计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(3):169−177 [doi: 10.15888/j.cnki.csa.008370]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

工业物联网数据流自适应聚类方法①

朱维富[1,2,3], 曾智霞[1,3], 肖如良[1,2,3,4]

[1](福建师范大学 计算机与网络空间安全学院, 福州 350117)
[2](福建省应用数学中心 (福建师范大学), 福州 350117)
[3](福建师范大学 数字福建环境监测物联网实验室, 福州 350117)
[4](福建师范大学 福建省网络安全与密码技术重点实验室, 福州 350007)
通信作者: 肖如良, E-mail: xiaoruliang@fjnu.edu.cn

摘　要: 5G 通讯技术的迅猛发展使工业物联网得到了全面提升, 工业物联网数据规模将越来越大、数据维度也越来越高, 如何高效利用流聚类进行工业物联网数据挖掘工作是一个亟需解决的问题. 提出了一种基于工业物联网数据流自适应聚类方法. 该算法利用微簇之间的高密性, 计算各微簇节点的局部密度峰值以自适应产生宏簇数; 采用引力能量函数对微集群进行递归在线更新; 并且去除边缘相交微簇之间的计算以达到降低维护宏簇所需的计算量. 理论分析和实验对比表明所提出的方法跟当前主流的流聚类算法相比有着更高质量的聚类效果.

关键词: 工业物联网; 流数据; 自适应流聚类; 微簇

引用格式: 朱维富,曾智霞,肖如良.工业物联网数据流自适应聚类方法.计算机系统应用,2022,31(3):169–177. http://www.c-s-a.org.cn/1003-3254/8370.html

Adaptive Clustering Method of Industrial Internet of Things Data Stream

ZHU Wei-Fu[1,2,3], ZENG Zhi-Xia[1,3], XIAO Ru-Liang[1,2,3,4]

[1](College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350117, China)
[2](Fujian Provincial Center for Applied Mathematics (Fujian Normal University), Fuzhou 350117, China)
[3](Digital Fujian Environment Monitoring IoT Laboratory, Fujian Normal University, Fuzhou 350117, China)
[4](Fujian Provincial Key Lab of Network Security and Cryptology, Fujian Normal University, Fuzhou 350007, China)

Abstract: The rapid development of 5G communication technology has led to a comprehensive enhancement of the industrial Internet of Things (IIoT). The scale of IIoT data will become larger, and the dimensionality of data will become higher. As a result, how to efficiently use stream clustering for IIoT data mining is an urgent problem. In this regard, this study proposes an adaptive clustering method of an IIoT data stream. The algorithm exploits the high density between micro-clusters and calculates the local density peaks of each micro-cluster node to adaptively generate the number of macro-clusters. It employs a gravitational energy function to recursively update the micro-clusters on line and removes the computation between edge-intersecting micro-clusters to achieve a reduction in the computational effort required to maintain the macro-clusters. The theoretical analysis and experimental comparison show that the proposed method has higher-quality clustering results than the current mainstream stream clustering algorithms.

Key words: industrial IoT (IIoT); stream data; adaptive stream clustering; micro-cluster

5G 技术的发展促使工业物联网得到全面提升, 使全球加速进入第 4 次工业革命时代[1]. 工业物联网在提升生产效率的同时, 会产生海量、超高维、复杂异构的实时流数据, 业界称之为工业数据流[2−4]. 这些流数据不能再做静态数据假设, 数据量大实时性强, 又必须在有限内存内处理[5,6], 从而工业物联网所产生的海量

① 基金项目: 国家自然科学基金 (61772004); 福建省科技计划重大项目 (2020H6011); 福建省自然科学基金 (2020J01161)
收稿时间: 2021-05-10; 修改时间: 2021-06-14; 采用时间: 2021-06-30; csa 在线出版时间: 2022-01-24

实时数据给数据流聚类分析带来了巨大的挑战.

近年来, 数据流聚类已经成为了工业物联网数据挖掘分析的关键性研究领域, 其目的是为了识别无界且无序观测流中的模式[5,7,8]. 数据流聚类技术通常是采用在线和离线两个阶段. 在线阶段主要通过优化数量和更新簇的位置, 以便更好地表示底层数据; 在离线阶段提取相关信息, 而不对数据进行存储并且重新评估所有的结果. 数据流聚类技术发展至今, 主要可以分为这 3 类: 基于距离的流聚类方法、基于网格的流聚类方法、基于预测的流聚类方法.

第 1 类: 基于距离的流聚类方法. 该类方法是最流行的方法, 它通过简单的插入规则来构建簇, 基于概要数据结构来总结与簇相关的观测值, 而不存储每个单独的数据点. 最具有代表性的是: CluStream[9], DBSTREAM[10] 以及 BOCEDS[11] 等. 2003 年, Aggarwal 等人提出了 CluStream 算法[9]. 其算法核心思想主要是通过在线阶段利用微族的概要存储结构存储数据流的汇总结果, 并按金字塔式时间结构将中间结果进行保存; 而其离线部分则根据用户指定的观察阶段及聚类数量, 快速生成聚类结果. 在 2016 年, Baer 等人提出了 DBSTREAM 算法[10]. 该算法采用了微簇之间密度共享机制来确定所属宏族, 该机制利用保持微族的共享密度来作为它们之间的半径, 并且在离线阶段具有高共享密度的微簇归属于同一个宏族. 在 2019 年 Islam 等人提出了 BOCEDS 算法[11], 它仍然采用了概要技术存储微簇信息, 该算法主要增加了一个缓存机制来处理数据流演化以及漂移问题.

