计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(11):247−252 [doi: 10.15888/j.cnki.csa.009309]

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

基于集束搜索的可解释阈值树构造①

李钰群, 何振峰

(福州大学 计算机与大数据学院, 福州 350108)
通信作者: 李钰群, E-mail: 593471409@qq.com

摘　要: 传统的聚类算法能够将数据集划分成不同的簇, 但是这些簇通常都是难以解释的. IMM (iterative mistake minimization) 是一种常见的可解释聚类算法, 通过单个特征来构造阈值树, 每个簇都可以用根节点到叶子节点路径上的特征-阈值对进行解释. 然而, 阈值树在每一轮划分数据时仅考虑错误最少的特征-阈值对, 这种贪心的方法容易导致局部最优解. 针对这一问题, 本文引入了集束搜索, 通过在阈值树的每一轮划分过程当中保留预定数量的状态来减缓局部最优, 进而提高阈值树提供的聚类划分与初始聚类划分的一致性. 最后, 通过实验验证了该算法的有效性.

关键词: 可解释聚类; 集束搜索; 阈值树; K-means

引用格式: 李钰群,何振峰.基于集束搜索的可解释阈值树构造.计算机系统应用,2023,32(11):247–252. http://www.c-s-a.org.cn/1003-3254/9309.html

Explainable Threshold Tree Construction Based on Beam Search

LI Yu-Qun, HE Zhen-Feng

(College of Computer and Data Science, Fuzhou University, Fuzhou 350108, China)

Abstract: Traditional clustering algorithms can split the dataset into different clusters, whereas these clusters are usually difficult to explain. Iterative mistake minimization (IMM) is a common explainable clustering algorithm, which constructs a threshold tree from a single feature, and each cluster can be explained by feature-threshold pairs on the path from the root node to the leaf node. However, the threshold tree only considers the feature-threshold pair with the fewest errors when dividing the data in each round, and this greedy method is easy to lead to the local optimal solution. To solve this problem, this study introduces beam search, which slows local optimization by retaining a predetermined number of states in each round of division, thereby improving the consistency between the clustering provided by the threshold tree and the initial clustering. Finally, the effectiveness of the algorithm is verified by experiments.

Key words: explainable clustering; beam search; threshold tree; K-means

聚类是一种典型的无监督学习方法, 它基于相似性将一组数据分为若干簇, 使簇内的数据相似性尽可能大, 不同簇中的数据相似性尽可能小[1]. 聚类的一个常见应用是利用未标记的数据点来识别数据集中的模式以及发现数据集的底层结构特点[2]. 在一些比较经典的聚类算法 (如 K-means 算法) 中, 尽管 K-means 聚类获得的簇有简单的数学解释, 但该解释对用户来说不一定容易理解[3]. 由于生成的簇中心集以复杂的方式依赖于数据集中的数据点. 因此, 数据点与距离其最近的簇中心之间的关系可能是许多特征的不透明组合, 这给关于聚类的解释带来了挑战.

近几年以来, 随着人们对机器学习模型透明度需求的不断增加, 可解释机器学习的呼声越来越高. 人们开始对一些黑盒模型持怀疑态度, 转而关注关于预测结果的决策依据[4]. 作为可解释机器学习的一个重要子领域, 可解释聚类也开始受到关注, 尤其是在缺少监督

① 收稿时间: 2023-03-21; 修改时间: 2023-05-11; 采用时间: 2023-07-03; csa 在线出版时间: 2023-08-22
CNKI 网络首发时间: 2023-08-23

信息的无监督场景. 可解释聚类算法研究发展至今, 阈值树是更流行的解释形式, 它更加清晰地突出了重要的特征且产生了更容易解释的可见结果[5]. 文献 [6] 提出一种基于密度测量的 CLTree 算法, 通过将数据空间划分为密集和稀疏区域进行聚类分析, 并根据簇之间的密集性来决定拆分. 文献 [7] 受 CART 算法[8] 启发, 提出了一种基于无监督二叉树的 CUBT 算法, 利用递归分区算法生成二叉树之后再进行剪枝、聚合等操作, 但是该算法只适用于连续数据. 针对这个问题, 文献 [9] 则提出了基于信息熵以及互信息的方法将 CUBT 算法拓展到应用于标称数据. 文献 [10] 提出了 ICOT 算法, 该算法建立在最优分类树[11] 上, 并将其拓展到无监督模型. ICOT 利用混合整数优化框架生成聚类模型, 从而实现在特征空间的高质量分区.

