一种基于边缘度密度距的聚类算法

吴明晖[1] 张红喜[1,2] 金苍宏[2] 蔡文明[2]
(浙江大学城市学院计算机科学与工程学系 杭州 310015)[1]
(浙江大学计算机科学与技术学院 杭州 310027)[2]

摘 要 传统网格聚类算法聚类质量低,而密度聚类算法时间复杂度高。针对两类算法各自的缺点,结合它们的聚类思想提出了一种新的聚类算法。该算法提出了边缘度密度距作为新的密度度量,并在此基础上逐步确定了类的定义和聚类过程的定义。算法前期通过网格划分操作统计记录了待聚类数据的初始信息,以供随后的 k 近邻统计使用。在寻找聚类中心点时使用了桶排序的策略,使得算法能快速地选出下一个聚类中心点。随后的聚类步骤是迭代搜索并检验当前类中未检验的 k 近邻是否满足密度可达性来完成聚类。理论分析和实验测试的结果表明,该算法不仅保持了较高的聚类精度,而且有接近线性的低时间复杂度。

关键词 聚类,网格,密度,Caed,Dbscan,Kmeans
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.052

Cluster Algorithm Based on Edge Density Distance

WU Ming-hui[1] ZHANG Hong-xi[1,2] JING Cang-hong[2] CAI Wen-ming[2]
(Department of Computer Science and Engineering, Zhejiang University City College, Hangzhou 310015, China)[1]
(College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)[2]

Abstract Clustering algorithms based on grid have a drawback of low clustering precision, and most clustering algorithms based on density have high time complexity. In order to improve clustering performance, a cluster algorithm based on edge density distance was proposed in this paper. The new cluster algorithm makes new definitions of density and category. In the clustering process, data are divided into grids and some initial information is recorded firstly for the operation of finding k near points. Then in the process of finding a new clustering center, a method come from bucket sort is used, which makes it fast to find the clustering center. A subsequent procedure is to iteratively analyse k near points of one category to judge whether they are density accessible. Analysis in theory and result of experiments show that the proposed algorithm has both high quality in clustering result and low time complexity.

Keywords Cluster, Grid, Density, Caed, Dbscan, Kmeans

1 前言

随着互联网的兴起和发展,数据挖掘近年来受到各界的广泛关注。聚类算法作为一种非监督的分类方法,是数据挖掘技术中发现数据规律和数据分布最重要的方法之一[1],是揭示数据的规则、发现数据相似性和区别性的有效途径[2]。从大量的数据中挖掘数据的信息已经成为一种有力的商业竞争。通过聚类,人们能够识别数据密集和稀疏的区域,因而发现数据的分布模式[3];聚类分析技术可以帮助人们更快地找到所需要的信息,在现实生活中也有着重要的意义[4]。聚类分析技术已经广泛研究了许多年[5],并应用于诸多领域中,包括模式识别、数据分析、图像处理、网络路由算法等[4],而新兴的应用领域甚至包括人体免疫学分析[6]。目前主流的聚类算法可分为如下几类:基于划分的方法、基于层次的方法、基于密度的方法、基于网格的方法以及基于模型的方法。上述算法中,基于密度的方法和基于网格的方法是两种较为常用的聚类算法。基于密度的聚类算法的指导思想就是:只要一个区域中的点的密度大过某个阈值,就把它加到与之相近的聚类中。基于密度的聚类算法(如 Dbscan)的优点在于:抗噪音性强,能克服基于距离的算法只能发现"类圆形"的聚类的缺点,能识别任意形状的类[7]。基于网格的聚类方法采用一个多维度的网格数据结构,它将空间量化为有限数目的单元,所有的聚类操作都在网格上进行。与其它聚类算法相比,基于网格密度的聚类算法不需要对输入数据的分布作任何假设,得到的结果与数据的输入顺序无关;能够较好地处理高维数据表格对象,能够处理多种数据类型;同时执行效率上都有了很大的提高,所需时间与要处理的总的记录条数呈线性关系。但这两种聚类方法也有各自的缺点。当处理的数据量较大

