一种基于密度和滑动窗口的数据流聚类算法

胡 睿 林昭文 柯宏力 马 严
(北京邮电大学 北京 100876)

摘 要 总结目前主流数据流聚类算法的优缺点后,提出了一种新的数据流聚类算法——DsStream。该算法采用双层聚类框架,应用滑动窗口技术,基于密度对数据流进行动态聚类,可以挖掘具有任意形状的数据流,且能够动态掌握数据流的分布特征。

关键词 数据流,聚类,密度,滑动窗口

DataStreams Clustering Algorithm Based on Density and Sliding Window

HU Rui LIN Zhao-wen KE Hong-li MA Yan
(Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract Summarizing the advantages and disadvantages of the current main datastreams clustering algorithms, this paper presented a new datastreams clustering algorithm——DsStream. The algorithm uses the Double-layer clustering framework, makes use of sliding window technology, clusters the datastreams dynamically based on the density. This algorithm can mine the datastreams with arbitrary shape and grasp distribution of datastreams dynamically.

Keywords DataStreams, Clustering, Density, Sliding window

近年来,许多应用中的数据都是以流的形式产生的,例如网络流、传感器数据以及网页点击流等。分析和挖掘这类数据,日益成为一个热点问题。作为一种基础的数据挖掘手段,聚类分析在数据流环境下得到了学术界和工业界的广泛关注。与传统数据不同,数据流具有如下特点:(1)数据总量的无限性;(2)数据到达的快速性;(3)数据到达次序的无约束性;(4)除非可以保存,每个元素均只能被处理一次。

数据流的上述特点对数据流上的聚类挖掘提出了如下要求:首先,算法必须能够进行实时在线挖掘,快速处理每一个元组,并实时输出挖掘处理结果。其次,相对于无限规模的数据流内存通常是有限的,算法的空间复杂度要低,往往需要在数据量的对数范围内。再次,由于实时在线挖掘以及对空间复杂度的限制,算法往往只能得到近似解,且需要具有一定的精确度保证。最后,算法要具有较强的适应性,包括对数据流不断进化的底层模型的适应性、处理离群点的能力以及挖掘任意形状簇的能力等。

2003年,Barbard总结了数据流聚类算法的要求[1],并对一些可能适用于数据流的聚类算法做了一次总结。他认为在数据流中聚类要满足3个要求:(1)压缩地表达;(2)快速、增长地处理新的数据点;(3)快速、清晰地判断独异点。

1 相关工作

2000年,Guha等人运用分治的思想提出LSEARCH[2]算法,他将到达的数据流分为不相交的块,分别对各个块中的数据运用K均值算法进行聚类。2002年,O'Callaghan提出了STREAM[3]算法。上述两种算法都能达到比较好的聚类效果,但是都只能提供对当前数据流的一种描述,而不能反映数据流的变化情况。

2003年,Aggarwal等人提出一种数据流聚类的框架CluStream[4]。他们把数据流聚类分成两个步骤:在线聚类和离线聚类。在线聚类利用一种被称为微簇的数据结构,对数据进行初步的聚类,并按一种金字塔框架(pyramid time frame)将结果以快照的形式保存下来。离线聚类则可以根据用户需求进行比较复杂的分析,从而取得最终结果。该框架非常的优秀,但是运用了BIRCH[5]的思想,所以只对球形的聚类可达到比较好的效果,对非球形的聚类不能很好的工作,且对存在噪声的数据流也得不到非常满意的效果。

2006年,Gao等人提出了一种基于密度的算法DenStream[6],该算法运用DBSCAN[7]的思想,解决了数据流中噪声的问题,且能够处理任意形状的数据。但该算法采用全局一致的绝对参数,使得聚类结果对参数十分敏感。

2 DsStream算法

针对上述算法的缺点及不足,本文提出一种新的数据流基于密度和滑动窗口的聚类算法——DsStream算法。该算法受CluStream算法的启发,分为在线和离线两个部分。在线部分获取数据流的摘要信息,应用滑动窗口技术,每隔一定时间对摘要信息和全局参数进行动态维护,以保证有效地利用有限的内存空间和消除参数敏感性。离线部分利用在线部分保存的数据流摘要信息,响应用户的查询,对摘要信息应用改进的DBSCAN算法进行再处理,以得到最终的聚类结果。

