计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(2):305−310 [doi: 10.15888/j.cnki.csa.008310]

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

基于 KD 树改进的 DBSCAN 聚类算法①

陈文龙, 时宏伟

(四川大学 计算机学院, 成都 610065)
通信作者: 陈文龙, E-mail: 1225965831@qq.com

摘　要: 针对 DBSCAN 聚类算法随着数据量增大, 耗时越发非常严重的问题, 提出一种基于 KD 树改进的 DBSCAN 算法 (以下简称 KD-DBSCAN). 通过 KD 树对数据集进行划分, 构造邻域对象集, 提前区分出噪声点和核心点, 避免聚类过程中对噪声的邻域集计算以及加快了核心点对象的邻域集查询速度. 文中以浮动车 GPS 数据为实验数据, 对比传统 DBSCAN 算法和 KD-DBSCAN 算法的聚类效果和时间性能, 实验结果表明 KD-DBSCAN 算法的聚类效果和传统的 DBSCAN 基本一致, 但时间性能有很大的提升.

关键词: 聚类; DBSCAN 算法; KD 树

引用格式: 陈文龙,时宏伟.基于 KD 树改进的 DBSCAN 聚类算法.计算机系统应用,2022,31(2):305–310. http://www.c-s-a.org.cn/1003-3254/8310.html

Improved DBSCAN Clustering Algorithm Based on KD Tree

CHEN Wen-Long, SHI Hong-Wei

(College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract: To tackle the problem that density-based spatial clustering of applications with noise (DBSCAN) clustering algorithm is increasingly time-consuming with the increase in data volume, this study proposes an improved DBSCAN algorithm based on a K-dimensional (KD) tree (hereinafter referred to as KD-DBSCAN). The KD tree is used to divide the data set, construct the neighborhood object set, and distinguish the noise point and the core point in advance to avoid the calculation of the noise neighborhood set in the clustering process and speed up the neighborhood set query of the core point object. In this study, the global positioning system (GPS) data of a floating car is used as experimental data to compare the traditional DBSCAN algorithm and KD-DBSCAN algorithm in aspects of the clustering effect and time performance. The experimental results show that the KD-DBSCAN algorithm is comparable to the traditional DBSCAN algorithm in the clustering effect but has greatly improved time performance.

Key words: clustering; density-based spatial clustering of applications with noise (DBSCAN) algorithm; KD tree

聚类 (clustering) 是数据挖掘中的重要手段, 其核心思想是“同类相近, 异类相远”, 将数据集划分为不同的簇, 同一簇中的数据对象之间有较高的相似度, 不同簇之间的数据对象差异较大. 通过聚类的方法可以分析数据之间的联系和价值, 目前已被广泛应用于诸如数据分析, 生物学, 模式识别[1] 等多领域的研究工作中.

目前聚类算法主要分为基于层次的方法如 BIRCH 算法, 基于网格的聚类方法如 STING 算法, 基于划分的方法如 K-means 算法, 基于密度的方法如 DBSCAN 算法以及基于模型的聚类方法等[2−6]. 其中 DBSCAN 算法是密度聚类中最经典的一种聚类算法, DBSCAN 算法不需要预先指定簇的个数, 且能在含有“噪声”的数据中找出任意形状的簇, 具有很好的抗干扰性, 但当数据量增大时, 由于算法自身的特性会导致时间的消

① 收稿时间: 2021-04-19; 修改时间: 2021-05-19; 采用时间: 2021-05-28; csa 在线出版时间: 2022-01-17

耗急剧增加.