第 2 类: 基于网格的流聚类方法. 该类方法通过网格将数据空间沿各维度进行分隔, 以创建多个网格结构. 通过将数据点映射到单元格, 可以保持密度估计[8]. 最具有影响力的代表性算法如: 2007 年 Chen 等人提出的 D-Stream 流聚类方法[12], 它将新的数据点映射到所属单元格中, 并通过将所有密集单元格分配给单个簇进行初始化, 并经相邻网格扩展; 还有 2014 年 Amini 等人所提出的 HDCStream[13] 以及其在 2016 年提出的 Mudi-Stream[14] 等. 基于网格的流聚类方法可以识别任意形状的簇, 这也是该方法所具备的重要特征.

第 3 类: 基于预测的流聚类方法. 该类方法是一种高维数据流的流聚类方法. 该类方法可以在数据流维度十分大的空间中识别簇, 从而为高维数据流提供了一个利基. 该类方法最具有代表性的是: HPStream[15]、HDDStream[16] 等. 2004 年 Aggarwal 等人提出了 HPStream 算法[15], 它基于 CluStream 算法[9] 在高维上进行了扩展. 通过定期采样当前簇的标准差, 调整现有的聚类, 使每个维度标准化, 通过更新簇分配每个簇的关联维度. 在每个簇中暂定添加一个新的观测点, 以更新维度. 如果聚类数据点不超过阈值则不增加聚类半径, 且将其添加到最近的聚类中.

以上方法各有优点: 基于距离的流聚类算法通常计算负荷低、基于网格的流聚类算法可以识别任意形状的簇; 基于预测的流聚类算法能有效应对维度灾难. 但是也存在如下的缺陷: 基于距离的流聚类算法往往依赖于参数的设定, 基于网格的聚类算法却由于网格是动态确定的而增加了计算负荷, 而基于预测的流聚类算法增加了宏族选择子空间的复杂性.

面对着内存受限、高维度灾难、低聚类质量以及低处理速度数据流特性来说, 目前数据流聚类研究主要存在着以下 3 个方面的困难:

(1) 对于持续、快速、以及高维数据流来说, 数据源源不断地流入, 导致微簇数量的持续性增加, 聚类方法中隐含高负荷剪枝操作.

(2) 聚类簇数的确定一直是流聚类方法中的困难问题, 同时聚类簇数也对聚类质量有非常大的影响.

(3) 生产环境是开放的, 所产生的数据流会不断演化, 许多数据流聚类方法未能将一些离核心微簇较远、信息量较少的微簇进行异常检测.

针对工业物联网实时数据流的上述挑战问题, 在我们小组已有的基础性工作[17] 的基础之上, 本文提出了一种新的工业物联网数据流自适应聚类算法 (简称 MCStream). 本文的方法与现有的方法完全不同, 第一, 在处理海量高维数据的聚类上, 目前基于密度的流聚类算法对于参数值的变化很敏感, 往往由于参数的细微变化在很大程度上影响了聚类质量; 第二, 在动态数据聚类过程中, 存在着大量数据的插入以及移除操作, 进一步地引起大规模的交叉微簇连接操作, 目前基于密度的流聚类算法未能有效的解决这样复杂计算问题. 本文算法引入一种新的引力能量函数的方法对参数进行递归的更新操作, 很好地解决了参数敏感性问题, 并且通过高斯核函数计算微簇密度峰值方式代替了大规模交叉微簇连接操作, 进一步地加快了数据流聚类映射速度. 因此本文的主要贡献如下:

(1) 提出了一种新的微簇构建方法. 该方法采用引

力能量更新函数, 对微集群进行递归在线更新; 同时取消交叉微簇连接操作, 从而达到微簇构建与数据映射实时响应.

(2) 提出了一种新的自适应计算聚类簇数的方法. 该方法以微簇作为参与宏簇聚类的样本数据, 利用各微簇的局部密度以及距离值, 计算微簇的密度峰值, 并通过这两个变量值来自适应求出聚类簇数, 从而更好地处理大规模数据.

(3) 构建了一种新的检测异常微簇的判定方法. 该方法采用一个局部密度上界来标识宏簇, 以区分密集簇和离群簇, 使异常族检测更加精确.

1 运动目标检测工业物联网聚类框架及基本概念

1.1 工业物联网聚类框架

工业物联网数据流的自适应聚类方法是工业物联网时代信息处理要解决的基本问题. 本文构建了工业物联网数据流挖掘总体框架. 它主要分为: 数据收集层, 数据挖掘层以及数据应用层, 如图 1 所示. 由于工业物联网数据量巨大、协议标准众多、安全性考虑不足等缺陷, 数据收集层大多采用传统传感器技术来获取数据, 如 GPS、Network Flow 以及海量信息化数据等[4,18,19]. 数据挖掘层接收来自数据收集层中的数据, 以流的形式传入处理器中; 在数据挖掘层中采用的是先利用本文提出的工业物联网自适应聚类技术进行聚类分析, 再将聚类结果集通过数据传输存储到总服务器中. 在数据应用层中, 数据存储总服务器通过类别分别传入到对应的应用上, 从而达到分布式管理. 对数据达到针对性应用. 在对工业物联网自适应聚类算法详细介绍之前, 我们先对工业物联网数据流及其相关概念进行介绍.

1.2 基本概念

定义 1 (工业数据流).设 D 是一个工业数据流: $D=\{x_i\}_1^{\infty}$, 其中, $\{x_i\}$代表着在 t 时刻到达的一个数据样本, $\{x_i \in R^d\}$代表着这个数据样本点是一个d维向量.

定义 2 (概念漂移). 设x为特征向量, 其中$P_t(l|x)$是x在 t 时刻的条件分布, 随着时间 t 的推移, x的条件分布满足$\exists x: P_{t0}(l|x) \neq P_{t1}(l|x)$. 即聚类统计特性正在以不可见的方式进行着变化, 从而聚类精度不断地降低.

