基于相对密度的数据流模糊聚类算法

刘青宝 王文熙 马德良

(国防科学技术大学信息系统与管理学院 长沙 410073)

摘 要 提出的基于相对密度的数据流模糊聚类算法结合了相对密度聚类和模糊聚类的优点,能形成任意形状、多密度分辨率的层次聚类结果。同时,利用微簇空间位置重叠关系,定义了微簇集合间的差运算,从而有效地支持了用户指定时间窗口内的数据流聚类要求。通过与CluStream算法在聚类质量和处理时间两个方面的比较分析,发现基于相对密度的数据流模糊聚类算法具有明显的优势。

关键词 多分辨率聚类,模糊聚类,数据流,相对密度

中图法分类号 TP391 **文献标识码** A

Data Stream Fuzzy Clustering Algorithm Based on Relative Density

LIU Qing-bao WANG Wen-xi MA De-liang

(College of Information System and Management, National University of Defense Technology, Changsha 410073, China)

Abstract This paper provided a relative density based data stream fuzzy clustering algorithm which inherits the advantages of relative density based clustering and fuzzy clustering, so it can discover arbitrary-shape and multi-resolution clusters. With the subtraction operator on the set of micro-clusters which is defined according to the spatial overlapping relations among micro-clusters, this algorithm can do clustering on any user-specified data stream window. Compared with CluStream algorithm on the two areas of clustering quality and processing time, this algorithm demonstrates a clear advantage.

Keywords Multi-resolution clustering, Fuzzy clustering, Data stream, Relative density

随着计算机和传感器技术的发展和应用,数据流挖掘技术在国内外得到广泛研究。它在网络监控、证券交易分析、电信记录分析等方面有着巨大的应用前景。特别在军事应用中,为了及时获得战场态势信息,大量使用了各种传感器。分析处理这些传感器数据流已显得极为重要。数据流聚类问题一直是吸引许多研究者关注的热点问题,已提出多种一次性扫描的方法和算法,如文献[1,2]等,但它们的聚类结果通常是球形的,不能支持对任意形状类的聚类[5]。

本文提出的基于相对密度的数据流模糊聚类算法利用微簇的空间位置重叠关系,定义了微簇集合间的差运算,能支持用户指定时间窗口内的数据流聚类要求。同时,本算法结合相对密度聚类和模糊聚类的优点,能形成任意形状、多密度分辨率的层次聚类结果。

本文第1节简要介绍数据流聚类的相关研究;第2节给出基于相对密度的数据流模糊聚类算法所使用到的基本概念;第3节给出一个完整的基于相对密度的数据流模糊聚类算法,并详细解析算法的执行过程;第4节通过实验进行算法性能的分析对比;最后总结本文的主要工作和贡献。

1 相关研究

Aggarwal在2003年提出了一个解决数据流聚类问题的框架CluStream[1]。CluStream聚类框架是近年来多种数据流聚类算法[2-5]的参考,但由于它采用距离作为度量参数,在线过程micro-cluster的结果通常是球形的,不能支持对任意形状类的聚类。

2006年,Cao Feng等人提出了数据流聚类DenStream算法[3]。它相对CluStream有很大的改进,能够在有噪声条件下对数据流进行任意形状的聚类。但它只能提供对数据流当前状态的描述,无法反映用户指定时间窗口内的数据流变化情况。

朱蔚恒等人提出的ACluStream聚类算法[4],通过定义有空间位置信息的聚类块,在对数据流进行初步聚类的同时,尽量保留数据的空间特性,有效地克服了CluStream算法不能支持对任意形状聚类的缺陷。但由于采用绝对密度作参数,聚类结果对参数设置的敏感性强,且不能产生多密度分辨率的结果。

文献[5]提出了基于网格的数据流聚类算法GClustream,通过对数据空间采用网格化处理,提高了算法处理速度,并能在噪声干扰条件下发现任意形状的类,但与DenStream算法类似,只能描述数据流的近期聚类结果。为此,本文提出的基于相对密度的数据流模糊聚类算法RDFCluStream利用微簇的空间位置重叠关系,定义了微簇集合间的

