第36卷 第19期 计 算 机 工 程 2010年10月
Vol.36 No.19 Computer Engineering October 2010
·软件技术与数据库· 文章编号：1000—3428(2010)19—0056—03 文献标识码：A 中图分类号：TP311.133

基于中心点及密度的分布式聚类算法

冯少荣，张东站
(厦门大学信息科学与技术学院，福建 厦门 361005)

摘 要：针对分布式聚类算法DBDC存在的不足，提出一种基于中心点及密度的分布式聚类算法DCUCD。将数据分布计算出的虚拟点作为核心对象，核心对象的代表性随算法的执行次数提高，聚类即是对所有核心对象分类的过程。理论分析和实验结果表明，该算法能有效处理噪声和分布不规则的数据点，时间效率和聚类质量较好。

关键词：数据挖掘；分布式聚类；中心点；噪声

Distributed Clustering Algorithm Based on Centers and Density

FENG Shao-rong, ZHANG Dong-zhan
(School of Information Science and Technology, Xiamen University, Xiamen 361005, China)

【Abstract】In order to overcome the shortcomings of the DBDC, a distributed clustering based on centers and density which called DCUCD is proposed. It works based on the centers and the density. The virtual core objects are generated from the distributed data and the quality is better if the algorithm runs more times. Clustering is the same as the process to classify all of the core objects. Theoretical analysis and experimental results testify that DCUCD can effectively deal with the problem of local noise, and discover clusters of arbitrary shape. It can generate high quality clusters and cost a little time.

【Key words】data mining; distributed clustering; centers; noise

1 概述

基于密度的聚类[1-2]研究是聚类分析研究的一个重要内容，而基于密度的分布式聚类算法[3-5]是分布式聚类算法[3-7]研究的热点，典型的算法当属DBDC[3]。本文在研究DBDC算法的基础上，提出一种新的基于中心点和密度的分布式聚类算法DCUCD，实验结果表明，该算法是有效可行的，且具有比DBDC和DBSCAN算法更好的性能。

2 基于中心点和密度的聚类算法

基于中心点和密度的聚类算法CUCD采用一定的核心对象代表若干个数据点，核心对象通过计算数据点的密度来获得，它们是虚拟的点，不是实际输入的数据。密度通过计算满足一定距离设置的点数来衡量。通过扩展核心对象形成聚类。利用降维技术，该算法可以支持多维数据的聚类。

2.1 核心对象的生成

为计算对象之间距离，必须记录核心对象的坐标信息。核心对象是虚拟的点，代表了其对应区域内所有点，为了更新核心对象的坐标信息，要求记录每个核心对象的代表区域内所有点的坐标和。为了统计密度，需要记录代表区域内对象数目。

核心对象的选取分为2个步骤:(1)寻找候选核心对象集；(2)过滤不符合密度阈值条件的候选对象。过滤根据阈值对候选核心对象集进行一次扫描即可。生成候选核心对象集的算法如下：

```
//输入：数据集pointSet，半径r，扫描次数Iterate
//输出：候选核心对象集合candidateSet
Void FindingCoreObject(pointSet, r, Iterate, candidateSet) {
candidateSet·add(pointSet·get(0)); //初始化至少一个核心对象
  for (i=0; i<Iterate; i++) { //进行Iterate次迭代
      SingleScan(pointSet, r, candidateSet);
      RegenerateCandidateSet(candidateSet); }
SingleScan(pointSet, r, candidateSet);
}
void SingleScan(pointSet, r, candidateSet) {
//输入：数据集pointSet，半径r，候选核心对象集合
candidateSet
//输出：处理后的候选核心对象集合candidateSet
  for (i=1; i<|pointSet|; i++) {
currentP=pointSet·get(i); //取得一个数据点
  if (!CheckUp(currentP, candidateSet, r)) //生成一个新的核心
//对象
candidateSet·add(currentP); }
}
boolean CheckUp(currentPoint, candidateSet, r) { //检测是否有核
//心对象包含了该点，若有，将数据点的坐标信息加入到包含该点候
//选核心对象的所有代表区域；否则，返回false;
flag=false;
for (j=0; j<|candidateSet|; j++) {
  currentC=candidateSet·get(j);
if (Distance(currentP, currentC)<=r) {
currentC·sumOfxPoint+=currentP·xPoint;
      currentC·sumOfyPoint+=currentP·yPoint;
currentC·density++;
```

基金项目：国家自然科学基金资助项目(50604012)