定义 3 (微簇结构). 设mc表示一个微簇, 该微族定义为一个六元组$mc=(CN_t, N_t, SN_t, C, \theta, W)$, 其中:

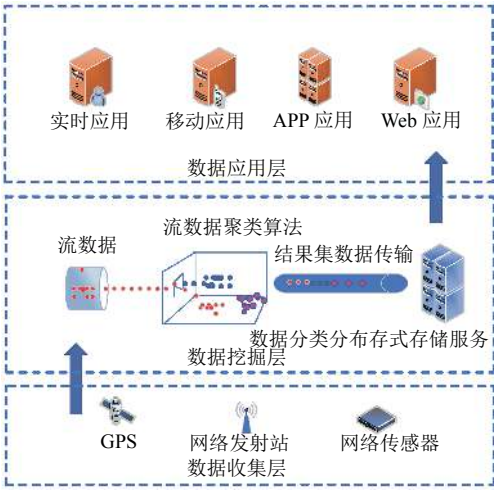

图 1 工业物联网聚类框架

(1) CN_t表示每个微簇核心区数据点数, 微簇核心区是指在距离微簇中心 $r/2$ 范围内的区域.

(2) N_t表示在 t 时刻数据点到来时相应微簇的数据点数, t+1 时刻使用式 (1) 更新.

$$N_{t+1}=N_t+1 \tag{1}$$

(3) SN_t表示微簇壳区总样本点数, 微簇壳区表示距离微簇核心 r 处边缘; 壳区数据点数用式 (2) 计算. 每个微簇分为核心区和壳区两部分.

$$SN_t=N_t-CN_t \tag{2}$$

(4) C 为微簇中心, 表示微簇在空间中的位置. C_t^k表示在 t 时刻聚类中心 C 的第 K 维所对应的值; x_t^k表示在 t 时刻到来的数据点 X 第 K 维的值; 随着时间推移, 数据点不断到来, 微簇中心 C 会进行更新改变, 如式 (3) 所示.

$$C_{t+1}^k=\frac{(SN_t-1)\times C_t^k+X_{t+1}^k}{SN_{t+1}} \tag{3}$$

(5) 微簇的衰减因子我们使用θ表示, 表示单位时间内到达的数据点数目, 当微簇需要进行微簇更新时, 每更新一次权重都会进行一次衰减因子的更新.

(6) W_t表示微簇的权重值, 采用引力能量函数进行递归在线更新; $dis(x_{t+1}, C_t)$表示在 t+1 时刻到来的数据点距离所属微簇 C 的簇中心距离值; r 表示微簇半径. 每个微簇的生存或者死亡都是依赖于微簇的权重, 权重值等于或者小于 0 的微簇都不会参与到聚类当中, 其中微簇权重更新使用式 (4) 进行更新.

$$W_{t+1}=W_t-\left\{\frac{r-dis(x_{t+1}, C_t)}{r}\right\}\times\frac{1}{\theta} \tag{4}$$

定义 4 (核心簇). 核心簇用 CoreClusters 表示, 在 t 时刻的核心簇中, 簇总数据点数$N_t > minPts$最小密度值), 微簇权重$W_t > 0$, 它存储在主存储器当中.

定义 5 (潜在核心簇). 潜在核心簇用 pCoreCluster 表示, 在 t 时刻的潜在核心簇中, 簇总数据点数$N_t < minPts$微簇权重值$W_t > 0$, 它代表目前这个簇由于数据点不足而无法构成核心簇, 但在未来时间内有可能随着数据点的增加而成为核心簇.

定义 6 (离线缓冲簇). 离线缓冲簇用 OBuffer-Cluster 表示, 在 t 时刻的离线缓冲簇中, 簇总数据点数$N_t > minPts$, 微簇权重$W_t < 0$, 它存储在缓冲存储器当中, 它表示核心簇由于随着时间的增加权重逐渐降至为 0 之后变成了离线缓冲簇, 但它并不是就没有关联了, 有可能在未来某一段时间它会随着数据点的到来再次变为核心簇.

定义 7 (微簇局部密度). 微簇局部密度P_{mi}是簇群中与 CoreCluster 之间的距离小于截断距离$d_{\min}$的微簇. 其中d_{ij}表示微簇 i 和微簇 j 中心点间的欧式距离. 局部密度P_{mi}的计算方式可使用一种高斯核函数来进行计算, 见式 (5).

$$P_{mi} = \sum_{j \in CoreCluster \backslash \{i\}} \mathrm{e}^{-\left(\frac{d_{ij}}{d_{\min}}\right)^2} \tag{5}$$

定义 8 (微簇距离). 基于定义 7, 对微族局部密度进行排序, 当微族q_i的局部密度最大时, 微簇距离δ_{qi}的距离是微簇群中与q_i最大的距离$\max\{\delta_{qj}\}$, 否则表示簇群中所有微簇局部密度大于q_i的微簇中与q_i最近距离的微簇$\min\{d_{qi}d_{qj}\}$. 从而微簇距离δ_{qi}可用式 (6) 计算.

$$\delta_{qi} = \begin{cases} \min\{d_{qi}, d_{qj}\},\ j < i,\ i \geqslant 2 \\ \max\{\delta_{qj}\},\ j \geqslant 2,\ i = 1 \end{cases} \tag{6}$$

在上述定义当中, 微簇中心是通过计算壳区域数据点的平均值, 而不是通过计算整个簇数据点的平均值, 这是因为他们通过限制微簇的移动来阻止微簇无休止地跟随数据流的漂移[20], 在聚类过程中, 只有核心簇参与到簇聚类, 对于离群簇则会进行移除操作, 当微簇 CoreCluster 具有最大局部密度时, δ_{qi}表示在核心簇 CoreClusters 中与 CoreCluster 距离最大的微簇点与 CoreCluster 之间的距离; 否则, δ_{qi}表示在所有局部密度大于 CoreCluster 的微簇点中, 与 CoreCluster 距离最小的那些微簇与 CoreCluster 之间的距离.