到稿日期:2009-09-28 返修日期:2009-12-17 本文受国家自然科学基金项目:模糊、动态多维数据建模理论与方法研究(70771110)资助。

刘青宝(1967-),男,博士,副教授,主要研究方向为数据仓库技术和数据挖掘,E-mail:liuqingbao@nudt.edu.cn;王文熙(1984-),男,硕士生,主要研究方向为数据挖掘技术;马德良(1980-),男,硕士生,主要研究方向为信息系统与智能决策技术。

差运算，能够支持用户指定时间窗口内的数据流聚类要求。

2 基本概念

定义 1 在数据空间的 d 个属性轴上定义一个单位长度，采用网格方式将数据空间划分为若干个网格单元。一个网格单元是数据空间中各个属性轴上具有单位长度的 d 维超立方体，即以 d 维向量 o 为起点，向各维的正方向延伸单位格长所形成的一个 d 维区间，记为 $grid(o)$。

定义 2 由若干个网格单元组成的超立方体，称为聚合块，记为 $block(o,\vec{r})$，其中 $\vec{r}$ 为聚合块的各维边长组成的向量。

定义 3 设起点为 o_i 的网格单元(或聚合块)包含 n 个分别在时刻 $t_1,t_2,\cdots,t_n$ 到达的数据对象 $p_1,p_2,\cdots,p_n$，则该网格单元(或聚合块)的特征值向量记为 $(o_i,n,\vec{F^1},\vec{F^2},\vec{r})$，其中 $F^1=\sum_{j=1}^{n}p_j$ 是 n 个 d 维向量求和，$\vec{F^2}=\sum_{j=1}^{n}p_j^2$ 是 n 个 d 维向量的平方和。该网格单元(或聚合块)在 t_{now} 时刻的活跃度记为 $a_{now}=\sum_{j=1}^{n}2^{-\lambda(t_{now}-t_j)}$。

特征值向量用于记录该网格单元(或聚合块)在空间中的位置和其中点的分布情况。活跃度 a_{now} 值越大，则反映其中的数据点越是在接近 t_{now} 时刻达到的，即在 t_{now} 时刻该网格单元(或聚合块)是比较活跃的。

定义 4 给定噪声因子 ϕ_{noise} 和活跃参数 a_0，设起点为 o_i 的网格单元(或聚合块)的特征值向量为 $(o_i,n,\vec{F^1},\vec{F^2},\vec{r})$，在 t_{now} 时刻的活跃度为 $a_{now}=\sum_{j=1}^{n}2^{-\lambda(t_{now}-t_j)}$，若满足 $n\geqslant\phi_{noise}\prod_{i=1}^{d}r_i$ 且 $a_{now}\geqslant a_0$，则称该网格单元(或聚合块)在 t_{now} 时刻为密集微簇。

3 算法描述

基于相对密度的数据流模糊聚类算法 RDFCluStream 分为两个阶段：在线微簇聚类过程和离线模糊聚类过程。在线微簇聚类过程维护着数据流当前时刻的所有微簇的聚类特征，在线过程称为 RDMic_Clusting；离线过程则根据用户指定的时间窗口，对其间的所有微簇进行模糊聚类，形成多分辨率的层次聚类结果，离线过程称为 RDFMac_Clusting。

3.1 RDMic_Clusting 过程描述

在给定噪声因子 ϕ_{noise}、活跃参数 a_0、相对密度阈值 rd、微簇最大尺寸 $\vec{r}_{\max}$ 和缓冲窗口宽度 w 的条件下，在线进程 GMic-Cluster 的具体步骤如下。

①初始化

初始时，积累一定数量的数据对象后，采用文献[6]中提出的基于相对密度的聚类算法进行聚类，再把聚类的结果划分成互不相交的密集微簇，计算这些密集微簇的特征值向量和活跃度。

