ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.4, April 2007, pp.933−942
DOI: 10.1360/jos180933

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

高维类别属性数据流离群点快速检测算法*

周晓云+, 孙志挥, 张柏礼, 杨宜东

(东南大学 计算机科学与工程系,江苏 南京 210096)

A Fast Outlier Detection Algorithm for High Dimensional Categorical Data Streams

ZHOU Xiao-Yun+, SUN Zhi-Hui, ZHANG Bai-Li, YANG Yi-Dong

(Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

+ Corresponding author: Phn: +86-25-83795451, E-mail: zxy0724@seu.edu.cn, http://www.seu.edu.cn

Zhou XY, Sun ZH, Zhang BL, Yang YD. A fast outlier detection algorithm for high dimensional categorical data streams. *Journal of Software*, 2007,18(4):933−942. http://www.jos.org.cn/ 1000-9825/18/933.htm

Abstract: This paper considers the problem of outlier detection in data stream, proposes a new metric called weighted frequent pattern outlier factor for categorical data streams, and presents a novel fast outlier detection algorithm named FODFP-Stream (fast outlier detection for high dimensional categorical data streams based on frequent pattern). FODFP-Stream computes the outlier measure through discovering and maintaining the frequent patterns dynamically, and can deal with the high dimensional categorical data streams effectively. FODFP-Stream can also be extended to resolve continuous attributes and mixed attributes data streams. The experimental results on synthetic and real data sets show the promising availabilities of the approaches.
Key words: data stream; outlier detection; frequent pattern; high dimension; concept drift

摘 要: 提出类别属性数据流数据离群度量——加权频繁模式离群因子(weighted frequent pattern outlier factor,简称 WFPOF),并在此基础上给出一种快速数据流离群点检测算法 FODFP-Stream(fast outlier detection for high dimensional categorical data streams based on frequent pattern).该算法通过动态发现和维护频繁模式来计算离群度,能够有效地处理高维类别属性数据流,并可进一步扩展到数值属性和混合属性数据流.对仿真数据集和真实数据集的实验检测均验证该算法具有良好的适用性和有效性.
关键词: 数据流;离群点检测;频繁模式;高维;概念转移
中图法分类号: TP311 文献标识码: A

随着计算机技术的广泛应用,数据流(data stream)作为一类重要的数据来源,受到越来越多的关注,基于数据流模型的管理系统及其知识发现算法等已成为重要的研究课题[1−3].网络事件日志、电话呼叫记录、信用卡交易流、传感器网络等均可以看作是基于数据流模型的数据集.它们具有数据量大、潜在无限、到达速率不确定等特点,若对这类数据存储后多次扫描进行数据挖掘,其代价昂贵甚至是不可能的,且缺少实时性.因此,迫切需要提出高效、可行的基于数据流模型的算法,使得在给定的有限运行空间上,能够通过对数据流进行一次或

* Supported by the National Natural Science Foundation of China under Grant No.70371015 (国家自然科学基金); the Doctor Science Research Foundation of the Education Ministry of China under Grant No.20040286009 (国家教育部高等学校博士学科点科研基金)
Received 2005-11-02; Accepted 2006-02-23

较少次数的线性扫描,对其进行管理以及进一步的知识发现.

1 相关工作

离群点检测问题是数据挖掘技术的重要研究领域之一,它被广泛应用于网络入侵抵御、信用卡恶意透支检测等风险控制领域.离群点检测技术由于其独特的知识发现功能而得到较为深入的研究.到目前为止,离群点还没有一个正式的、为人们普遍认同的定义.Hawkins 的定义[4]揭示了离群点的本质:“如果一个数据样本与其他样本之间存在足以引起怀疑的差异,则称其为离群点”.

假设检验是最早用来发现异常样本点的基于统计学原理的方法,它基于对小概率事件的判别来实现对数据样本异常性的鉴别.然而,基于统计学的离群点检测方法必须假设数据集符合特定的分布模型,显然,在面对大量分布特征未知的数据时,这种先验假设存在很大的局限性.

离群点的检测也经常作为聚类算法研究的一种副产品出现[5].在这些算法中,离群点被定义为不属于任何聚类的数据点.由于这些算法着重于从聚类的角度进行考虑,而离群点的检测通常不需要了解其他数据具体的聚类情况,因此,这类检测方法通常不能满足实际应用的效率要求.

近年来,基于数据挖掘概念的离群点检测研究取得了一定的进展,提出了一些有效的检测算法,并得到一定程度的应用.例如,Johnson 等人提出的基于深度的算法 DEEPLOC[6],Knorr 等人提出的基于距离的算法 FindAllOutsD[7], Breunig 等人提出的带离群度的离群点检测算法 LOF[8],Papadimitirou 等人提出的 LOCI 算法[9].这些算法一方面具有 $O(N\log N)$以上的时间复杂度;另一方面,数据集的高维性常造成上述算法失效.为了克服“维数灾难”问题,Aggarwal 等人提出了基于空间投影的离群点检测算法 EvolutionaryOutlierSearch[10],Wei 等人提出了基于超图模型的离群点检测算法 HOT[11],但时间复杂度较高,在处理大数据集时无法获得令人满意的响应速度.特别是不能够处理只能进行一遍扫描的数据流数据,限制了其在数据流数据上的进一步应用.

