass data stream mining algorithm to mine the recent frequent itemsets in data streams with a sliding window based on transactions. To reduce the cost of time and memory needed to slide the windows, denoted each items a bit-sequence representations. Basing on dealing with the representations, can find frequent patterns in data streams efficiently, and the sequent of frequent 2-items is correct. This paper named the method MRFI-SW(mining recent frequent itemsets by sliding window) algorithm.

Key words: online data stream; frequent items; sliding window

数据流可以分为离线数据流(offline data stream)[1]和在线数据流[2]。离线数据流处理的数据是成块到来的,应用在如数据仓库等系统中;在线数据流的特点是实时地更新,在线数据流中流数据的到来是连续的,如实时产生事务的网络监控系统。挖掘频繁模式最著名的算法是 Apriori[3] 算法,但是它需要多遍扫描数据集,不适应数据流环境。很多算法的提出致力于降低扫描数据集的次数,如 Partition[4] 算法;Manku 等人提出了两种单遍扫描的算法,即 sticky sampling 和 lossy counting,在离线数据流中挖掘频繁模式;Chang 等人[4] 提出一种基于 BTS 的算法;SWFI-stream[5] 算法是基于事务滑动窗口模式[6] 的在线数据流中挖掘频繁项集的算法。

针对在线数据流挖掘近期频繁项集的问题,提出一种基于对事务敏感的滑动窗模式的算法 MRFI-SW(mining recent frequent itemsets over online data stream with sliding window),设计了一种新的事务表示方法,这种表示方法可以记录数据项在事务中的支持计数和位置。使用基于 Apriori 理论的算法,通过处理事务的表达式,频繁项集可以被高效输出,并解决了使用基于 Apriori 理论的算法时,由候选频繁 1-项集生成频繁 2-项集时数据项顺序出错的问题。

1 问题定义

$\Psi=\{i_1,i_2,i_3,\cdots,i_n\}$ 为一个数据项的集合,事务是一系列数据项的集合,表示为 $T=\{\mathrm{id},x_1,x_2,\cdots,x_n\}$。其中:$x_i\in\Psi$ $(1\leqslant i\leqslant n)$,id 是事务的惟一标志符。$DS=\{T_1,T_2,\cdots,T_N\}$ 是一个连续的事务数据流,N 表示数据流大小和最后一个事务 T_N 的 id 值。这个数据流也可以表示为 $DS=\{W_1,W_2,\cdots,W_m\}$,这里 W_i 表示为一个对事务敏感的滑动窗口。每个窗口内包含固定数目的事务,w 是窗口内事务的数目。所以当前的窗口可以表示为 $SW_{N-w+1}=[T_{N-w+1},T_{N-w+2},\cdots,T_N]$,$N-w+1$ 是当前窗口的 id 号。一个事务 T 在 SW 的支持度是在 SW 中包含 T 作为子集的事务的数目,如果 T 的支持度大于 $s\times w$,则称 T 是频繁项集(FI),这里 s 是用户定义的最小支持阈($s\in[0,1]$)。

2 MRFI-SW 算法

2.1 项集表示方法 bit-order 序列的设计

本算法为每个包含于当前窗口内事务中的数据项都构造一个序列形式的表示结构,这个序列结构的长度为 w,即窗口内的事务数目。序列内的每个元组的形式为(bit,order),用 $R(x)$ 来表示。如果一个数据项 x 出现在窗口内第 i 个事务中,则表示该数据项的 $R(x)$ 内的第 i 个元组的 bit 域置 1;order 域表示的是 x 在它所在的事务中的位置,若该数据项没有出现在这个事务中,则 $R(x)$ 的该个元组为 0。这个为每个事务中的

收稿日期: 2009-09-15; **修回日期**: 2009-10-26 **基金项目**: 国家"863"计划资助项目(2009AA01Z433);河北省自然科学基金资助项目(F2008000888)

作者简介:李可(1983-),女,黑龙江齐齐哈尔人,助理实验师,硕士研究生,主要研究方向为数据挖掘(like19830319@gmail.com);任家东(1964-),男,教授,博导,主要研究方向为数据挖掘、时态数据建模、软件安全技术.