②加入新数据对象

对于在 t_c 时刻到达的新数据对象 p，若它属于一个已存在的微簇，则修改该微簇的特征值向量为 $(o_i,n+1,\vec{F^1}+p,\vec{F^2}+p^2,\vec{r})$，活跃度更新 $a_{now}=a_{now}\times 2^{-\lambda(t_c-t_{now})}+1$；否则，直接定位其所在的网格单元，并计算该网格单元的特征值向量和活跃度，把它加入到候选密集微簇集合。

③缓冲窗口满处理

调整噪声因子 ϕ_{noise}，使 ϕ_{noise} 随数据流规模线性增长；更新所有微簇的活跃度；对所有聚合微簇进行数据分布情况计算(根据特征值向量计算均值和方差)，若一微簇的数据分布严重不均，则从其方差最大且边长大于 1 的维进行居中切分成两个独立的微簇，并根据契贝晓夫不等式进行特征值向量的分割，活跃度等于原微簇的活跃度；对所有微簇进行密集性条件的判断，据此调整它们在密集微簇集合和候选密集微簇集合之间的隶属关系。

④候选密集微簇的淘汰

随着数据流数据对象的不断到达，候选微簇数目在逐渐增加，必须把无潜力升级为密集微簇的候选密集微簇进行淘汰，以节省内存空间。淘汰的策略为：清除候选密集微簇集合中活跃度$<\frac{a_0}{2}$的所有微簇。

⑤微簇聚合处理

若通过步骤④后，仍然缓解不了空间需求压力，则分别聚合密集微簇集合和候选密集微簇集合中的微簇，以节省空间消耗。两微簇聚合的具体条件为：属于同一集合、形状相同($\vec{r}_1=\vec{r}_2$)、相邻(两起点 o_1 与 o_2 只在某一维上的取值相差 $\vec{r}$ 的该维分量，其他维上取值相同)、相对密度$\geqslant rd$、聚合后的新微簇的尺寸$\leqslant\vec{r}_{\max}$。

两微簇聚合形成新微簇，其特征值向量是原来两微簇的特征值向量之和，活跃度是原来两微簇的活跃度的平均值。

若还解决不了空间紧张问题，则动态扩大 $\vec{r}_{\max}$，重复步骤⑤，直到问题缓解。

⑥把微簇的特征向量写入磁盘

采用 CluStream 算法提出的 pyramid time frame 存储策略，按一定的时间跨度将 RDMic_Clusting 过程的内存结果(微簇及其特征值向量)写入磁盘。

RDMic_Clusting 过程的算法输入为噪声因子 ϕ_{noise}、活跃参数 a_0、相对密度阈值 rd、微簇最大尺寸 $\vec{r}_{\max}$、缓冲窗口宽度 w、数据流 ds，输出为 pyramid time frame 存储结构的微簇集。算法的伪代码描述如下：

```
RDMic_Clustering(ds, φnoise, a0, rd, r→max, w)
BEGIN
  SetofDmiclus=InitMiclus(ds, φnoise);//第①步：初始化密集微簇集合
  SetofCmiclus=∅;//初始化候选密集微簇集合
  REPEAT
    FOR i FROM 1 TO w DO
      dataobj=GetDataObject(ds);//第②步开始
      Setofmiclus=SetofDmiclus∪SetofCmiclus;
      miclus=SearchMiclus(Setofmiclus,dataobj);
      IF miclus≠NULL THEN
      //若 dataobj 属于已存在的微簇 miclus
      UpdateMiclus(miclus,dataobj);
      //修改微簇 miclus 的特征值向量和活跃度
      ELSE
      c_miclus=NewCMiclus(dataobj);//创建一个新的候选密集微簇
      SetofCmiclus=InsertCset(SetofCmiclus,c_miclus);
    END IF;
  END FOR;//第②步结束，第③开始
  φnoise=φnoise+func(w);//φnoise 线性增长
  Setofmiclus=SetofDmiclus∪SetofCmiclus;
  FOR EACH miclus∈Setofmiclus DO
```