到稿日期:2013-09-23 返修日期:2013-12-13 本文受浙江省重点科技创新团队项目(2010R50009)资助。
吴明晖(1976-),男,博士,教授,硕士生导师,CCF会员,主要研究领域为人工智能、软件工程和移动互联网应用,E-mail:mhwu@zucc.edu.cn(通信作者);张红喜(1987-),男,硕士生,主要研究领域为数据挖掘;金苍宏(1982-),男,博士生,主要研究领域为大数据技术;蔡文明(1988-),男,硕士生,主要研究领域为数据挖掘。

时，一般的基于密度聚类的算法，由于计算复杂度高，造成聚类操作的速度很慢。而后者在聚类数据集合时，参数的设置依靠经验难以确定，而且网格划分操作基于将要进行聚类的数据的特性，如果划分过于稠密，则聚类的网格数量将会非常庞大；如果划分过于稀疏，则会导致不够精确的聚类边界，一定程度上降低了聚类的质量和准确性。

本文结合这两类算法的设计思想，定义一种新的密度度量方法和与之对应的聚类规则，提出了边缘度密度距聚类算法(clustering algorithm based on edge density，简称 Caed 算法)。该聚类方法在时间复杂度上接近线性而且在聚类精度上也保持着较高的精度。

2 相关工作

聚类问题的理论是建立在“一个类之内的数据比不同类的数据会更加相似”的准则的基础上的[8]。由于此概念的重要性，研究者提出了各种各样的算法来定义和解决这个问题。一部分具有代表性的聚类算法将在下文中简述。

Wei Wang 等人提出了 Sting (Statistical Information Grid-based method)的聚类算法。Sting 是一种基于网格的多分辨率聚类技术，针对不同级别的分辨率，通常存在多个级别的矩形单元，这些单元形成了一个层次结构：高层的每个单元被划分为多个低一层的单元。网格结构最底层的粒度决定了 Sting 算法聚类的质量。Sting 算法的时间复杂度是 $O(n)$，簇的质量和精度比较低[9]。Clique (由 Agrawal 等 1998 年提出)是一个基于密度和基于网格的能自动识别子空间的聚类算法[10]。它把每个维度划分成一个 m 格的数据区间，整个空间被划分为互不相交的矩形单元，同时识别其中的密集单元，然后运用深度优先算法来发现空间中的聚类。Clique 是一种简化了的密度聚类算法，聚类的精度有所降低。Chameleon (由 Karypis 等 1999 年提出)结合了图形划分方法和层级聚类方法的动态聚类模型，可以应用于任何定义了相似度函数的数据[11]。对需要处理的数据集合建立稀疏图，通过一个图划分算法将数据对象聚类为大量相对较小的子聚类，然后通过反复合并 RI 和 RC 分别超过 TRI 和 TRC 的簇对子类来找到真正的结果簇。该算法具有较好的聚类效果，但存在较大的时间复杂度。Kmeans 算法是经典的聚类算法之一[12]，根据选择的参数 k，随机选择 k 个类的初始中心，根据剩下的其它对象及它们与这些聚类中心的相似度，分别将它们分配给与其最相似的聚类，然后逐次更新各聚类中心的值，直至得到最后的聚类结果[13]。该算法的优点是具有较低的时间复杂度，缺点是准确地初始化 k 个聚类中心存在困难[14]，并且不能聚类出任意形状的类。Clara 算法是一种基于采样的方法，它能够同时处理大量的数据的聚类算法。Clara 算法从实际数据中抽取多个采样，在每个采样上都用 PAM 方法得到相应的 k 个聚类中心，然后在这当中选取总代价最小的一个作为最终的结果。可以处理的数据集比 PAM 大，但算法的有效性依赖于样本集的大小，而且由于样本会发生倾斜，因此样本代价最小的聚类方式对整个数据集来说不一定是代价最小的聚类。Racm(A robust adaptive clustering method)[2] 是 Mok P，Huang H 等提出的一种新式的聚类算法，算法能识别所需的聚类簇个数，是一种综合性的可靠的聚类方法。该算法可获取多个分类方法对数据集所分类出来的聚类结果，比如 Fuzzy C-Means，Kmeans，通过判定矩阵来整合多种分类方法所分类的结果。但该算法的复杂度由所选择的聚类算法来决定。为了兼顾较高聚类精度和较低的时间复杂度，本文提出了一种综合密度聚类和网格聚类思想的聚类方法。