到稿日期:2010-06-14 返修日期:2010-09-15 本文受中央高校基本科研业务费项目(2009RC0502)资助。

胡 睿 主要研究方向为计算机网络应用,E-mail:phyurui@163.com;林昭文 博士,主要研究方向为IPv6及网络安全技术;柯宏力 主要研究方向为计算机应用技术;马 严 教授,博士生导师,主要研究方向为校园计算机网的建设。

2.1 基本概念

2.1.1 滑动窗口模型

现有的滑动窗口技术更新方法一般都采用即刻更新方法，即每当有新数据到达时，就进行窗口更新。

由于数据流大多与时间有关系，本文定义的模型将是基于时间的滑动窗口，而且限制内存不能存储所有当前窗口中的数据，也无法记录所有数据到达的先后次序。

定义 1(滑动窗口模型) 设当前时间为 T，窗口大小为 W，滑动窗口模型包含在$[T-W,T]$之间到达的所有数据流数据，即在任意时刻，滑动窗口模型只包含当前 W 时间段内到达的数据。

2.1.2 双层数据流聚类框架

图 1 给出了数据流挖掘模型。

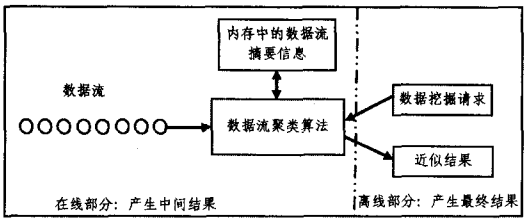


图 1 数据流挖掘模型

近几年来，数据流聚类算法逐渐发展成分层的算法框架。通常情况下，分层聚类算法将算法结构分为两个部分：在线部分和离线部分。在线部分算法负责对数据进行快速的处理，并负责保存中间结果，通常称为概要数据；离线部分算法在中间结果的基础上进行精确而复杂的分析，并得到最终结果。

2.1.3 核心微簇

定义 2(核心微簇) 一个核心微簇称为 core-micro-cluster，对于一组相互邻近的数据点 $p_{i_1},p_{i_2},\cdots,p_{i_n}$，其到达的时间为 $T_{i_1},T_{i_2},\cdots,T_{i_n}$，core-micro-cluster 定义为$\{\overrightarrow{CF^1},\overrightarrow{CF^2},n,t\}$，其中$\overrightarrow{CF^1}$表示数据点的线性和，即$\overrightarrow{CF^1}=\sum_{j=1}^{n}p_{i_j}$；$\overrightarrow{CF^2}$表示数据点的平方和，即$\overrightarrow{CF^2}=\sum_{j=1}^{n}(p_{i_j})^2$；$n$ 为核心微簇中数据点的个数；t 表示最近一个数据点进入核心微簇的时间，即 $t=T_{i_j}$。设当前时间为 T，那么 $T-t\leqslant W$，W 为滑动窗口的大小。核心微簇的中心为 $C=\frac{\overrightarrow{CF^1}}{n}$，半径为 $r=\sqrt{\frac{|\overrightarrow{CF^2}|}{n}-\left(\frac{\overrightarrow{CF^1}}{n}\right)^2}$。核心微簇必须满足$\frac{n}{\pi r^2}\geqslant\rho$，其中 ρ 为一个动态的全局参数，要受到核心微簇与孤立微簇密度的共同影响，用来标识核心微簇密度的阈值。

由于限制了密度的大小，核心微簇的半径就受到限制(不能无限增大，否则密度会被稀释)，因此核心微簇的个数一般远大于自然聚簇的个数，这样有利于最后得到更准确的聚类结果。此外，因为有密度的限制，核心微簇的个数更远小于数据流中点的个数。

