基于z值的分布式密度峰值聚类算法

卢 晶[1],段 勇[1],刘海波[2]
(1. 沈阳工业大学信息科学与工程学院,辽宁沈阳 110870;
2. 河北大学计算机科学与技术学院,河北保定 071002)

摘 要: 密度峰值聚类算法由于在发现任意形状簇且不需指定聚类个数等方面具有一定的优势而被广泛关注.但是该算法需要计算数据集中所有点的密度和点对之间的距离,因此不适合处理大规模高维数据集.为此,本文提出了一种基于z值的分布式密度峰值聚类算法,DP-z.本方法利用空间z填充曲线将高维数据集映射到一维空间上,根据数据点的z值信息对数据集分组.为了能够得到正确的结果,需要对分组间数据进行交互,然后并行计算每个点密度和斥群值.DP-z算法在分组间数据交互时采用过滤策略,减少大量无效距离计算和数据传输开销,有效提高算法的执行效率.最后,本文在云计算平台上对DP-z算法进行了验证,实验表明在保证DP-z算法与原始密度峰值聚类算法聚类结果相同的情况下有效的提高了算法执行效率.

关键词: 聚类;分布式计算;云计算;z填充曲线;密度峰值
中图分类号: TP301.6 **文献标识码:** A **文章编号:** 0372-2112 (2018)03-0730-09
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2018.03.031

Distributed Density Peaks Clustering Based on z-Value

LU Jing[1], DUAN Yong[1], LIU Hai-bo[2]
(1. *School of Information Science and Engineering, Shenyang University of Technology, Shenyang, Liaoning* 110870, *China*;
2. *School of Computer Science and Technology, Hebei University, Baoding, Hebei* 071002, *China*)

Abstract: Density peak clustering is an effective and novel clustering algorithm, it is concerned as its superiority of finding arbitrary shape of clusters and number of clusters. However, this algorithm is required to measure the density and distance between any pair of objects. This limits the practicability of this algorithm when clustering high-volume and high-dimensional data set. In order to improve efficiency and scalability, we propose a distributed density peak clustering algorithm based on z-value, and DP-z. It utilizes z-values to map points in multidimensional space into one dimension, and then splits the data set into several partitions according to the z-values of points. In order to get the correct result, we make use of the character of points' z-values to filter the data object while exchanging data among groups, which reduces a huge amount of useless distance measurement cost and data shuffle cost. Then we compute the density and distance value in parallel. Finally, we test the DP-z algorithm based on the cloud computing platform, the experiments show that DP-z can achieve higher performance at speed without reducing the accuracy.

Key words: clustering; distributed computing; cloud computing; z-order curve; density peaks

1 引言

聚类分析是数据挖掘和模式识别等领域广为研究的问题之一.它将数据库中的数据分割成不同的族(类),并使族内数据之间的相似性比族间数据之间的相似性大.大量聚类分析算法[1~3]在社会网络分析、统计数据分析、智能商务、图像模式识别、Web搜索等领域得到广泛应用.

发表于《Science》学术期刊的密度峰值聚类算法(Density Peaks Clustering, DPC)[4]一个新型的基于密度的聚类算法.由于该算法能够发现任意形状的聚类,也不依赖于数据集的维度;算法的实现过程只需要计算出每个点密度值ρ(由某一范围内点的个数来刻画)和

收稿日期:2016-10-31;修回日期:2017-05-31;责任编辑:梅志强
基金项目:辽宁省自然科学基金(No. 2015020010);辽宁省高等学校优秀科技人才支持计划(No. LR2015045);河北省自然科学基金青年基金(No. F2015201140)

斥群值 δ(与密度值比自己大的点的距离的最小值刻画),不需要多次迭代. 重要的是该算法不必事先指定聚类个数,而是用户根据每个点的两个属性值而决定. 密度峰值聚类算法拥有以上优点而被广泛关注.

由于该算法需要计算数据集中所有点对之间距离,导致算法执行效率低、降低适用性,尤其高维海量数据. 随着云计算技术的发展,很多算法可以在多台机器的分布式环境下执行,文献[5]基于 MapReduce 计算模型提出了一种高效的分布式密度中心聚类算法:EDDPC. 由于其粗略的数据复制方案导致其仍然存在大量的冗余数据复制和计算开销.

本文在深入分析密度峰值聚类算法的基础上,提出一种基于 z 值[6]的分布式密度峰值聚类算法(Distributed Density Peaks Clustering based on z-value,DP-z),并在 Hadoop 集群上进行了实验测试,验证 DP-z 算法在处理大规模高维数据上的高效性.

DP-z 算法利用空间 z 填充曲线将高维数据映射成一维空间 Z 轴上的点,根据数据点 z 值的分布将数据集分组. 将各组数据分给不同的机器进行运算,在组内计算每个点的密度值 ρ 和斥群值 δ. 由于采用 z 填充曲线对高维数据点进行了降维处理,可能会导致原始数据的近邻性受损,导致相邻的数据被分到不同的组中. 在计算分组中每个点密度值 ρ 和斥群值 δ 时可能会用到其它组中的数据. 对此,本文基于数据点 z 值携带高维空间位置信息的性质提出一种基于 z 值上下限的数据复制模型和数据分发模型,将其他组内的少量数据复制到本组中. 各机器根据分组数据和复制数据进行并行计算各点的密度值 ρ 和斥群值 δ,并从理论上保证其正确性.

本文的主要贡献如下:

① 对原始 DPC 算法进行了分布式扩展,基于 MapReduce 模型提出基于 z 值的分布式密度峰值聚类算法,DP-z,并在开源云计算平台 Hadoop 上实现.

② 为正确计算 ρ 值,设计基于 d_c 范围内 z 值最小下限和最大上限的复制策略;为正确计算分组内密度最大值点外其他点的全局斥群值 δ,设计基于 δ' 范围内 z 值最小下限和最大上限的复制策略;为正确计算组内密度值最大点的全局斥群值 δ,设计基于各组最大密度值点的 δ"范围内 z 值下限和上限分发策略.

③ 在多个数据集上对分布式密度峰值聚类算法进行了多方面 实验评估.

2　密度峰值聚类算法及其分布式

由 Alex Rodriguez 等人于 2014 年在《Science》上发表. 文献[5]在此基础上针对算法复杂度高等问题,提出了分布式密度中心聚类算法:EDDPC.