3 算法描述

3.1 算法分析

综合利用网格聚类算法和密度聚类算法的优势是本文算法主要的研究内容。所述 Caed 算法包括密度度量的定义和聚类方法定义两个组成部分。本章将先阐述算法的相关定义，然后给出算法的具体实现步骤。

3.2 聚类模型相关定义

进行聚类操作的空间点集设为集合 $P=\{p_1,p_2,p_3,\cdots,p_n\}$，其中 n 表示空间点的个数，p_i 代表空间中的点。点的每个维的值是数据特征的量化值，点代表了从某个对象中抽取的值向量。边缘度密度距是针对向量化的空间点集定义的密度度量参数。类的定义以及聚类规则都是在它的基础上实现的。边缘度密度距涉及 k 近邻和边缘度向量两个概念。它们的含义将在下文中详细阐述。

定义 1(k 近邻) 对于空间点 $p\in P$，$p(k)$表示点 p 的最近邻的 k 个空间点集。

定义 2(边缘度向量) $\vec{v_{(k)}}(p)=\frac{1}{k}\sum_{i=0}^{k}(\vec{pk_i}-\vec{p})$，其中 $pk_i\in PK$，$\vec{v_{(k)}}(p)$的模称为边缘度量，向量描述点处于边界的位置情况。如图 1 所示，为图 1 上方矩阵各点的边缘向量的方向和大小情况。

[1,1,1;1,1.6,1;1,1,1,1.7;1.5,1,1;1.5,1.5,1.5;1.5,1.5,1;2,2,2;1,2,2;1.5,1,2;]

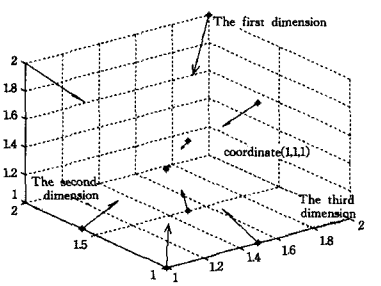

图 1 边缘度向量实例

箭头的起点是上述三维向量的端点，向量是起点对应$\vec{v_{(k)}}(p)$的计算结果。

在当前点的 k 近邻领域中，当前点越处在点集的边缘位置，边缘向量的模就越大。原因是 k 近邻点集会偏向于边缘点的一边，当前点的边缘度向量趋于叠加增强的状态，否则就具有互相抵消的效果。$\vec{v_{(k)}}(p)$第一个维度可以用 x_1 来表示。$x_1=\frac{\sum_{i=0}^{k}(x_{pk_i1}-x_p)}{k}=\frac{\sum_{i=0}^{k}x_{pk_i1}}{k}-x_p$，可见，在第一维上，点 p 越偏离 k 近邻在第一维的均值 x_1，$\vec{v_{(k)}}(p)$在第一维上的绝对值就越大，其他维度情况也是如此。因此$\vec{v_{(k)}}(p)$向量的模反映了空间各点的边缘的情况。