虽然在决策树上进行可解释聚类取得了一定的研究进展, 但是这些工作没有与初始聚类划分相比较从而提供聚类质量分析. 可解释聚类只有与流行的聚类算法结合, 才能得到较好的发展. 于是, 文献 [2] 提出了 IMM 算法, 在生成初始 K-means 聚类划分之后, 运用贪心算法的思想迭代地寻找错误点数量最少的特征-阈值对, 以此来构造一棵阈值树, 其中每个叶子节点与生成的簇一一对应, 该算法可以高效并简明地解释聚类分配. 然而, 阈值树在每一轮划分数据时仅考虑对于当前数据来说错误点数最少的那个特征-阈值对, 这种短视的方法很容易导致局部最优解. 对此, 我们在构造阈值树期间使用集束搜索算法来减缓局部最优.

集束搜索是一种流行的启发式搜索方法, 目前已广泛应用于多个领域, 包括自然语言处理[12]、组合优化问题[13] 等. 集束搜索定义了一个集束宽 B, 在状态空间中允许保留 B 个启发式值最高的状态, 以此来拓展搜索空间. 文献 [14] 将集束搜索结合到决策树归纳算法, 在构造决策树期间存储 B 个状态, 旨在通过集束搜索来减缓决策树产生局部最优的问题. 文献 [15] 则是提出了一种使用集束搜索来学习最优树模型的算法, 研究了在集束搜索背景下的贝叶斯最优性, 提高了树模型在信息检索方面的性能.

为了减缓 IMM 算法结果的局部最优问题, 本文在构造阈值树期间引入了集束搜索来拓展搜索范围, 在阈值树的每一轮划分过程当中通过保留更多的状态来获得更好的解决方案, 使得阈值树提供的聚类划分更加贴近初始聚类划分. 本文的主要内容包括 4 节: 第 1 节介绍 IMM 算法, 第 2 节介绍改进的 IMM 算法, 第 3 节通过实验对改进的 IMM 算法进行了验证, 并对实验结果进行了分析, 第 4 节对本文的工作进行了总结.

1 IMM 算法简介

IMM 算法又称为迭代错误最小化算法, 它在生成初始 K-means 聚类划分的基础之上, 通过迭代生成阈值树来构成数据集的近似 K-means 聚类, 达到可解释的目的[2]. 该算法的主要思想是: 首先使用生成的初始 K-means 簇标签来标记所有数据点, 在每一轮迭代划分数据的过程中基于单个特征, 利用贪心算法的思想以及标记信息找到错误点数最少的特征-阈值对, 最终生成具有 k 个叶子的阈值树.

给定数据集 $X=\{x^1,x^2,\cdots,x^n\}\in R^{d\times n}$ 以及整数 k, 其中 n 是数据集 X 的大小, d 是样本实例的维度, k 为簇数. 将 k 个簇中心表示为 $\mu^1,\cdots,\mu^k$, K-means 的目标是找到一个最小化以下目标函数的聚类:

$$cost(\mu^1,\cdots,\mu^k)=\sum\nolimits_{x\in X}\|x-c(x)\|_2^2 \tag{1}$$

其中, $\|\cdot\|_2$ 表示欧几里得范数, 样本实例 x 与距离其最近的簇中心 μ 之间的距离 $c(x)$ 可表示为:

$$c(x)=\mathop{\arg\min}_{\mu\in\{\mu^1,\cdots,\mu^k\}}\|\mu-x\|_2 \tag{2}$$

然而, 以上有关聚类的数学描述并不容易解释. 鉴于此, IMM 算法使用阈值树来解释聚类. 当簇数 k 为 2 时, IMM 算法使用阈值分割的方式来分离这两个簇, 即根据单个特征的阈值对数据进行分区. 这两个簇可以用 $\hat{C}^{\theta,i}=\left(\hat{C}^1,\hat{C}^2\right)$ 来表示, 其中 $\theta\in R$ 表示阈值, $i\in\{1,\cdots,d\}$ 表示特征序号. 对于每个输入点 $x\in X$, 令 $x=\{x_1,\cdots,x_d\}$. 若 $x_i\leqslant\theta$, $x\in\hat{C}^1$, 否则 $x\in\hat{C}^2$. 而当簇数 k 大于 2 时, 通过迭代地寻找特征-阈值对的方式来构造阈值树. 阈值树有以下特点: (1) 阈值树的每个内部节点都包含着一个阈值分割. (2) 阈值树恰好具有 k 个叶子节点, 每个簇都与树的叶子节点一一对应. (3) 阈值树的深度以及使用的特征总数最多为 k–1.