为了解决上述问题, 许多学者提出了不同的改进方案. 文献 [7] 提出的 IF-DBSCAN 通过选取核心点邻域中的代表对象来扩展类, 从而减少邻域中的查询次数, 但效率提升有限. 文献 [8] 通过按顺序选择邻域外未被标记的点作为种子点, 分不同情况进行聚类扩展, 有效地减少了核心点邻域重叠区域的查询次数, 同时也提升聚类的质量. 文献 [9] 提出选取距离最远且半径内样本个数大于 *MinPts* 的点为核心点, 以避免因核心点随机选取而导致计算量过大, 但优化效率还需提高. 文献 [10] 通过避免公共邻域内对象的重复查询来优化算法的执行速度, 但聚类质量还需提高. 文献 [11] 采用贪心策略进行聚类, 在寻找核心对象的过程中使用邻域查询的方法来提升算法的运行效率, 但需手动输入密度阈值. 文献 [12] 采用重心点转移的方法来改进 St-DBSCAN 算法, 以提升聚类的时间性能和质量, 但效率还有待提高. 本文提出的 KD-DBSCAN 算法能减少传统 DBSCAN 算法在聚类时需要计算数据样本中每个点 *Eps* 邻域所带来的开销, 从而极大的降低聚类时间, 且保证了聚类的效果与传统的 DBSCAN 算法基本一致.

1 DBSCAN 算法分析

DBSCAN 算法是一种基于密度的空间聚类算法, 该算法可以在具有“噪声”的数据中发现具有一定密度条件下的任意形状的簇. DBSCAN 中的基本概念如下.

定义 1. *Eps* 邻域: 给定某对象点 X, 其半径 *Eps* 范围称为该点的 *Eps* 邻域, 对于 X 的 *Eps* 邻域内的任意一点 Y 都有: $\text{dist}(X, Y) \leqslant Eps$.

定义 2. 核心点: 给定某对象点 X, 其 *Eps* 邻域内数据点数量超过 *MinPts* 的点.

定义 3. *MinPts*: 簇的最小点集, 即核心点邻域内的点数量必须不少于 *MinPts*.

定义 4. 噪声点: 既不是核心点也不是边界点的其它点.

定义 5. 边界点: 不是核心点, 但在核心点的邻域内.

定义 6. 直接密度可达: 对象点 Y 在 X 的 *Eps* 邻域内, 且对象点 X 是核心点, 则称 X 到 Y 是直接密度可达的.

定义 7. 密度可达: 某一对象链 $X_1, X_2, \cdots, X_n$, 若满足任意 X_i 到 X_{i+1} 是直接密度可达的, 则称 X_0 到 X_n 密度可达.

定义 8. 密度相连: 对某一对象 Z, 若 Z 到 X 和 Y 都是密度可达的, 则称 X 和 Y 是密度相连的.

DBSCAN 算法聚类过程思想为: 从给定的数据集 D 中随机选取某个对象点 X, 查询 X 的 *Eps* 邻域半径内的所有密度可达点是否大于 Min*Pts*, 如果大于则创建一个以 X 为核心点的聚类簇. 之后迭代聚集核心点密度可达的所有对象, 该过程可能会将核心点密度可达的簇进行合并, 在算法执行的过程中, 某些核心点的密度可达对象会被重复查询. 当一个簇聚类完成后会开始选取下一个点开始聚类, 直到没有任何新的点添加到任何一个簇中时, 聚类结束, 算法流程图 (如图 1 所示) 及部分伪代码如算法 1.

算法 1. DBSCAN 算法

输入: D: 包含 n 个样本点的数据集; *Eps*: 半径大小; *MinPts* : 最少样本点个数;
输出: 簇的集合.

```
1) Initialize all points in D are unvisited
2) Select a point p in unvisited
3) Mark p as visited
4) Set N as Eps-neighbor of p
5) if size of N ⩾ MinPts
6)       Create a new cluster C, and put p in C
7)       for each point p∈N
8)          If p in unvisited
9)             Mark p as visited
10)            Set N as Eps-neighbor of p
11)            if size of N ⩾ MinPts
12)               Put those points in N
13)            endif
14)         endif
15)         if p is not yet a number of any cluster
16)            Put p to C
17)      endfor
18) else mark p as noise
19) Until all points are marked as visited
```

2 KD-DBSCAN 算法

传统的 DBSCAN 聚类算法在生成邻域集时会扫描整个数据样本, 导致大量且不必要的计算开销. 因此, 在聚类之前使用有效的数据结构对数据进行预处理, 可以快速地找出给定对象的邻域集, 减少近邻点的搜索时间. 文中将利用 KD 树这种数据结构对数据进行划分, 由于 KD 树是基于数据属性进行构建的, 因此相似度高的数据对象在 KD 树中联系更加紧密, 在查找的过程中也更加快速, 避免了在全局范围内查找邻域对象的时间开销.