定义 3(密度距) 表示空间点到 k 近邻点的平均距离，反映了点与周围点的集中或稀疏情况，其计算的公式如下所示：

$$r_{(k)}(p)=\frac{\sum_{i=0}^{k}|\vec{pn_i}-\vec{p}|}{k} \quad (1)$$

表示点 p 与 k 近邻点的平均距离。

定义 4(边缘度密度距) 表示空间点的物理密度的度量值,是一个综合了点所处边缘位置和周围的点的分布情况的度量值,值越大,点的周围越偏稀疏或边缘,其计算的公式如下所示:

$$vr_{(k)}(p)=r_{(k)}(p)+\frac{|\vec{v}_{(k)}(p)|+\sum_{i=1}^{k}|\vec{v}_{(k)}(pn_i)|}{k+1} \tag{2}$$

接下来定义 5 至定义 9 是在定义类的过程中所涉及的相关参数或相关方法。

定义 5(边缘度密度距容忍度) $vr\ \max_C=\max(vr_k(p)),\forall p\in C, vr\ \min_C=\min(vr_k(p)),\forall p\in C$,其中 C 为如下所定义的类。该值在类的定义中会使用到。向量密度距容忍度定义为:

$$tlec=\frac{vr\ \max_C}{vr\ \min_C} \tag{3}$$

定义 6(聚类起点) 设 Cs 表示已经被聚类算法搜索出来的类的集合,聚类起点简写为 o,则 o 满足的性质:

$$o\in P-\{p\mid p\in\bigcup_{C\in Cs}C\}$$

$$\forall p\in P-\{p\mid p\in\bigcup_{C\in Cs}C\}, vr(o)\leqslant vr(p)$$

定义 7(密度可达) 聚类起点 o 密度可达 q,若存在(o, $p_1,p_2,\cdots,p_n$)序列,其中 $q=p_n$,并且序列之间满足如下关系(其中 α 位某个类中最大密度距与最小密度距的比例):

1) $p_{i+1}\in p_i(k)$;

2) $\frac{vr(p_k)}{vr(o)}\leqslant\alpha * tlec$,并且 $\frac{vr(p_{i+1})}{vr(p_i)}\leqslant tlec$。

定义 8(密度连接) 点 p 从点 q 密度连接,若存在聚类起点 o,使得($o,p_1,\cdots,p_n$),($o,q_1,\cdots,q_m$),其中 $p_n=p,q_m=q$ 且有 o 密度可达 $p_i(i\in(1,n))$,o 密度可达 $q_i(i\in(1,m))$。有图 2 所示示例。

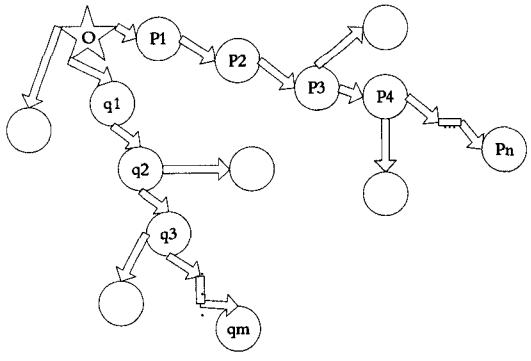

图 2 密度连接示意图

图 2 中箭头起点标记为 start(如 p_3 相对 p_4 来说为 start,p_4 相对 p_3 来说为 end),终点标记为 end。如图 2 中的 q_1 和 q_2 由箭头连接,q_1 为起点 start,q_2 为终点 end。箭头表示的关系是 $\frac{vr(start)}{vr(end)}\leqslant tlec$,并且 $\frac{vr(p)}{vr(o)}\leqslant\alpha * tlec$($p$ 为图 2 中任何 o 的密度可达的点),中间的空心矩形表示省略路径中的点。图 2 所示 p_n 与 q_m 是密度连接的。

定义 9(类) 非空集合 C 是一个类,当且仅当 C 满足以下条件:

1)存在唯一聚类起点 o,$o\in C$;

2)对于 p,若 o 密度可达 p,则 $p\in C$;

