文章编号: 1001-0920 (2015) 01-0039-07 DOI: 10.13195/j.kzyjc.2013.1501

基于连通分量的分类变量聚类算法

周红芳[1], 周 扬[1], 张晓鹏[2], 谈姝辰[1]

(1. 西安理工大学 计算机科学与工程学院，西安 710048；2. 陕西应用物理化学研究所，西安 710061)

摘 要: 针对分类变量相似度定义存在的不足, 提出一种新的相似度定义. 利用新的相似度定义, 将数据集抽象为无向图, 将聚类过程转化为求无向图连通分量的过程, 进而提出一种基于连通分量的分类变量聚类算法. 为了定量地分析该算法的聚类效果, 针对类别归属已知的数据集, 提出一种新的聚类结果评价指标. 实验结果表明, 所提出的算法具有较高的聚类精度和聚类效率.

关键词: 聚类；分类变量；相似度；连通分量；聚类精度

中图分类号: TP311.13 **文献标志码:** A

A clustering algorithm for categorical variables based on connected components

ZHOU Hong-fang[1], *ZHOU Yang*[1], *ZHANG Xiao-peng*[2], *TAN Shu-chen*[1]

(1. School of Computer Science and Engineering，Xi'an University of Technology，Xi'an 710048，China；2. Shanxi Applied Physics and Chemistry Research Institute，Xi'an 710061，China. Correspondent：ZHOU Hong-fang，E-mail: zhouhf@xaut.edu.cn)

Abstract: For the insufficient similarity concepts for categorical variables, a new more reasonable concept is proposed. Firstly, a data set is organized into an undirected graph by the new definition. The clustering process is converted into the problem of determining connected components in the undirected graph. Then a novel clustering algorithm for categorical variables based on connected components is proposed. In order to analyze the clustering results quantitatively, a new index is proposed for the known labels. Finally, the experimental results show that the proposed algorithm has a higher clustering precision and faster execution speed compared with several existing ones.

Keywords: clustering；categorical variables；similarity；connected components；clustering precision

0 引 言

聚类是数据挖掘的一个重要研究课题, 早期的聚类算法采用距离来度量2条记录之间的相异度, 如K-means[1]、DBSCAN[2]等算法. 对于分类变量数据集, 可以利用已有的标准化方法[3-6]将其转化为区间标度变量, 从而采用传统方法进行聚类. 但分类变量属性值之间通常不存在数量关系, 导致标准化工作具有很大的盲目性. 因此, 采用传统方法处理分类变量会影响聚类效果. Guha等[7]提出的ROCK算法引入了链接(Link)的概念, 从而可以利用相关的全局信息来度量记录之间的相似度. 实验表明, 利用ROCK算法对分类变量数据集进行聚类时, 所得到的聚类结果明显优于传统聚类算法. 但它也存在一定的缺陷, 如需要预先给定判定是否为近邻的参数θ和聚类数k. 目前, 已有一些学者提出了基于ROCK算法思想的改进算法, 如VBACC[8]、QROCK[9]、DNNS[10]和GE-ROCK[11]. VBACC采用基于商品价格的相似度定义, 对于属性稀疏的商品数据集而言, 聚类效果较好, 而对于一般分类变量数据集(如UCI标准数据集)而言, 聚类效果不及ROCK. QROCK认为期望得到的聚类数k依赖于相似度阈值θ, 通过适当选择θ可以消除参数k. 与ROCK算法相比, QROCK算法速度较快, 但

收稿日期: 2013-10-29；**修回日期:** 2014-03-20.

基金项目: 国家自然科学基金项目(61402363, 61272284)；陕西省工业攻关项目(2014K05-49)；陕西省自然科学基础研究计划项目(2014JQ8361)；西安市碑林区科技计划项目(GX1405)；西安市科学计划项目(CXY1339(5))；校特色研究计划项目(116-211302).

作者简介: 周红芳(1976−), 女, 副教授, 博士, 从事数据仓库、数据挖掘等研究；周扬(1987−), 男, 硕士生, 从事数据挖掘的研究.