具有 k 个叶子节点的阈值树 T 引导了 k 个簇的生成. 将这些簇表示为 $\hat{C}^j\subseteq X, j\in k$, 树的 K-means 分区成本可以定义为:

$$cost(T)=\sum\nolimits_{j=1}^{k}\sum\nolimits_{x\in\hat{C}^j}\|x-mean(\hat{C}^j)\|_2^2 \tag{3}$$

为了减少在阈值树上的 K-means 分区成本, IMM 算法在确定阈值分割时采用了一种贪心标准, 即最小

化每次分割的错误数据点数. 当一个阈值分割在划分簇时, 如果数据点与其对应的簇中心被划分到不同的树节点, 则认为该数据点发生了错误. IMM 算法框架如算法 1 所示.

算法 1. IMM 算法

输入: 数据集$X=\{x^1,x^2,\cdots,x^n\}\in R^{d\times n}$, 簇数 k
输出: 阈值树

1) 采用 K-means($x^1,x^2,\cdots,x^n,k$)算法得到参考簇中心$\mu^1,\cdots,\mu^k$以及簇标签y^j, $j\in\{1,\cdots,n\}$
2) return build_tree($\{x^j\}_{j=1}^n,\{y^j\}_{j=1}^n,\{\mu^j\}_{j=1}^k$) //迭代构建阈值树, 详见步骤 (3)–步骤 (18)
3) build_tree($\{x^j\}_{j=1}^n,\{y^j\}_{j=1}^n,\{\mu^j\}_{j=1}^k$):
4) if ($\{y^j\}_{j=1}^n$中y^j都相同)
5) *leaf.cluster*←y^1
6) return *leaf*
7) for each $i\in\{1,\cdots,d\}$
8) $l_i\leftarrow\min_{1\leqslant j\leqslant n}\mu_i^{y^j}$
9) $r_i\leftarrow\max_{1\leqslant j\leqslant n}\mu_i^{y^j}$
10) end for
11) $i,\theta=\arg\min_{i,l_i\leqslant\theta<r_i}\sum_{j=1}^n \text{mistake}(x^j,\mu^{y^j},i,\theta)$ //详见步骤 (19) 和步骤 (20)
12) $M\leftarrow\{j|\text{mistake}(x^j,\mu^{y^j},i,\theta)=1\}_{j=1}^n$
13) $L\leftarrow\{j|(x_i^j\leqslant\theta)\wedge(j\notin M)\}_{j=1}^n$
14) $R\leftarrow\{j|(x_i^j>\theta)\wedge(j\notin M)\}_{j=1}^n$
15) *node.condition*←"$x_i\leqslant\theta$"
16) *node.left*←build_tree($\{x^j\}_{j\in L},\{y^j\}_{j\in L},\{\mu^j\}_{j=1}^k$)
17) *node.right*←build_tree($\{x^j\}_{j\in R},\{y^j\}_{j\in R},\{\mu^j\}_{j=1}^k$)
18) return *node*
19) mistake(x,μ,i,θ):
20) return $(x_i\leqslant\theta)\neq(\mu_i\leqslant\theta)$?1:0

如算法 1 所示, 阈值树由内部节点 *node* 和叶子节点 *leaf* 构成. 内部节点 *node* 包含数据域 *condition* 和指针域 l*eft*、*right*. 其中, 数据域 *condition* 存储着阈值分割(i,θ), 指针域 *left*、*right* 则分别存储着指向左右子节点的指针. 而叶子节点 *leaf* 只包含一个数据域 *cluster*, 用来存储节点内的簇标签.