3)对于 p,q,有 $p\in C$ 和 $q\in C$,则 p 和 q 是密度连接的。

3.3 算法描述

按照所述定义,Caed 算法接受输入空间向量点集,识别输出点集所包含的类别。算法初步需要统计 k 近邻的邻接情况和边缘度密度距的值。随后需选择合适的聚类中心,从中心处展开点集的聚类操作。在选择合适的聚类中心步骤中,由算法 1 提供桶排序的快速选择算法和相关的结构数据。算法 2 是从聚类中心展开聚类操作的算法描述。算法 3 是 Caed 算法的详细文字描述,在它的步骤中有调用到算法 1 与算法 2。

3.3.1 算法 1

Algorithm 1 splitNode(sonNode,maxPoint)

输入:带有已经计算 vr 的点链表的结构体 node 和最大节点个数 maxPoint。

输出:按 vr 大小划分好的 node。

```
1. if node.inList.size > maxPoint
2.     len←ceil(node.inList.size / maxPoint);
3.     node.son←new node[len];
4.     len←ceil((node.max−node.min) / maxPoint);
5.     for point:node.inList
6.         place← (point.vr−node.mine) / len
7.         if node.son[place].min>point.vr
8.             node.son[place].min← point.vr;
9.         if node.son[place].max<point.vr
10.            node.son[place].max←point.vr node.son[place].inList.add(point);
11. node.inList.delete(point);
12.   node.inList←null;
13.   for sonNode:node.son
14.       splitNode(sonNode,maxPoint).
```

上述步骤完成按 vr 大小划分 point 链表的任务。判断当前节点的点个数是否大于最大点限度值 *maxPoint*,当大于最大点限度值 *maxPoint* 时,根据当前节点中所记录的最大值与最小值,新建合适大小的 node 数组。遍历当前节点点集,计算各点的 *len* 的模数,将其放入到数组以模数为下标的节点中,然后将当前节点的本结构体指针指向新建数组。同时,删除本节点中的点集。算法 1 为边缘度密度距划分操作,为算法 3 做准备工作。

3.3.2 算法 2

Algorithm 2 pointCluster(point,category)

输入:聚类起点

输出:聚类起点所对应的类

```
1. sourceQueue.add(point);
2. point.inQ ← false;
3. for nextPoint:sourceQueue
4.   if nextPoint.calculate
5.     nextPoint.calculate ← false;
6.     for near:nextPoint.near
7.       if near.value ! =null && near.value.inQ
8.         near.value.inQ ← false;
9.         sourceQueue.add(near);
10.      end if
11.    end for
12.  end if
13.  category.pointList.add(nextPoint);
```

14. end for

15. return category.

上述代码描述从聚类起点开始进行的聚类步骤。算法 2 选中聚类起始点后，采用先进先出队列的方式，将当前 k 近邻的点和以当前点为 k 近邻的点放入队列，然后循环取出队列中的点(直到队列为空)，计算该点是否与当前聚类起始点密度可达。密度可达则把它归类于同一类的列表中，同时将该点的未被处理的 k 近邻放入队列。

3.3.3 算法 3(聚类操作)

Algorithm 3 cluster(pointListFile)

输入：记录待聚类点集 P 的文件

输出：聚类好的类

Step 1 从文件读入聚类点集，确定单维上的分割数大小。将维度上的分割数和点集表作为 Step 2 的输入。算法的格在单个维上的分割数$\propto n^{\frac{1}{dimension}}$，格数的总规模是$\prod_{i=1}^{dimension} len_i = \prod_{i=1}^{dimension} n^{\frac{1}{dimension}} = n$，与 n 处于同数量级。

Step 2 获取单维数分割数和点集，遍历输入的点集，将空间点收入到对应的格对象中，然后建立网格和网格的近邻关系。将格链表作为 Step 3 的输入。

Step 3 获取格链表，遍历建立的格及每个格中的所有点集，对每一个点集，按照格的近邻关系，计算当前点的 k 最近邻；再将格链表传至 Step 4。

Step 4 计算格和相邻格中的点与当前点的边缘向量和平均向量模，然后按照式(2)计算各自边缘度密度距(vr)，并记录密度距的最大值和最小值。将处理好的点集加入新建的 node 结点，同时将 vr 的最大值与最小值记录在 node 相关属性中。将 node 作为 Step 5 的输入。