聚类效果与ROCK算法相同. DNNS算法利用动态近邻选择模型, 将相似度作为权重作用于聚类的全过程. 此外, DNNS引入内聚度度量函数指导聚类过程, 可以自动寻找最佳聚类效果, 并得到较高的聚类精度. 虽然在最坏的情况下, DNNS算法与ROCK算法时间复杂度相同, 但由于DNNS在执行的过程中考虑了更多的近邻, 导致该算法通常慢于ROCK. GE-ROCK借助遗传算法的思想, 将相似度作为权重作用于聚类的全过程, 得到了较好的聚类结果, 但基因遗传算法本身的特点决定了GE-ROCK算法速度较慢. 文献[12]利用分布式思想实现了ROCK算法, 提高了算法的速度, 但算法精度与ROCK相同. 在ROCK及其改进算法中, DNNS在未提高时间和空间复杂度的情况下, 提高了算法的精度, 因此本文选用DNNS作为对比算法. Squeezer[13]算法与ROCK算法思想不同, 该算法通过定义数据点和簇的相似度, 仅需一次遍历数据集即可完成聚类, 因此速度远快于ROCK及其改进算法.

聚类速度慢或聚类结果不理想是现阶段分类变量聚类算法的一个最主要问题. 本文分析了以上问题产生的原因, 并提出了一种基于连通分量的分类变量聚类算法CABOC (Clustering algorithm for categorical variables based on connected components). 标准数据集的实验结果表明, 与现有的分类变量聚类算法相比, CABOC算法具有较高的聚类精度和较快的聚类速度.

1 属性权重相似度

1.1 传统相似度定义存在的问题

Jaccard Coefficient[14]被广泛应用于分类变量相似度的计算, 计算公式如下:

$$\mathrm{Sim}(T_1,T_2)=\frac{|T_1\bigcap T_2|}{|T_1\bigcup T_2|}. \tag{1}$$

其中: T_1, T_2 为2条记录属性值的集合; $|T_1\bigcap T_2|$ 为2条记录属性值交集的基数; $|T_1\bigcup T_2|$ 为2条记录属性值并集的基数. 数据集举例如表1所示.

表 1 数据集举例

ID	属性 A	属性 B	属性 C	属性 D
1	a	b	c	d
2	a	b	c	e
3	a	b	c	f
4	a	m	c	f

对于表1中的数据集, 属性 B 的值域为 $\{b,m\}$. 对于整个数据集而言

$$P(B=b)=\frac{1}{2},\ P(B=m)=\frac{1}{2},$$

即对于任意2条记录 X, Y 都有

$$P(X_B=b)=\frac{1}{2},\ P(Y_B=b)=\frac{1}{2}.$$

因为 $X_B=b, Y_B=b$ 是相互独立事件, 所以

$$P(X_B=b,Y_B=b)=$$
$$P(X_B=b)P(Y_B=b)=\frac{1}{4},$$

同理可得

$$P(X_B=m,Y_B=m)=$$
$$P(X_B=m)P(Y_B=m)=\frac{1}{4}.$$

又因为 $X_B=b$ 且 $Y_B=b$, $X_B=m$ 且 $Y_B=m$ 是互斥事件, 所以

$$P(X_B=Y_B)=$$
$$P(X_B=b,Y_B=b)+P(X_B=m,Y_B=m)=\frac{1}{2}.$$

属性 D 的值域为 $\{d,e,f\}$, 对于整个数据集而言

$$P(D=d)=\frac{1}{3},\ P(D=e)=\frac{1}{3},$$
$$P(D=f)=\frac{1}{3}.$$

对于任意2条记录 X, Y 都有

$$P(X_D=d)=\frac{1}{3},\ P(Y_D=d)=\frac{1}{3}.$$

因为 $X_D=d, Y_D=d$ 是相互独立事件, 所以

$$P(X_D=d,Y_D=d)=$$
$$P(X_D=d)P(Y_D=d)=\frac{1}{9},$$

同理可得

$$P(X_D=e,Y_D=e)=$$
$$P(X_D=e)P(Y_D=e)=\frac{1}{9},$$
$$P(X_D=f,Y_D=f)=$$
$$P(X_D=f)P(Y_D=f)=\frac{1}{9}.$$