本节对 DPC 和 EDDPC 做简要介绍,并对其中存在的复杂度高、计算量大等问题进行简要分析.

2.1　密度峰值聚类算法简介

考虑待聚类的数据集 $S=\{x_i\}_{i=1}^{N}$,对于 S 中的任何数据点 x_i,密度峰值聚类算法要计算其局部密度 ρ_i 和斥群值 δ_i 两个属性.

局部密度 ρ_i:

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \tag{1}$$

其中 $\chi(x)=\begin{cases}1, x<0\\0, x>0\end{cases}$

参数 $d_c>0$ 为截断距离,d_c 可以根据如下参数估计方法[4]来确定,将所有点对的距离从小到大排序的 1%~2% 处. 例如 100 个点,共有 4950 种组合,将这些组合距离排序,在 50~100 处选取某一距离为 d_c 值. 由式(1)可知,ρ_i 表示的是 S 中与 x_i 之间的距离小于 d_c 的点的个数(不考虑 x_i 本身).

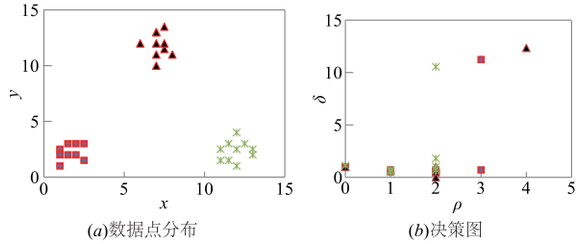
点 i 的斥群值 δ_i 定义为:

$$\delta_i = \min_{j:\rho_j>\rho_i}(d_{ij}) \tag{2}$$

斥群值 δ 代表与密度值比自己大的点的距离的最小值. 假设密度比 ρ_i 大的点中,点 j 与点 i 的距离最近,那么 $\delta_i=d_{ij}$(d_{ij}代表点 i 到点 j 的距离),而点 j 就是点 i 的斥群值依附点 σ_i[5],$\sigma_i=\arg\min_{j\in S,\rho_j>\rho_i}(d_{ij})$. 说明点 i 可以归属于点 j 所属聚类. 斥群值 δ_i 越小,这种依附可能性越大,说明点 i 越有可能归属于点 j 所属的聚类;斥群值 δ_i 越大,点 i 距离点 j 越远,依附关系就越弱,点 i 越有可能与点 j 不属于同一个聚类,或者是离群点. 当某个点 m 的密度值 ρ_m 是所有点中密度最大值,那么点 m 的斥群值 δ_m 则为

$$\delta_m = \max_j(d_{mj}) \tag{3}$$

根据密度值 ρ 和斥群值 δ 可绘制二维决策图,横轴为密度值,纵轴为斥群值. 如图 1 所示,图 1(b)为图 1(a)数据集的决策图. 根据图中决策点的分布情况,可以看出密度值 ρ 和斥群值 δ 都非常大的点分布在决策图中的右上角,将这些点圈出作为聚类中心点,然后根据数据集中每个点的 δ 值的依附关系,反推出每个点所属聚类.



(a)数据点分布　　(b)决策图

图1　密度峰值聚类算法

从 DPC 算法的实现过程中可以看出,在计算每个点的 ρ 和 δ 值时需要计算数据集中所有点对之间的距离,复杂度为 $O(N^2)$.

2.2 MapReduce 简介

MapReduce 是当前流行的分布式编程模型,它提供了两个接口函数 map 和 reduce. 函数 map 功能由用户自己定义,并且以〈key,value〉对的形式输出:map($k1$,$v1$)→list($k2$,$v2$),然后系统将 map 输出的具有相同 key 值的 value 收集起来,以〈key,list(value)〉的形式把具有相同 key 值的键值对发送给同一个 reduce 函数进行处理,该过程为数据 shuffle,最终 reduce 函数以〈key,value〉的形式输出:reduce($k2$,list($v2$)) → list($k3$,$v3$).

本文将基于 MapReduce 模型设计基于一种 z 值的分布式密度峰值聚类算法,并在 MapReduce 模型的开源框架 Hadoop 上实现了该算法.

2.3 分布式密度峰值聚类算法冗余计算问题分析

为了提高 DPC 算法的执行效率,文献[5]提出一种对基本 DPC 算法的改进算法,EDDPC. 但是由于这种方法在数据复制方面所存在的问题导致其大量的冗余复制和计算开销.

EDDPC 首先用 Voronoi 图分割技术将数据集分割成互不相交的组,然后发送到不同的机器中执行. 该分组方法的不足之处在于种子对象随机性非常大,可能会造成计算机集群中的负载不均衡.

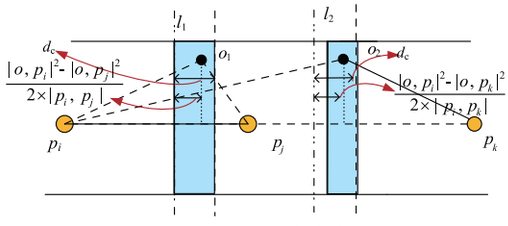


图2 EDDPC的d_c复制方案示例

其次,在计算每个点的密度值 ρ 时,EDDPC 算法根据如下公式对数据进行组间复制.

$$\frac{|o,p_i|^2-|o,p_j|^2}{2\times|p_i,p_j|}<d_c \tag{4}$$

其中 p_i, p_j 是分组 P_i,P_j 的种子点,这样可以将相邻组在分界线 d_c 范围内的点相互复制,从而在组内计算时能够得到正确密度值 ρ. 但是该复制方法可能会导致如图 2 的问题,o_2 所在区域为多余复制. 因为冗余的复制,导致计算点的密度值 ρ 时造成冗余计算开销.

在计算斥群值 δ 时,为了精确计算每个点的 δ 值,需要根据如下公式再次对数据进行复制

$$|o,p_i|\leqslant B(P_i)+\operatorname{smax}\delta(P_i),\rho_o>\min\rho(P_i) \tag{5}$$

$$|o,p_i|\leqslant\min\{2|m,p_i|+|p_j,u|+|p_i,p_j|\},\ \rho_o>\max\rho(P_i) \tag{6}$$

其中 $B(P_i)=\max\{|o,p_i|,o\in P_i\}$, $\operatorname{smax}\delta(P_i)$ 为组内次大的 δ',m 为分组 P_i 中密度最大的点,$u\in P_j$. 但是该分组方法可能会出现如图 3 所示,阴影部分全部被复制到 P_i 中(可能包括其他分组中的所有数据或者分组中的大部分数据,实际上只需要虚线所绘制的圆内的数据即可),因此存在大量的冗余复制. 又因为在 δ 值求精的过程中需要组内数据与复制数据计算距离,导致大量不必要的计算开销.