Step 5 设置区间点数的大小 maxPoint。将参数 node 与 maxPoint 作为算法 1 的输入，将边缘度密度距按大小划分到合适的位置。将处理后的 node 作为 Step 6 的输入，转至 Step 6。

Step 6 查看输入本步骤的 node 结点点集是否为空，空则将 node 作为 Step7 的输入，跳至 Step 7；否则对 node 的 inList 进行排序(常数内完成，因为结点个数小于 maxPoint)。遍历排序后的点，如点未被分类则将点作为 Step 8 的聚类起点，转至 Step 8。遍历完，转至 Step 9。

Step 7 查看当前 node 的 son 指针是否为空指针，是则退出本步骤，否则将本指针所指数组元素，逐个作为输入 Step 6 的 node，调至 Step 6。

Step 8 categoryList 中增加新的类项，将输入的聚类起点作为算法 2 的输入，转至算法 2 进行从聚类起点开始的聚类操作。

Step 9 返回 categoryList。

聚类的步骤是严格按照类的定义来进行聚类的过程，也是所定义的聚类步骤的具体实现。

3.4 时间复杂度分析

在算法 3 中各步骤所进行的处理工作的时间复杂度分析如下。

1)确定输入数据集 P 各维度的边界，建立网格后将 P 放置于格内的时间复杂度为 n；建立格相邻关系的时间复杂度为 $n * 3^d$(d 为数据的维度，常数)。

2)计算所有空间点的 k 近邻，然后计算各自的密度距。由于 k 近邻的计算限制在了相邻格，因此存在常数 c 使得任意点的统计操作的时间复杂度时间小于 c，所以处理过程的时间复杂度为 $c * n$。

3)对 n 个点进行聚类分析的时间复杂度是 $n * k$，因为总共具有 $n * k$ 个 k 近邻。

由上述统计可知 Caed 算法的时间复杂是 $n * 3^d + (C+1+k)n = (3^d + C + 1 + k)n = O(n)$。其与 Kmeans 和 Clique 算法具有相同的时间复杂，比基于密度的 Dbscan 算法的 $O(n^2)$ 时间复杂度小，理论上具有较领先的时间效率。

4 实验与分析

实验部分使用了两个数据集，其中一个是人工数据集，另一个是 UCI 页面上用于测试聚类算法的真实数据集。测试算法过程中，所有数据集的点都进行了分类处理，故实验结果只包括算法的精度和算法所消耗的时间，以此来进行实验结果的展示和对比。其中有部分算法不能将少部分的点归类，比如 Dbscan 和 Clique 中密度低的点，因为处于低密度区域的点被视为孤立点。处理方式是将其归类为距离聚类中心最近的点。为了避免数据因不同单位而造成的影响，对数据集同时进行了正规化和规范化两种方式处理。正规化公式为：

$$x_i' = \frac{x_i - \min\limits_{1 \leqslant j \leqslant n}\{x_j\}}{\max\limits_{1 \leqslant j \leqslant n}\{x_j\} - \min\limits_{1 \leqslant j \leqslant n}\{x_j\}} \tag{4}$$

规范化公式为：

$$x_i' = \frac{x_i - \bar{x}}{s} \tag{5}$$

这里

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{6}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{7}$$

式(4)－式(6)中参数 $i \in (1, n)$。各种算法的参数设置中，Caed 算法设置的 $tlec$(建议范围为 1.6～2.5)为 1.85，α 为 2。其他算法密度阈值均设置为平均的空间密度 ϵ。对于 Dbscan 算法的参数，$\epsilon = number/(\frac{4 * pi * Eps^3}{3})$，其中 Eps 设置为 $\sqrt[2.5]{number} * \log10(number)$，$number$ 表示待聚类点的个数，而 Clique 也设置密度阈值为 ϵ。Kmeans 初始聚类中心由随机数方法生成。