He 等人提出了基于频繁模式的离群点检测算法 FindFPOF[12],算法给出了新的离群点度量——频繁模式离群因子(frequent pattern outlier factor,简称 FPOF),认为频繁模式为通常模式,一个数据中包含的频繁模式越少,则其成为离群点的可能性越大. FindFPOF 具有高维数据离群点检测能力,比 EvolutionaryOutlierSearch 和 HOT 时间复杂度要低.但仍然需要多次扫描数据集,因此同样不适合数据流环境.

而在数据流算法的研究中,离群点检测问题同样引起了研究者的关注.文献[13]首次提出了一种针对大规模数据流的异常检测算法,该算法引用文献[14]中提出的异常(deviant)作为离群点的概念,但该算法只能处理低维数值型数据,并且没有针对概念转移情况进行考虑.

本文的工作主要包括以下几个方面:在频繁模式的离群因子概念基础上,给出类别属性数据流数据离群度量——加权频繁模式离群因子(weighted frequent pattern outlier factor,简称 WFPOF),并针对数据流特点提出基于 WFPOF 的高维数据流离群点检测算法 FODFP-Stream(fast outlier detection for high dimensional categorical data streams based on frequent pattern). FODFP-Stream 利用 WFPOF 能够有效度量高维数据离群度的优点,结合适应离群点检测特点的数据流频繁模式发现维护方法,快速而有效地检测离群点.同时,通过数据衰减系数的设定,可以有效地处理数据流数据中的概念转移问题.FODFP-Stream 可以进一步扩展到数值属性和混合属性数据流上的离群点检测.实验表明, FODFP-Stream 算法是可行而有效的.

2 问题描述及相关定义

设 $D=\{D_1,D_2,\ldots,D_k\}$是一个有序集合, $A_j=\{a_j^{(1)},a_j^{(2)},\ldots,a_j^{(n_j)}\}$ 是类别属性 D_j 上的值的集合,其中,n_j 表示类别属性 D_j 的可取值的个数,相应地,有 k 维类别属性空间 $S=D_1\times D_2\times\ldots\times D_k$.设 $X=\langle x_1,x_2,\ldots,x_t\rangle$是数据流 DS 在第 t 个数据到达时的数据集,其中,$x_i=\langle x_{i1},x_{i2},\ldots,x_{ik}\rangle$,$x_{ij}\in A_j$.为简化描述,我们假设数据在每个单位时间到达,对于其他情况也可以很容易地进行转化处理.

本文对所研究的高维类别属性数据流中离群点检测问题描述为:给定一个 k 维类别属性数据流 DS,在一定内存和时间限制下,动态维护相对于当前窗口(宽度为 w)数据流分布的候选离群点集合 $CandOutlier$.当用户提

出查询 $TopQ$($0<Q<100\%$)的离群点时,算法对 $CandOutlier$ 进行筛选得到离群点集合 $Outlier$,并提供给用户.候选离群点集合的大小为预先设置的参数,在算法实际运行中,设定$|CandOutlier|=P\times w$ ($0<P<100\%,P\geq Q$).

文献[12]中提出了频繁模式离群因子的概念,并在其基础上给出了一个新的离群点定义.本文将对 FPOF 作进一步的修正,提出加权频繁模式离群因子的概念.首先对频繁模式离群因子进行简单的描述.

设 $I=\{i_1,i_2,\ldots,i_m\}$为 m 个项目的集合,$D=\{t_1,t_2,\ldots,t_n\}$为含有 n 个交易的数据集,t_j 为一条交易,是 I 的非空子集($1\leq j\leq n$).

定义 **1**(频繁模式离群因子). 对于任意交易 t,其频繁模式离群因子为

$$FPOF(t)=\frac{\sum support(Y)}{\|FPS(D,\text{mini}support)\|},$$

其中:Y 为 I 的非空子集;mini$support$ 为最小支持度阈值;$support(Y)=\|\{t\in D|Y\subseteq t\}\|/\|\{t\in D\}\|$;$FPS(D,\text{mini}support)=\{Y\subseteq I|support(Y)\geq\text{mini}support\}$;$\|\ldots\|$表示集合所含元素个数.

由定义 1 可知:若一个交易 t 包含的频繁项目集越多,其 FPOF 越大,则它成为离群点的可能性越小;相反地,具有较小 FPOF 值的交易,成为离群点的可能性越大.显然,FPOF 的取值在 0 和 1 之间.但是,不同长度的频繁模式的贡献是不同的,长度较长的模式往往更有意义.如果仅仅考虑频繁模式的支持度,显然会丢失一些有用信息.因此,本文先对其作必要的修正,并应用于类别属性数据流的离群点检测.

定义 **2**(模式 ***p***). $(D_{m_1}=v_{m_1},D_{m_2}=v_{m_2},\ldots,D_{m_q}=v_{m_q})$ 表示一个模式 p,其中,$1\leq m_1\leq m_2\leq\ldots\leq m_q\leq k$;第 i 个等式表示属性 D_{m_i} 的取值为 v_{m_i},$v_{m_i}\in A_{m_i}$.对任意数据 x,若其 $D_{m_l}=v_{m_l}$ $(1\leq l\leq q)$,则称 x 满足模式 p.