作者简介：冯少荣(1964－)，男，副教授、博士，主研方向：并行分布式数据库，数据仓库，数据挖掘；张东站，副教授、博士

收稿日期：2010-03-04 **E-mail：**shaorong@xmu.edu.cn

© 1994-2013 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
if  (!flag)   flag =TRUE;  }
}
return  flag;
}
```

RegenerateCandidateSet 函数用于调整候选核心对象集，在 SingleScan 函数第一次执行完成后，已经生成一个候选核心对象集，并且保存了每个候选核心对象代表区域内所有点的坐标和等信息。为了使得核心对象的能够更接近于聚类的中心，将候选核心对象 R 用它当前所代表的所有对象的中心 R′替代。即

```
R′·xPoint=R·sumOfxPoint/R·density;
R′·yPoint=R·sumOfyPoint/R·density; R′·density=1;
R′·sumOfxPoint=R·xPoint; R′·sumOfyPoint=R·yPoint;
```

处理完后，再次执行 SingleScan 得到的候选核心对象能更好地反映输入数据的空间集合形状特征并且更靠近聚类的中心。

主程序 FindingCoreObject 执行 *Iterate* 次，该参数人为设定。通过多次调用 SingleScan 与 RegenerateCandidateSet 函数，保留最后一次执行 SingleScan 产生的候选核心对象的集合。

由于可能存在如下情况：1 个数据点与多个候选核心对象的距离都小于或者等于阈值，下面证明将该数据点归于其中任意一个候选核心对象的代表集合，不会对聚类结果产生影响。

定理 点 p 具体与哪个参考点之间建立映射不会对聚类的结果产生影响，只需满足点 p 与该参考点的距离小于等于 r。

证明：如果核心对象 R_1、R_2 与点 p 的距离都小于等于 r，由三角形边长定理：任意一边的边长小于其余两边的边长之和。可知，核心对象 R_1 和 R_2 之间的距离小于 2 倍的 r。因此，核心对象 R_1 和 R_2 是邻接核心对象。在多维空间中，由于任意不在同一直线上的 3 个点形成一个平面，因此上面的三角形定理仍然适用。当点 p、R_1、R_2 在同一直线上时，R_1 和 R_2 之间的距离小于 2 倍 r，R_1 和 R_2 是邻接核心对象。算法中邻接核心对象具有属于同一个聚类的基本信息。因此，点 p 无论与 R_1 还是 R_2 建立映射，点 p 最终都属于同一个聚类，该定理得证。

主程序 FindingCoreObject 执行完之后，对候选核心对象集合进行过滤，剔除密度小于密度阈值 t 的候选核心点，这样得到的核心对象集合可以较为准确地反映输入数据的集合特征。

2.2 核心对象聚类

核心对象聚类算法如下：