又因为 $X_D=d$ 且 $Y_D=d$、$X_D=e$ 且 $Y_D=e$ 和 $Y_D=f$ 且 $X_D=f$ 是互斥事件, 所以

$$P(X_D=Y_D)=$$
$$P(X_D=d,Y_D=d)+P(X_D=e,Y_D=e)+$$
$$P(X_D=f,Y_D=f)=\frac{1}{3}.$$

由Jaccard Coefficient定义可知, 记录2与记录3之间的相似度等于记录3与记录4之间的相似度. 但由上述推导可以看出, 属性 B 相等的概率为 $\frac{1}{2}$, 而属性 D 相等的概率为 $\frac{1}{3}$, 即属性 D 相等的条件更加难以满足. 因此, 一旦属性 B 与属性 D 同时相等, 属性 D 对于整条记录相似度的贡献应该大于属性 B. 但Jaccard Coefficient将属性 B 与属性 D 同等对待, 不能

反映这种差异.

1.2 属性权重相似度定义

定义 1 (数据集)　以 4 元组 (R, A, V, f) 定义数据集 DS. 其中: R 为数据集中所有记录组成的集合; A 为数据集中所有记录属性的集合; V 为 A 中所包含属性的属性值集合; f 为映射函数 f: $R \times A \to V$, 可表示为

$$(\forall x)(\forall a)((x \in R) \bigwedge (a \in A) \to f(x, a) \in V_a).$$

定义 2 (属性权重相似度)　记录 x, y 的属性权重相似度定义为

$$\mathrm{Sim}(x, y) = \frac{\sum\limits_{a \in A} \varphi(x, y, a)}{\sum\limits_{i=1}^{d} |V_i|}. \tag{2}$$

其中: $\mathrm{Sim}(x, y)$ 为记录 x 与记录 y 的属性权重相似度; d 为数据集的维度; V_i 为第 i 维属性值的集合, $|V_i|$ 为第 i 维属性值集合的基数, 即第 i 维属性中不同属性值的个数.

$$\varphi(x, y, a) = \begin{cases} |V_i|, & f(x, a) = f(y, a); \\ 0, & f(x, a) \neq f(y, a). \end{cases}$$

对于表 1 所示的数据集, 属性权重相似度可以客观地反映 2 条记录之间的相似度. 此外, 利用 Jaccard Coefficient 计算相似度时, 考虑到属性的有序性, 假定 d 为数据集的维度, 则所有数据点的相似度只有 d 个孤立的值. 若以 θ 作为判断是否为近邻的参数, 则 θ 在一定范围内变化时, 聚类结果完全不变, 当 θ 超过某值时, 聚类结果会出现较大波动. 此种情况会导致难以通过选择 θ 来控制聚类效果. 而采用属性权重相似度, 根据计算公式, 任意 2 个数据点的属性权重相似度分母相同, 仅考虑分子. 假定 d 为数据维度, d 个属性的属性值构成了 d 个集合 $V_1, V_2, \cdots, V_d$, 假定 d 个集合的基数 $|V_1|, |V_2|, \cdots, |V_d|$ 各不相同. 若 2 个数据点的属性都不相等, 则有 1 个属性权重相似度 (属性权重相似度为 0); 若 2 个数据点有 1 个相等的属性, 则有 C_d^1 个不同的属性权重相似度; 若 2 个数据点有 2 个相等的属性, 则最多有 C_d^2 个不同的属性权重相似度. 依此类推, 任意 2 个数据点不同的属性权重相似度个数的最大值为

$$1 + C_d^1 + C_d^2 + \cdots + C_d^d =$$
$$C_d^0 + C_d^1 + C_d^2 + \cdots + C_d^d =$$
$$(1 + 1)^d = 2^d.$$

以 22 维数据集 mushroom 为例, 各数据点之间最多有 2^{22} 个不同相似度, 适合利用 θ 控制聚类效果.

2 CABOC 算法

2.1 算法思想

根据任意 2 个数据点之间属性权重相似度与参数 θ 的关系 (参数 θ 预先给定), CABOC 算法将数据集抽象为无向图. 当数据集中 2 个数据点的属性权重相似度大于等于 θ 时, 认为 2 个数据点之间有边; 当数据集中 2 个数据点的属性权重相似度小于 θ 时, 认为 2 个数据点之间无边. 确定该无向图后, 无向图的每个连通分量即为 1 个簇, 簇中的记录为连通分量中的各顶点. CABOC 算法实质是寻找无向图各连通分量所包含的顶点, 可以采用图遍历算法的思想指导聚类过程, CABOC 算法步骤如下所述.