定义 **3**(模式 ***p*** 的支持度). 在任意时刻,模式 p 的支持度 $Sup(p)=\frac{p.Count}{N}$,其中,$p.Count$ 为数据流中满足模式 p 的数据个数;N 为数据流总量.

定义 **4**(频繁模式). 对任意模式 p,若 $Sup(p)\geq\text{min}sup$,则称模式 p 是频繁的,其中,minsup 为用户指定最小支持度阈值.

定义 **5**(加权频繁模式离群因子). 对于 k 维类别属性数据流 DS 上的任意数据点 x,其加权频繁模式离群因子为

$$WFPOF(x)=\frac{\sum Sup(p)\times(|p|/k)}{\|FPS(DS,\text{min}sup)\|},$$

其中:$|p|$为模式 p 的长度;k 为数据空间的维数;$FPS(DS,\text{min}sup)=\{p|Sup(p)\geq\text{min}sup\}$.

WFPOF 继承了 FPOF 的优点,与其他离群点定义相比,不仅能够有效度量高维数据的离群度,而且计算所需的时空复杂度较低,有利于应用于高维数据流环境.同时,WFPOF 将模式长度与数据流空间维数的比值作为权重,更能体现不同长度频繁模式对离群度的贡献大小,使得 WFPOF 比 FPOF 能够更准确地度量离群点.

文献[12]中的 FindFPOF 算法通过计算交易数据集中的 FPOF 来发现高维数据集中的离群点,具有简单、直观的特点.但是,FindFPOF 算法是针对传统静态数据集,需要多次扫描数据集:首先,通过传统频繁模式发现算法,获得整个数据集的频繁模式集,至少需要两次扫描;然后再次扫描数据集,计算每个交易的 FPOF 值,判断其是否为离群点.与以前的离群点检测算法一样,FindFPOF 仍然是将整个数据集作为检测对象.而本文所提出的算法继承了 FindFPOF 的优点,并在一定内存、时间限制下,通过动态地维护数据流的频繁模式,同时计算 WFPOF 值,判断其是否为离群点,只需扫描数据一次,从而能够有效地应用于数据流环境.

3 FODFP-Stream 算法及其构造

3.1 动态发现和维护频繁模式

FODFP-Stream 算法的重点和难点是设计适应数据流离群点检测的频繁模式发现和维护方法.由于数据流的特殊性和频繁模式挖掘算法本身的要求,基于数据流的频繁模式挖掘成为当前研究的难点之一.针对静态数

据集的频繁模式挖掘已进行了大量的研究,如 Apriori 算法[15]、FP-growth 算法[16]等,这些算法一方面需要多次扫描数据集,另一方面空间复杂度较大,无法在有限的内存空间中进行数据保存和计算,因此无法适应高维数据流的应用要求.近年来,数据流频繁模式的挖掘已有一些研究成果,如 Manku 等人提出的 Lossy Counting 算法[17]和 Jin 等人提出的 StreamMining 算法[18].上述两种算法在设定支持度阈值 s 和误差因子 ε 的情况下,得到频繁模式的估计值.能够保证支持度大于 s 的所有频繁模式均被输出,支持度小于 $s-\varepsilon$ 的所有模式均不输出,一个模式的被估计的支持度小于真实值最多为 ε,其存在假正(false positive)区间为 $[s-\varepsilon,s]$.但是,上述两种算法并没有提供对出现假正的频繁模式数量的保证,也没有对该数量进行估计.在最坏情况下,得到的所有频繁模式都落在这个假正区间内,这势必对进一步的离群点检测的精度造成影响.

为了克服上述问题,本文提出一种新颖的数据流频繁模式发现和维护方法,对出现假正的频繁模式数量提供概率保证.

通过对频繁模式挖掘的特点分析可以发现:类 Apriori 算法的瓶颈在于产生大量低维的项目集,无法将其保存至有限的内存中.若保存至硬盘,必将影响算法的响应速度,因此无法应用于数据流.对于类别属性数据流,当属性较多(即高维)、各属性取值域大和支持度阈值 s 较小时,时空效率更差.而类 FP-growth 算法,由于数据流的动态性,动态维护 FP-tree 结构的开销很大,对于处理出现概念转移的数据流更加困难.Apriori 算法将作为本文方法的基本思想之一.

本文方法的另一思想来源于 Bloom Filter 算法[19]及其改进算法 SBF(spectral Bloom filters)[20].Bloom Filter 的最大特点是:仅使用一小块远小于数据集数据范围的内存空间来表示数据集,并且各个数据仍然能被区分开来.假设所申请的内存大小为 m 比特位,该方法创建 g 个相互独立的哈希函数,能将数据集均匀映射到 $[1\ldots m]$ 中去.对任何元素,利用哈希函数进行计算,得到 g 个 $[1\ldots m]$ 之间的数,并将内存空间中这 g 个对应比特位都置为 1.这样,就可以通过检查一个元素经过 g 次哈希操作后,是否所有对应的比特位都被置为 1 来判断该元素是否存在.这种判断方法可能产生假正——虽然某元素并不存在,但是它所对应的 g 个比特位都已经被其他元素所设置了,从而导致被误认为存在.然而,假正发生的概率随着内存的增加可以非常小.SBF 对其进行了改进,在每一个位置上都用计数器代替比特位,从而不仅能够判断元素是否存在,而且能够估算元素的值[3].