UpdateMiclusActi($miclus, t_{now}$);
CheckMiclusSplit($miclus$);
//若聚合微簇的数据分布严重不均,则进行分裂
CheckMiclusDensity($miclus$);
//进行密集性条件的判断,并调整集合隶属关系
IF ($miclus.a_{now} < \frac{a_0}{2}$) AND ($miclus \in SetofCmiclus$) THEN
ClearMiclus($miclus$);//第④步
END IF;
END FOR;//第③步结束
WHILE LowSpace() DO //开始第⑤步
MergeMiclus($SetofDmiclus, rd, \vec{r}_{max}$);
MergeMiclus($SetofCmiclus, rd, \vec{r}_{max}$);
IF LowSpace() THEN
$\vec{r}_{max}$=Enlarge($\vec{r}_{max}$);
END IF;
END WHILE;//结束第⑤步
IF $\exists i((t_c \bmod \beta^i)=0) \wedge ((t_c \bmod \beta^{i+1}) \neq 0)$ THEN //第⑥步
SaveToSnapshotSet(i, $SetofDmiclus \cup SetofCmiclus$);
ClearOldestSnapShot(i);
END IF;
UNTIL;
END RDMic_Clustering.

3.2 RDFMac_Clusting 过程描述

RDFMac_Clusting 过程描述如下。

首先,定位有关微簇的存储快照。RDFMic_Clusting 得到的微簇结果集以 pyramid time frame 存储结构保存在磁盘上,假设用户指定的查询窗口为 $W[t_c-h, t_c]$,其中 t_c 为当前时刻,h 为用户查询窗口宽度。若 t_s 为刚好在 t_c-h 之前的微簇快照存储时刻,则有 $(t_c-t_s) \leqslant (1+1/\alpha^{l-1}) \times h$(证明过程参见文献[1],这里从略)。

其次,比较 t_c 和 t_s 两个时刻的微簇结果集 $Setofmiclus_{t_c}$ 和 $Setofmiclus_{t_s}$。由于微簇有严格的空间意义,因此可以很容易找到有重叠关系的微簇对 $\{(miclus_{t_c}, miclus_{t_s}) \mid miclus_{t_c} \in Setofmiclus_{t_c}, miclus_{t_s} \in Setofmiclus_{t_s}\}$,对重叠关系分如下几种情况进行处理:

①当 $miclus_{t_s} \subseteq miclus_{t_c}$ 时,不须分裂处理;

②当 $miclus_{t_s} \supset miclus_{t_c}$ 时,须把 $miclus_{t_s}$ 分裂成 $miclus_{t_s}^{(1)}$ 和 $miclus_{t_s}^{(2)}$,使得 $miclus_{t_s}^{(1)} = miclus_{t_c}$;

③否则,当两者之间只相交、不相互包含时,须把相交部分分别从 $miclus_{t_c}$ 和 $miclus_{t_s}$ 中分裂出来。

经过上面的处理,使所有的重叠关系只能以第①情况存在。

第三,在有重叠关系的微簇对 $(miclus_{t_c}, miclus_{t_s})$ 上定义如下的减法关系:$miclus_{t_s}^{t_c} = miclus_{t_c} - miclus_{t_s}$。设各微簇 $miclus_{t_s}^{t_c}$,$miclus_{t_c}$ 和 $miclus_{t_s}$ 的特征值向量分别为 $(o_{t_s}^{t_c}, n_{t_s}^{t_c}, \vec{F}^1{}_{t_s}^{t_c}, \vec{F}^2{}_{t_s}^{t_c}, \vec{r}_{t_s}^{t_c})$,$(o_{t_c}, n_{t_c}, \vec{F}^1{}_{t_c}, \vec{F}^2{}_{t_c}, \vec{r}_{t_c})$ 和 $(o_{t_s}, n_{t_s}, \vec{F}^1{}_{t_s}, \vec{F}^2{}_{t_s}, \vec{r}_{t_s})$,则有

$$o_{t_s}^{t_c} = o_{t_c}, n_{t_s}^{t_c} = n_{t_c}, \vec{r}_{t_s}^{t_c} = \vec{r}_{t_c}$$

$$\vec{F}^1{}_{t_s}^{t_c} = \vec{F}^1{}_{t_c} - \vec{F}^1{}_{t_s}, \vec{F}^2{}_{t_s}^{t_c} = \vec{F}^2{}_{t_c} - \vec{F}^2\vec{F}_{t_s}$$