本实验的环境为 CPU：Intel core i7 2630 qm，内存：8G，硬盘：500G 单碟，操作系统：windows 7 spl 32 位，编程语言：java。

人工构造数据集的点集分布图形如图 2 所示。人工数据的生成使用了 C++QT 所写的简单程序，由人为绘制标注得到不同类型的数据。不同种类分别由连通性、密度大小、邻接情况人工预先确定。

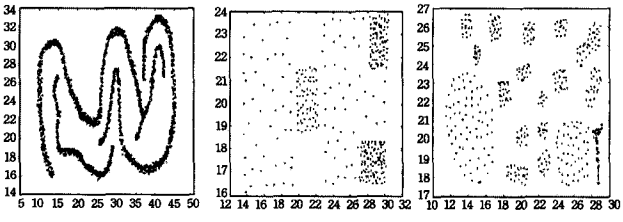

图 3 DataSet1，DataSet2，DataSet3

图 3 左边数据密度具有相似性，不同的类主要是由连通性予以区分的。中间数据集主要区分不同类的密度与类的连通性。将测试结果与人工标记的期望值进行比较。将上述数据分别用作为 Dbscan、Clique、Kmeans 和 Caed 的输入点集 P，运行处理后，统计分析实验数据的结果。测试的各算法的精度如表 1 所列。

表 1　算法精度对比实验结果

	种类数	Dbscan	Clique	Kmeans	Caed
DataSet1	3	86%	85%	56%	100%
DataSet2	18	92%	90%	60%	98%
DataSet3	5	91%	70%	80%	99%

各聚类方法的精度趋势变化图如图 4 所示，时间消耗对比图如图 5 所示。

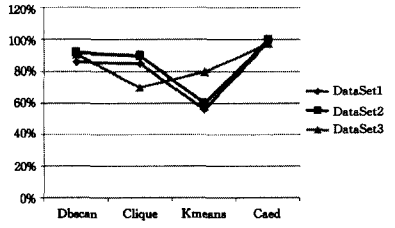

图 4　DataSet 测试精度对比图

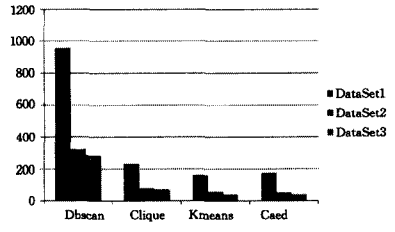

图 5　DataSet 时间消耗对比图(毫秒)

另外一类用于聚类分析真实数据集。其中 Iris 以鸢尾花的特征作为数据来源，是在数据挖掘、数据分类中非常常用的权威的测试集数据集。收集来自 3 种不同品种的 Iris，包括 Iris 花瓣的长度和宽度、花萼的长度和宽度 4 个属性。第二个数据集是随机选自 Kama、Rosa 和 Canadian 3 个品种的小麦种子的 7 个属性的数据。将它们作为各算法的输入数据，测试的结果如表 2 所列。

表 2　实验结果 2

	种类数	Dbscan	Clique	Kmeans	Caed
Iris	3	75%	69%	78%	95%
Seeds	3	85%	82%	71%	91%

精度和消耗时间对比图如图 6 和图 7 所示。

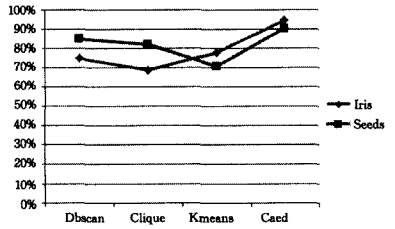

图 6　Iris 与 Seeds 数据集测试精度对比图

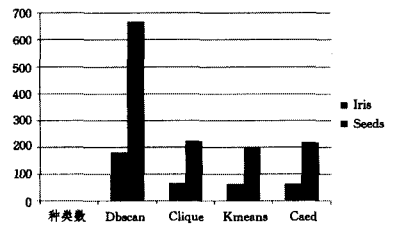

图 7　Iris 与 Seeds 数据集消耗时间对比图(毫秒)