数据项创建表示序列的过程被称为 bit-order 变形。

下面用一个实例说明这种数据项的表示方法。假设一个数据流的前四项为〈T_1,(acd)〉,〈T_2,(bce)〉,〈T_3,($abce$)〉和〈T_4,(be)〉,T_1、T_2、T_3 和 T_4 是事务(项集),a、b、c、d 和 e 是数据项。使滑动窗大小为 3,所以这个事务流包含两个滑动窗 $SW_1=[T_1,T_2,T_3]$ 和 $SW_2=[T_2,T_3,T_4]$。在 SW_1 中,数据项 a 出现在 T_1 和 T_3 中而且都是第一个数据项,所以 a 的表示序列 $R(a)=\langle(1,1),0,(1,1)\rangle$。同理 $R(c)=\langle(1,2),(1,2),(1,3)\rangle$,$R(d)=\langle(1,3),0,(0,0)\rangle$,$R(b)=\langle0,(1,1),(1,2)\rangle$,$R(e)=\langle0,(1,3),(1,4)\rangle$。对于一个 k 项式,bit-order 序列就只有一个表示存在的域,如 $R(bc)=\langle1,1,0\rangle$。

2.2 MRFI-SW 算法的设计

MRFI-SW 算法包含三个阶段:窗口初始化阶段(window initialization phase);窗口滑动阶段(window sliding phase);频繁项集产生阶段(mining frequent itemsets phase)。

1)窗口初始化阶段

这个阶段的条件是窗口未满,当窗口内事务数目少于事先设定的窗口大小 w 时,就一直处于窗口初始化阶段。当滑动窗口满的时候,窗口中的 w 个事务分别表示成 bit-order 形式。

2)窗口滑动阶段

窗口滑动阶段被激活的条件是窗口被事务充满,然后执行对事务的 bit-order 变形操作。新事务到来时被插入窗口中,而最先到来的事务被从窗口中移除。为了完成新事务的插入和旧事务的移除,对于 bit-order 表示,算法设计了一种简单却高效的处理。因为一个事务(数据项)的 bit-order 表示是序列的结构,对它执行左移操作。最早的那个事务的元组被从 bit-order 结构中移除,新到事务的 bit-order 形式被插入序列尾部。为了提高内存使用效率,一个删除序列的操作在窗口滑动后进行,即当一个数据项的 bit-order 形式为 0 时,就删除该序列。如果数据项 x 在窗口滑动并且更新完 bit-order 序列后,所有事务都不包含 x,则对 x 的 bit-order 序列 $R(x)$ 进行删除。

第一阶段窗口初始化和第二阶段窗口滑动算法如下:

算法:Initialize windows and slide windows。

```
输入:数据流 DS, 最小支持阈 s,窗口大小 w。
输出: updated bit-order sequence//更新后的 bit-order 序列。
initialize sliding window and bit-order sequence//窗口和序列初始化
while each new coming transaction T_i in SW do//新事务到来
if(SW is full)
   transform all of items in SW to bit-order sequence//bit-order 变形
else
   do left_shift operation on bit-order sequence of all items
   //对 bit-order 序列左移
   for each item X arrives in SW
      transform X to bit sequence representation
   end for
end if
for each R(X) in SW
if sum(R(X).bit) =0//删除不存在的数据项的元组
   drop X from SW
end if
end for
```

3)频繁项集产生阶段

频繁项集产生阶段当 bit-order 序列更新完毕和用户查询频繁的时候被激活。算法根据 bit-order 序列的结构特点,使用一种基于 Apriori 理论的算法挖掘频繁项集,即在 k-1 阶频繁项集已知的情况下挖掘 k 阶频繁项集,而 Apriori 理论的精髓就是如果一个模式是频繁的,那么它的子模式都是频繁的。