由此,本文所提出方法的主要思路为:利用类 Bloom Filter 进行近似估计,减少低维项目集的规模,再根据 Apriori 的单调性性质获得高维项目集.数据流数据的一个重要特点是其动态性,其中存在着概念转移现象.为解决这一问题,可以对历史数据给予较小的权重,使其影响逐步减小,这样能够更好地反映当前数据的分布情况.为此,在 Bloom Filter 的每一个位置上设置计数器 *count* 的同时,又设置一个计时器 *time*,用于记录这个位置最近一直修改的时间.

定义数据权重的衰减系数 $\lambda(0\le\lambda\le1)$,并设每个数据在某个时刻 t 的权重为 $\lambda^{t-t_{occur}}(t\ge t_{occur})$,其中,$t_{occur}$ 为数据第 1 次出现的时间,初始状态时数据的权重为 1.可以看到,通过对衰减系数的设定,数据的权重随着时间的推移逐渐减少,符合算法需要.为设定衰减系数 λ,设一个数据点的生命周期为 t_{life},权重阈值为 $weight_{\min}$,并满足:$\lambda^{t_{life}}\le weight_{\min}$,即数据在经过 t_{life} 后,其权重将不大于 $weight_{\min}$.通过不同的 t_{life} 和 $weight_{\min}$,就可以设置相应的 λ.

性质 **1**. 在时刻 $t(t\ge latest)$时,Bloom Filter 计数器 *count* 满足 $count_t=\lambda^{t-latest}\times count_{latest}$,其中 *latest* 为计时器 *time* 的值.

设当前时间为 t_{cur},由性质 1 可知,算法可以对保存信息进行增量更新:$count=\lambda^{t_{cur}-t_{lastest}}\times count+1$,$time=t_{cur}$.

3.2 FODFP-Stream算法

当新数据加入时,FODFP-Stream 对频繁模式集进行更新,同时,根据定义 5 和当前的频繁模式集计算该数据的 WFPOF.然后,对数据按 WFPOF 从小到达顺序排列,维护前 $P\times w$ 个数据点,由于算法维护数据流中当前窗口的 *CandOutlier*,因此,对于 *time* 距离当前时间大于窗口宽度 w 的数据,需要对其进行删除操作.

频繁模式挖掘方法将数据流 *DS* 上的所有低维模式进行编码,映射成 $[1\ldots M]$ 的整数,在内存中保存 m 个元

组,每个元组结构为(*id*,*count*,*time*),利用 *g* 个哈希函数将[1…*M*]映射到[1…*m*].这样,在较高正确概率的情况下,得到数据流的低维频繁模式估计,再根据 Aporior 的单调性性质,获得更高维的频繁模式.具体将多少维以下的模式进行编码,将根据内存大小和精度需要,由用户提供维数阈值参数 *l* 设定.

FODFP-Stream 在初始化时,可以一次性读入若干数据,得到初始的频繁模式集.离群点检测是基于大规模数据上的统计信息,这样可以避免在初期数据较少时不必要的操作.对于信息的衰减操作也可以推迟到有新数据加入或用户进行查询时进行.

为了方便描述,定义以下符号:*CandSet* 表示高维候选频繁模式集; $Lattic=\bigcup_{r=1}^{k}L_r$ 表示频繁模式集;L_r 表示 $r(1\le r\le k)$维频繁模式集;*CandSet* 和 *Lattic* 保存的元组结构为(*item_id*,*count*,*time*,*level*),*item_id* 为模式的编码,*level* 为模式的长度,*count* 为模式的支持数,*time* 为模式最近一次更新的时间.算法实现时,采用 TreeHash[18]对元组进行存储、搜索和更新操作.具体算法描述如下:

算法. FODFP-Stream.

输入: *k* 维数据流 *DS*;维数阈值 *l*;哈希函数 $h_1,h_2,\ldots,h_g$;最小支持度 min*sup*;衰减系数λ,*CandOutlier* 大小 $P\times w(0<P<100\%)$.

输出: *k* 维数据流 *DS* 的离群点集合 *Outlier*.

步骤:

```
Initialization;
For each datum x arrived in DS {
 Support=0;FPS=0;
 N=N×λ+1
 For all f-itemset(1≤f≤l) s of x {
  id=convert(s,DS,l);   /*convert(s,DS,l)将 s 转换成[1…M]之间的整数*/
  For (i=1;i<=g;i++) {
   V[h_i(id)].count = V[h_i(id)].count × λ^(t−V[h_i(id)].time); V[h_i(id)].time = t; }
   /*V[i]表示获得第 i 个位置上的元组,t 为 x 到达时间*/
   If (s not in L_f) {
    If (all V[h_i(id)].count≥minsup×N) {
     insert s into L_f;
     L[s].count=min(V[h_i(id)].count);L[s].time=t;
     Support=Support+((|s|/k)×L[s].count)/N;
     FPS=FPS+1;}
     /*L[s]表示获得 s 在 Lattic 中的元组*/
    }
    Else If (all V[h_i(id)].count≥minsup×N){
        L[s].count=L[s].count×λ^(t−L[s].time)+1;L[s].time=t;
         Support=Support+((|s|/k)×L[s].count)/N;
         FPS=FPS+1;}
        Else delete s and all superset(s) from Lattic;
         /* superset(s)表示 s 的所有超集*/
 }
 j=l+1;
 While (L_j≠∅) and (j≤k) do {
   For all j-itemset s of x {
```