Step 1: 输入数据集 DS 和相似度阈值 θ.

Step 2: 求任意 2 个数据点的属性权重相似度.

Step 3: 将任意 2 个数据点的属性权重相似度与 θ 作比较, 当 2 个数据点的属性权重相似度大于或等于 θ 时, 在 2 个数据点之间建立一条边.

Step 4: 若数据集中所有数据点已被标记为已聚类, 则执行 Step 6; 否则, 在数据集中取未被聚类的数据点 i, 执行 Step 5.

Step 5: 将 i 加入一个新簇 C 中, 将 i 标记为已聚类. 将所有与 i 之间有通路且未被聚类的数据点加入到簇 C 中, 再将这些数据点标记为已聚类, 之后执行 Step 4.

Step 6: 输出聚类数和每个簇包含的数据点.

与 ROCK 及其改进算法相比, CABOC 算法避免了计算数据集的链接矩阵, 提高了算法的运行速度. 与 Squeezer 算法相比, CABOC 算法聚类结果仅与数据集相关, 与数据集中数据点的排列顺序无关, 聚类结果全局最优. CABOC 算法主程序伪代码如下所示:

```
1) Begin
2) Input similarity threshold θ.
3) Calculate weighted-attribute similarity between
   each pair of data points.
4) Create undirected graph according to θ and simi-
   larity between each pair of data points
5) for (i = 0; i < n; i++) {
   //n is the number of data points in data set
6)    if(i is not clustered){
7)       create a new cluster C ;
8)       addDataPointToCluster(C, i);
9)    }
```

10) }

11) End

子函数function addDataPointToCluster(C, i)的伪代码如下所示:

1) function addDataPointToCluster(C, i) {

2) C. add(i);

3) i. setClustered();

4) for every data point j connected with i {

5) if (j is not clustered)

6) addDataPointToCluster(C, j);

7) }

8) }

在主程序中, 代码2)∼4)根据2个数据点之间的属性权重相似度和参数θ的关系创建无向图. 从代码5)开始执行聚类算法: 首先, 对于每1个数据点i, 如果未被聚类, 则建立一个新簇C; 然后, 调用子函数addDataPointToCluster, 将该数据点添加到簇C中, 并将i标记为已聚类; 最后, 扫描所有与数据点i相邻的数据点j, 如果j未被聚类, 则递归调用子函数addDataPointToCluster, 当递归过程执行完毕时, 一个簇形成. 返回主程序5), 建立一个新簇, 重复以上步骤, 直到循环结束, 算法执行完毕.

2.2 时间复杂度与空间复杂度分析

CABOC算法在构建无向图时, 需要计算任意2个数据点的属性权重相似度, 并进行$0.5n(n+1)$次运算(n为数据集记录数), 此步骤的时间复杂度为$O(n^2)$. 以邻接表的方式存储构建无向图, 聚类过程采用图的深度优先遍历思想, 该步骤的时间复杂为$O(n+e)$(e为无向图边数). 因此, 总的时间复杂度为$O(n^2+e)$. 在空间复杂度方面, CABOC运行过程中需要缓存所有的数据点以及与每个数据点相邻的数据点. 最坏的情况是所有的数据点都相邻, 此时所需空间为$n+n(n-1)$; 最好的情况是所有的数据点都不相邻, 此时所占空间为n. 因此, CABOC算法的空间复杂度为$O(n+e)$, 其值界于$O(n)$与$O(n^2)$之间.

在ROCK及其改进算法中, 进行聚类前需要预先求出表示近邻关系的邻接矩阵和表示相似性关系的链接矩阵. 求邻接矩阵时, 需求任意2个数据点的相似度, 时间复杂度为$O(n^2)$. 求链接矩阵时, 需将邻接矩阵与其本身相乘, 时间复杂度为$O(n^{2.87})$[15]. 仅此2步已消耗大量时间. 此外, 层次聚类本身也决定了ROCK及其改进算法在聚类时迭代次数较多, 从而导致执行速度较慢. 因此, CABOC算法的执行速度要快于ROCK及其改进算法.

