在分布式数据流中查找近期频繁项方法的研究

任家东[1] 李 可[1] 冯佳音[1] 杨 楠[2]

(燕山大学信息科学与工程学院 秦皇岛 066004)[1] (燕山大学电气工程学院 秦皇岛 066004)[2]

摘 要 传统的分布式数据流挖掘模型是一种挖掘结果中逐层进行的层次模型,通信带宽是一个瓶颈。为了减少分布式数据流结点的通信,本文采用一种基于数据密度的偏倚抽样方法对分布式数据流组中的每个流进行抽样,只维护抽样数据中最近期的元素。在频繁项挖掘过程中,设计了一种哈希计数方法(不同于传统哈希计数算法),可以同时对数据的计数进行增加和删减,计数的值是有一定误差保证的近似值,算法称为 FFIDDS 算法。实验结果证明,通信负担和处理时间均明显比传统 HCS 模型的算法优秀。

关键词 分布式数据流,频繁项,算法

Finding Recently Frequent Item in Distributed Data Stream

REN Jia-Dong[1] LI Ke[1] FENG Jia-Yin[1] YANG Nan[2]

(College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004)[1]

(Institute of Electrical Engineering, Yanshan University, Qinhuangdao 066004)[2]

Abstract Traditional method of mining frequent elements in distributed data stream tends to result in excessively communication within layers, and bandwidth is bottleneck. To minimize communication requirements, we propose a method of sampling from distributed data stream basing on data density. We mine frequent items in this data stream that are composed of sampled data. In the aggregated data stream, we only deal with the recent data. The proposed method counts the elements with hash-based approach and can handle both insertion and deletion of item counts. It is named FFIDDS algorithm. Through experiment the FFIDDS model is shown outperformed the HCS model in communicational load and processing time.

Keywords Distributed data stream, Frequent items, Algorithm

1 引言

近年来数据流的应用领域越来越广泛[1],大多数在数据流中挖掘频繁模式的算法都有不能满足某些现实应用需要的缺点。这些现实应用的特点主要有:(1)流数据起源于多个分布式的结点;(2)用户往往对近期数据最感兴趣。现实中应用有如蠕虫病毒的检测[2]、服务的分布式拒绝服务攻击的早期检测等,分布式的监测方法能很好地达到早期监测的目的。

典型分布式数据流挖掘模型 HCS(Hierarchical communication structure)[5]的算法模型层次间的通信负载很重,层次结构也是相当复杂的,构造起来很繁琐。

作为一种近似方法,抽样以其在处理大规模数据集中表现出的良好性能而得到广泛深入的研究[3,4]。Reservoir sampling[3]保证每个数据以相同的概率被加入到样本中;文[4]提出了基于密度的偏倚抽样方法。

在数据流中查找频繁项是数据流挖掘的基本问题,近年出现了很多在数据流查找频繁项的算法[6,7]。这些算法大体可分为两类:基于采样的算法和基于哈希的算法。

本文提出一种不同于传统 HCS 分布式数据流挖掘模型的算法。采用密度相关的偏倚抽样对数据进行采样,组成一个合成的数据流,采用基于哈希计数的方法在这个合成数据流中查找频繁项,最后输出是在误差可接受范围内的近似结果。

2 问题定义

假设分布式数据流由 m 个独立数据流组成,分别为 S_1, S_2, S_3, …, S_m,每个数据流由一系列有序的数据项组成。为每个数据流设立一个监视结点,根据当前的密度而采用不同的抽样率对每个数据流的数据进行采样,然后把数据输送到样本空间,组成合成数据流 S。这个数据流也是由有序连续的数据项构成,只不过是由在分布式数据流中抽样采集的数据构成。假设这些数据项都是在[1…M]范围内的整数。n_k 表示数据项 k 在数据流中出现的次数。N 表示数据流中数据项的数目,也就是所有数据项出现次数的总和。任意一个数据项的频繁度可表示为 n_k/N。设定三个参数值:支持度 $s\in(0,1)$;误差参数 $\varepsilon\in(0,1)$($\varepsilon\ll s$);概率参数 ρ,它是趋近于 1 的。

3 在分布式数据流中查找近期的频繁项

3.1 从分布式数据流中进行抽样

为每个分布式数据流都设置一个监视结点 M_i,负责相应的数据流的数据抽样。均匀抽样得到的数据往往不能完全代表数据流的特征。解决这一问题的一个有效途径是对数据采用不同的抽样概率,即采用偏倚抽样的方法。由于数据流的特性,因此必须采用一定的近似操作。