2.1.4 孤立微簇

定义 3(孤立微簇) 一个核心微簇称为 outlier-micro-cluster，对于一组相互邻近的数据点 $p_{i_1},p_{i_2},\cdots,p_{i_n}$，其到达的时间为 $T_{i_1},T_{i_2},\cdots,T_{i_n}$，core-micro-cluster 定义为$\{\overrightarrow{CF^1},\overrightarrow{CF^2},n,t\}$。孤立微簇必须满足$\frac{n}{\pi r^2}<\rho$，其他定义和定义 2 中相同。

孤立微簇主要是为了保留在转变成核心微簇之前的数据信息。由于孤立微簇本身包含的点数较少，因此在计算时并不增加过多的系统负担，同时有效地保留了聚类信息，对最后得到正确的聚类结果具有重要作用。

2.1.5 成长半径

定义 4(成长半径) 我们为孤立微簇提供成长半径，以向孤立微簇提供一个成长的环境。在孤立微簇成长为核心微簇之前，孤立微簇凭此为依据，吸收新到的数据点。成长半径为一个动态变化的全局参数 ε，它的大小受到核心微簇的影响。

2.2 相关性质

性质 1 核心微簇和孤立微簇能够在线增量保持。

证明：对于一个核心微簇 $C_i=\{\overrightarrow{CF^1},\overrightarrow{CF^2},n,t\}$来说，如果有一个新的数据点 p_{i_j} 在时间 T_{i_j} 进入，那么 C_i 可以更新为$\{\overrightarrow{CF^1}+p_{i_j},\overrightarrow{CF^2}+p_{i_j}^2,n+1,T_{i_j}\}$。对于孤立微簇来说，可以推得同样的结果。

2.3 算法思想

算法采用双层聚类框架。在线部分存储当前窗口中数据的近似摘要信息；离线部分利用在线部分保存的数据摘要信息，响应用户查询，得到最后聚类结果。

2.3.1 算法框架

算法 1 DsStream(DS,W,InitNum)

```
//DS 为数据流，W 为滑动窗口大小
//全局参数：ρ 为核心微簇密度阈值；ε 为孤立微簇成长半径
init();
while(1)
  getNextPoint(p,t);
  Merge(p);
  if(t mod T==0 )
    Maintain(t);
  end if
  if( a clustering request arrives )
  DBSCAN_improved();
  End if
end while
```

1)初始化：利用 DBSCAN 算法，扫描 DS 最初 InitNum 个数据点，获取最初的一组核心微簇，并以所有核心微簇的最小密度的 1/2 作为最初的 ρ，以所有核心微簇的平均半径作为最初的 ε。

2)获取下一个数据点：于当前时间 t 从 DS 中取得下一个数据点 p。

3)处理数据点：调用 Merge(p)方法，将 p 添加到核心微簇或孤立微簇中。

4)动态维护：每过一个时间单元，调用一次 Maintain(t)方法，对现存的概要信息以及全局参数进行动态维护。

5)查询请求：如果有离线的查询请求到达，则调用改进的 DBSCAN 算法，利用概要信息生成自然微簇，即为最终的聚类结果。

2.3.2 在线部分

算法 2 Merge(p)

```
找到与点 p 距离最近的核心微簇 Ci；
计算包括点 p 在内的 Ci 的密度 ρi；
If(ρi≥ρ) then 将点 p 合并入 Ci；
else
  找到和点 p 距离最近的孤立微簇 Oi；
  计算包括点 p 在内的 Oi 的半径 ri；
  If(ri≤ε) then
```

```
    将 p 并入 O_i;
    计算包含 p 的 ρ_i;
    If(ρ_i≥ρ) then 将点 O_i 转为 C_i;
    end if
   else 利用 p 创建一个新的 O_i;
  end if
end if
```