```
void  FormCluster(candidateSet, T, R) {
//输入：核心对象集合 candidateSet，密度阈值 T，全局半径 R
//输出：经过处理的核心对象集合 candidateSet
  int  clusterId=1;
  for  (int i=0; i<candidateSet·size(); i++)  {
    p=candidateSet·get(i);
    if  (p·clusterId==UNCLASSIFIED) {
      ExpandCluster(p, clusterId, candidateSet, R);
      clusterId++; }
  }
}
void  ExpandCluster(p, clusterId, candidateSet, R){ //扩展一个
//聚类
candidateSeeds·add(p);
while  (!candidateSeeds·isEmpty())  {
  pp= candidateSeeds·get(0);
  for  (int i=0; i<candidateSet·size(); i++)  {
currentP=candidateSet·get(i);
if  (currentP·clusterId==UNCLASSIFIED)  {
if  (eDistance(pp, currentP)<=2*R) { //R 为设定参数
currentP·clusterId=clusterId;
candidateSeeds·add(currentP);  }
}
}
  candidateSeeds·remove(0);
}
}
```

3 基于中心点与密度的分布式聚类算法

3.1 局部聚类

局部聚类的基本原则是：传递的数据量要尽可能小；同时还要保证传递的信息精度高、代表性强。

DCUCD 的局部聚类传递给主站点的是局部数据的核心对象集合。子站点上局部核心对象的数据结构与 CUCD 的核心对象相同，抽取核心对象的过程类似于上节所述 FindingCoreObject 函数。

在任意一个子站点上经过多次扫描，得到一个核心对象集合。由于这些核心对象在全局聚类时都有可能形成聚类，在传输条件允许的情况下可以先不进行候选核心对象的筛选，以免丢失聚类。另外，如果生成核心对象集合大小仍然超过传输负载，可以仅选取一个完全子集传输。

假设站点 i 用 $CandidateSet_i$ 来产生局部模型。利用各个子站点传递的核心对象的坐标信息、密度信息，在主站点可以进行聚类。局部模型是一个二元组，$LocalModel_i=<CandidateSet_i, r_i>$，其中，$r_i$ 是局部模型抽取核心对象使用的距离参数。

3.2 全局聚类

对该全局模型应用全局参数 r_G 执行 FindingCoreObject 算法，重新扫描生成全局候选核心对象。该算法与生成局部模型算法类似。算法得到的全局候选核心对象包含的信息是它代表的所有对象的信息求和之后的值。接着对全局候选核心对象集合进行筛选，剔除不满足全局密度阈值条件 T 的候选对象，小于阈值的对象鉴定为噪声。继而通过对核心对象的扩展形成聚类。扩展过程是对邻接对象合并的过程。

经过筛选、扩展，形成对全局核心对象的聚类。聚类中的对象是一系列虚拟的点。将聚类的信息传回子站点，子站点根据聚类集合的信息，计算输入数据与聚类中对象的距离。如果某个数据点在聚类中某一对象的距离邻域内，该数据点属于这个对象的聚类；否则，标识为噪声。

4 实验与分析

4.1 算法效率分析

假设某站点的数据量为 N，候选核心对象的数目为 M，循环次数为 I，扫描得出的核心对象数目为 K。则 SingleScan 函数的时间复杂度为 $O(K·N)$, RegenerateCandidateSet 函数的时间复杂度为 $O(K)$，寻找候选核心对象的时间复杂度总共为 $O(K·(I\text{-}1)+K·N·I)$。对候选参考点的筛选在 $O(K)$可以完成。对核心对象集合分类采用图的广度优先算法，时间复杂度为 $O(M·M)$。最后，对所有数据点进行标识的时间复杂度为 $O(M·N)$。可见，CUCD 的时间复杂度为 $O(K·(I\text{-}1)+K·N·I)+O(K)+O(M·M)+O(M·N)$，通常 K、I、$M<<N$，因此，CUCD 算

© 1994-2013 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

法的时间复杂度近似为 $O(I\cdot K\cdot N+M\cdot N)$，算法执行效率很高。

所有局部模型在主站点集成后，假设主站点上数据量为 N_G，候选核心对象的数目为 M_G，循环次数为 I_G，扫描得出的核心对象数目为 K_G，生成全局候选核心对象的时间复杂度为 $O(K_G\cdot(I_G-1)+K_G\cdot N_G\cdot I_G)$。

全局候选核心对象筛选过程的时间复杂度为 $O(M_G)$。如果通过生成图再搜索连通子图的方式形成聚类，全局聚类的时间复杂度为 $O(K_G\cdot(I_G-1)+K_G\cdot N_G\cdot I_G)+O(M_G)+O(M_G\cdot M_G)$，一般 K_G、I_G、$M_G <<N_G$，全局聚类可得到线性时间复杂度。