在迭代划分簇的过程中, 若内部节点 *node* 当中包含至少两个簇中心μ^j时, IMM 算法通过步骤 (8) 和步骤 (9) 中计算出的 l_i 和 r_i 来限制阈值θ的取值范围, 保证了由内部节点分裂而成的每个子节点当中至少包含一个簇中心. 在步骤 (11) 中, 通过使用贪心思想来搜索错误点数最少的阈值分割(i,θ), 利用(i,θ)计算出步骤 (12) 中 M 存放的错误点并将其丢弃. 之后在步骤 (16) 和步骤 (17) 中递归生成左右子节点. 步骤 (19) 和步骤 (20) 描述了数据点与其对应的簇中心在划分簇的过程中是否发生了错误. 在步骤 (4)–步骤 (6) 中, 当树节点中的所有数据点都具有相同的簇标签时, 此树节点为叶子节点并将该簇标签赋予该叶子节点. 由于算法初始时的聚类数量为 k, 故最终的阈值树将会生成 k 个叶子节点, 此时算法终止.

2 基于集束搜索的可解释阈值树构造

我们将 IMM 构造阈值树的过程看作是一个状态空间搜索问题. 其中, 由根节点作为初始状态, 将算法运行各阶段拓展生成的树定义为状态. 具体而言, 由于阈值树是一种具有 k 个叶子节点的决策树, 每个叶子节点都与簇一一对应. 因此, 阈值树包含 k–1 个内部节点. 换而言之, 对于一个 k 簇的问题, 我们将根节点定义为初始状态, 在整个状态空间当中我们需要进行 k–1 步搜索来寻找到最终目标状态. 我们使用 (*data*, *labels*, *centers*, *condition*, *mistakes*, *cluster*) 表示一个树节点. 其中, *data*、*labels*、*centers* 分别为在该节点内的数据点集合、簇标签集合以及簇中心集合, *mistakes* 为在该节点划分数据时产生的错误点数.

但是, IMM 算法在每一步搜索当中采取的是贪心策略来选择分割, 从而生成阈值树, 这往往会导致局部最优的目标状态, 因为每一步搜索都会选择当前的最优解. 为了减缓这一问题, 本文提出了基于集束搜索的 IMM 算法 (beam search based iterative mistake minimization, BSIMM). BSIMM 算法在阈值树的每一轮划分中根据启发式值存储预定数量的状态 (即集束宽 B), 而并非仅是一个状态. 实际上, IMM算法采用的贪心策略本质上就是一种 B=1 时的集束搜索, 而当 B 的值无限制时, 集束搜索就会演变为广度优先搜索.

2.1 基于集束搜索的 IMM 算法

在阈值树的生成过程中, BSIMM 算法所采取的策略是对 IMM 算法的一种拓展. 除了贪心算法产生的局部最优状态, 在每次迭代中 BSIMM 算法通过使用预定义的集束宽 B 来允许在状态空间中探索其他候选状态. 我们定义, 如果当前节点包含两个及以上的簇中心, 则称该节点是待拓展的. 首先从包含所有数据的根节点开始, BSIMM 算法根据启发式值保留当前最优的 B 个状态, 启发式采用当前状态所产生的错误点数. 在每一轮的划分当中, 如果这 B 个状态中存在包含待拓展节点的状态, 则针对每个待拓展节点继续生成这些状态的后继状态. 如果这些拓展后的后继状态为最终目标状态, 则算法停止. 否则从这些后继状态中保留

B 个最优的状态继续进行拓展, 直至找到最终目标状态.

2.2 算法描述

BSIMM 算法框架如算法 2 所示.

算法 2. BSIMM 算法

输入: 数据集$X=\{x^1,x^2,\cdots,x^n\}\in R^{d\times n}$, 聚类数量 k, 集束宽 B, 节点考虑的分割数量 C
输出: 阈值树 t

1) 采用 K-means($x^1,x^2,\cdots,x^n,k$)算法得到参考簇中心$\mu^1,\cdots,\mu^k$以及簇标签y^j, $j\in\{1,\cdots,n\}$
2) $root$←使用 ($\{x^j\}_{j=1}^n$,$\{y^j\}_{j=1}^n$,$\{\mu^j\}_{j=1}^k$, ∅, ∅, ∅) 初始化根节点
3) $curr\_states$←$root$ //初始化当前搜索状态为根节点
4) v←1
5) while (v<k)
6) $next\_states$←[] //初始化下一步搜索状态为空
7) for each $state\in curr\_states$ //遍历当前搜索状态
8) for each $node\in state$
9) if (|$node.centers$|≥2) //|·|代表集合元素个数
10) c_states←generate(d, C, $node$, $\{\mu^j\}_{j=1}^k$) //详见算法 3
11) $next\_states$.append(c_states)
12) end for
13) end for
14) $next\_mis$←[]
15) for each $state\in next\_states$
16) $next\_mis$.append(cal_mistakes($state$)) //详见算法 4
17) end for
18) $curr\_states$←从 $next\_states$ 中根据 $next\_mis$ 选取前 B 个最小的状态
19) v←v+1
20) end while
21) t←$\arg\min_{s\in curr_states}$cal_mistakes($s$)
22) return t