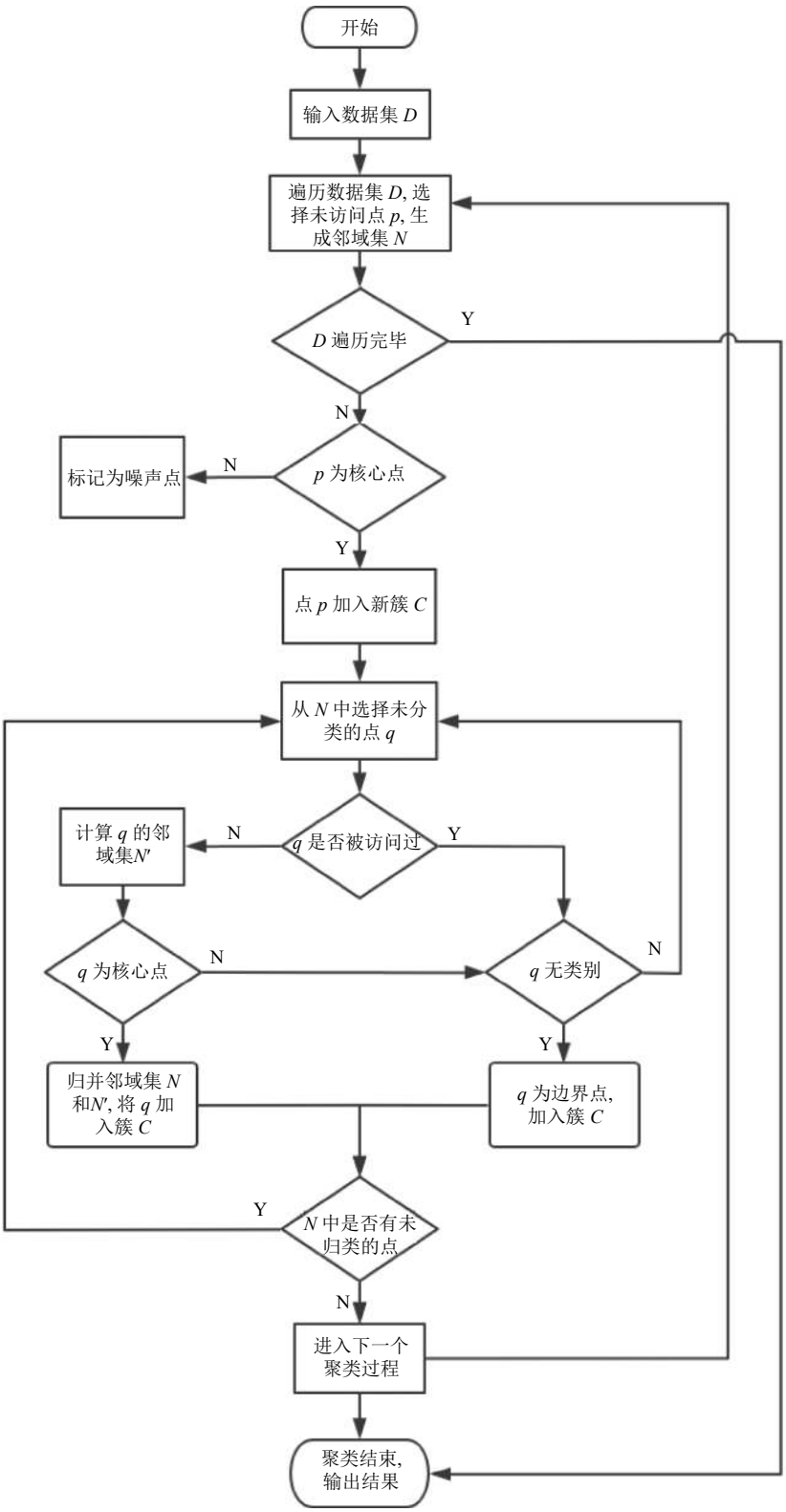

图 1 DBSCAN 算法流程图

2.1 相关概念

KD 树是 K-dimensional tree 的缩写, 是一种在存储和查询方面非常高效的数据结构. 最早由 Bentley[13] 提出, 用于将数据点在 K 维空间中进行构造, 是一种特