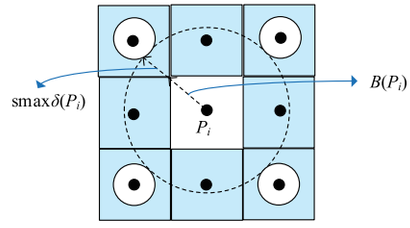


图3 EDDPC的δ'复制方法示例

3 基于 z 值的分布式密度峰值聚类算 法

综上所述,目前密度峰值聚类及其改进算法中存在大量的冗余复制和计算效率不高等问题. 基于此,本文根据空间 z 填充曲线和高维数据点 z 值携带点位置信息的特点提出一种基于 z 值的分布式密度峰值聚类算法,并使用基于云计算开发框架 Hadoop 实现基于 z 值的分布式密度峰值聚类算法.

3.1 空间 z 填充曲线与高维点 z 值

空间 z 填充曲线是一种把 d 维空间映射成一维空间的方法,每个点 p 的编号对应一个 z 值 z_p,如图 4 所示为一条二维空间中的 z 填充曲线贯穿一个 3×3 的二维空间区域.

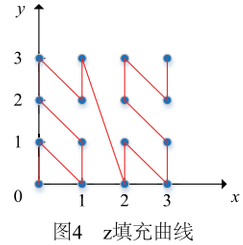


图4 z填充曲线

d 维坐标点 $P(m_1, m_2, m_3,\cdots m_d)$ 的 z 值计算方法如下:将十进制表示成二进制 $P'(b_1, b_2, b_3,\cdots b_d)$,其中 b_i 是 m_i 的二进制表示. 将 P'从最高有效位至最低有效位,按维度从小到大的顺序交叉排列转化为一个新的二进制数 $p_{(2)}$. 将此二进制数 $p_{(2)}$ 用十进制数表示为 $p_{(10)}$,$p_{(10)}$ 即是点 P 的 z 值. 如图 5 所示,一个二维空间中的一个点 $P(2,6)$,表示成二进制数 $p(010,110)$,因此点 P 的 z 值 $z_p=(011100)_2=(28)_{10}$.

3.2 基于 z 值的分布式密度峰值聚类算法

密度峰值聚类算法的主要过程是计算数据集中每

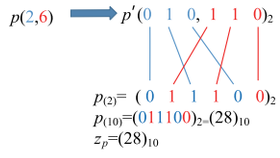


图5　点$p(2,6)$的z值

个点的两个属性值:密度值 ρ 和斥群值 δ. 而计算密度值 ρ 时,每个点只需要与附近的点计算距离即可. 同理,计算斥群值 δ 时,每个点只需要与附近比它密度大的点计算距离即可. 恰好数据点的 z 值携带有位置信息,而且通过该位置信息可以粗略估计其距离关系,为此本节提出了基于 z 值的分布式密度峰值聚类算法:DP-z. DP-z 算法步骤如算法 1:

算法 1　基于 z 值的分布式密度峰值聚类算法

1)对数据集 D 采样得到样本 ξ,计算样本中点的 z 值和点对间的距离,并对距离值和 z 值排序,估计 d_c 值和分组分位点集 $N=\{n_1,n_2\cdots,n_{n-1}\}$.

2)根据分位点 n_i 的 z 值对数据分组,同时统计每个分组中所有点 d_c 范围内 z 值最小下限 $\mathrm{lz}^{d_c}(P_i)$ 和最大上限 $\mathrm{uz}^{d_c}(P_i)$.

3)根据该 $\mathrm{lz}^{d_c}(P_i)$ 和 $\mathrm{uz}^{d_c}(P_i)$ 对数据进行分组间复制. 根据式(1)和(2)计算每组数据的密度值 ρ 和局部斥群值 δ'.

4)寻找每组中除具有最大密度之外数据点的局部斥群值 δ' 范围内的 z 值最小下限 $\mathrm{lsz}^{\delta'}(P_i)$ 和最大上限 $\mathrm{usz}^{\delta'}(P_i)$.

5)根据该 $\mathrm{lsz}^{\delta'}(P_i)$ 和 $\mathrm{usz}^{\delta'}(P_i)$ 重新对数据进行组间复制,根据式(2)计算组内每个数据点的斥群值 δ. 而将分组中具有最大密度值的点发送到其他分组中,然后计算该点的斥群值 δ.

根据算法 1,首先寻找分布式密度峰值聚类算法计算密度值时所需的 d_c 值和分组时所需的分位点 n_i. 由于没有计算点集中所有点对之间的距离,故文献[4]所提供的取所有点对距离从大到小排序后的 1% ~2% 处的值作为 d_c 值的方法不可行. 本文采用文献[5]基于对数据采样的方法,近似估计合适的 d_c 值. 同样,当数据量非常大时,寻找精确的分位点也比较困难,本文同样适用采样的方法对分组分位点进行估计,得到近似分位点,然后对数据根据所得近似分位点分组.

MapReduce 作为 Hadoop 云计算平台的主要模块用于执行大规模数据处理的分布式计算. 在得到 d_c 值和分位点后我们利用 MapReduce 模型设计基于 z 值的分布式密度峰值聚类算法. 该算法总共包括一个 Map 和两个 MapReduce.

3.2.1　第一个 Map 过程

该过程实现对数据集分组,同时统计每个分组中 $\mathrm{lz}^{d_c}(P_i)$ 和 $\mathrm{uz}^{d_c}(P_i)$ 值. 其对应的 Map 过程如下所示:

(1) $\forall p\in D$,如果 $z_{ni}<z_p<z_{ni+1}$,则将点 p 发送到分组 P_{i+1},即 $p\in P_{i+1}$.