2 工业物联网数据流自适应聚类方法

本文算法主要分为 6 个阶段, 分别为: 初始化微族, 微族映射, 更新微族, 移除离群族, 构建微族决策树, 更新族图.

2.1 初始化微簇

在数据流 D 中第一个数据点到来时候, 它不属于任何簇结构, 因此将创建一个微簇来存储信息. 这一步与之后的新的微簇创建同时发生. 微簇的创建首先要初始化特征向量. 新簇的簇中心 C 和半径 r 定义了微簇在数据空间中位置以及覆盖范围; 簇中心 C 初始设置为x_i, 衰减因子θ是根据相关应用程序的专家知识所设置的, 壳数据点数据以及微簇数据点数都设置为 1. 这些值被记录以用来对微簇中心的递归更新, 权重 W 值是用来确定微簇群的时间长度; 它是使用一个时间衰减函数来进行递归更新, 这在后面详述. 当数据点到来之时, 它会判断是否存在簇结构; 如果不存在, 则会创建一个微簇结构.

2.2 微簇映射

在任意 t 时刻, 数据流$D = \{x_i\}_1^{\infty}$中数据点x_i到达的时候, 该算法首先将计算数据点x_i与微簇的欧式距离; 如果距离 dis 满足式 (7), 则将数据点映射到所属微簇当中. 数据点有可能映射 3 种微簇集合当中.

$$dis(x_i, clusters) \leqslant r \tag{7}$$

第 1 种就是参与集群聚类的核心簇 (CoreClusters), 第 2 种就是潜在核心簇 (pCoreCluster), 第 3 种就是存储在缓冲器当中的离线缓冲簇 (OBufferCluster). 为了找到目标微簇, 该算法首先计算核心簇与数据点x_i的欧式距离. 如果满足式 (7), 则将数据点映射到核心簇当中, 并记录该核心簇索引. 如果在核心簇中没有找到, 则同寻找核心簇方法一样在另外两种微簇集群当中进行映射. 如果数据点x_i即满足核心簇的映射条件也满足其他一种或者两种核心簇, 则选择将该数据点映射到核心簇当中.

2.3 微簇的更新

在图像预对于集群当中任何微簇结构. 任何一个微簇集群只要接收到了新数据, 那么该微簇结构的概要信息都会进行更新. 如果在 t 时刻数据点属于核心簇 CoreCluster, 那么将会使用式 (1) 更新微簇数据点的数量; 如果它是映射到核心簇壳区域, 那么会使用式 (3) 更新映射微簇中心. 这种簇中心更新机制是为了防止微簇集群由于数据点的增加而出现数据漂移, 并且

会使用式 (2) 更新壳区数据点的数量. 使用式 (4) 更新微簇权重.

$$0 \leqslant dis \leqslant r/2 \tag{8}$$

如果数据点x_i^t是映射到核心区域, 则不会进行簇中心的更新. 如果数据点x_i^t映射到潜在核心簇 pCoreCluster, 那么同核心簇一样会对所映射的 pCoreCluster 概要信息进行更新; 并且会对 pCoreCluster 微簇数据样本点进行判断, 看是否满足$N_t > minPts$. 如果满足, 则将该映射 pCoreCluster 移入到核心簇当中, 并从潜在核心簇去除该簇. 如果数据点映射到离线缓冲簇当中, 那么在同核心簇一样更新微簇概要之后还会将该簇从离线缓冲簇中移除; 并将它移入核心簇当中, 这是由于数据流的演化特性. 由于在数据演化过程中微簇权重可能会越来越低, 当它权重低于 0 则会与当前数据流无关; 但是在未来某一段时间有可能会随着新的数据的到来而重新相关; 因此它会被重新移入核心簇当中参与集群聚类.

2.4 移除离群簇

伴随着流数据的不断流入, 在数据点不断地映射到微簇群之后; 该算法会对所有簇群中的微簇进行权重更新. 在核心簇、潜在核心簇、离线缓冲簇中, 由于有些微簇持续性没有新的数据映射进来; 其微簇权重都会不断地降低, 这也体现着数据流演化特性. 对于核心簇群来说, 每个核心微簇如果权重小于 0, 那么该微簇将会从核心簇当中移入缓冲簇中; 并且会对其能量设置为原有初始能量的一半. 而在对潜在核心簇以及离线缓冲簇来说; 如果长时间的没有新的数据对其微簇进行更新, 那么说明这些微簇跟数据流内容长期无关并且是正在消亡的微簇. 对于流式算法来说低内存是其中一个不可或缺的评价标准; 因此这些正在消亡的微簇就需要从内存中永久性的移除.

2.5 构建微簇决策树

当核心簇以及其他两个簇群发生改变时, 该算法会进行维护聚类图操作. 而对于聚类我们都需要保证聚类中心的密度最大, 以及各个微簇聚类中心的距离相对较远. 只有这样才能让各簇之间区分明显, 并且自适应的确定宏簇聚类簇数.

在微簇进行更新之后. 为了获取根据核心簇的数据特性而自适应的求出微簇所需要聚类簇数; 我们首先需要计算核心微簇群当中所有微簇之间相互的欧式距离值. 通过计算这个距离值可以构建一个核心微簇的距离矩阵. 在获取到距离矩阵后我们通过对这个距离进行排序以此来获取一个截断距离 *dmin*; 其中等于比 *dmin* 更接近核心簇 *i* 的数据点数. 由于这个值的选择跟核心簇中不同的微簇距离相关, 因此 *dmin* 的选择是鲁棒性的. 通过式 (5) 可知, 与微簇 *i* 之间的距离小于 *dc* 的越多, 那么与微簇 *i* 的局部密度就越大. 当微簇 *i* 为核心簇局部密度最大的点时, 我们通过计算与微簇 *i* 距离最大的点的距离; 而对于其他局部密度的微簇, 则通过计算比它局部密度更大的点中距离最小的微簇之间的距离; 通过这两个值, 我们可以直观的反应聚类中心的特性. 局部密度大以及各聚类中心之间相差很远.