在上述与 Dbscan、Clique、Kmeans 具有代表性的算法的比较中，Caed 在综合精度和时间复杂度方面都具有明显的优势。实验结果也反映出各个算法各自的特点，如 Kmeans、Caed 算法的低时间复杂度。Caed 算法能根据密度的差异，发现任意形状的不同密度的类，由于对密度的敏感性，能快速检测出不同密度的数据点，并将其归并为与其相邻的密度相近的类。实验结果表明，所定义的新的密度度量值及算法在聚类精度和聚类复杂度上都有较好的表现。

结束语　本文通过结合基于密度的聚类方法和基于网格的聚类方法两类算法的聚类思想，定义了一种新的密度度量和与之对应的聚类规则，即边缘度密度距聚类方法及其聚类规则。所述算法具有聚类结果与集合输入顺序不敏感，聚类过程可以识别不同密度、任意形状的类的特点。理论分析和后来的实验结果说明，Caed 算法不仅在时间复杂度上有接近线性的优势，而且也保持了基于密度的聚类方法的较高精度，获得了较出色的实验结果。

参 考 文 献

[1] Escudero L F, Garín M A, Pérez G, et al. Scenario Cluster Decomposition of the Lagrangian dual in two-stage stochastic mixed 0-1 optimization [J]. Computers & Operations Research, 2013, 1(40): 362-377

[2] Wang W, Yang J, Muntz R. STING: A statistical information grid approach to spatial data mining [C]// Proc. of the 23rd Very Large Databases Conf. (VLDB 1997). Athens, Greece. 1997: 186-195

[3] Rakai L, Farshidi A, Behjat L, et al. A New Length-Based Algebraic Multigrid Clustering Algorithm [J]. VLSI Design, 2012, 2012

[4] Demirtas E A. A Data Envelopment-Based Clustering Approach for Public Sugar Factories in Privatizing Process [J]. Mathematical Problems in Engineering, 2011, 2011

[5] Xu X-W, Ester M, Kriegel H P, et al. A distribution-based clustering algorithm for mining in large spatial databases [C]// Proc. 14th Internat. Conf. on Data Eng. (ICDE 98). Orlando, FL, 1998: 324-331

[6] Zhong Y-F, Zhang L-P. A New Fuzzy Clustering algorithm Based on Clonal S election for Land Cover Classification [J]. Mathematical Problems in Engineering, 2011, 2011

[7] Elbatta M T, Bolbol R M, Ashour W M. A Vibration Method for Discovering Density Varied Clusters [J]. ISRN Artificial Intelligence, 2012, 2012

[8] Karypis G, Han E H, Kumar V. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling [J]. IEEE Computer, 1999, 32(8): 68-75

[9] 孙吉贵，刘杰，赵连宇. 聚类算法研究[J]. 软件学报，2008，1(19)：48-64

[10] 赵慧，刘希玉，崔海青. 网格聚类算法[J]. 计算机技术与发展，2010，9(20)

[11] Bhattacharya G, Ghosh K, Chowdhury A S. An affinity-based new local distance function and similarity measure for KNN algorithm [J]. Pattern Recognition Letters, 2012, 33(3): 356-363

[12] Emre Celebi M, Kingravi H A, Vela P A. A comparative study of efficient initialization methods for the Kmeans clustering algorithm [J]. Expert Systems with Applications, 2013, 40(1): 200-210

[13] Tsai C, Chiu C. Developing a feature weight self-adjustment mechanism for a Kmeans clustering algorithm [J]. Computational Statistics & Data Analysis, 2008, 10(52): 4658-4672

[14] Mok P, Huang H, Ylkwok E. A robust adaptive clustering analysis method for automatic identification of clusters [J]. Pattern Recognition, 2012, 45(8): 3017-3033