(2)在分组 P_{i+1} 中,将点 p 每一维坐标都减(加)d_c 并计算其 z 值 $z_p^{d_c-}(z_p^{d_c+})$,然后更新 $\mathrm{lz}^{d_c}(P_i)$ 和 $\mathrm{uz}^{d_c}(P_i)$,$\mathrm{lz}^{d_c}(P_i)=\min\{z_p^{d_c},\mathrm{lz}^{d_c}(P_i)\}$,$\mathrm{uz}^{d_c}(P_i)=\max\{z_p^{d_c+},\mathrm{uz}^{d_c}(P_i)\}$.

3.2.2　第一个 MapReduce 过程

该过程实现对密度值 ρ 的计算和局部斥群值 δ' 的计算,具体步骤如下所示:

(1)Map:$\forall q\notin P_i$,且 $\mathrm{lz}^{d_c}(P_i)<z_q<\mathrm{uz}^{d_c}(P_i)$,则将 q 复制到分组 P_i 中.

(2)Reduce:根据式(1)计算每组中数据点的密度值 ρ,然后根据式(2)计算每组数据点的斥群值 δ'.

在分组内部计算每个点密度值 ρ 时可能导致错误的结果,如图 6 所示,点 $p\in P_i$,而点 $q\notin P_i$,且 $|p,q|<d_c$. 那么计算出的点 p 的密度值 ρ_p 将是一个错误的值. 因此需要将其它组中类似于 q 的点复制到分组 P_i 中. 为此本文提出了一种基于 d_c 范围内 z 值最小下限和最大上限的复制策略.

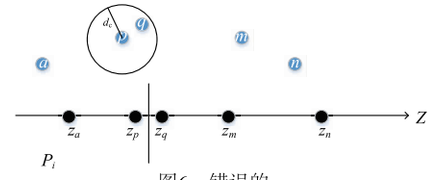


图6　错误的ρ_p

根据式(1)可得,计算数据点 $p\in P_i$ 的密度值 ρ 就是计算点 p 周围 d_c 范围内的点. 因此只要将其他组中的点 $q\notin P_i$,且 $|p,q|<d_c$ 的点复制到 P_i 中,分组 P_i 中的点便可得到正确的密度值. 为了寻找点 q,首先给出如下定理.

定理 1　$\forall q\notin P_i,p\in P_i$,且 $|p,q|<d_c$ 时,那么 z_q 一定满足如下关系:

$$\mathrm{lz}(P_i)<z_q<\mathrm{uz}(P_i)\tag{7}$$

其中 $\mathrm{lz}(P_i)=\min\{z_p^-,\forall p\in P_i\}$,$\mathrm{uz}(P_i)=\max\{z_p^+,\forall p\in P_i\}$,$z_p^-=\min\{z_q,|p,q|<d_c\}$ 和 $z_p^+=\max\{z_q,|p,q|<d_c\}$.

证明　$\forall q\notin P_i,p\in P_i$,且 $|p,q|<d_c$ 时,根据 $\mathrm{lz}(P_i)$ 和 $\mathrm{uz}(P_i)$ 的定义可得,$\mathrm{lz}(P_i)<z_p^-$,$\mathrm{uz}(P_i)>z_p^+$,根据 z_p^- 和 z_p^+ 的定义可知 $z_p^-<z_q<z_p^+$. 因此 $\mathrm{lz}(P_i)<z_q<\mathrm{uz}(P_i)$　证毕

但是精确寻找 z_p^- 和 z_p^+ 是一个非常困难的事情(列举 d_c 范围内所有点的坐标,并计算其 z 值),为此本文使用了文献[6]所采用的方法. 将每一维坐标都减 d_c 并计算其 z 值 $z_p^{d_c-}$,每一维坐标都加 d_c 并计算其 z 值 $z_p^{d_c+}$. 根据 z 值的定义可知,$z_p^{d_c-}<z_p^-$,$z_p^{d_c+}>z_p^+$. 由此得出如下定理:

定理 2　$\forall q\notin P_i,p\in P_i$,且 $|p,q|<d_c$ 时,那么 z_q

一定满足如下关系：

$$\mathrm{lz}^{d_c}(P_i) < z_q < \mathrm{uz}^{d_c}(P_i) \tag{8}$$

其中 $\mathrm{lz}^{d_c}(P_i) = \min\{z_p^{d_c-}, \forall p \in P_i\}$，$\mathrm{uz}^{d_c}(P_i) = \max\{z_p^{d_c+}, \forall p \in P_i\}$.

证明 因为 $z_p^{d_c-} < z_p^-$，$z_p^{d_c+} > z_p^+$，所以 $\mathrm{lz}^{d_c}(P_i) < \mathrm{lz}(P_i)$，$\mathrm{uz}^{d_c} > \mathrm{uz}(P_i)$，又因为 $\mathrm{lz}(P_i) < z_q < \mathrm{uz}(P_i)$，所以 $\mathrm{lz}^{d_c}(P_i) < z_q < \mathrm{uz}^{d_c}(P_i)$. 证毕

由定理 2 可得如下推论：

推论 1 $\forall q \notin P_i$，且 $\mathrm{lz}^{d_c}(P_i) < z_q < \mathrm{uz}^{d_c}(P_i)$ 时，点 q 因有可能成为分组 P_i 中某个点的 d_c 范围内的点而被复制到分组 P_i 中

根据推论 1 可以将其他分组中满足式(8)的点复制到分组 P_i 中. 然后，组中每个点与该组中其它点和复制进来的点计算距离，得到每个点的正确密度值 ρ.

由于在计算每个点的密度值 ρ 时并没有计算所有点对之间的距离，而是在每个分组内部和根据定理 2 和推论 1 对从其他分组中复制进来的点之间进行了距离计算. 从而节省大量计算开销，提高算法执行效率.

计算出组中每个点的密度值 ρ 之后根据式(2)可以计算组内每个点的斥群值 δ. 由于不知道由其它组复制进来的点的密度，因此只能组内点之间进行运算，所以该斥群值只是一个局部斥群值 δ'，该斥群值并不是原始算法中全局范围内的值. 如图 7 所示，点 p 的斥群值依附点在分组 P_{i-1} 中. 因此需将其他组中的带有密度值 ρ 的数据点复制进来，计算全局斥群值 δ.