2.6 更新簇图

在簇图更新中, 我们在式 (5) 得到了微簇局部密度以及距离. 我们将每一个参与聚类的微簇作为一个数据点. 首先将每个微簇的局部密度以及距离进行归一化处理; 然后将他们的乘积 λ 作为以微簇为聚类点的聚类中心的评判标准. 如果微簇 $C[i]$ 是聚类中心, 并且属于第 *k* 簇, 那么将其宏簇属性设定为 *k*; 如果不是微簇, 那么将其属性赋值为−1. 在所有聚类中心确定之后, 我们需要对非聚类中心微簇进行归类操作. 这里是按照密度值从大到小的顺序进行的遍历; 这样做可以逐层的扩充每一个微簇. 微簇通过这两步操作, 即完成了聚类中心及微簇的归类操作. 我们进一步将每个宏簇分为 cluster core 以及 cluster halo 两类; 这里是通过属性来进行标识. 对于 cluster core, 是指那些局部密度较大者; 而对于 cluster halo 是指那些局部密度较小的, 我们通过了为每一个宏簇设定一个局部密度平均上界来确定 cluster core 以及 cluster halo. 这样能将离聚类中心以及信息量较少的微簇进行标识.

3 相关理论分析

3.1 时间复杂度

算法时间的消耗主要来源于数据点的映射以及簇图的聚类过程, 假设在T时刻M维数据流产生了N个微簇, 数据点映射时间与微簇数以及维度呈正相关关系, 因此在数据点映射的时间复杂度为$O(MN)$, 在簇图聚类过程中, 假设产生了d个核心簇, 产生了K个类, 所需时间复杂度为$O(d^2)$, 因此本文算法整体时间复杂度为$O(MN)+O(d^2)$.

3.2 空间复杂度

该算法内存消耗主要来源于微簇的存储以及簇图

的存储, 维度为 M 的数据流产生了 N 个微簇且其中产生了 C 个核心簇, 在簇图聚类过程中, d 个核心簇在簇图聚类过程中聚成了 K 个类, 因此本文算法整体空间复杂度为 $O(N+d)$, 且 $d \ll N$.

3.3 微簇数量上界值分析

进行对于任何聚类算法来说, 空间复杂度以及时间复杂度两个值都决定着聚类的优劣. 而对于流聚类算法来说, 微簇数量的边界决定着算法运行速度以及空间存储需求, 因此我们针对本文算法分析了在每一个周期内微簇数量 M 的边界值.

对于每一个时间周期内, 微簇数量 M 的产生总是满足:

$$M \leqslant \frac{3\theta}{2minPts}$$

证明: 在任意的一个时间周期上, 衰减因子 θ 是从数据流中在一个单位时间内到达的数据点数, 由于数据点最新映射到潜在核心簇中, 但参与聚类的簇为核心簇, 潜在核心簇到核心簇满足条件至少需要 $minPts$ 个数据点, 因此在一个时间内核心簇的最大数量为 $\frac{d}{minPts}$, 从而核心簇在一个周期内满足 $0 \leqslant C \leqslant \frac{\theta}{minPts}$ 的微簇数, 由于离线缓冲簇是来自核心簇, 且权重比为核心簇的 1/2, 因此离线缓冲簇的数量跟核心簇的数量成正比, 从而离线缓冲簇在一个周期内产生的微簇数满足:

$$0 \leqslant C\_of\_c \leqslant \frac{\theta}{2minPts}$$

因此, 对于一个时间周期内, 该算法所产生的微簇数量 M 为 C_of_c, 即:

$$M \leqslant \frac{3\theta}{2minPts}$$

3.4 微簇低延迟处理

假设 W_t 为某一时间窗口, C_t 为当前时间, T_s 为 $C_t - W_t$ 时间之前任意序列中最后一次存储微簇概要信息的时间, 那么 $C_t - T_s \leqslant 2 \cdot W_t$.

证明: 设 δ 是最小整数, β 是一个整数且 $\beta \geqslant 1$, β^{δ} 表示第 δ 次存储微簇概要信息时间间隔使得 $\beta^{\delta} \geqslant W_t$. 那么 $\beta^{\delta-1} \geqslant W_t$. 因为可以知道存在序列为 $(\delta-1)$ 的 β 微簇概要信息, 则在 $C_t - W_t$ 之前必须始终存在至少一个序列为 $(\delta-1)$ 的微簇概要信息, 让 T_s 是发生在 $C_t - W_t$ 之前的 $(\delta-1)$ 微簇概要信息, 那么 $C_t - W_t - T_s \leqslant \beta^{\delta-1}$. 从而满足:

$$C_t - T_s \leqslant W_t + \beta^{\delta-1} < 2 \cdot W_t$$

4 仿真实验与分析

为验证本文所提出流聚类方法的先进性, 下面将与当前已发表的同类前沿方法进行对比试验, 针对各项性能进行评测. 所设定的实验 PC 硬件环境为: RAM 12 GB, 主频 2.4 GHz; 操作系统 Windows 10 专业版, 语言平台: Python 3.6. 我们使用了多个数据集来仿真工业物联网中海量传感器数据. 在快速处理、有限内存的约束下, 流聚类技术的高聚类质量是我们的追求目标, 而聚类纯度、聚类精度是评价聚类质量的重要指标, 因此我们拟采用与目前基于密度的聚类算法分别从数据点处理速度, 聚类纯度以及算法内存消耗 3 个方面进行对比, 并设计了 3 组实验.

根据以上 3 个方面, 所进行的 3 组实验如下.

实验 1. 测试不同数据流长度对各类密度聚类算法处理能力进行对比.