3 实验及分析

3.1 聚类精度

文献[10]利用文献[16]所提出的召回率和精度来衡量聚类结果的优劣. 召回率和精度最初是衡量分类算法结果的指标, 对于分类情况已知的数据集, 在聚类算法得到的聚类数和实际类数相等或相差不大时, 召回率和精度确实能反映聚类结果的质量. 而现实的情况是, 数据集实际的类数和通过运行聚类算法得到的簇数往往相差很大. 以标准数据集mushroom为例, 实际数据集中共有2类(可食与有毒), 但当前任何一种聚类算法发现的聚类数皆为20个左右. 此时, 若仍以召回率和精度评判聚类结果, 则退化为比较2个最大簇的精度, 其余的小簇被忽略. 针对此种情况, 对于分类情况已知的数据集, 本文提出一种新的评判聚类结果的标准, 详述如下.

定义3(簇纯度) 簇纯度为簇中同种类别数量最多的记录在本簇所有记录中所占的比例

$$\mathrm{CP}_i = \frac{\max(|\mathrm{CF}_1|,\cdots,|\mathrm{CF}_r|,\cdots,|\mathrm{CF}_a|)}{\sum_{j=1}^{a}|\mathrm{CF}_j|}\times 100\%. \tag{3}$$

其中: CP_i为第i个簇的纯度, CF_r为在第i个簇中来自第r个实际类的数据点集合, a为数据集中实际类数.

定义4(聚类精度)

$$\mathrm{CPrecision} = \frac{\sum_{i=1}^{k}\mathrm{CP}_i}{k+|k-a|}. \tag{4}$$

其中: CPrecision为聚类精度, CP_i为在第i个簇的簇纯度, k为算法发现的聚类数, a为数据集的实际类数. 新提出的聚类结果评价指标综合考虑了聚类数和簇纯度2个因素. 在各簇纯度相同的情况下, 算法发现的聚类数与实际类数差别越大, 聚类精度越低; 在算法发现的聚类数相同的情况下, 各簇纯度越高, 聚类精度越高.

为了讨论新提出的聚类结果评价指标与已有指标的不同, 现以基于熵的评价指标为例进行说明. 基于熵的评价指标首先需要求得聚类结果中各个簇的熵, 然后根据数据点数对各个簇的熵进行加权平均. 基于熵的评价指标没有考虑算法发现的簇数和数据集中实际类数的差异. 考虑一种比较极端的情况, 假定某个算法将数据集中的每个数据点都当作一个簇, 这种聚类效果是没有实际意义的. 但使用基于熵的评

价指标评价聚类结果时, 熵反而为 0 (熵为 0 说明聚类效果较好). 出现这种不合理现象的原因是基于熵的评价指标没有考虑算法发现的簇数和数据集中实际类别数的差异. 此外, 基于 Rand 系数的评价指标和基于 Jaccard 系数的评价指标都有类似的问题.

3.2 实验结果与分析

实验的数据选自 UCI 标准数据集 (mushroom 数据集和 hayes-roth 数据集). 在众多分类变量聚类算法的文献中, 实验对象是数据集 Zoo 和数据集 Vote. 本文未引用是因为 Zoo 的 16 个属性有 15 个是布尔型, Vote 的 16 个属性全部为布尔型, 当分类变量数据集的属性为布尔型时, 属性权重相似度与式 (1) 所定义的相似度等价. 实验结果不能反映属性权重相似度的优势. 数据集 mushroom 共有 8 124 条记录, 实际分为 2 类, 记录数分别为 4 208 和 3 916; 数据集 hayes-roth 共132条记录, 实际分为 3 类, 记录数分别为 51, 51 和 30.

本文利用 Java 实现 ROCK、Squeezer、CABOC 和 DNNS 算法. 实验中所使用的操作系统为 Windows 7 version 6.1.7600, CPU 为 Intel Core i3-2310 M 2.1 GHz, 内存为 4 G. 对于 ROCK、Squeezer 和 CABOC 三种算法, 需要给定参数, 多次运行算法, 取不同参数的最优结果. 对于 DNNS 算法, 根据文献[10]的思想, 在内聚度度量函数减小时结束算法即可.