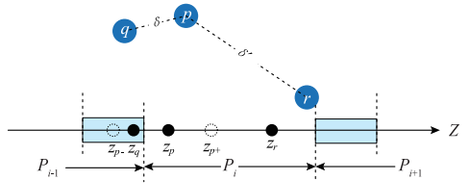


图7 局部斥群值δ'

3.2.3 第 2 个 MapReduce 过程

该过程实现对局部斥群值的进一步求精，得到全局斥群值 δ 具体步骤如下所示：

(1) Map：$\forall q \notin P_i$，且 $\mathrm{lsz}^{\delta'}(P_i) < z_q < \mathrm{usz}^{\delta'}(P_i)$，$\rho_p > \min\rho(P_i)$，则将 q 复制到分组 P_i 中. $\forall m = \mathrm{argmax}\{\rho_m, \forall m \in P_j\}$，将点 m 每一维坐标都减(加)δ'_m 并计算其 z 值 $z_p^{\delta'-}(z_p^{\delta'+})$，当 $z_p^{\delta'-} < \max^z(P_j)$，$z_p^{\delta'+} > \min^z(P_j)$，$\rho_m > \max\rho(P_i)$，则将 m 复制到分组 P_j 中，其中 $\min^z(P_j) = \max\{z_p, p \in P_j\}$.

(2) Reduce：根据式(2)计算每组数据点斥群值 δ.

计算 δ 值的复制方法不同于计算密度值 ρ 的复制方法，在计算密度值 ρ 时，由于距离阈值 d_c 是已知条件，因此可以找到每个点的 d_c 范围内所有点的 z 值上下限. 而在计算斥群值 δ 时，在没有得到每个点正确的斥群值 δ 之前，不能确定每个点的 δ 范围内所有点的 z 值上下限. 为此本文提出了基于 δ' 范围内 z 值最小下限和最大上限的复制策略.

在分组内部计算出每个点的局部斥群值 δ'，根据的斥群值 δ 定义点可得点 p 的 δ' 与 δ 满足如下关系：

$$\delta_p \leqslant \delta'_p \tag{9}$$

因此 δ' 可以作为 δ 值的上限值.

由此可以得出如下定理.

定理 3 $\forall q \in P_j$，δ'_p 为点 p 在分组 P_i 中计算出的局部斥群值. 如果 $\sigma_p = q$，那么点 q 的 z 值满足如下关系：

$$\mathrm{lsz}^{\delta'}(P_i) < z_q < \mathrm{usz}^{\delta'}(P_i) \tag{10}$$

其中 $\mathrm{lsz}^{\delta'}(P_i) = \min\{z_p^{\delta-}, \forall p \in P_i, p \neq m\}$，$\mathrm{usz}^{\delta'}(P_i) = \max\{z_p^{\delta+}, \forall p \in P_i, p \neq m\}$，$m = \mathrm{argmax}\{\rho_m, \forall m \in P_j\}$.

证明 $\forall q \notin P_i, p \in P_i$，且 $\sigma_p = q$，则 $|p,q| = \delta_p$，又因为 $\delta_p \leqslant \delta'_p$，所以 $|p,q| \leqslant \delta'_p$. 由定理 2 可得 $z_p^{\delta'-} < z_q < z_p^{\delta'+}$，根据 $\mathrm{lsz}^{\delta'}(P_i)$ 和 $\mathrm{usz}^{\delta'}(P_i)$ 的定义可知 $\mathrm{lsz}^{\delta'}(P_i) \leqslant z_p^{\delta'-}$，$\mathrm{usz}^{\delta'}(P_i) \geqslant z_p^{\delta'+}$，因此，$\mathrm{lsz}^{\delta'}(P_i) < z_q < \mathrm{usz}^{\delta'}(P_i)$ 证毕

由斥群值 δ 的定义可知，如果点 $\sigma_p = q$，那么 p 和 q 的密度值满足 $\rho_q > \rho_p$. 根据定理 4 和斥群值 δ 的定义可得如下推论：

推论 2 $\forall q \in P_j$，将 q 复制到分组 P_i 中的条件为：$\mathrm{lsz}^{\delta'}(P_i) < z_q < \mathrm{usz}^{\delta'}(P_i)$，$\rho_p > \min\rho(P_i)$，其中 $\min\rho(P_i) = \min\{\rho_p, p \in P_i\}$.

根据推论 2 可以将其他组中的点复制到分组 P_i 中，然后 P_i 分组中的点与复制进来的点进行距离计算，从而得到每个点的最终正确斥群值 δ.

基于 δ' 值范围内 z 值最小下限和最大上限的复制策略只能将组内不具有最大密度值的点的依附点复制到本组中. 由于组内具有最大 ρ 值点的 δ' 值无法根据式(2)得到(组内没有比该点密度大的点)，又因为该点未必是整个数据集中所有点的密度值最大点，因此也不能根据式(3)求出. 为此本文提出了基于各组最大密度值点的 δ'' 范围内的 z 值下限和上限分发策略.

数据的分组数为 n，数据集中点的个数为 N，则 $n \ll N$. 因此，把所有分组中具有最大密度值 ρ 的点收集到起来，然后在这些点之间根据式(2)计算出每个点的一个斥群值近似值 δ'' 值，根据斥群值的定义，分组 P_i 中具有最大密度值的点 m 的 δ''_m 值和 δ_m 值一定满足 $\delta_m \leqslant \delta''_m$. 而在所有的这些点中除具有最大全局最大密度的点之外都有一个 δ''，最简单的方法是根据定理 1 将其他分组中满足如下关系的点复制到 P_i 中

$$z_p^{\delta''-} < z_q < z_p^{\delta''+} \tag{11}$$

但是这样会造成大量的冗余复制，首先 δ'' 值由不同组之间的点计算得到，因此该值一般会比较大，其次只

是计算分组中这一个点的斥群值 δ 却复制大量的数据. 另一种方法是将该点复制到其他分组中,但是如果将这些组内最大密度值点复制到所有的组中计算距离,也会带来非常大的计算开销,因此只需要将这些组内密度最大值点发送到其他那些可能存在这些点的 δ 值依附点的分组中即可,为此本文提出如下定理:

定理 4　$m=\text{argmax}\{\rho_m, \forall m\in P_i\}$, $\exists \sigma_m\in P_j$,那么一定满足如下关系

$$z_p^{\delta^-}<\max^z(P_j)\text{且}z_p^{\delta^+}>\min^z(P_j) \tag{12}$$

其中 $\max^z(P_j)=\max\{z_p, p\in P_j\}$, $\min^z(P_j)=\min\{z_p, p\in P_j\}$.