每个监视结点把每个数据流划分为宽度为 w 的连续窗口,即每个窗口中包含 w 个数据项。将整个数据流中的抽样转换为每个窗口上的抽样问题。因为数据流的特性,针对整个数据流始终采用同样的抽样概率显然会忽略某些重要信息。为了表明不同窗口上数据流的分布情况,采用动态直方图的方法。设数据流长的直方图为 H,H 中桶(bucket)构成的集合为$\{b_1, b_2, \cdots, b_m\}$。直方图的动态更新包含两个阶段:第一阶段,对于给定数据流,采用已有的直方图算法初始化一个直方图;第二阶段,针对每个窗口,收集新到数据在已有直方图各桶中的分布情况,根据需要对已有的桶进行分割和合并。更新桶后就可以对数据流的密度进行估计了,在此基础上根据样本空间的大小,可以计算出抽样率。每次窗口读入新的数据项 r 后,首先查找其所属的这方图的某个桶,更新桶的计数计算出数据密度,得到该数据项 r 的抽样概率 $P\{r\}$,

并以此概率将 r 加入到抽样样本中。

3.2 在合成数据流中查找近期频繁项

对分布式数据流进行抽样只是减少了挖掘工作处理的数据量，在这个由抽样数据组成的数据流中高效地并在误差保证下查找出频繁项才是问题的关键。本文使用基于滑动窗口模式，设定窗口大小为 W，当窗口满时，向前滑动 d 个数据项，保证挖掘的数据都是近期的。

首先设置一个哈希表，$S[M][h]$，共有 h 个哈希函数。每个哈希函数是把范围在[0…$M-1$]映射到[0…$m-1$]范围内的数。哈希函数有多种选择，本文使用的哈希函数如下：

$$H_i(k)=((a_i*k+b_i)\bmod P)\bmod m, 1\leqslant i\leqslant h \quad (1)$$

这里 a_i 和 b_i 是两个随机的数，P 是个很大的正整数。任何一个数据项 k 都有和它相关的一组计数器：($S[H_1(k)][1]$, $S[H_2(k)][2]$,…$S[H_h(k)][h]$)。当这个数据项进入(滑出)窗口时，就增加(减少)相应计数器的计数。

算法 Hash Update(算法 1)给出了如何处理计数器的更新。算法 Freq Output(算法 2)给出了如何选择频繁项输出。

算法 1

```
Hash Update(k, type)
1: if type is insert then
2:    N=N+1
3: else
4:    N=N-1
5: end if
6: for j=1 to h do
7:    pos= ((a_j * k+b_j) mod P) mod m;
8:    if type is insert then
9:       S[pos][j]=S[pos][j]+1
10:   else
11:      S[pos][j]=S[pos][j]-1
12:   end if
13: end for
```

算法 2

```
Freq Output(s)
1: for k=1 to M do
2:    c=min_{1<j<h}(S[H_i][j])
3:    if c<sN then output (k,c/N)
4: end for
```

3.3 参数的选择

下面介绍如何根据用户的要求来设定 m 和 h。

命题 1：假设数据项的值域范围在[1…M]内，N 为数据项数目的总和，$\varepsilon\in(0,1)$ 是误差参数。在保证计数估计的误差不超过 εN 的概率是 ρ 的情况下，将有 $\frac{e}{\varepsilon}*\ln(-\frac{M}{\ln\rho})$ 个计数器被用于数据项计数的估计，并且

$m=e/\varepsilon, h=\ln(-M/\ln\rho)$。

证明：对于任意一个数据项 k，对它进行计数的相关的计数器有 h 个，$S[H_1(k)][1]$, $S[H_2(k)][2]$,…, $S[H_h(k)][h]$，每个计数器都可能对好几个数据项进行计数。假设每个计数器都被利用，那么每个计数器平均有 M/m 个数据项映射。这就意味每个计数器期望的计数值为 N/m，每个计数误差的期望值都不超过 N/m。设随机变量 Y 表示这个误差值，并且 $E[Y]\leqslant N/m$