If (s in *CandSet*) {
$C[s].count=C[s].count\times\lambda^{t-C[s].time}+1;C[s].time=t;$}
/*$C[s]$表示获得 s 在 *CandSet* 中的元组*/
Else if all ($j-1$)-itemset of s in (*CandSet* or L_l){
insert s into *CandSet*;
$C[s].count=1;C[s].time=t;$}
If (s not in L_j) {
If ($C[s].count\geq \min sup\times N$) {
insert s into L_j;
$L[s].count=C[s].count;L[s].time=C[s].time;$
$Support=Support+((|s|/k)\times L[s].count)/N;$
$FPS=FPS+1;$}
}
Else If ($C[s].count\geq \min sup\times N$){
$L[s].count=L[s].count\times\lambda^{t-L[s].time}+1;L[s].time=t;$
$Support=Support+((|s|/k)\times L[s].count)/N;$
$FPS=FPS+1;$}
Else delete s and all *superset*(s) from *Lattic*;
}
$j=j+1;$
}
$$WFPOF(x)=\frac{Support}{FPS};$$
}
Sort Top $P\times w$ datum x with least *WFPOF* which in current windows as *CandOutlier*;
If requested Output *TopQ* datum x with least *WFPOF* as *Outlier*.

定理 **1**. 算法 FODFP-Stream 频繁模式挖掘出现假正的概率 $P\approx(1-e^{-gM/m})^g$.

证明:在 FODFP-Stream 中,对于维数大于维数阈值 l 的频繁模式,其候选集是根据低维频繁模式产生的,所以误差也由低维频繁模式决定.因此,我们只需考察低维(维数不大于 l)的频繁模式的误差.由文献[20]可知,SBF 的假正概率 $P\approx(1-e^{-gM/m})^g$.本文算法中,低维频繁模式发现方法是 SBF 算法的简单扩充,满足该式.所以,该算法频繁模式挖掘出现假正的概率 $P\approx(1-e^{-gM/m})^g$,其中,M 为根据 l 和数据流 *DS* 类别属性得到的编码个数. □

与 Lossy Counting 和 StreamMining 算法相比,FODFP-Stream 频繁模式挖掘能够提供出现假正的概率保证,进而有效地保证了离群点检测的精度.由定理 1 可知,在实际运用中,出现假正的概率是很低的,并且与内存元组数 m 成反比,与维数阈值 l 成正比.

3.3 讨 论

算法可以进一步扩展到数值属性和混合属性的数据流.对于数值属性数据流或是混合属性数据流中的数值属性,预先作离散化.有很多算法可以对连续属性进行离散化,可参考文献[21].比较简单的方法是,可以将连续属性维归一化后,平均分割为ξ个区间,每个区间对应一个类别属性值.在下面的实验部分,为了与现有算法进行比较,我们采用了这种简单方法.

随着数据流的不断到达,在 l 一定的情况下,产生的高维候选频繁模式集 *CandSet* 的规模也随之增大.对于分布稳定的数据流,*CandSet* 规模最终将趋于稳定;而对于发生概念转移的数据流,*CandSet* 会发生显著变化.对于那些不再是候选的模式,删除操作可推迟到可用内存不足的时候再进行.

4 算法性能分析与实验结果

4.1 复杂度分析

本文构造的高维类别属性数据流离群点检测算法 FODFP-Stream 具有良好的空间与时间效率.

在算法运行过程中,需要维护频繁模式集以及候选离群点集合 *CandOutlier*.其中,频繁模式集维护的空间复杂度包括 4 个部分:Bloom Filter 所需内存空间 $O(m)$、高维(维数大于 l)候选频繁模式所需内存 $O(|CandSet|)$、频繁模式集存储所需空间 $O(|Lattic|)$和候选离群点所需空间 $O(|CandOutlier|)$.在类 Apriori 算法中,低维候选频繁模式需要大量的存储空间,随着维数的增加,高维候选频繁模式显著减少.FODFP-Stream 用 $O(m)$空间对低维候选频繁模式进行估计,并且用户可以根据内存大小和精度需要设定维数阈值参数 l,因此,$O(m+|CandSet|)$可以控制在有限的内存范围内,所以 FODFP-Stream 可以在有限内存空间中得以实现.下一节的实验结果进一步说明了本文所提算法仅所需较少的内存.

定理 **2**. FODFP-Stream 算法具有相对于数据流数据量 N 线性规模的时间复杂度.

证明:在最坏情况下对频繁模式的更新与维护,计算 WFPOF 的时间负责度为 $O(2^kN)$,对 *CandOutlier* 维护需 $O(|CandOutlier|N)$,FODFP-Stream 算法总共需要 $O((2^k+|CandOutlier|)N)$时间复杂度,因此,整个算法对于数据量 N 具有线性复杂度. □

在实际情况下,高维频繁模式数量很少,并且存在的频繁模式的最高维数远小于 k.因此,FODFP-Stream 算法的计算量远小于最坏情况的时间估计.