证明:$m=\text{argmax}\{\rho_m, \forall m\in P_j\}$,且 $q=\sigma_m\in P_j$,则 $z_p^{\delta^-}<z_q<z_p^{\delta^+}$,根据 $\max^z(P_j)$ 和 $\min^z(P_j)$ 的定义可得 $\min^z(P_j)<z_q<\max^z(P_j)$,因此 $z_p^{\delta^-}<\max^z(P_j)$, $z_p^{\delta^+}>\min^z(P_j)$. 证毕

由此可得如下推论:

推论 3　$m=\text{argmax}\{\rho_m, \forall m\in P_i\}$,则为计算 m 的斥群值 δ 而将 m 复制到分组 P_j 中的条件为:

$$z_p^{\delta^-}<\max^z(P_j), z_p^{\delta^+}>\min^z(P_j), \rho_m>\max\rho(P_i) \tag{13}$$

根据推论 3 可以将每组中具有最大密度值 ρ 的点复制到其他分组中,并计算出其全局斥群值 δ. 由于组内密度最大值点 m 可能被复制到多个分组中,得到多个 δ 值,因此最后需要一个合并的过程,比较所有 δ 值,选出最小值为最终值. 而全局密度最大值点无需计算其斥群值,可将该点直接视为某个聚类中心点.

在计算每个点的斥群值 δ 时,根据定理 3 和定理 4 在数据分组复制和分发过程中采用了数据过滤方案,因此减少了不必要的数据传输开销和距离计算开销,提高了算法执行效率,增强算法的实用性.

3.3　算法分析

计算密度值 ρ 的第 1 个 MapReduce 过程中,假设每个点的平均副本个数为 a,则计算密度值 ρ 的 shuffle 开销为 $O(a\cdot N)$,其中 N 为数据集点的个数;距离计算开销为 $O((a+1)\cdot(N/n)^2)$,其中 n 为数据分组个数.

在计算斥群值 δ 的第 2 个 MapReduce 过程中,假设每个点的平均副本个数是 b,则计算 δ 时的 shuffle 开销为 $O(b\cdot N)$,其中 N 为数据集点的个数;距离计算开销为 $O((b+1)\cdot(N/n)^2)$,其中 n 为数据分组个数.

4　实验结果及分析

为了验证本文提出方法的有效性,本文分别在低维数据集和真实大规模高维度数据集上对 DP-z 算法进行了聚类精度和执行效率上的测试. 实验在基于云计算开发环境的开源分布式框架 Hadoop 实现,并通过 3 个真实数据集在本地集群上对 DP-z 算法和 DPC 算法以及 EDDPC 算法进行了实验对比和性能分析.

4.1　实验数据集

本文的实验数据集使用 KDD'99_10%[7], FCoverType[7] 和 facial[8] 真实数据集. KDD'99_10% 是 1998 年 DARPA 在 MIT 林肯实验室进行的一项入侵检测评估项目中统计到的网络入侵记录. 数据集由 494021 个拥有连接时间、传输数据量等 42 个属性的数据点组成的数据集,本文只截取其中 34 个实值属性. FCoverType 是美国某一区域的森林覆盖类型的记录数据,由 581012 个包括经纬度在内的 54 个属性数据点组成的数据集. Facial 数据集是 27936 张人脸图组成的数据集,每个人脸图包括 300 个像素点. 此外,为了便于可视化显示实验结果和验证算法效果,本文实验也使用了低维数据集 *S1*[9] 和数据集 Spiral[10].

4.2　算法性能分析

本文首先通过实验验证本文算法的聚类效果,然后重点进行算法执行效率实验,并进行对比分析.

4.2.1　聚类效果对比

实验首先对本文研究的 DP-z 算法和使用最广泛的 K-means 聚类算法进行了实验对比和分析. 本实验使用二维数据集 Spiral,包含 312 个数据点,聚类簇为 3 个. 实验结果发现,K-means 算法没有实现正确聚类,聚类的正确率为 37.82%. 这是因为 K-means 算法在执行聚类过程时只考虑数据距离邻近特性,而没有考虑数据的密度分布特征. 而本文研究的 DP-z 算法实现了对 312 个数据点的正确分类,表明算法对非线性分割的数据集取得了较好的聚类效果,算法的执行兼顾了数据的相似性和分布特征. 同时本文算法 DP-z 也不需要事先指定聚类簇个数,而是能够根据数据集的分布自适应的产生聚类簇数目.

本文同时对 DP-z 算法,EDDPC 算法和 DPC 算法的聚类效果进行了实验对比. 本实验使用二维数据集 *S1* 和高维数据集 FCoverType 两个真实数据集. 其中数据集 S1 具有 15 个类簇,5000 个点. FCoverType 具有 54 个类簇,581012 个点.

本实验对 DP-z 算法,EDDPC 算法和 DPC 算法在数据集 S1 进行聚类实验,三种算法均得到 15 个聚类中心,并且对于每一个数据点得到了一致的正确聚类结果. 本文统计了在高维数据集 FCoverType 上三种算法所得到的密度值 ρ 和斥群值 δ,统计结果显示,每个点由三种算法计算所得到的密度值 ρ 和斥群值 δ 均相等. 实验结果证明本文研究的 DP-z 算法和 EDDPC 算法及 DPC 算法在聚类效果上取得了相同的结果,验证了本文研究算法的正确性和有效性.

本文算法的重点在于对高维大数据处理时,如何

提高聚类效率. 因此下面将详细讨论和验证在不同数据集上对本文的 DP-z、EDDPC 和 DPC 三个算法的运行效率进行比较.

本文用同一数据集分别截取不同数量的点和不同维度对 DP-z 和 EDDPC 进行了实验对比与分析. 最后针对不同规模的数据集测试了 DP-z 算法和 EDDPC 算法的速率差. 以上算法均使用相同硬件环境和 Mapreduce 分布式计算模型实现.

4.2.2 不同数据集算法执行效率对比

图 8 为 KDD'99_10% 和 FCoverType 数据集的前 100k 条数据上本文算法 DP-z、EDDPC 和 DPC 算法运行时间的比较. 结果显示 DP-z 和 EDDPC 算法效率明显高于原始的 DPC 算法.