算法 3 Maintain(T)

```
foreach C_i and O_i
  If( T−t>W) then delete C_i or O_i;
  end if
end foreach
ρ_min=Min(ρ_1^c,ρ_2^c,…,ρ_n^c);//ρ_i^c 为核心微簇 C_i 的密度
ρ_max=Max(ρ_1^o,ρ_2^o,…,ρ_n^o);//ρ_i^o 为孤立微簇 O_i 的密度
ρ=(ρ_min+ρ_max) /2;
ε=Avg(r_1^c,r_2^c,…,r_n^c);//r_i^c 为核心微簇 C_i 的半径
```

滑动窗口大小为 W，由于它是当前窗口中的数据聚类，因此需要测定是否有核心微簇和孤立微簇过时，即未包含任何当前窗口中数据。

对于参数 ε，取当前所有核心微簇的平均半径作为它的取值，以便为孤立微簇提供一个类似于当前核心微簇的成长环境。

对于参数 ρ，取当前核心微簇最小半径和孤立微簇最大半径的均值作为它的取值，这样既能反映当前微簇的密度分布，又能保持现有核心微簇和孤立微簇不变。

2.3.3 离线部分

在线微聚类保持过程中，核心微簇和孤立微簇代表了当前数据窗口中的摘要信息，离线过程通过使用这些摘要信息来得到聚类结果。在对当前窗口数据聚类的要求到达时，使用改进的 DBSCAN 算法来聚类在线保持的数据流摘要信息。

定义 5（直接密度可达） 如果一个微簇 X_i 和另一个微簇 X_j 之间的距离 $\mathrm{dist}(X_i,X_j)\leqslant r_i^r+r_j^r$，且 X_j 是一个核心微簇，则称 X_i 到 X_j 是直接密度可达的。

定义 6（密度可达） 如果存在一个微簇链 $Z_1,\cdots,Z_n,Z_1=Z_j,Z_n=Z_i$，满足 Z_{k+1} 到 Z_k 是直接密度可达的，则称 Z_i 到 Z_j 是密度可达的。

定义 7（密度相连） 如果存在一个核心微簇 Z_k，使得 Z_i 到 Z_j 都是能够到 Z_k 密度可达，则称 Z_i 到 Z_j 是密度相连的。

算法 4 ExDBSCAN()

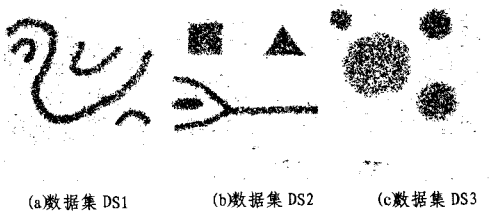
```
//改进的 DBSCAN 算法
do
  得到一个未处理的簇微 x_i;
  If(x_i 是核心微簇 ) then
    找出所有到 x_i 密度可达的微簇;
  利用它们形成一个自然簇;
  else // x_i 是孤立微簇
    break;// 跳出本次循环
  endIf
until 所有的微簇都被处理;
```

3 实验结果与分析

本节对所提出的 DsStream 算法进行性能测试。实验平台配置如下：Xeon 2.13GHz * 2，2GB 内存，Ubuntu 操作系统，算法采用 C++编写。

实验所用的数据有两种：仿真数据集和真实数据集。仿真数据集采用 DBSCAN 中 database2(DS1)，database3(Ds2)，CURE 算法中 data set1(DS3)和通过 10 次随机选择 DS1，DS2，DS3 数据集得到的进化数据流 EDS。DS1，DS2 和 DS3 的数据量都是 100000，如图 2 所示。真实数据集采用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。

流速 v 设置为单位时间有 2000 个数据点通过，滑动窗口 W 大小为 3 个时间单元。



(a)数据集 DS1　(b)数据集 DS2　(c)数据集 DS3

图 2 类数据集

3.1 聚类形状

DsStream 算法是基于密度的算法，这里应用仿真数据集 EDS 测试 DsStream 算法。图 3 是对进化数据流 EDS 在不同时刻的聚类效果实验结果。从图中可以看出，DsStream 算法能够发现任意形状的聚类，屏蔽异常数据的影响。

t1 时刻　t2 时刻　t3 时刻

图 3 DsStream 算法聚类结果

3.2 聚类质量

聚类质量定义为该数据集中数据点聚类正确的比例。图 4、图 5（横轴为正确率，纵轴为不同时间单元）分别显示了在仿真数据集和真实数据集下与 DenStream 算法的质量比较结果。真实数据集选择其中 20 个连续值属性维，仿真数据集为 EDS。由于采用了动态的全局参数维护策略，因此 DsStream 算法的聚类质量比 DenStream 算法好。