算法对每个数据项的 bit-order 序列的每个元组逐位相加(sum operation),查找出当前窗口内的频繁 1-项集,然后对每个元组的对应 bit 位进行“与”(and)操作,这样得出的是 2-项集,再把各位进行“加”(sum)操作,找出频繁 2-项集,非频繁数据项的元组就可以删除了。这个过程一直进行到没有新的频繁 $k+1$ 项集产生为止。

第三阶段频繁项集产生的算法如下:

算法:Frequent itemset outputting。

```
输入:updated bit-order sequence, 最小支持阈 s。
输出:a set of frequent itemsets//频繁项集。
find frequent 1-itemsets FI_1//查找出频繁 1-项集
for(k =2;FI_{k-1} ≠ null; k ++)
   do and operation on R(FI_{k-1}).bit to find candidate FI_k
   /* 对频繁k-1-项集的 bit-order 序列的个位进行与操作,找出候选频繁 k-项集 */
   for each FI do
   do bitwise sum operation on R(candidate FI_k)//查找频繁项集
     if sum(R(candidate FI_k).bit ) ≥ s * w
       if k =2
      scan R(candidate FI_k).order //调整项集内数据项的位置
      output FI_k
      end if
    end if
   end for
```

2.3 实例

用一个实例来说明 MRFI-SW 算法。在窗口被事务充满时,每到来一个事务都要被执行 bit-order 变形操作。例如表 1 所示的事务流,在第一个滑动窗 SW_1 中有三个事务:T_1,T_2 和 T_3。滑动窗 SW_1 中数据项的 bit-order 序列表达形式如表 1 示。

表 1 数据项的 bit-order 序列

数据流中的事务	滑动窗口	事务	数据项的 bit-order 序列
〈T_1,(acd)〉	SW_1	〈T_1,(acd)〉	$R(a)=\langle(1,1),0,(1,1)\rangle$
〈T_2,(bce)〉		〈T_2,(bce)〉	$R(c)=\langle(1,2),(1,2),(1,3)\rangle$
〈T_3,($abce$)〉		〈T_3,(abce)〉	$R(d)=\langle(1,3),0,0\rangle$
〈T_4,(be)〉			$R(b)=\langle0,(1,1),(1,2)\rangle$
			$R(e)=\langle0,(1,3),(1,4)\rangle$
	SW_2	〈T_2,(bce)〉	$R(a)=\langle0,(1,1),0\rangle$
		〈T_3,($abce$)〉	$R(c)=\langle(1,2),(1,3),0\rangle$
		〈T_4,(be)〉	$R(d)=\langle0,0,0\rangle$
			$R(b)=\langle(1,1),(1,2),(1,1)\rangle$
			$R(e)=\langle(1,3),(1,4),(1,2)\rangle$

当 T_4 到来时,T_1 要从当前的窗口中移除。SW_1 中的 bit-order 序列的每个元组需要向左移。$R(a)$ 由〈(1,1),0,(1,1)〉变为〈0,(1,1),0〉。同样,$R(c)$、$R(d)$、$R(b)$ 和 $R(e)$ 分别变为〈(1,2),(1,3),0〉,〈0,0,0〉,〈(1,1),(1,2),(1,1)〉和〈(1,3),(1,4),(1,2)〉。数据项 d 被移除,$R(d)$ 为〈0,0,0〉,它的支持度为 0。

在第三阶段,由于用户定义的最小支持阈为 0.6 且每个窗口中有三个事务,项集称为频繁的条件是窗口内的支持计数不少于 1.8。下面介绍在 SW_2 中挖掘频繁项集的具体过程,如图 1 所示。

根据数据项的 bit-order 序列 $R(a)=\langle0,(1,1),0\rangle$、$R(c)=\langle(1,2),(1,3),0\rangle$、$R(b)=\langle(1,1),(1,2),(1,1)\rangle$、$R(e)=\langle(1,3),(1,4),(1,2)\rangle$,各个数据项的支持度计数 sup($a$)=1、sup($c$)=2、sup($b$)=3、sup($e$)=3,得到频繁 1-项集,只有数据项 a 不是频繁的;然后整合频繁 1-项集并同时扫描每个 bit-order 元组的 order 位,产生频繁 2-项集。扫描 order 位的目的是确定数据项在 2-项集中的顺序(使用 Apriori 理论时,频繁项是 ad 还是 ba 往往确定不了,只能确定频繁模式中含有 a