实验 2. 比较不同聚类算法的平均聚类纯度.

实验 3. 设置不同衰减因子, 测试 CMStream 算法从低维到高维数据流处理的响应时间.

工业物联网所产生的数据数量庞大, 超高维度, 因此为了更好验证 CMStream 算法在工业物联网环境的性能, 我们使用了 3 个海量、高维的数据集, 为了让数据集仿真真实环境, 本文通过时间窗口的形式将静态数据集进行动态化模拟测试.

(1) KDDCUP'99: 数据集包含 4 898 431 个实例, 每个实例包含着 41 维向量, 它产生于现代工业的网络流量记录.

(2) Bag of Words: 数据集包含着 8 000 000 条实例, 每个实例包含着 100 000 维向量, 它从收集于文本数据集.

(3) EPM: 数据流包含着 230 318 条数据集, 每个数据集 13 维向量, 它是一个学习分析数据集.

4.1 对比算法

(1) CEDAS[20]: 2016 年 Hyde 等人提出了 CEDAS 算法. 该算法是一种基于完全在线的算法将演化数据流聚成任意形状的簇, 它主要分为两个阶段, 一个是微簇维护阶段, 一个是宏簇聚类阶段.

(2) DBCLPG[21]: 2019 年 Halim 等人提出了 DBCLPG 算法. 该算法是基于密度的大概率图聚类, 与其他算法不同的是, 它在聚类过程是利用节点度以及邻域信息为引导, 通过图的密度对大概率图进行聚类.

(3) BOCEDS[11]: 2018 年 Islam 等人提出了

BOCEDS 算法. 该算法是一种基于缓冲区的演化数据流在线聚类方法, 在 CEDAS[20] 的基础上提出了一个缓存机制来存储无关微簇以及从这个缓冲区提取暂时无关微簇的在线剪枝聚类算法.

(4) microTEDAclus[22]: 2019 年 Maia 等人提出了 microTEDAclus 算法. 该算法是基于典型性混合的进化聚类算法, 基于 TEDA 框架所提出来的, 将聚类问题划分为两个子问题, 一个是微簇构建, 一个是微簇进化成宏簇.

4.2 聚类处理速度对比

为了探究在随着数据集中数据点不断流入的情况下 MCStream 算法聚类处理速度, 以及对于在同样数据集长度下不同维度数据集的聚类处理速度, 在实验过程中, 我们分别与当前最前沿的多个方法如: CEDAS[20] 方法 DBCLPG[21] 方法 microTEDAclus[22] 方法以及 BOCEDS[11] 方法在不同维度的数据集上进行了对比. 我们在数据集 EPM 以及 KDDCUP'99 分别设置不同长度来测试这些聚类算法的处理速度 (如图 2、图 3 所示), 其中每 1 000 个数据点设置为一个时间窗口长度, 图 3 显示了 MCStream以及其他算法在不同维度数据集上响应时间.

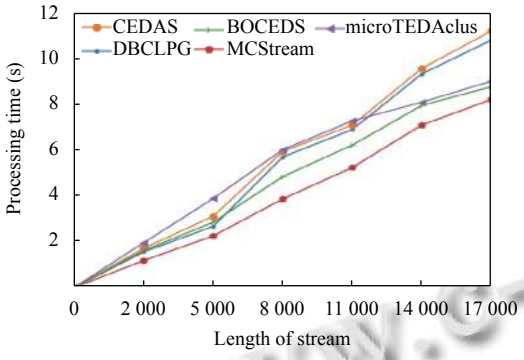

图 2 在 EPM 数据集上测试对 MCStream 算法进行测试

从图 2、图 3 中可以看出, MCStream 聚类算法随着数据点的不断增大在聚类处理时间上明显低于其他聚类算法. 而且在图 2 的结果数据中可以看出, 面对着数据量更大以及维度更高的 EPM 数据流来说, 各类聚类算法处理时间长度都显著增加. 这是由于维度大小与时间复杂度存在着线性关系. 对于 CEDAS[20] 以及 BOCEDS[11] 来说, 随着数据量不断增加, 微簇数量也呈线性增加; 而创建新的微簇来维护簇群以及更新微簇的边缘, 这是一个十分巨大的耗时任务. MCStream 算法都是明显优于其他聚类算法, 这说明 MCStream 算法聚类算法可以在较低的处理时间和延迟代价来扩展到更高维的数据空间中.

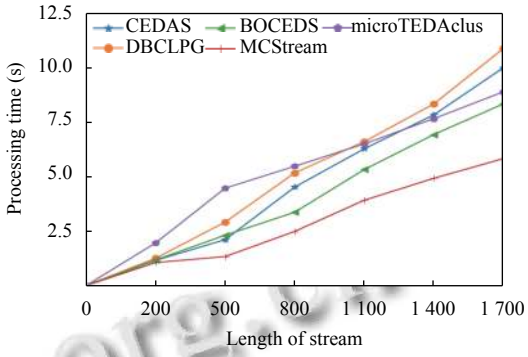

图 3 在 KDDCUP'99 数据集上测试对 MCStream 算法进行测试

4.3 聚类纯度对比

本组实验的目的是为了研究 MCStream 算法在数据流中聚类的纯度从而反映其聚类性能. 在评价聚类的指标中聚类纯度是其中比较流行的一种评价方法, 在式 (9) 给出了纯度的详细定义.

$$purity = \frac{1}{K}\sum_{i=1}^{K}\frac{m_i}{m}\max_j\left(\frac{m_{ij}}{m_i}\right) \tag{9}$$

其中, K 是所有宏簇数, m 是所有参与聚类的数据点数, m_i是聚类 i 中所有数据点数, m_{ij}是聚类 i 中数据类 j 的数据点数, 这里取平均值是为了降低误差性.