把这种减法关系应用到集合的差运算中,便可以得到用户指定查询窗口 $W[t_c-h, t_c]$ 上的数据流微簇结果集 $Setofmiclus_w = Setofmiclus_{t_c} - Setofmiclus_{t_s}$。这样的集合差运算是合理的,并且能保证在用户查询窗口之前创建的微簇不会对用户查询窗口中的聚类分析结果造成影响。

最后,对 $Setofmiclus_w$ 中的微簇进行模糊聚类,形成多分辨率的层次聚类结果。基于微簇的模糊聚类算法思路描述如下:以微簇为顶点,微簇之间的邻接关系为边,并以两个微簇的相对密度作为邻接强度(也是两个微簇之间的相似度),在没有邻接关系的顶点之间建立邻接强度为零的虚边,则构建了一个微簇的连通图。这个连通图的矩阵表示便是微簇集合的模糊相似矩阵。为了减少计算量,基于这个模糊相似矩阵可以采用直接聚类法[7],从而避免计算模糊相似矩阵的传递闭包。其具体步骤为:

①建立模糊相似度。设两邻接的微簇为 C_i 和 C_j,且设两微簇的 k 近邻平均密度分别为 $C_i.nnad$ 和 $C_j.nnad$,则定义微簇 C_i 和 C_j 间的模糊相似度 $\tilde{r}_{ij}$ 为

$$\tilde{r}_{ij} = \frac{\min\{C_i.nnad, C_j.nnad\}}{\max\{C_i.nnad, C_j.nnad\}}$$

②取模糊相似矩阵 $\tilde{R}=(\tilde{r}_{ij})_{n\times n}$ 中最大的模糊相似度 λ_1,这时有 $\lambda_1=1$。设 $(C_i)_{\lambda_1}=\{C_j \mid \tilde{r}_{ij}=1\}$,即将满足 $\tilde{r}_{ij}=1$ 的微簇 C_i 和 C_j 作为一组,构成模糊相似类 $(C_i)_{\lambda_1}$。不同的模糊相似类可能有公共元素,这种情况下,只要将有公共元素的模糊相似类进行归并,即可得关于 $\lambda_1=1$ 的模糊等价类 $[C_i]_{\lambda_1}$。

③取模糊相似矩阵 $\tilde{R}=(\tilde{r}_{ij})_{n\times n}$ 中第二大的模糊相似度 λ_2,直接找出模糊相似度等于 λ_2 的微簇对 (C_i, C_j),即 $\tilde{r}_{ij}=\lambda_2$。相应地,将模糊等价类 $[C_i]_{\lambda_1}$ 与 $[C_j]_{\lambda_1}$ 进行归并,将所有这种情况都归并后可得到关于 λ_2 的模糊等价类 $[C_i]_{\lambda_2}$。

④取模糊相似矩阵 $\tilde{R}=(\tilde{r}_{ij})_{n\times n}$ 中第三大的模糊相似度 λ_3,如步骤③一样进行归并。如此进行下去,直到成为一类为止。这样就建立了微簇集上的多分辨层次聚类结果。

4 实验分析

这里以算法 CluStream 为参照对比,从聚类质量和处理时间两个方面进行比较分析。

4.1 实验条件

(1)实验环境:PC 机配置为 P4 2.4GHz / 512MB / 160GB;操作系统为 Windows XP;编程语言为 C++。

(2)数据集:http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html 的网络入侵检测数据集,数据集描述如表 1 所列。