殊的二叉搜索树结构, 树中的每一层对应一个维度. 传统二叉搜索树只对一维的数据进行构建, KD 树通过数据点的属性个数循环的构造 K 维二叉搜索树. KD 树中左子树在给定维度上的值小于父节点, 而右子树则大于父节点, 以二维数据为例, 假设有一数据集为{(2, 11), (9, 12), (2, 13), (12, 12), (11, 14), (10, 10), (15, 13)}对其进行 KD 树的构造, 在第一层中首先按第一个维度上的值进行排序, 将中间数据值的点作为根节点, 如 (10, 10), 此时将数据集划分为两个区域, 左边的子数据集为{(2, 11), (2, 13), (9, 12)}在第一个维度上的数值均小于根节点, 右边为{(11, 14), (12, 12), (15, 13)}在第一个维度上的数值都大于根节点, 再分别在左右子树中选择第 2 个维度上的数据进行排序并选中间值, 左子树中以点 (9, 12), 右子树为点 (15, 13) 分别构造左右子树, 到第 3 层又回到第一维进行比较构造, 图 2 为构造的效果图.

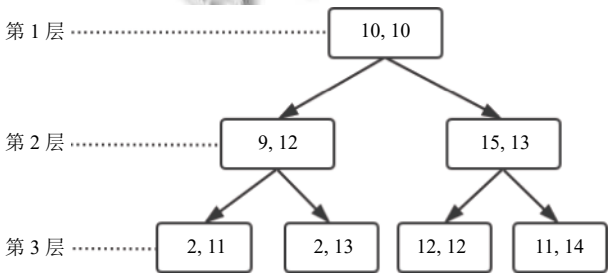

图 2 KD 树样例

2.2 算法相关步骤

利用 KD 树对数据集进行构造, 在聚类之前生成每个数据点的邻域对象集, 遍历每个数据点, 并通过构造好的 KD 树进行近邻点搜索, 找出所有数据对象的邻域集, 包含核心点, 边界点以及噪声点, 将找出的核心点用于后续的聚类步骤. 虽然构造 KD 树需要花费一些时间且需要全局查询, 但是当数据量增大时, 可以明显减少近邻节点查找次数, 因此是非常值得的.

(1) KD 树的构造

将数据点在 K 维空间中进行分割并构造 KD 树的节点, 先在一个维度上对数据点的某一属性值进行排序, 选取中间值点作为根节点, 再构造其左右子树, 其左子树节点在此维度上的数值均小于根节点在此维度上的值, 右子树则相反; 在子树的构造过程中, 在第 2 个维度上对数据点的第 2 个属性值进行排序, 选取中间值的数据点为子树的根节点, 反复执行, 直至构造完毕. 利用 KD 树对 DBSCAN 算法改进可将算法的时间复杂度由$O(n^2)$降至$O(n\log_2 n)$.

(2) 查找邻近节点集合

利用 KD 树进行近邻节点的查询, 此步骤将遍历数据集中的每一个节点, 让每一个节点根据输入的 Eps 参数计算出其正方形边界范围, 再从 KD 树的根节点开始查找其近邻点, 比较树中的每一层对应维度上的值是否在其边界范围内, 如果不满足条件则去子树中查找, 如果在其边界范围内还要继续计算其两点之间的距离是否满足 $\text{dist}(X, Y)\leqslant Eps$, 因为构造的正方形边界范围会略大于 Eps 为半径的圆形范围, 两者对应关系如图 3 所示. 将满足上述条件的数据对象添加到该点的 neighbors 邻域集合中, 每个数据点都有一个属性 Key 用于唯一标识该点, 构造后的$neighbors(Key_n)$集合形式如$\{Key_1, Key_2, \cdots, Key_n\}$, 其含义是$Key_1$点在 Eps 邻域内包含的数据点有$Key_2, \cdots, Key_n$. 算法中采用的是 haversine 距离[14], 具体公式如下:

$$d = \sin^2(\theta) + \cos(lat1)\times\cos(lat2)\times\sin^2(\omega) \tag{1}$$

$$\theta = \frac{lat2 - lat1}{2} \tag{2}$$

$$\omega = \frac{lon2 - lon1}{2} \tag{3}$$

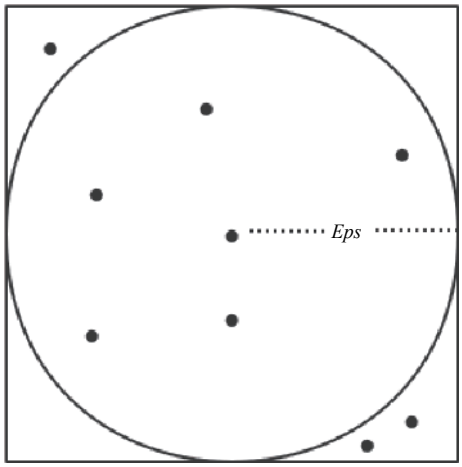

图 3 正方形边界与 Eps 半径圆范围

两点的距离为:

$$\text{dist}(P_1, P_2) = 2\times R\times\arcsin(\text{sqrt}(d)) \tag{4}$$

其中, R 为地球半径, $lat1$, $lon1$ 和 $lat2$, $lon2$ 分别对应 P_1, P_2 两点的经纬度, $\text{dist}(P_1, P_2)$为 P_1, P_2 两点之间的距离.

(3) KD-DBSCAN 聚类

将上一步中查找到的邻近节点集合中长度大于等

于 *MinPts* 的数据点用于聚类, 即 len(neighbors(Key_n)) ≥ *MinPts* 的数据点, 通过过滤噪声点来进一步减少聚类时间.

算法的部分伪代码如算法 2 和算法 3.

算法 2. 构建 KD 树

```
CreateKDTree(points, tree_depth):
1) //根据树的深度来选择维度
2) axis = tree_depth mod K
3) //选择第 axis 维度的中间值为根节点进行 KD 树的构造
4) select median by axis from points
5) //构造节点和子树
6) node = median
7) node.left = KDTree(points in subList(0,median),axis, tree_depth + 1)
8) node.right = KDTree(points in subList(median + 1, points.size()),axis,
tree_depth + 1)
9) Return node
```

算法 3. 找出邻近节点集合 find_neighbors(kdtree,node,eps)

```
1) //遍历节点 node, 根据 Eps 大小计算边界范围
2) bounding = getbounding(node, Eps)
3) Set bounding_max as the max value of bounding
4) Set bounding_min as the min value of bounding
5) q = deque(kdtree)
6) while size of q > 0
7)       current_node = q.popleft()
8)       if is_inbounding(current_node, bounding)
9)          if haversine(node,current_node) <= eps and current_node.
key!= node.key
10)             Put current_node into neighbors
11)          endif
12)       endif
13)       Set axis as current_node's axis
14)       if bounding_max[axis] > current_node[axis] and bounding_
min[axis] > current_node[axis]
15)       and current_node has rightTree:
16)          q.add(current_node.rightTree)
17)       elif bounding_max[axis] < current_node[axis] and bounding_min
[axis]<current_node[axis]
18)       and current_node has leftTree:
19)          q.add(current_node.leftTree)
20)       elif bounding_max[axis] >= current_node[axis] >= bounding_
min[axis]:
21)             if current_node has rightTree
22)                q.add(current_node.rightTree)
23)             endif
24)             if current_node has leftTree
25)             q.add(current_node.leftTree)
26)             endif
27)       endif
28) Return neighbors
```

3 实验结果与分析

3.1 实验环境和数据集

本实验以成都市网约车 GPS 数据为实验数据, 在 Intel Core i7-9750H CPU 2.60 GHz, 16 GB 内存的笔记本电脑上运行传统 DBSCAN 算法与改进的 KD-DBSCAN 算法, 统计两种算法的运行时间以及聚类的数目并用聚类评价指标对两种算法进行评估.