目前在据流聚类时比较受欢迎一类流聚类数据集是 KDDCUP'99 数据集, 它可以测试不断演化的数据流聚类算法. 我们通过以 500 个数据点为一个长度分别设置不同长度来测试 CEDAS[20]、DCLPG[21]、microTEDAclus[22]、BOCEDS[11] 以及本文提出的 MCSTream 在该数据集中聚类纯度, 实验结果在图 4 柱状图中显示.

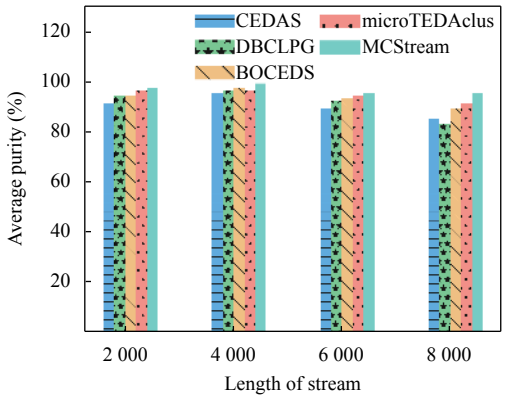

图 4 测试不同算法的聚类纯度

从图 4 数据中可以看出, 在不同的时期聚类纯度都有所不同, 在后期所有聚类纯度都有所降低. 这是因为随着数据流长度的不断增长, 微簇数量也在不断增加, 数据点错分的概率也随着线性增加, 从而降低聚类纯度. 尽管这样, MCStream 聚类算法仍然保持了一个平稳的较高聚类纯度. 这是由于在宏簇聚类中 MCStream 移除了信息量较少且离簇中心较远却又参与到宏簇聚类的异常簇, 从而保证了高质量聚类, 这表明 MCStream 算法在大规模数据集上有着良好的稳定性.

4.4 参数影响以及聚类精度定义

在本文所规划的实验中, 需要设定相关参数以研究权重衰减因子对聚类响应速度, 以及对于聚类精度的影响.

聚类精度的定义在式 (10) 中给出, 该公式中变量与式 (9) 中的变量定义是一样的. 在本实验中, 我们在高维数据流 Bag of Words 以及 KDDCUP'99 中分别测试了不同衰减因子在不同维度对于 MCStream 聚类反应时间, 以及不同衰减因子对于聚类精度的影响. 为了模拟数据流的大规模性, 本文采取了以 1 000 个数据点为一个数据窗口长度, 设置不同数据流长度的方式对 MCStream 算法进行测试. 图 3 的实验结果显示了不同衰减因子在不同维度下算法 MCStream 对于每个数据点的平均处理速度. 在图 4 的柱状图中显示了对于不同的衰减因子对于聚类精度的影响.

$$accuracy = \frac{\sum_{i=1}^{n} |c_i^d|}{\sum_{i=1}^{n} |c_i|} \times 100\% \tag{10}$$

从图 5 可以看出, 各类算法在不同维度下无论设置什么衰减因子其处理速度都在显著的增加. 这是因为数据维度是和时间复杂度呈线性相关, 维度的升高, 数据映射时间也会显著性增加. 而对于衰减因子来说, 衰减因子是和微簇数量呈正相关. 衰减因子增加会导致权重衰减得越慢, 从而微簇数量势必会增加, 这对于新数据点所归属簇群的映射计算量也会显著性增强. 从簇群方面来说, 微簇的量增加, 那么对于维护簇群压力也会增大. 因此, 衰减因子越大, 数据点平均处理延迟也会越大.

从图 6 可以看出, 随着衰减因子不断增加, 各类聚类算法的聚类精度也会随之下降, 这是因为上述所说的衰减因子与微簇数量呈正相关. 微簇数量的增加, 将会导致异常微簇数量增加, 从而导致整体聚类精确度降低. 对 CEDAS[20] 以及 BOCEDS[11] 来说, 微簇数量的增加, 会把更多的异常簇加入到聚类宏簇当中; 而对于 MCStream 来说, 由于在宏簇聚类过程中有一个平均局部密度上界来区分一些边缘微簇, 从而大大提升了聚类的精度.

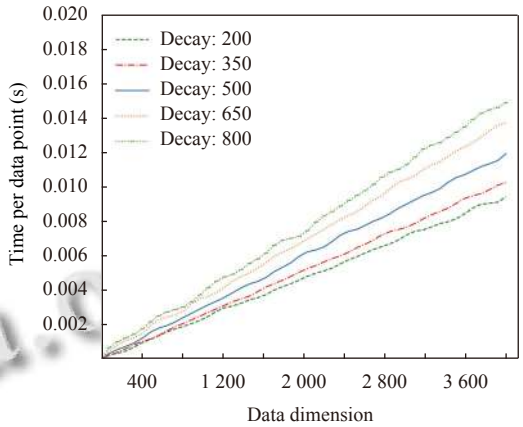

图 5 不同衰减因子在不同维度上对数据点处理速度

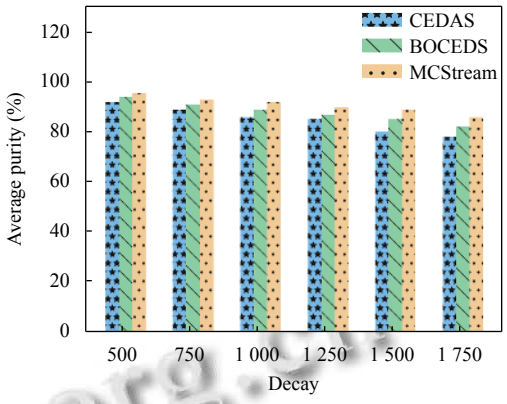

图 6 不同衰减因子参数下对聚类精度的影响

在本部分我们通过使用了 3 个数据集在处理时间、聚类纯度、精确度以及衰减参数上分别验证了基于微簇结构的流聚类算法的有效性. 实验结果表明 MCStream 具有较高的聚类能力, 然而对于高维数据仍然对聚类处理时间有着较大的影响. 在与流数据聚类算法进行对比的过程中, 我们不仅得出了高速聚类的重要性, 也验证了 MCStream 的有效性和高效性.