4.2 实验结果

本节对所提出的 FODFP-Stream 算法进行性能测试.文献[13]中所提算法采用的是动态规划方法,针对时间序列模型数据流中的异常检测问题,与本文研究问题有一定的差别,因此我们选取 FindFPOF, EvolutionaryOutlierSearch 作为对比算法.两种算法都是处理高维数据的,并且 FindFPOF 是基于频繁模式的,具有一定的可比性.但是,它们都是针对静态数据集的算法,不存在时间维度的概念,因此,对它们在整个数据集上进行测试.实验平台配置如下:Intel 1.8G/512MB,Windows2000 (Server 版),所用代码均用 Visual C++ (6.0)实现.实验所使用的数据共有两种:第 1 种是网络入侵检测数据集 KDD-CUP-99,该数据集中的数据对象分为 5 大类,包括正常的连接、各种入侵和攻击等.为了进行实验,对 KDD-CUP-99 数据作适当的修改,使得攻击(即离群点)只占数据集的 2%.对连续属性维进行归一离散化,每维平均分割成 10 个区间;第 2 种是仿真数据,用户可以通过输入参数来控制产生数据集的大小、维数和聚类个数等.

为测试算法的精确度和执行效率,我们对比了 FODFP-Stream,FindFPOF,EvolutionaryOutlierSearch.采用 KDD-CUP-99 数据集和仿真数据集 Synthetic DataSet1 作为测试集,其中,Synthetic DataSet1 为 30 维,800K,包含 3 个聚类,均匀加入 3%和聚类具有较大偏差的离群点.实验中,取 P=10%,w=100K,minsup=0.2%,l=10.精度采用以下量度对算法进行评价:

$$Precision = \frac{Number\ of\ \ correct\ outliers}{|Outlier|}.$$

实验结果如图 1 和图 2 所示.FODFP-Stream 虽然精度没有 FindFPOF 和 EvolutionaryOutlierSearch 高,但仍在 94%以上.而由于 FODFP-Stream 采用动态发现维护频繁模式,一遍扫描数据集,在运行速度上表现出了其优越性.

为了测试算法的内存使用情况,分别生成 10,15,20,25 和 30 维仿真数据集,大小为 100K,并设算法 FODFP-Stream 的窗口长度为 w=10K,P=10%,l=8.数据集包括 3 个主要聚类,数据集中加入 3%和 3 个聚类都具有较大偏差的离群点.算法运行稳定时,使用的内存如图 3 所示.可以看到:算法所需的内存保持在一个较小的范围内.其内存使用对数据维数具有良好的伸缩性.支持度阈值越大,需要动态维护的频繁模式越少,所用内存也越少.

为了测试算法对概念转移的处理能力,生成服从不同分布的两组 10 维数据,大小均为 15K,类似地,均匀插入 3%偏离较大的数据点作为离群点,并将两组数据合并,模拟发生概念转移的数据流.取窗口宽度为 w=6K,

t_{life}=6K,$weight_{min}$=0.1,minsup=0.4%,l=5.我们对比了第 6K,12K,18K 和 24K 数据开始的 4 个窗口(分别标记为第 1~第 4 个窗口)中离群点的精度,如图 4 所示.由于在第 2 个窗口,两组数据发生混合现象,造成检测精确度降低.但随着时间的推移,第 1 组数据的影响逐渐减少,算法精确度也随之提高.

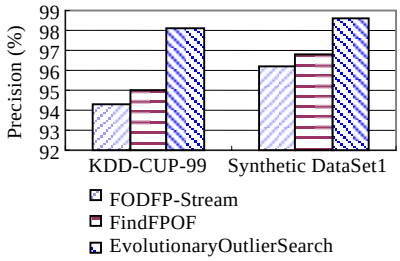

Fig.1 Precisions of FODFP-Stream, FindFPOF, EvolutionaryOutlierSearch

图 1 不同算法的精度对比

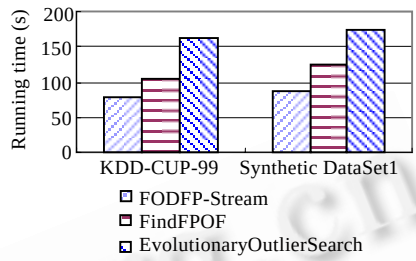

Fig.2 Running times of FODFP-Stream, indFPOF EvolutionaryOutlierSearch

图 2 不同算法的执行时间对比

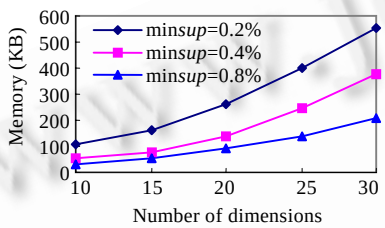

Fig.3 Memory usage for different dimensions

图 3 不同维数下的内存使用

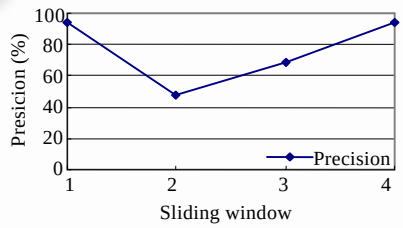

Fig.4 Precisions for different windows

图 4 不同窗口中的算法精度

为了测试算法对数据集大小的伸缩性,分别生成大小为 100K,200K,400K,800K 和 1600K 的数据集,数据维数均为 30,均包括两个聚类,均匀插入 3%偏离两个聚类较大的数据点作为离群点.窗口长度为 w=20K,P=10%,minsup=0.4%,l=8.如图 5 所示,算法执行时间与数据集大小呈线性变化.