表 1 实验数据集

数据集	记录数	维数	预先类标识
KDD CUP99 数据集	311k	34	有标识

4.2 聚类质量分析

RDFCluStream 算法克服了 CluStream 算法的固有缺陷,能很好地支持任意形状的聚类,而且吸取了 CluStream 算法和 DenStream 算法对孤立点和陈旧微簇淘汰机制上的优点,引入了活跃度参数。与基于密度的 ACluStream 算法[4]相比,其避免了 ACluStream 算法中的随机淘汰机制所带来的缺陷,而且在整个算法的两个阶段中,都采用相对密度作参数,从而能形成多密度分辨率的层次聚类结果。

在对比聚类质量时,由于基于密度聚类算法不能采用"距离平方和"作为聚类质量的评价标准,这里采用聚类精度来评价聚类质量。RDFCluStream 算法的聚类精度定义描述如下:在指定时间窗口 h 上,把带有预先类标识的数据集按数据点

在数据空间中的位置，分配到相应的聚类结果簇中，计算该数据集中数据点分配正确的比例，此比例则为在指定时间窗口 h 上 RDFCluStream 算法的聚类精度。

同样，可以定义 CluStream 算法的精度如下：在指定时间窗口 h 上，把带有预先类标识的数据集按数据点在第一阶段聚类获得的微簇 id 号，分配到相应的聚类结果簇中，计算该数据集中数据点分配正确的比例。此比例则为在指定时间窗口 h 上 CluStream 算法的聚类精度。

图 1 所示为 KDD CUP99 数据集 corrected 在流速固定为 1000/s 条记录、时间窗口 h=60s、微簇数目为 1024 块时在数据流不同查询时点上的聚类质量的比较情况。由于 KDD CUP99 数据集 corrected 中的网络连接类型分布严重不均匀，从图 1 可以看出，能很好地支持任意形状聚类的 RDFCluStream 算法在精度上明显优于 CluStream 算法。

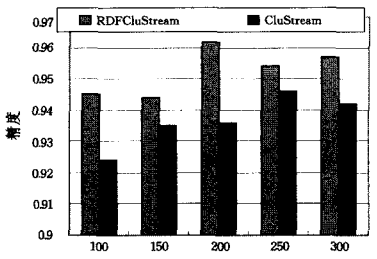

图 1　数据流不同查询点上的聚类精度比较(1000 条/s，h=60s)

4.3　算法处理时间

对于数据流处理算法来说，在线处理速度的高低是至关重要的。质量再高的处理算法，若处理速度无法达到数据流的峰值速度，也是无法使用的。为此，这里重点比较分析算法 *RDFCluStream* 和 *CluStream* 的在线处理速度。

RDFCluStream 算法由于对数据空间采用了网格化处理，每当新数据点到达，则可直接定位其所属网格单元，时间复杂度为常数 1。而 CluStream 算法则要通过计算新数据对象到各子聚类中心的距离，并比较之后才能决定其所属的子聚类，其时间复杂度为 $O(k)$，其中 k 为子聚类的数量。

在处理微簇聚合时，RDFCluStream 算法判断两个微簇是否可以合并，主要看它们是否相邻，即两者之间是否存在一个公共的超平面，只须进行一次减法运算，时间复杂度为常数 1。而 CluStream 算法判断两个微簇是否可以合并，则要计算各微簇中心之间的距离，并进行比较之后才能决定，其时间复杂度为 $O(k)$。

从上面的分析可知，RDFCluStream 算法的处理时间不随数据流的大小、微簇数目多少而明显改变，能很好地完成连续数据流环境下的在线聚类任务。相比可知，CluStream 算法的速度虽然也可以不随数据流的大小而改变，但随微簇的数量增多而明显下降。由于在线处理阶段的微簇数目多少直接影响到离线阶段的最终聚类质量，因此算法 RDFCluStream 相比 CluStream 而言，能更好地支持数据流的高质量聚类。