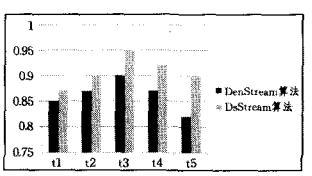


图 4 两种不同算法对 EDS 的聚类结果

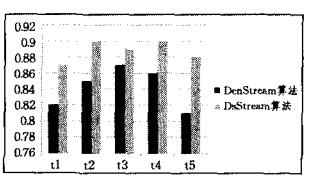


图 5 两种不同算法对真实数据集的聚类结果

3.3 执行效率

实验选取 DenStream 算法作为对比算法，图 6（横轴为流量，单位 10000，纵轴为执行时间，单位为 s）显示了在网络入侵检测数据集 KDD-CUP-99 下 DsStream 和 DenStream 算法执行效率的比较结果。可以看出，DsStream 算法执行效率还是比较高的，比 DenStream 慢不了多少。

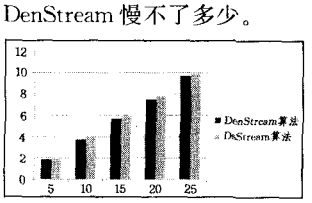


图 6 两种不同算法的执行速度

结束语 本文提出一种有效的聚类进化数据流的算法

DsStream。算法既保持了基于密度的聚类算法可以发现任意形状的聚类和对噪声数据不敏感的优点，又能够动态地调整全局参数，故算法是参数不敏感的。算法应用了滑动窗口技术对数据进行动态更新，并提供在线维护策略，从而能够在保障聚类结果尽可能精确的基础上有效地控制内存的消耗。

参 考 文 献

[1] Barbar D. Requirements for Clustering Data Streams[J]. SIGKDD Explorations, 2003, 3(2): 23-27

[2] Guha S, Mishra N, Motwani R, et al. Clustering Data Stream[C]// FOCS 2000. 2000: 359-366

[3] O'Callaghan L, Mishra N, Meyerson A, et al. Streaming-Data algorithms for high-quality clustering[C]// ICDE Conf. 2002: 685-704

[4] Aggarwal C C, Han J, Wang J, et al. A framework for clustering envolving data streams[C]// VLDB. 2003: 81-92

[5] Zhang T, Ramakrishnan R, Livny M. BIRCH: An Efficient Data Clustering Method for Very Large Databases[C]// ACM SIGMOD Conference. 1996: 103-113

[6] Cao Feng, Ester M, Qian Weining, et al. Density-based clustering over an evolving data stream with noise[C]// 2006 SIAM Conference on Data Mining. 2006: 326-337

[7] Ester M, Kriegel H P, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databased with noise[C]// Proc of KDD. 1996

[8] Chang Jianlong, Cao Feng, Zhou Aoying. Evolving datastreams clustering based on the sliding window[J]. Journal of Software, 2007, 17(4): 905-918

[9] Udommanetanakit K, Rakthanmanono T, Waiyamai K. E-Stream: Evolution-based Technique for Stream Clustering[C]// Proceedings of ADMA. Berlin Heidelberg: Springer-Verlag, 2007: 605-615

(上接第 126 页)

对于不存在 commit 的逻辑程序而言，做出这两次选择已经足够。对于出现了剪枝算子的程序，则不仅同样需要做出这两次选择，而且剪枝算子还引入了计算过程中新的不确定性：给定搜索树 T_i，首先需要选择下一步操作是扩展还是剪枝。在选定剪枝操作后，还需要决定对哪棵子树进行剪枝，即选择剪枝的依据并选择截断点。