表 2 和表 3 分别为 ROCK 算法和 Squeezer 算法对数据集 mushroom 的聚类结果, 表 4 和表 5 分别为 DNNS 算法和 CABOC 算法对数据集 mushroom 的聚类结果. 在表中: E 为 mushroom 中的 edible 类, P 为 mushroom 中的 poisonous 类. DNNS 算法与 Squeezer 算法在簇数分别为 23 和 24 时, 有不纯的簇出现. 而 CABOC 算法在簇数为 22 时, 所有的簇纯度仍然为 100%. 直观上, CABOC 算法发现簇的个数少且每个簇纯度都为 100%, 所以聚类精度高. 本文所提出的聚类精度的定义也验证了这一结果.

表 2　ROCK 算法对 mushroom 聚类

ID	E	P	ID	E	P
1	96	0	13	0	288
2	0	256	14	192	0
3	704	0	15	32	72
4	96	0	16	0	1 728
5	768	0	17	288	0
6	0	192	18	0	8
7	1 728	0	19	192	0
8	0	32	20	16	0
9	0	1 296	21	0	36
10	0	8	–	–	–
11	48	0	–	–	–
12	48	0	–	–	–

在表 2 中, 各参数为

$$\theta = 0.8,\ \mathrm{ClusterNum} = 21,$$

$$\mathrm{CPrecision} = 51.82\%.$$

表 3　Squeezer 算法对 mushroom 聚类

ID	E	P	ID	E	P
1	0	256	13	48	0
2	512	0	14	1	72
3	768	0	15	48	0
4	96	0	16	0	32
5	96	0	17	0	8
6	192	0	18	0	359
7	1 728	0	19	192	0
8	0	1 296	20	288	0
9	0	192	21	0	36
10	0	288	22	31	0
11	192	0	23	0	8
12	0	869	24	16	0

在表 3 中, 各参数为

$$\theta = 16,\ \mathrm{ClusterNum} = 24,$$

$$\mathrm{CPrecision} = 51.08\%.$$

表 4　DNNS 算法对 mushroom 聚类

ID	E	P	ID	E	P
1	96	0	13	0	288
2	672	0	14	192	0
3	0	256	15	32	0
4	96	0	16	0	869
5	768	0	17	288	0
6	0	192	18	0	8
7	1 728	0	19	192	0
8	32	859	20	16	0
9	48	0	21	0	36
10	0	8	22	0	72
11	0	1 296	23	0	32
12	48	0	–	–	–

在表 4 中, 各参数为

$$\mathrm{ClusterNum} = 23,\ \mathrm{CPrecision} = 52.19\%.$$

表 5　CABOC 算法对 mushroom 聚类

ID	E	P	ID	E	P
1	0	256	13	0	72
2	704	0	14	48	0
3	768	0	15	0	32
4	96	0	16	0	8
5	96	0	17	192	0
6	1 728	0	18	288	0
7	0	1 296	19	32	0
8	0	192	20	0	36
9	0	288	21	0	8
10	192	0	22	16	0
11	0	1 728	–	–	–
12	48	0	–	–	–

在表 5 中, 各参数为

$$\theta = 0.89,\ \mathrm{ClusterNum} = 22,$$

$$\mathrm{CPrecision} = 52.38\%.$$

表 6 ROCK 算法对 hayes-roth 聚类

ID	Calss 1	Calss 2	Calss 3
1	2	1	0
2	2	1	0
3	0	4	0
4	1	2	0
5	2	2	0
6	2	0	0
7	3	0	0
8	2	2	0

在表 6 中, 各参数为

$$\theta = 0.8,\ \text{ClusterNum} = 69,$$

$$\text{CPrecision} = 45.51\%.$$

表 7 Squeezer 算法对 hayes-roth 聚类

ID	Calss 1	Calss 2	Calss 3
1	7	13	3
2	17	14	3
3	10	4	3
4	0	0	6
5	16	19	4
6	1	1	3
7	0	0	3
8	0	0	3

在表 7 中, 各参数为

$$\theta = 2.1,\ \text{ClusterNum} = 9,$$

$$\text{CPrecision} = 45.00\%.$$

表 8 DNNS 算法对 hayes-roth 聚类

ID	Calss 1	Calss 2	Calss 3
1	3	0	0
2	5	5	0
3	3	1	0
4	0	3	0
5	0	3	0
6	0	3	0
7	3	0	0
8	0	3	0