由 Markov 不等式可得

$$P[|Y|-\lambda E[|Y|]>0]\leqslant 1/\lambda$$

这里 λ 是个正数。因为 Y 是个正整数，并且 $E[Y]\leqslant N/m$，也可以写成

$$P[Y-\lambda N/m>0]\leqslant 1/\lambda$$

上面的式子表示出，变量 Y 的值大于 $\lambda N/m$ 的概率不超过 $1/\lambda$，这个概率用 p 表示。如果我们重复 h 次，那么使所有值都大于 $\lambda N/m$ 的概率不超过 p^h。使 $Y_{\min}$ 表示 h 个 Y 的值中最小的那个，可以得到以下的式子：

$$P[Y_{\min}-\lambda N/m<0]\geqslant 1-1/\lambda^h \quad (2)$$

ρ 表示所有 M 个数据项都满足上述式子的概率。

$$P=(1-1/\lambda^h)^M\approx \exp(-M/\lambda^h) \quad (3)$$

设置用户定义的误差参数 ε 为 λ/m。

为了降低对内存空间的要求，使 V 表示哈希表的大小，$V=m*h$，通过式(2)和(3)可以得到下式：

$$V=\frac{1}{\varepsilon}\ln(-\frac{M}{\ln\rho})*\lambda/(\ln\lambda)$$

这里 λ 是个正数，$\min(\frac{\lambda}{\ln\lambda})=e$。可以得出 $V=\frac{e}{\varepsilon}\ln(-\frac{M}{\ln\rho})$，所以可以算出 h 和 m 应如何取值：

$$h=(-\frac{M}{\ln\rho}), m=\frac{e}{\varepsilon}$$

例子 1：假设一个数据流，它的数据项值域的范围在[1…16]内。窗口大小 $W=30$，$d=10$，支持阈 $s=0.1$。创建哈希表 $S[M][h]$，其中 $m=5$ 和 $h=4$。哈希函数(1)，四个哈希参数：(a_1,b_1):(7,13)，(a_2,b_2):(22,6)，(a_3,b_3):(24,11)，(a_4,b_4):(14,27)，$P=31$。

表 1 数据流的数据

ti	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
k	2	1	6	3	9	6	16	1	13	2	4	3	16	1	5	3	10	5	2	11
ti	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
k	11	2	1	3	8	2	1	4	11	3	7	5	1	1	9	2	2	13	12	16

表 2 数据流实际计数和哈希估计计数的比较

Item	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
True count	4	5	3	2	3	0	1	1	1	1	3	1	1	0	0	2
Estimated count	5	4	4	1	4	2	3	2	2	5	3	2	2	3	1	2

表 1 表示的是一个由 40 个数据项组成的数据流，t_i 表示该项是第 i 个项，k 表示的是要处理的数据项。当滑动窗口没满时，对每个到来的数据项都执行 insert 操作，就是增加相关计数器的计数。图 1(a)是窗口未满时前 4 个数据项到来时哈希表的情况，图 1(b)是窗口满时刻哈希表的状态，此后再有数据项到来时候，对窗口内最先到来的 d 个数据执行和 insert 相反的操作。图 1(c)就是对窗口中前 10 个到来的数据项移除后哈希表的状态。从表 2 是根据图 1(d)得到的当时计数器对数据项的计数情况和实际数据项的出现次数。此时执行 Freq Output 操作，当 $s=0.1$ 时，算法输出数据项为 1，2，3，5，7，10，11，14。从表 2 可以看出实际的频繁项是 1，2，3，5 和 11，数据流中所有的频繁项都被输出了。

1	0	1	1	1
1	0	0	1	2
1	1	0	1	1
1	0	1	1	1

(a) t_4时的哈希表

8	2	6	7	4
7	1	0	10	9
2	5	3	9	6
8	3	6	0	8

(b) t_4时的哈希表

8	1	2	5	1
6	1	0	6	7
0	4	1	7	4
8	3	5	0	5

(c) 前10个项被移除

9	2	8	6	2
7	1	5	11	7
2	4	4	9	7
8	3	5	4	10

(d) t_{40}时的哈希表

图1 处理数据流时的哈希表状态图

4 性能分析

传统HCS分布式数据流处理模型是把单个数据流的挖掘结果,传输到上一层,反复挖掘上一层的结果,直到挖掘出最后的全局频繁模式。本文提出的方法也需要把从每个分布在各个结点的数据流的抽样数据传输到一个执行挖掘工作的结点上,所以两种模型都需要结点间进行通信。将从通信负载和算法运行时间两方面对HCS模型和本算法FFIDDS模型进行比较。

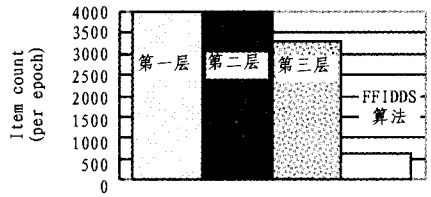

图2 传统HCS算法模型和FFIDDS模型结点通信的比较

分布式数据流挖掘可应用在网络安全中分布式拒绝攻击的早期检测和大规模分布式网络系统的"热点"的检测上。所以使用现实数据网络通信日志Internet2[8],并查找近期的热点来验证算法。使用一个由216个网络结点组成的分布式数据流。设置阈值s为0.01,传统分布式数据流挖掘模型中使每6个数据流为一组,根结点便在第4层,挖掘频繁项的算法使用lossy counting算法[6]。