3.2 实验结果与分析

设置参数 *Eps* = 100 m, *MinPts* = 60, 数据集从 9 300 行依次增加到 58 000 行, 运行结果如表 1 所示.

表 1 不同数据行数的测试结果

算法	指标	9 300	18 500	38 000	58 000
DBSCAN	时间 (s)	158	583	2 612	7 298
	簇个数	12	33	78	109
KD-DBSCAN	时间 (s)	2	6.5	29.5	94
	簇个数	10	33	78	107

图 4 和图 5 分别为耗时以及聚类簇个数的对比情况.

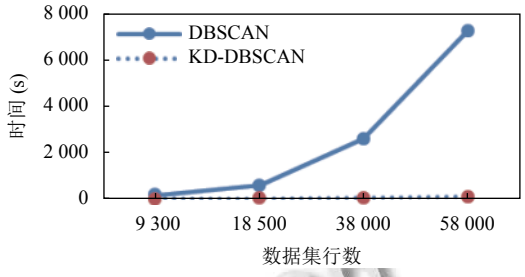

图 4 算法耗时对比图

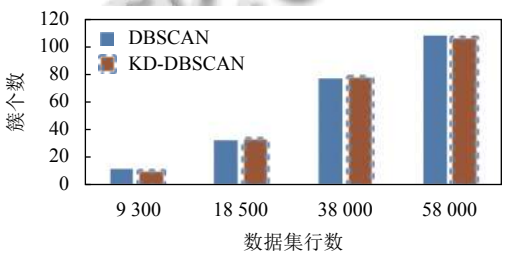

图 5 聚类簇个数

为比较两种算法的聚类质量, 文中引入聚类效果评价指标 Calinski-Harabasz (CH)[15,16], 其定义为:

$$CH(k)=\frac{BGSS}{k-1}\Big/\frac{WGSS}{n-k} \tag{5}$$

其中, n 为数据集样本数, k 为类别数.

$$WGSS=\frac{1}{2}\left[(n_1-1)\overline{d}_1^{\,2}+\cdots+(n_k-1)\overline{d}_k^{\,2}\right] \tag{6}$$

$$BGSS=\frac{1}{2}\left[(k-1)\overline{d}^{\,2}+\cdots+(n-k)A_k\right] \tag{7}$$

其中, $\overline{d_i}^2$表示第 i 类中样本间的平均距离, i=1, 2…, k; $\overline{d}^2$是所有样本间的平均距离, 其中A_k为:

$$A_k = \frac{1}{n-k}\sum_{i=1}^{k}(n_i-1)\left(\overline{d}^2-\overline{d_i}^2\right) \tag{8}$$

CH 指标可以用于评估 DBSCAN 这类事先无法得知聚类个数的算法, 即在真实的分类标签 label 不知道的情况下用于评估聚类结果的性能, 其指标值越大说明簇内数据对象之间的间距越小而簇间距离越大, 聚类结果的 CH 指标如图 6 所示.

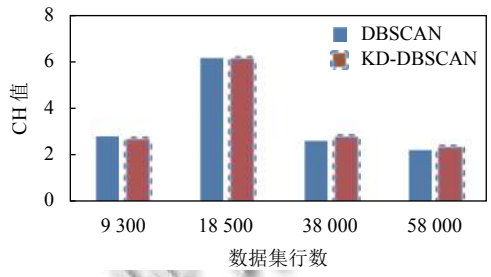

图 6 聚类结果的 CH 指标

实验结果分析:

(1) 聚类时间

从图 4 中可以看出, 随着数据量的增加, 传统 DBSCAN 算法耗时是急剧增加的, 而改进的 KD-DBSCAN 算法耗时依旧平稳, 且二者耗时的差距越来越大. KD-DBSCAN 算法体现出了 KD 树对于减少邻近节点查询时间的优势.

(2) 聚类数量

两种算法在相同数据量的情况下, 聚类产生的簇个数基本一致, 在数据量较少和较大时, KD-DBSCAN 算法产生的簇个数略微少于传统 DBSCAN 算法, 这可能是由于 KD-DBSCAN 算法对于数据的划分更严格导致的.