在子站点上对各个数据对象进行聚类标识的过程相当于一趟扫描，时间复杂度为 $O(M\cdot N)$。可以看出，DCUCD 算法的总体效率很高。

4.2 实验与性能比较

手工构造包含 3 086 个数据点的数据集 $Test_1$。将它们分布于 3 个子站点，数据量分别为：站点 A(930)，站点 B(950)，站点 C(1 206)。

DCUCD 聚类算法的时间消耗由各站点的局部聚类时间与全局聚类时间求和获得，不考虑网络的传输时间。由于对子站点数据的标识相当于对数据集的一趟扫描，耗费的时间很少，不计入分布式聚类时间内。

为了比较聚类的质量，在所有 3 086 个数据点上执行 DBSCAN 算法。DBSCAN 的执行时间为 563.0 ms，结果中噪声点数目为 590。

在 $Test_1$ 上分别执行分布式聚类算法 DBDC 与 DCUCD，DBDC 算法的时间耗费 260 ms。DCUCD 算法的时间耗费为 78.5 ms。

DBDC 在 3 个子站点上产生的噪声数为 657，其中，计算错误的点 67 个。按照实验采取的统计方法，DBDC 的聚类质量为(3 086-67)/3 086=97.82%。DCUCD 在 3 个子站点上产生的噪声数为 758，其中，计算错误的点 168 个；DCUCD 的质量为 94.52%。

对手工构造的另外 3 个数据集($Test_2$:7 438, $Test_3$: 20 553, $Test_4$: 64 590)进行相同的测试过程。

图 1 是 DBDC、DCUCD 与 DBSCAN 3 个算法的聚类时间比较。

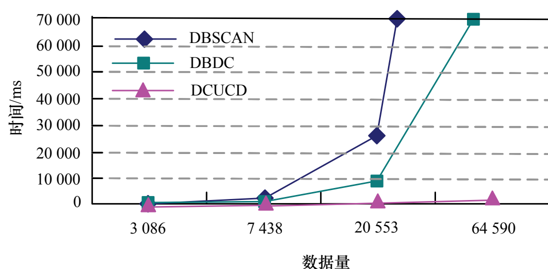

图 1 聚类时间比较

在数据量比较大的情况下，DBSCAN、DBDC 算法的时间效率迅速降低。DCUCD 算法仍然保持近似于线性的时间复杂度变化，是一种有效的方法。

图 2 是 DBDC、DCUCD 算法聚类质量比较。可以看出，CUCD 算法明显优于 DBDC 算法。

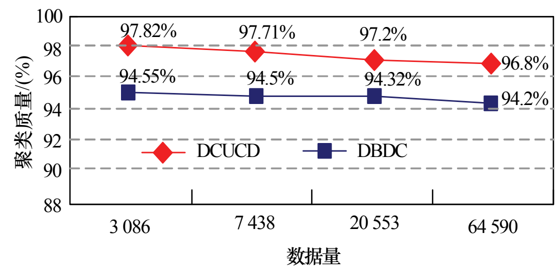

图 2 聚类质量比较

实验及分析还发现，DCUCD 的质量与子站点数据的分布有一定的关系：若各子站点间数据比较独立，聚类的质量非常接近于集中式 CUCD 的质量。如果各子站点数据分布比较均匀，聚类的质量有所降低但仍然相当稳定，而且处于一个可接受的范围内。

5 结束语

本文在深入研究目前典型的基于密度的分布式聚类算法的基础上，提出了基于中心点和密度的聚类算法 CUCD。将 CUCD 应用于分布式环境，提出 DCUCD 算法。DCUCD 首先在各子站点生成局部模型，主要步骤是在各子站点生成局部核心对象。然后，将子站点的局部模型传递至主站点。主站点根据收集到的局部核心对象信息，再次扫描生成全局核心对象。经过筛选以及分类，得到全局核心对象聚类的结果。最后，各子站点根据该结果对输入数据进行聚类标识。DCUCD 与 CUCD 算法类似，具有很高的时间效率。由于传递给主站点的是局部核心对象，因此数据传输消耗很低。实验对 DCUCD 的聚类时间以及聚类质量做了详细的分析。

DCUCD一个难点在于局部聚类和全局聚类参数的设定，这在一定程度上影响了最终聚类质量。可以考虑利用遗传算法对聚类结果进行优化，以保证聚类的质量。另外，对增量式的数据聚类可以考虑使用 FCM(Fuzzy C-Means)方法来实现，是下一步要考虑和研究的问题。

参考文献

[1] 黄学宇，魏　娜，陶建锋. 基于人工免疫聚类的异常检测算法[J]. 计算机工程, 2010, 36(1): 166-169.

[2] 纪洲鹏，周　军，何　明. 基于变精度粗糙集的 Web 用户聚类方法[J]. 计算机工程, 2010, 36(3): 44-46.

[3] Januzaj E, Kriegel H P, Pfeifle M. DBDC: Density Based Distributed Clustering[C]//Proc. of EDBT'04. [S. l.]: Springer, 2004: 88-105.

[4] Januzaj E, Kriegel H P, Pfeifle M. Scalable Density-based Distributed Clustering[C]//Proc. of PKDD'04. Pisa, Italy: Springer, 2004: 231-244.

[5] 郑金彬，卓义宝. 基于密度的分布式聚类算法研究[J]. 计算机工程, 2008, 34(17): 65-68.

[6] Zhou Jun, Liu Zhijing. Distributed Clustering Based on K-means and CPGA[C]//Proc. of FSKD'08. Jinan, China: [s. n.], 2008.

[7] Jiang Guoxing, Yang Zhiya. A Distributed Clustering Algorithm Based on Cluster Stability for Mobile Ad Hoc Networks[C]//Proc. of WiCOM'08. Dalian, China: [s. n.], 2008: 1-6.

编辑　金胡考

© 1994-2013 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net