图 2 所示是算法 RDFCluStream 与 CluStream 在不同微簇数目设置下的运行时间比较。图 2 的实验结果表明，RDFCluStream 算法的速度总体上明显高于 CluStream 算法。只是在微簇数目与最终结果类数目相近时，RDFCluStream 算法因微簇之间的合并与分裂操作十分频繁，使得速度有所下降，而 CluStream 算法的速度却达到最高，不过这时因微簇数目过少，使得最终聚类质量很差，难以使用。

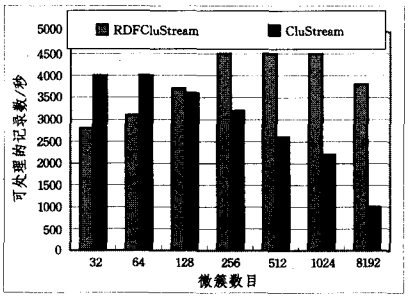

图 2　不同微簇数目时的在线处理速度比较

结束语　基于相对密度的数据流模糊聚类算法利用微簇的空间位置重叠关系，定义了微簇集合间的差运算，以支持用户指定时间窗口内的数据流聚类要求。同时，此算法在借鉴 CluStream 算法的两阶段聚类思想和 pyramid time frame 的快照储存结构的基础上，结合相对密度聚类[8]和模糊聚类[7]的优点，克服了 CluStream 算法的固有缺陷，能形成任意形状、多密度分辨率的层次聚类结果。

参 考 文 献

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

[2] Aggarwal C, Han J, Wang J, et al. A framework for projected clustering of high dimensional data streams[C]// Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04). Toronto, Canada, 2004, 8:852-863

[3] Cao F, Martin E, Qian W, et al. Density-based Clustering over an Evolving Data Stream with Noise[C]// Proc. of the 2006 SIAM Conference on Data Mining (SDM'2006). 2006

[4] Liu Qing-Bao, Deng Su, Lu Chang-Hui, et al. Relative density based K-nearest neighbors clustering algorithm[C]// Proc. 2003 Int. Conf. on Machine Learning and Cybernetics. 2003:133-137

[5] 朱蔚恒，印鉴，谢益煌. 基于数据流的任意形状聚类算法[J]. 软件学报，2006，3：379-387

[6] 刘青宝，戴超凡，邓苏，等. 基于网格的数据流聚类算法[J]. 计算机科学，2007(3)：159-162

[7] 孙即祥，等. 现代模式识别[M]. 长沙：国防科技大学出版社，2002

[8] 刘青宝，何勇，邓苏，等. 基于相对密度的多分辨率聚类算法[J]. 小型微型计算机系统，2007(6)：1287-1292

[9] 刘青宝，金燕，邓苏，等. 基于模糊聚类的属性匹配算法[J]. 模糊系统与数学，2006，20(6)：96-102

(上接第 142 页)

[5] Diaz G, Cuartero F, Valero V, et al. Automatic Verification of the TLS Handshake Protocol[C]// Proc. of the 2004 ACM Symposium on Applied Computing. 2004, 1:789-794

[6] Diaz G, Larsen K G, Pardo J, et al. An Approach to Handle Real Time and Probabilistic Behaviors in E-commerce: Validating the SET Protocol[C]// Proc. of the 20th Annual ACM Symposium on Applied Computing. 2005, 1:815-820

[7] Larsen K, Pettersson P, Wang Yi. UPPAAL in a Nutshell[J]. International Journal on Software Tools for Technology Transfer, 1997:134-152

[8] Larsen K G, Petterson P, Wang Yi. Diagnostic Model-checking for Real-time Systems[C]// Proc. of Workshop on Verification and Control of Hybrid Systems III. 1995, 1066:575-586

[9] Diaz G, Cambronero M E, Pardo J J, et al. Model Checking Techniques Applied to the Design of Web Services[J]. CLEI Electronic Journal, 2007, 10