(3) 聚类质量

从图 6 中可以看出两种算法在相应的数据量下的聚类质量基本相同, 且随着数据量的增大, KD-DBSCAN 算法的 CH 值总是略大于传统 DBSCAN 算法的.

综上所述, 改进后的 KD-DBSCAN 算法在保证了聚类的数量和质量与传统 DBSCAN 算法保持基本一致的前提下, 对聚类的效率有了大幅度的提升, 因此利用 KD 树改进的 KD-DBSCAN 算法是切实可行的.

4 结束语

本文通过 KD 树对传统 DBSCAN 算法进行改进, 在保证了两种算法的聚类个数以及质量基本一致的前提下, 极大地提高了算法的运行效率. 但是, 当数据量进一步增大时两种算法都会出现内存不足的情况, 因此下一步的研究工作会聚集于如何利用分布式计算框架来解决这类问题, 以提升算法的性能.

参考文献

1 孙吉贵, 刘杰, 赵连宇. 聚类算法研究. 软件学报, 2008, 19(1): 48–61. [doi: 10.3724/SP.J.1001.2008.00048]

2 Cohen-addad V, Kanade V, Mallmann-trenn F, *et al*. Hierarchical clustering: Objective functions and algorithms. Journal of the ACM, 2019, 66(4): 26.

3 Wang YW, Zhou YC, Liu Y, *et al*. A grid-based clustering algorithm for wild bird distribution. Frontiers of Computer Science, 2013, 7(4): 475–485. [doi: 10.1007/s11704-013-2223-2]

4 Hencil PJ, Antonysamy A. An optimised density based clustering algorithm. International Journal of Computer Applications, 2010, 6(9): 20–25. [doi: 10.5120/1102-1445]

5 Rodriguez A, Laio A. Clustering by fast search and find of density peaks. Science, 2014, 344(6191): 1492–1496. [doi: 10.1126/science.1242072]

6 Hoijtink H, Notenboom A. Model based clustering of large data sets: Tracing the development of spelling ability. Psychometrika, 2004, 69(3): 481–498. [doi: 10.1007/BF02295648]

7 王桂芝, 王广亮. 改进的快速 DBSCAN 算法. 计算机应用, 2009, 29(9): 2505–2508.

8 许虎寅, 王治和. 一种改进的基于密度的聚类算法. 微电子学与计算机, 2012, 29(2): 44–47, 53.

9 安计勇, 韩海英, 侯效礼. 一种改进的 DBscan 聚类算法. 微电子学与计算机, 2015, 32(7): 68–71.

10 彭波, 史春雷, 高万林. DBSCAN 算法优化及在村镇管理决策中的应用. 农业机械学报, 2016, 47(10): 346–350. [doi: 10.6041/j.issn.1000-1298.2016.10.044]

11 冯振华, 钱雪忠, 赵娜娜. Greedy DBSCAN: 一种针对多密度聚类的 DBSCAN 改进算法. 计算机应用研究, 2016, 33(9): 2693–2696, 2700. [doi: 10.3969/j.issn.1001-3695.2016.09.029]

12 刘勇, 何婧, 姚绍文, 等. 基于重心点转移的 St-DBSCAN 改进算法. 计算机技术与发展, 2018, 28(11): 6–11. [doi: 10.3969/j.issn.1673-629X.2018.11.002]

13 Bentley JL. Multidimensional binary search trees used for associative searching. Communications of the ACM, 1975, 18(9): 509–517. [doi: 10.1145/361002.361007]

14 Soe NC, Thein TLL. Haversine formula and RPA algorithm for navigation system. International Journal of Data Science and Analysis, 2020, 6(1): 32–40. [doi: 10.11648/j.ijdsa.20200601.14]

15 冯柳伟, 常冬霞, 邓勇, 等. 最近最远得分的聚类性能评价指标. 智能系统学报, 2017, 12(1): 67–74.

16 Caliński T, Harabasz J. A dendrite method for cluster analysis. Communications in Statistics, 1974, 3(1): 1–27.