在表 8 中, 各参数为

$$\text{ClusterNum} = 60,\ \text{CPrecision} = 46.10\%.$$

表 9 CABOC 算法对 hayes-roth 聚类

ID	Calss 1	Calss 2	Calss 3
1	10	0	0
2	5	5	0
3	10	0	0
4	10	0	0
5	0	10	0
6	0	10	0
7	0	10	0
8	5	5	0

在表 9 中, 各参数为

$$\theta = 0.75,\ \text{ClusterNum} = 9,$$

$$\text{CPrecision} = 47.53\%.$$

表 6∼表 9 的数据是对数据集 hayes-roth 运行各算法所得到的实验结果. 各算法都出现了大量单个数据点的簇, 此种情况与数据集本身有关. 仅将前 8 个簇列出. 与其他 3 种算法相比, 只有 CABOC 发现了 6 个纯度为 100% 的较大簇.

ROCK 算法和 DNNS 算法的时间复杂度为 $O(n^2 + nm_m m_\alpha + n^2 \log n)$, 空间复杂度为 $O(\min\{n^2, nm_m m_\alpha\})$ [7, 10]. 其中: n 为数据集中的记录数, m_m 为最大近邻数, m_α 为平均近邻数. Squeezer 算法的时间复杂度为 $O(n{\cdot}k{\cdot}p{\cdot}m)$, 空间复杂度为 $O(n{+}k{\cdot}p{\cdot}m)$[13]. 其中: n 为数据集中的记录数, k 为最终簇数, p 为属性值的取值范围, m 为数据集维度, 通常 p, k 和 m 的值远远小于 n. 综合第 2.2 节的分析, 在时间复杂度和空间复杂度方面, Squeezer 算法最小, CABOC 算法次之, ROCK 算法和 DNNS 算法最大. 从 mushroom 中选取不同数据量的记录运行各个算法, 运行时间对比如图 1 所示.

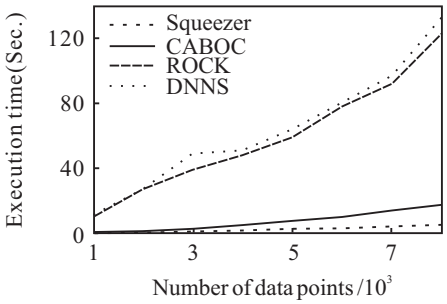

图 1 各算法运行时间对比

需要特别指出的是, 图 1 中 ROCK 算法和 DNNS 算法的运行时间仅指其完成聚类所需的时间, 不包含求链接矩阵的时间, 并且在实现 ROCK 算法和 DNNS 算法时都对其作了改进. 文献[7, 10]中为了选取指导合并函数的最大值, 对整个全局堆和相关局部堆按指导合并函数值从小到大排序. 实际上不必完成排序, 按堆排序的思想找到最大值即可, 从而提高算法运行速度. Squeezer 算法和 CABOC 算法的运行时间为完成整个聚类算法所需的总时间. 即便如此, Squeezer 算法和 CABOC 算法的用时仍然远远小于 ROCK 算法和 DNNS 算法. 在各算法运行时, 利用 jdk 自带的工具 jvisualvm 监控最大内存占用情况, 如图 2 所示. 由于 ROCK 算法和 DNNS 算法都要缓存链接矩阵, 程序运行时所占内存空间大于 Squeezer 算法和 CABOC 算法. 通过上述理论分析以及图中数据都可以得出 CABOC 算法比 Squeezer 算法的时空复杂度略大的结论. 但在精度方面, CABOC 算法要比 Squeezer 算法高

很多, 特别是在对数据集hayes-roth测试时, Squeezer算法发现的所有簇纯度都较低, 导致聚类精度较低.