DP-z 和 EDDPC 这两种算法在执行过程中都采用了过滤的措施,减少了大量的冗余计算. 而对 DP-z 和 EDDPC 两种算法进行比较可以看出在 KDD'99_10% 和 FCoverType 两个数据集上 DP-z 要比 EDDPC 算法执行时间短,而在 facial 数据集上 EDDPC 算法的执行时间要少于 DP-z,经分析发现,facial 数据集中包含的数据量较小,在进行距离计算时 DP-z 与 EDDPC 所用时间基本相同,而由于 DP-z 算法需计算数据点的 z 值,因此产生额外开销. 数据量非常大的 KDD'99_10% 和 FCoverType 两个数据集中,虽然有数据点 z 值计算的开销,但是由于 EDDPC 算法中包含大量的冗余距离计算,因此总体运行时间比 DP-z 长. 这说明在数据量越大的情况下,本文所提出的 DP-z 算法相比于 EDDPC 算法优势越明显.

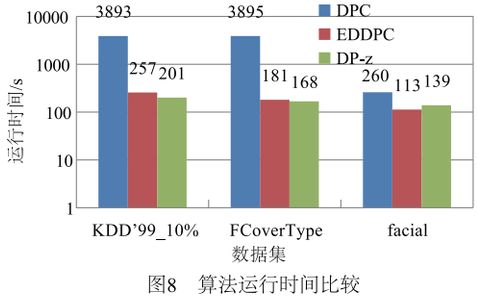


图8 算法运行时间比较

4.2.3 不同数据量算法执行效率对比

如图 9 显示了 KDD'99_10% 和 FCoverType 分别截取 5×10^4, 1×10^5, 1.5×10^5, 2×10^5 和 2.5×10^5 个数据点构成的 5 个大小不同的数据集的运行时间情况. 从图 9(*a*)和 9(*c*)中可以看出 DP-z 算法所需运行时间比 EDDPC 算法少,随着数据集的增长运行时间差距越来越大. 说明数据量越大,本文的 DP-z 算法处理效率越高. 为分析其造成该差距的原因,本文统计了算法执行过程中每个点因分组间的复制而产生的副本数量(图 9(*b*)和 9(*d*)),图中可以看出随着数据集中点的个数增加,DP-z 算法中的数据复制方法与 EDDPC 算法中数据复制方法相比,DP-z 算法中产生的数据点副本个数增长缓慢,副本数量变少,从而使数据点之间距离计算次数减少,缩短了程序执行时间.

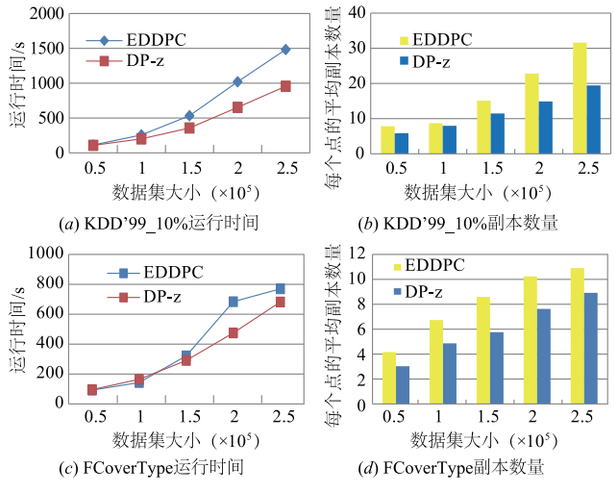


图9 数据量不同算法运行效率比较

4.2.4 不同数据维度算法执行效率对比

图 10 显示 DP-z 和 EDDPC 在 KDD'99_10% 和 FCoverType 数据集中前 150k 条记录中不同维度的运行时间对比. 实验结果显示,KDD'99_10% 数据集中在前 15 个属性中随着属性个数的增多,运行时间逐渐变长;15~20 个属性时,随着属性个数的增加,运行时间反而下降.

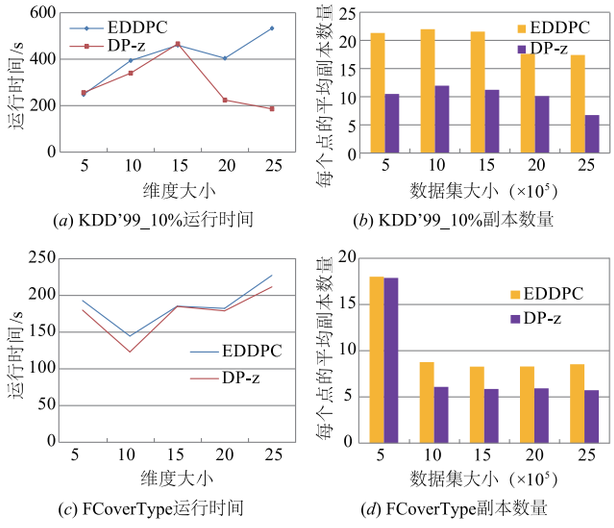


图10 数据维度不同时算法效率比较

在 FCoverType 数据集中,在前 5~10 个属性中随着属性个数的增多,运行时间下降,但从第 10 个属性往后,随属性的增多运行时间逐渐增大. 为分析其原因,我们对算法过程中副本的个数进行了统计,图 10(*b*)展示

了 KDD'99_10% 数据集在算法过程中产生的副本数量的变化,从图中可以发现数据集在有 15 个属性时副本个数基本维持稳定,由于维度的增加,导致其运行速度变慢,而在有 15 ~ 25 个属性时,副本个数急剧减少,虽然维度仍然增大,但是由于副本数量大量减少,因此运行速度加快. 观察发现原因是前 15 个属性中大部分的属性值为 0. 导致数据比较集中,因此数据副本个数非常大,而在 15 ~ 25 个属性中不为 0 的属性增多,数据分布开始稀疏,因此副本个数减少,运行时间反而加快. 图 10(d)展示了 FCoverType 数据集在算法过程中产生的副本数量,在数据集只有 5 个属性时数据产生的副本数量异常大,导致其运行时间长,而在大于 10 个属性时副本数量基本保持不变,而随着维度的增大距离计算开销变大因此运行时间变慢. 经观察发现,数据集只有 5 个属性时,数据比较密集,导致分组之间数据交互多,因此副本数量非常大.