值得注意的是，这些选择的具体实现可以有不同的方式。通常，在逻辑程序语言的实现中，采用积极的剪枝策略，这意味着在 commit 的范围内尽可能地进行搜索树的修剪，以达到提高程序效率的目的。这里，给出的积极剪枝操作实现策略如下[8]：

(C0) 若允许剪枝，则选择剪枝操作，否则选择扩展操作；

若(C0) 为扩展步骤，则

(C1) 选择最新建立的非空结点进行扩展(深度优先)；

(C2) 根据计算规则，选择结点中的最左非延迟子公式进行扩展；

若(C0) 为剪枝步骤，则

(C3) 选择剪枝的依据 J；

(C4) 选择 J 的所有截断点。

图 4 给出了 Gödel 语言控制机制的简要示意图，略去了各子过程的实现算法。该算法的思想已用于一个 Gödel 语言编译器雏形的研发，其中细节如包含动态类型的合一处理、延迟检查仍在进一步优化中[10]。

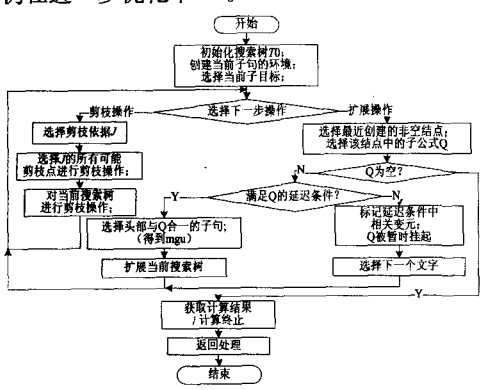


图 4 Gödel 语言控制机制示意图

结束语 逻辑程序设计有别于命令式程序设计，其特点在于计算存在不确定性和一致匹配的计算方式。其中，不确定性分为“不知不确定性”和“无关不确定性”。前者指的是通过回溯进行计算求解，后者可以通过剪枝操作去除无关的搜索空间。剪枝操作对提高逻辑程序的执行效率非常重要。在逻辑语言的发展中，Gödel 改进了传统逻辑语言中的语义问题，设计了一个更具说明性的剪枝算子，提供了 3 种不同的 commit 形式以满足不同的需要。剪枝算子和谓词的延迟声明结合更具有实用价值，事实上就是已知了谓词中参数的使用模式，能够进行更有效的剪枝。本文引入了带 commit 的逻辑程序及其求解目标的搜索树的定义，对剪枝操作的执行剪枝步骤作出了规定，为剪枝算子的实现提供了依据，并结合带有延迟声明的计算规则集合，讨论了 Gödel 的控制机制，为语言的编译实现提供了参考。

参 考 文 献

[1] 刘椿年，曹德和. Prolog 语言，它的应用与实现[M]. 北京：科学出版社，1990

[2] Saraswat V A. Concurrent Constraint Programming Languages [M]. MIT Press, 1993

[3] Hill P M, Lloyd J W. The Gödel programming Language[M]. London: MIT Press, 1994

[4] Lloyd J W. Foundation of Logic Programming (2 ed) [M]. Springer-Verlag, 1987

[5] Maluszynski N. Logic, Programming and Prolog (2 ed) [M]. John Wiley& Sonds Ltd, 1995

[6] Börger E. Logic+Control Revisited: and Abstract Interpreter for Gödel programs[M]. Levi G. editor. Advance in Logic Programming Theory. Oxford Univ. Press, 1994

[7] Jeffery D. Expressive Type Systems for Logic Programming Language[D]. 2002

[8] Brogi A, Buarino C. Pruning the Search Space of Logic Programs [J]. Lecture Notes in Computer Science, 1996, 1050: 35-49

[9] Lee Naish. Pruning in logic programming [R]. 95/16. Melbourne, Australia: Department of Computer Science, University of Melbourne, 1995

[10] Li Hui-qi, Zhao Zhi-zhuo. A Polymorphic Type System in Logic Programming[C]// Proceedings of 2008 3rd International Conference on Intelligent System and Knowledge Engineering (ISKE2008). 2008: 125-130