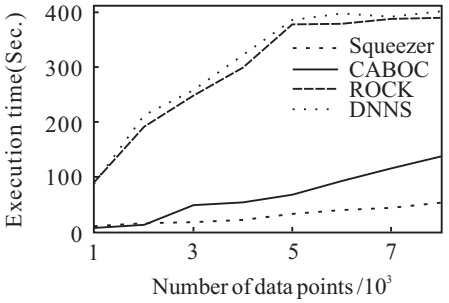

图 2　各算法运行时最大内存占用量对比

4　结　论

本文深入分析了各种分类变量聚类算法的优缺点, 提出了CABOC算法. 该算法利用属性权重相似度的定义, 将聚类过程转化为寻找无向图连通分量的过程. 该算法时间复杂度和空间复杂度较低, 并且改善了分类变量数据集的聚类效果. CABOC算法速度的瓶颈在于无向图创建的过程, 算法在执行速度方面仍有较大的提升空间, 下一步的工作重点是利用分布式的思想实现CABOC算法, 以提高算法执行速度.

参考文献(References)

[1] James B M. Some methods for classification and analysis of multivariate observations[C]. Proc of the 5th Berkeley Symposium on Mathematical Statistics and Probability. Berkeley: University of California Press, 1967: 281-297.

[2] Martin E, Hans P K, Jiirg S, et al. A density-based algorithm for discovering clusters in large spatial databases with noise[C]. Proc of the 2nd Int Conf on Knowledge Discovery and Data Mining. Portland: AAAI Press, 1996: 226-231.

[3] Wu S, Liu J J, Wei G. Clustering algorithm based on condensed set dissimilarity for high dimensional sparse data of categorical attributes[C]. Proc of the 3rd Int Conf on Advanced Computer Control. Harbin: IEEE Press, 2011: 445-448.

[4] Han J W, Kamber M. Data mining: Concepts and techniques[M]. Beijing: China Machine Press, 2008：253-260.

[5] Cao F Y, Liang J Y, Li D Y. A dissimilarity measure of the k-modes clustering algorithm[J]. Knowledge-Based Systems, 2012, 26(15): 120-127.

[6] Natthakan L, Tossapon B, Simon G, et al. A link-based cluster ensemble approach for categorical data clustering[J]. IEEE Trans on Knowledge and Data Engineering, 2012, 24(3): 413-425.

[7] Guha S, Rastogi R, Shim K. ROCK: A robust clustering algorithm for categorical attributes[C]. Proc of the 15th Int Conf on Data Engineering. Sydney: IEEE CS Press, 1999: 512-521.

[8] Joydeep G, Gunjan K G. Value balanced agglomerative connectivity clustering[C]. Proc of the 3rd Int Conf on Data Mining and Knowledge Discovery: Theory, Tools and Technology. Orlando: SPIE, 2001: 6-15.

[9] Dutta M, Dakoti M A, Pujari A K. QROCK: A quick version of the ROCK algorithm for clustering of categorical data[J]. Pattern Recognition Letters, 2005, 26(15): 2364-2373.

[10] 金阳, 左万利. 一种基于动态近邻选择模型的聚类算法[J]. 计算机学报, 2007, 30(5): 756-762.
(Jin Y, Zuo W L. A clustering algorithm using dynamic nearest neighbors selection model[J]. Chinese J of Computers, 2007, 30(5): 756-762.)

[11] Zhou Q B, Ding L X, Zhang S S. A genetic evolutionary ROCK algorithm[C]. Proc of the 1st Int Conf on Computer Application and System Modeling. Taiyuan: IEEE Computer Society, 2010: 347-351.

[12] Anil P, Ritesh J, Surendra M. Implementation of distributed ROCK algorithm for clustering of large categorical datasets and its performance analysis[C]. Proc of the 3rd Int Conf on Electronics Computer Technology. India: IEEE Press, 2011: 78-83.

[13] He Z Y, Xu X F, Deng S C. Squeezer: An efficient algorithm for clustering categorical data[J]. J of Computer Science and Technology, 2002, 17(5): 611-624.

[14] Richard O D, Peter E. Pattern classication and scene analysis[M]. New York: A Wiley-Interscience Publication, 1973: 103-114.

[15] Don C, Siimuei W. Matrix multiplication via arithmetic progressions[J]. J of Symbolic Computation, 1990, 9(3): 251-280.

[16] Fabrizio S. A tutorial on automated text categorization[C]. Proc of the 1st Argentinean symposium on artificial intelligence. Buenos Aires: AR, 1999: 7-35.

(责任编辑: 闫　妍)