在HCS数据流挖掘模型下采用216个单独数据流经过四层进行挖掘,需要通信三次才能把最终的挖掘结果输出,而FFIDDS模型只需要进行一次数据的通信。由图2可知,HCS模型每层通信数据量都比FFIDDS模型单次通信的信息量要大得多。另外,在系统的运行时间上,FFIDDS算法也要比基于传统HCS模型的算法少得多。

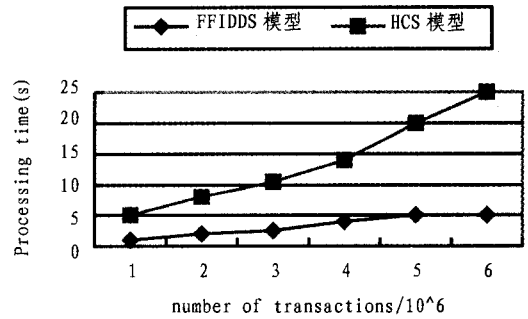

图3 算法运行时间的比较

结论 本文提出了一种在分布式数据流中查找近期频繁项的算法。它不同于先挖掘再聚合再挖掘的HCS模型,是先把分布式的数据流通过抽样聚合成一个数据流再进行频繁项挖掘。采用的抽样技术是基于密度的偏倚抽样,抽样操作由为每个单独数据流设置的监视结点完成。这种抽样算法根据数据密度改变抽样率,所以抽样的数据可以代表整个数据流的信息。频繁项的挖掘过程由基于滑动窗口模式和基于哈希的计数方式相结合的方法完成。实验结果表明,不论从通信负载和运行时间上,FFIDDS算法都要优于传统HCS模型。

参考文献

1 Arasu A, Manku G S. Approximate quantiles and frequency counts over sliding windows. In:PODS, 2004
2 Kim H A, Karp B. Autograph: Toward automated distributed worm signature detection. In:Proceedings of the 13th Usenix Security Symposium, 2004
3 Vitter J S. Random sampling with a reservoir[J]. ACM Transactions on Marhematical Software, 1985, 11(3): 37~57
4 Kollios G, Gunopoulos D, Koudas N. Efficient biased sampling for approximate clustering and outlier detection in large data sets [J]. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(5): 1170~1187
5 Manjhi A, Shkapenyuk V. Rinding(Recently) Frequent Items in Distributed Data Streams. In:Proceedings of the 21st ICDE, 2005
6 Manku G S, Motwani R. Approximate frequency counts over data streams. In:Proc. of VLDB, 2002
7 Karp R, Papadimitriou C, Shenker S. A simple algorithm for finding frequent elements in sets and bags. Transactions on Databas e Systems, 2003
8 Internet2. Internet2 Abilene Network. Hrrp://Abilene. internet2. edu

(上接第182页)

须对表进行物理的链接,这不仅浪费了空间,而且使得运行时间变长。因此,该算法具有较强的可伸缩性。

结束语 聚类是数据挖掘中的一个比较活跃的研究领域,目前在数据挖掘中已经开发出许多聚类算法。本文针对传统的BIRCH算法的局限性,提出了一种改进的BIRCH——IBIRCH算法,该算法首先通过ID传播把多个表联系起来,使得BIRCH算法可以适用于多表的情况,再通过计算共享最近邻密度,可以发现任意形状的簇。实验表明,该算法不仅具有较强的可伸缩性,还可以得到较高精确的聚类结果。

参考文献

1 Han Jiawei, Kamber M. Data mining: concepts and technique. China Machine Press, 2006
2 Zhang T, Ramakrishnan R, Livny M. BIRCH: An efficient data clustering method for very large databases[C]. In: Proc. ACM SIGMOD Conf. Management of Data, Montreal ,1996. 103~114
3 Ertoz L, Steinbach M, Kumar V. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data[C]. In: Proc. of the 2003 SIAM International Conference on Data Mining, San Francisco, 2003. 150~155
4 Yin X, Han J, Yang J, et al. CrossMine: Efficient Classification Across Multiple Database Relations[C]. In:ICDE, Boston, 2004. 399~410
5 Ester M, Kriegel H, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise[C]. In: Proc. 2nd Int Conf Knowledge Discovery and Data Mining (KDD'96), Portland , 1996. 226~231
6 Karypis G, Han E H, Kumar V. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling [J]. IEEE Computer, 1999, 32(8): 68~75