4.2.5　不同分组数算法执行效率对比

图 11 为 DP-z 在 KDD'99_10% 和 FCoverType 的前 100k 数据集上分组数量不同时算法执行时间. 结果显示,随着分组数量的增加,算法执行时间先变小然后增大,这是因为随着分组数量的增加,组内元素变少,加快程序运行时间. 而随着分组数量的不断增加,导致 Hadoop 的 Map 和 Reduce 线程数也不断增加,增加了系统开销,导致程序运行时间变慢.

4.2.6　速率差和数据量的关系

图 12 为 DP-z 和 EDDPC 在 KDD'99_10% 和 FCoverType 数据集上的运行速率差随数据集大小变化而变化的情况. 定义 DP-z 对 EDDPC 的速率差为$(t_{\text{EDDPC}} - t_{\text{DP-z}})/|D|$,其中$|D|$为数据集大小. 速率差表示计算平均每 1k 个数据点两个算法的运行效率差. 从图 12 中可以看出随着数量增大,KDD'99_10% 数据集上两个算法的运行速率差越来越大,说明 DP-z 处理数据量越大效率越高. 而在 FCoverType 数据集上,两个算法执行效率在数据集为 200k 时,DP-z 比 EDDPC 效率高,其他情况下基本相同. 这是因为 FCoverType 数据在 200k 时出现了数据倾斜,分布不均匀,导致 EDDPC 算法在执行时出现短板效应. 而 DP-z 由于机器之间运行时间比较均衡,因此避免了该现象,因此在 EDDPC 在 200k 时执行效率变慢.

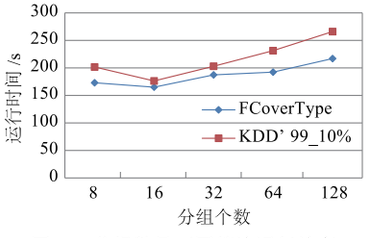


图11　分组数量不同算法运行效率

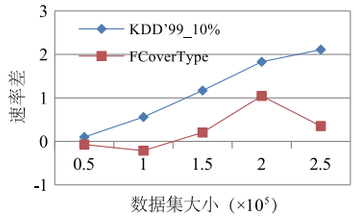


图12　速率差变化

5　结论

本文深入探究密度峰值聚类算法在执行过程中因大量距离计算而导致的效率问题,并且分析现有分布式密度峰值聚类算法所存在的弊端. 为了能够更好的发挥密度峰值聚类算法在聚类效果上的优势,增强其实用性,本文结合数据点的 z 值特性和云计算框架 MapReduce 提出基于 z 值的分布式密度峰值聚类算法:DP-z. 该算法使用基于 z 值分布情况的分组方法,缓解了 Voronoi 图分组方法中因随机性带来的负载不均衡问题. 同时为能够正确计算每个点的 ρ 值和 δ 值,提出了两种高效的组间数据复制策略,从而大大减少了数据副本数量,节省了程序运行过程中的时间和空间开销. 使密度峰值聚类算法的执行效率大大提升,增强了该算法的实用性和可行性.

虽然本文在算法执行效率上得到了大幅度提升,但是并没有对聚类质量提升的提升进一步研究,这将是下一步进行深入研究的问题.

参考文献

[1] 何宏,谭永红. 一种基于动态遗传算法的聚类新方法[J]. 电子学报,2012,40(2):254-259.
He H,Tan Y H. A novel clustering method based on dynamic genetic algorithm[J]. Acta Electronica Sinica,2012,40(2):254 - 259. (in Chinese)

[2] 王玲,薄列峰,焦李成. 密度敏感的谱聚类[J]. 电子学报,2007,35(8):1577 - 1581.
Wang Ling,Bo Lie Feng,Jiao Li Cheng. Density-sensitive spectral clustering[J]. Acta Electronica Sinica,2007,35(8): 1577. (in Chinese)

[3] Zhang Wei,Wang X,Zhao D,et al. Graph Degree Linkage: Agglomerative Clustering on A Directed Graph[M]. Belin Heidelberg: Springer,2012. 428 - 441.

[4] Rodriguez A,Laio A. Clustering by fast search and find of density peaks[J]. Science, 2014, 344 (6191): 1492 - 1496. .

[5] 巩树凤,张岩峰. EDDPC:一种高效的分布式密度峰值聚类算法[J]. 计算机研究与发展,2016,53(6):1400 - 1409.
Gong Shufeng,Zhang Yanfeng. EDDPC: An effect distrib-

uted density peaks clustering algorithm [J]. Journal of Computer Research and Development, 2016, 53(6), 1400 – 1409. (in Chinese)

[6] Zhang C, Li F, Jestes J. Efficient parallel kNN joins for large data in MapReduce[A]. Proc of EDBT '14[C]. Berlin, Germany: ACM, 2012. 38 – 49.

[7] Lichman M. UCI Machine Learning Repository [DB/OL]. http://archive. ics. uci. edu/ml, 2013

[8] Freitas F A, Peres S M, Lima C A M, et al. Grammatical facial expressions recognition with machine learning [A]. Proc of FLAIRS'14[C]. Menlo Park, CA: AAAI Press, 2014. 180 – 185.

[9] Fränti P, Virmajoki O. Iterative shrinking method for clustering problems [J]. Pattern Recognition, 2006, 39(5): 761 – 775.

[10] Chang H, Yeung D Y. Robust path-based spectral clustering[J]. Pat-tern Recognition, 2008, 41(1): 191 – 203.

作者简介

卢 晶 女, 1992 年 7 月出生, 辽宁盘锦人, 现为沈阳工业大学信息科学与工程学院硕士研究生. 主要研究方向为数据挖掘、大数据分布式计算.
E-mail: fionalujing@163. com

段 勇(通信作者) 男, 1978 年生于辽宁, 2007 年于东北大学信息科学与工程学院获博士学位, 现为沈阳工业大学信息科学与工程学院教师, 主要从事机器学习和智能机器人的研究工作.
E-mail: duanyong0607@126. com

刘海博 男, 1979 年 2 月出生, 河北廊坊人, 硕士. 现为河北大学计算机科学与技术学院副教授. 研究领域为大规模数据挖掘、个性化推荐与大规模社会网络分析.
E-mail: liuhaibo@hbu. edu. cn