为了测试算法对数据集维数的伸缩性,分别生成 15,20,25,30 和 35 维仿真数据集,大小为 400K,均包括两个聚类,均匀插入 3%偏离较大的数据点作为离群点.窗口长度为 w=20K,P=10%,minsup=0.4%,l=8.如图 6 所示,算法对数据维数具有较好的伸缩性.

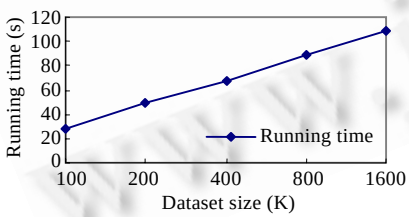

Fig.5 Scaling of running time with dataset size

图 5 对数据集大小的伸缩性

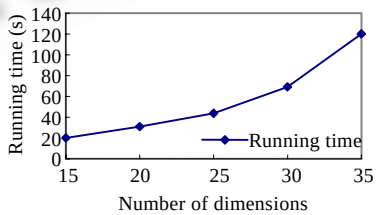

Fig.6 Scaling of running time with dataset dimensionality

图 6 对数据集维数的伸缩性

5 结 论

本文研究了数据流上离群点的检测问题,提出了 FODFP-Stream 算法.算法动态发现与维护频繁模式,通过对数据点的加权频繁模式离群因子的计算,得到候选离群点集合 *CandOutlier*,根据用户查询要求,进一步筛选得到离群点集合 *Outlier*.FODFP-Stream 算法能够提供出现假正的概率保证,有效地处理类别属性数据流的高维问题和概念转移问题.实验结果表明,本文提出的算法是可行而有效的.进一步提高算法的实现效率,将其扩展到

数值属性和混合属性数据流,以及与数据流中其他相关的数据挖掘算法(例如聚类等)的结合等,是我们下一步的研究内容.

References:

[1] Babcock B, Babu S, Datar M, Motawani R, Widom J. Models and issues in data stream systems. In: Popa L, ed. Proc. of the 21st ACM Symp. on Principles of Database Systems. Madison: ACM Press, 2002. 1−16.

[2] Muthukrishnan S. Data streams: Algorithms and applications. In: Proc. of the 14th Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 2003. 413. http://doi.acm.org/10.1145/644108.644174

[3] Jin CQ, Qian WN, Zhou AY. Analysis and management of streaming data: A survey. Journal of Software, 2004,15(8):1172−1181 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1172.htm

[4] Li CH, Sun ZH. GridOF: An efficient outlier detection algorithm for very large datasets. Journal of Computer Research and Development, 2003,40(11):1586−1592 (in Chinese with English abstract).

[5] Zhang T, Ramakrishnan R, Livny M. BIRCH: An efficient data clustering method for very large databases. SIGMOD Record, 1996, 25(2):103−114.

[6] Johnson T, Kwok I, Ng R. Fast computation of 2-dimensional depth contours. In: Agrawal R, Stolorz PE, Piatetsky-Shapiro G, eds. Proc. of the 4th Int'l Conf. on Knowledge Discovery and Data Mining. New York: AAAI Press, 1998. 224−228.

[7] Knorr EM, Ng RT. Algorithms for mining distance-based outliers in large datasets. In: Gupta A, Shmueli O, Widom J, eds. Proc. of the 24th Int'l Conf. on Very Large Databases. New York: ACM Press, 1998. 392−403.

[8] Breunig MM, Kriegel H, Ng RT, Sander J. LOF: Identifying density-based local outliers. In: Chen WD, Naughton JF, Bernstein PA, eds. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. Dallas: ACM Press, 2000. 93−104.

[9] Papadimitirou S, Kitagawa H, Gibbons PB, Faloutsos C. LOCI: Fast outlier detection using the local correlation integral. In: Dayal U, Ramamritham K, Vijayaraman TM, eds. Proc. of the 19th Int'l Conf. on Data Engineering. IEEE Computer Society Press, 2003. 315−326.

[10] Aggarwal C, Yu P. An effective and efficient algorithm for high-dimensional outlier detection. The VLDB Journal, 2005,14(2): 211−221.

[11] Wei L, Gong XQ, Qian WN, Zhou AY. Finding outliers in high-dimensional space. Journal of Software, 2002,13(2):280−290 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/280.pdf

[12] He ZY, Xu XF, Huang JZ, Deng SC. FP-Outlier: Frequent pattern based outlier detection. Computer Science and Information System, 2005,2(1):103−118.

[13] Muthukrishnan S, Shah R, Vitter J. Mining deviants in time series data streams. In: Proc. of the 16th Int'l Conf. on Scientific and Statistical Database Management. Los Alamitos: IEEE Computer Society Press, 2004. 41−50. http://doi.ieeecomputersociety.org/10.1109/SSDBM.2004.51

[14] Jagadish HV, Koudas N, Muthukrishnan S. Mining deviants in a time series database. In: Atkinson MP, Orlowska ME, Valduriez P, Zdonik SB, Brodie ML, eds. Proc. of the 25th Int'l Conf. on Very Large Data Bases. Edinburgh: Morgan Kaufmann Publishers, 1999. 102−113.