和 b)。候选频繁 2-项集为 bc、be 和 ce。这些频繁 2-项集的 bit-order 序列的 bit 位执行 sum 操作,$R(bc)$ 支持度为 2,同样 be 和 ce 的计数分别为 3 和 2。然后算法仍然基于 Apriori 理论产生候选频繁 3-项集,$R(bc)$ and $R(be)$ and $R(ce) = R(bce) = \langle 1,1,0\rangle$,bce 的支持度为 2,它为频繁项。此时没有新的候选项集产生,MRFI-SW 算法终止。SW_2 中一共有六个频繁项集 c、b、bc、be、ce 和 bce。

SW_2 中的事务	bit-order 序列	sup
$\langle T_2,(bce)\rangle$	$R(a)=\langle 0,(1,1),0\rangle$	1
$\langle T_3,(abce)\rangle$	$R(c)=\langle (1,2),(1,3),0\rangle$	2
$\langle T_4,(be)\rangle$	$R(b)=\langle (1,1),(1,2),(1,1)\rangle$	3
	$R(e)=\langle (1,3),(1,4),(1,2)\rangle$	3

FI_1	候选 FI_2	sup
c	cb:$R(c)$. bit and $R(b)$. bit $=\langle 1,1,0\rangle$	2
b	be:$R(b)$. bit and $R(e)$. bit $=\langle 1,1,1\rangle$	3
e	ce:$R(c)$. bit and $R(e)$. bit $=\langle 1,1,1\rangle$	3

FI_2	候选 FI_3	sup	FI_3
bc	bce:$R(bc)$ and $R(be)$	2	bce
be	and $R(ce)=\langle 1,1,0\rangle$		
ce			

图 1　SW_2 中频繁项产生的过程

3　算法性能分析和实验结果

本文使用 Agrawal 等人提出的 IBM Synthetic data generator 生成的数据,具体生成的方法同文献[3]。这个理论上的数据流表示为 T5I4D1000K(T 表示事务的平均长度;I 表示事务项的平均长度;D 表示事务的总数),把这些数据看成是持续地实时地到来的。最小支持阈值设置为 0.000 1,基于事务的滑动窗口大小为 20 KB。

由于本算法是基于对事务敏感的滑动窗口模式在在线数据流中挖掘频繁项集的算法,主要是通过它与 SWFI-stream 算法性能对比来分析。图 2 所示 MRFI-SW 算法在窗口初始化阶段仅需内存 2.1 MB,而 SWFI-stream 算法需要 11.2 MB ~ 109.7 MB。图 3 为窗口滑动阶段的内存使用情况,MRFI-SW 算法的内存需要 2.1 MB,而 SWFI-stream 算法的内存使用在 109.7 MB ~ 120.3 MB。图 4 所示是频繁项集产生阶段的内存使用情况比较,MRFI-SW 算法使用的内存在 33.5 MB ~ 39 MB,由图可见 SWFI-stream 算法的内存使用远远要高于这个区间。由于 MRFI-SW 算法是挖掘近期频繁项集的单遍算法,而且挖掘过程只处理项集的表示序列便可,内存使用上比 SWFI-stream 算法小得多。