5 结束语

工业物联网产生的工业数据流具有无限、高维、无序的特点, 因此构建高质量自适应流式聚类算法处理工业数据流具有十分重要的意义. 本文提出了一种工业物联网数据流自适应聚类算法, 根据微簇的高密度性, 将每一个微簇作为一个参与聚类的数据样本点,

计算每个微簇的局部密度以及微簇之间的距离, 通过微簇的局部密度以及微簇距离来构建宏簇聚类决策树从而确定聚类中心以及自适应确定宏簇聚类数. 并且通过引入了引力能量函数来不断地更新微簇权重, 从而来移除老化的微簇以防止概念演化以及数据漂移问题. 此外, 本文方法去除了微簇构建过程中相交微簇之间的计算, 维护了宏观簇所需的最小计算量. 通过在 3 个仿真的大规模数据流中对算法从多个方面进行了评测, 并且与当前前沿的聚类算法进行了充分的比较, 证明了该算法具有显著性优势. 由于该算法面向的是稠密数据集, 面对着稀疏数据集时由于会产生大量单个稀疏微簇, 聚类质量会显著性降低.

目前, 工业物联网正发展迅猛, 其数据结构越来越复杂, 其规模也越来越大. 在未来工作中, 我们将进一步提高算法在大规模稀疏数据集上的处理效果, 提高算法在现实工业领域的适应性.

参考文献

1 Sisinni E, Saifullah A, Han S, *et al*. Industrial Internet of Things: Challenges, opportunities, and directions. IEEE Transactions on Industrial Informatics, 2018, 14(11): 4724–4734. [doi: 10.1109/TII.2018.2852491]

2 常建龙, 曹锋, 周傲英. 基于滑动窗口的进化数据流聚类. 软件学报, 2007, 18(4): 905–918.

3 杨春宇, 周杰. 一种混合属性数据流聚类算法. 计算机学报, 2007, 30(8): 1364–1371.

4 沙乐天, 肖甫, 陈伟, 等. 面向工业物联网环境下后门隐私泄露感知方法. 软件学报, 2018, 29(7): 1863–1879. [doi: 10.13328/j.cnki.jos.005356]

5 Youn J, Shim J, Lee SG. Efficient data stream clustering with sliding windows based on locality-sensitive hashing. IEEE Access, 2018, 6: 63757–63776.

6 王桂玲, 韩燕波, 张仲妹, 等. 基于云计算的流数据集成与服务. 计算机学报, 2017, 40(1): 107–125.

7 Growe GA. Comparing algorithms and clustering data: Components of the data mining process [Master's Thesis]. Allendale: Grand Valley State University, 1999.

8 陈华辉, 施伯乐, 钱江波, 等. 基于小波概要的并行数据流聚类. 软件学报, 2010, 21(4): 644–658.

9 Aggarwal CC, Yu PS, Han JW, *et al*. A framework for clustering evolving data streams. In: Freytag JC, Lockemann P, Abiteboul S, *et al*., eds. Proceedings 2003 VLDB Conference. Amsterdam: Elsevier, 2003. 81–92.

10 Baer A, Casas P, D'Alconzo A, *et al*. DBStream: A holistic approach to large-scale network traffic monitoring and analysis. Computer Networks, 2016, 107: 5–19.

11 Islam K, Ahmed M, Zamli KZ. A buffer-based online clustering for evolving data stream. Information Sciences, 2019, 489: 113–135.

12 Chen YX, Tu L. Density-based clustering for real-time stream data. Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. San Jose: ACM, 2007: 133–142.

13 Amini A, Saboohi H, Wah TY, *et al*. A fast density-based clustering algorithm for real-time Internet of Things stream. The Scientific World Journal, 2014, 2014: 926020.

14 Amini A, Saboohi H, Herawan T, *et al*. MuDi-Stream: A multi density clustering algorithm for evolving data stream. Journal of Network and Computer Applications, 2016, 59: 370–385.

15 Aggarwal CC, Han JW, Wang JY, *et al*. A framework for projected clustering of high dimensional data streams. In: Nascimento MA, Özsu MT, Kossmann D, *et al*., eds. Proceedings 2004 VLDB Conference. Amsterdam: Elsevier, 2004. 852–863.

16 Ntoutsi I, Zimek A, Palpanas T, *et al*. Density-based projected clustering over high dimensional data streams. Proceedings of the 2012 SIAM International Conference on Data Mining. Anaheim: Society for Industrial and Applied Mathematics, 2012. 987–998.

17 Xiao RL, Su JW, Du X, *et al*. SFAD: Toward effective anomaly detection based on session feature similarity. Knowledge-Based Systems, 2019, 165: 149–156.

18 Ng RT, Han JW. CLARANS: A method for clustering objects for spatial data mining. IEEE Transactions on Knowledge and Data Engineering, 2002, 14(5): 1003–1016.

19 Haque A, Khan L, Baron M. SAND: Semi-supervised adaptive novel class detection and classification over data stream. Proceedings of the 30th AAAI Conference on Artificial Intelligence. Phoenix: AAAI Press, 2016. 1652–1658.

20 Hyde R, Angelov P, MacKenzie AR. Fully online clustering of evolving data streams into arbitrarily shaped clusters. Information Sciences, 2017, 382–383: 96–114.

21 Halim Z, Khattak JH. Density-based clustering of big probabilistic graphs. Evolving Systems, 2019, 10(3): 333–350.

22 Maia J, Junior CAS, Guimarães FG, *et al*. Evolving clustering algorithm based on mixture of typicalities for stream data mining. Future Generation Computer Systems, 2020, 106: 672–684.