[15] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. In: Bocca JB, Jarke M, Zaniolo C, eds. Proc. of the 20th Int'l Conf. on Very Large Databases. San Francisco: Morgan Kaufmann Publishers, 1994. 487−499.

[16] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Chen WD, Naughton JF, Bernstein PA, eds. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2000. 1−12.

[17] Manku GS, Motwani R. Approximate frequency counts over data streams. In: Bernstein P, Loannidis Y, Ramakrishnan R, eds. Proc. of the 28th Int'l Conf. on Very Large Databases. Hong Kong: Morgan Kaufmann Publishers, 2002. 346−357.

[18] Jin R, Agrawal G. An algorithm for in-core frequent itemset mining on streaming data. In: Proc. of the 5th IEEE Int'l Conf. on Data Mining. IEEE Computer Society, 2005. 210−217. http://doi.ieeecomputersociety.org/10.1109/ICDM.2005.21

[19] Bloom B. Space/Time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970,13(7):422−426.

[20] Cohen S, Matias Y. Spectral bloom filters. In: Halevy AY, Ives ZG, Doan AH, eds. Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. San Diego: ACM Press, 2003. 241−252.

[21] Dougherty J, Kohavi R, Sahami M. Supervised and unsupervised discretization of continuous features. In: Prieditis A, Russell SJ, eds. Proc. of the 12th Int'l Conf. on Machine Learning. Tahoe City: Morgan Kaufmann Publishers, 1995. 194–202.

附中文参考文献:

[3] 金澈清,钱卫宁,周傲英.流数据分析与管理综述.软件学报,2004,15(8):1172–1181. http://www.jos.org.cn/1000-9825/15/1172.htm

[4] 李存华,孙志挥.GridOF:面向大规模数据集的高效离群点检测算法.计算机研究与发展,2003,40(11):1586–1592.

[11] 魏藜,宫学庆,钱卫宁,周傲英.高维空间中的离群点发现.软件学报,2002,13(2):280–290. http://www.jos.org.cn/1000-9825/13/280.pdf

周晓云(1979－),男,江苏太仓人,博士生,主要研究领域为数据挖掘,知识发现.

张柏礼(1970－),男,博士生,主要研究领域为数据挖掘,数据仓库.

孙志挥(1941－),男,教授,博士生导师,CCF高级会员,主要研究领域为数据库系统、应用及知识发现.

杨宜东(1979－),男,博士生,主要研究领域为数据挖掘,知识发现.

△△△

2007 年全国开放式分布与并行计算学术年会

征 文 通 知

由中国计算机学会开放系统专业委员会主办、广西大学计算机与电子信息学院承办的 2007 全国开放式分布与并行计算学术年会（DPCS 2007）将于 2007 年 10 月 12 日~15 日在广西南宁市广西大学召开。本次年会录用的论文将由《小型微型计算机系统》和《微电子学与计算机》以正刊方式发表，欢迎大家积极投稿。会议将评选优秀论文，予以奖励并推荐到一级学报发表。同时，鼓励在年会召开期间组织讲座（Tutorial），有意者请与广西大学钟诚、李陶深教授联系。现将有关征文事宜通知如下：

一、征文范围

（1）开放式分布与并行计算模型、体系结构、算法及应用；(2）开放式网络、数据通信、网络与信息安全、业务管理技术；（3）开放式海量数据存储与 Internet 索引技术，分布与并行数据库及数据/Web 挖掘技术；(4）开放式机群计算、网格计算、Web 服务、P2P 网络及中间件技术；(5）开放式移动计算、移动代理、传感器网络与自组网技术；(6）分布式人工智能、多代理与决策支持技术；(7）分布、并行编程环境和工具；(8）分布与并行计算算法及其在科学与工程中的应用；(9）开放式虚拟现实技术与分布式仿真；(10）开放式多媒体技术与流媒体服务，包括媒体压缩、内容分送、缓存代理、服务发现与管理技术。

二、投稿要求

1．论文必须是未正式发表的、或者未正式等待刊发的研究成果。稿件格式应包括题目、作者、所属单位、摘要、关键词、正文和参考文献。

2．务必附上第一作者简历（姓名、性别、出生年月、出生地、职称、学位、研究方向等）、通信地址、邮政编码、联系电话和电子信箱。并注明论文所属领域。来稿一律不退，请自留底稿。

3．论文投稿需提交激光打印稿一式 2 份和电子版 Word 文件。论文投寄地址和电子信箱如下：

530004 广西南宁市大学东路 100 号 广西大学计算机与电子信息学院 钟诚、李陶深教授收；E-mail：dpcs2007@sina.com

三、重要日期

征文投稿截止日期：2007 年 6 月 15 日 论文录用通知日期：2007 年 7 月 10 日

四、联系方式

会议承办方：钟诚 0771-3236396，13607819333，chzhong@gxu.edu.cn；李陶深：0771-3236627，13768301390，tshli@gxu.edu.cn

专委会：南京大学计算机系 陈贵海；电话：025-58916715；E-mail：gchen@nju.edu.cn

大会网站：http://www.dpcs2007.com