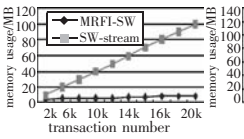

图 2　窗口初始化阶段内存使用情况比较

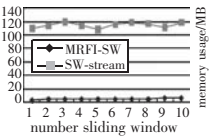

图 3　窗口滑动阶段内存使用情况比较

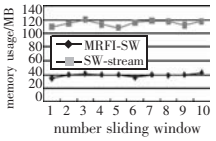

图 4　挖掘频繁模式阶段内存使用情况

4　结束语

对在线数据流的挖掘是一项应用广泛且具有挑战性的工作[7,8],在线数据流的特点使得传统的挖掘算法甚至离线数据流挖掘算法都不适合应用其中。本文提出了一种分为三个阶段的基于对事务敏感的滑动窗口模式的在在线数据流中挖掘近期频繁项集的算法,即 MRFI-SW 算法。为了适应在线数据流实时更新的特点和用户实时地查询,更是设计了一种对频繁数据项和频繁项集的序列结构表示方法,对它进行简单的移动和各位相加操作就可以实现动态更新和查找频繁模式的工作。实验表明,MRFI-SW 算法不仅在挖掘结果上获得了较好的精度,而且在内存利用率方面优于 SWFI-stream 算法。

参考文献:

[1] MANKU G S, MOTWANI R. Approximate frequency counts over data streams[C]//Proc of the 28th International Conference on Very Large Data Bases. 2002:346-357.

[2] JIANG N, GRUENWALD L. Research issues in data stream association rule mining[J]. ACM SIGMOD Record, 2006, 35(1):14-19.

[3] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules[C]//Proc of the 20th International Conference on Very Large Data Bases. 1994:484-499.

[4] CHANG J, LEE W. A sliding window method for finding recently frequent itemsets over online data streams[J]. Journal of Information Science and Engineering, 2004, 20(4):175-184.

[5] SAVERAERS A, OMIECINSKI E, NAVATHE S. An efficient algorithm for mining association rules in large databases[C]//Proc of the 21st International Conference on Very Large Data Bases. San Francisco: Morgan Kaufmann Publisher, 1995:432-444.

[6] LIN C H, CHIU D Y, WU Y H, *et al*. Mining frequent itemsets from data streams with a time-sensitive sliding window[C]//Proc of SIAM International Conference on Data Mining. 2005.

[7] YU J X, CHONG Z, LU H, *et al*. False positive or false negative: mining frequent itemsets from high speed transactional data streams[C]//Proc of the 30th International Conference on Very Large Data Bases. 2004:204-215.

[8] LI H F, LEE S Y, SHAN M K. An efficient algorithm for mining frequent itemsets over the entire history of data streams[C]//Proc of the 1st International Workshop on Knowledge Discovery in Data Streams. 2004:287-291.

(上接第 1701 页)进行定位,能有效地获取目标的空间位置;同时,该方法对无线网络定位、移动通信定位等工程问题也具有一定的应用价值。

参考文献:

[1] 刘利军,韩焱. 基于最小二乘法的牛顿迭代信源定位方法[J]. 弹箭与制导学报,2006,26(3):325-328.

[2] 蔡昭权,黄翰. 自适应变异综合学习粒子群优化算法[J]. 计算机工程,2009,35(7):170-171.

[3] 张安玲,刘雪英. 求解非线性方程组的拟牛顿—粒子群混合算法[J]. 计算机工程与应用,2008,44(33):41-42.

[4] 彭芳,左继章,吴军. 基于高斯—牛顿法改进的复合双基地雷达目标空间定位算法[J]. 系统工程与电子技术,2009,31(3):557-575.

[5] 唐勇,王兴春. 基于粒子群优化算法的空中目标定位[J]. 指挥控制与仿真,2007,29(4):31-32.

[6] 李俊峰,高洪元,庞伟正,等. 粒子群优化算法在 TDOA 定位中的应用[J]. 应用科技,2005,32(10):7-9.

[7] van den BERGH F, ENGELBRECHT A P. A study of particle swarm optimization particle trajectories[J]. Information Sciences, 2006, 176(8):937-971.

[8] JIANG Yan, HU Tie-song, HUANG Chong-chao, *et al*. An improved practicle swarm optimization algorithm[J]. Applied Mathematics and Computation, 2007, 193(1):231-239.

[9] 柳辉. 解非线性方程组的牛顿迭代法及其应用[J]. 重庆工学院学报,2007,21(8):95-98.

[10] 李荣钧,常先英. 一种新的混合粒子群优化算法[J]. 计算机应用研究,2009,26(5):1700-1702.