如算法 2 中的步骤 (2) 所示, 整个状态空间搜索过程由初始状态 ($\{x^j\}_{j=1}^n$,$\{y^j\}_{j=1}^n$,$\{\mu^j\}_{j=1}^k$, ∅, ∅, ∅) 开始. 通过对各状态中的待拓展节点上的分割分别进行拓展从而不断生成新树, 即新状态, 直至搜索到最终目标状态为具有 k 个叶子节点的阈值树. 接下来介绍详细的搜索过程.

对于算法 2, 步骤 (5)–步骤 (20) 在一次迭代期间共进行了 k–1 步搜索, 使用步骤 (4) 中定义的 v 来控制搜索步数. 步骤 (7)–步骤 (13) 首先寻找到当前状态空间中可拓展状态的所有待拓展节点, 然后针对每个待拓展节点进行逐一拓展. 其中, 步骤 (10) 中的 generate 算法描述了在每个待拓展节点上生成 C 个状态的具体过程.

算法 3. generate 算法

输入: 数据点的维度 d, 节点考虑的分割数量 C, 当前节点 $node$, 全局簇中心集合$\{\mu^j\}_{j=1}^k$
输出: 树集合 c_trees

1) $\{x^j\}_{j=1}^n$←$node.data$ //当前节点的数据点集合
2) $\{y^j\}_{j=1}^n$←$node.labels$ //当前节点的簇标签集合
3) $\{c^j\}_{j=1}^h$←$node.centers$ //当前节点的簇中心集合
4) r_cuts←[]
5) r_mis←[]
6) for each $i\in\{1,\cdots,d\}$
7) for each $t\in\{1,\cdots,h-1\}$
8) $\theta\leftarrow\arg\min_{c_i^t\leqslant\theta<c_i^{t+1}}\sum_{j=1}^n$ mistake(x^j,μ^{y^j},i,θ) //详见算法 1 中步骤 (19) 和步骤 (20)
9) $mis\leftarrow\min_{c_i^t\leqslant\theta<c_i^{t+1}}\sum_{j=1}^n$ mistake(x^j,μ^{y^j},i,θ)
10) r_cuts←r_cuts.append(i,θ)
11) r_mis←r_mis.append(mis)
12) end for
13) end for
14) c_cuts←从 r_cuts 中根据 r_mis 选取前 C 个最小的分割
15) c_trees←在 $node$ 上将 c_cuts 进行拓展, 得到 C 棵树
16) 更新 c_trees 中 $node$ 的 $condition$, $mistakes$
17) c_left, c_right←c_trees 中拓展后的左右子节点
18) 更新 c_left, c_right 中的 $data$, $labels$, $centers$
19) for each $c\in\{1,\cdots,C\}$
20) if (|c_left[c].$centers$|==1)
21) c_left[c].$cluster$←c_left[c].$centers$ 所对应的簇号
22) if (|c_right[c].$centers$|==1)
23) c_right[c].$cluster$←c_right[c].$centers$ 所对应的簇号
24) end for
25) return c_trees

在算法 3 定义的 generate 算法中, 当前我们结合聚类问题的特点, 对每个待拓展节点内分割的选择步骤加以约束. 由于具有较好分类能力的特征一般能基于簇中心大致地将各簇的点分开, 并且此时在两个相邻簇中心之间的多个潜在分割点效果相似. 如步骤 (6)–步骤 (13) 所示, 我们针对每个特征, 在每两个相邻的簇之间只选择最优的分割点, r_cuts 即为在该待拓展节点内的候选分割集合. 此外, 如果在某个待拓展节点内考虑的分割的错误点数都较小, 那么我们在该轮保存的 B 个状态极易来自该节点, 从而忽略了其他待拓展节点. 因此在步骤 (14) 中, 我们从每个待拓展节点的候选分割集合 r_cuts 中保存 C 个分割代表, 其中 C 应小于 B. 接着, 我们在步骤 (15)–步骤 (24) 中描述可拓展状态中待拓展节点的拓展过程. 其中, 如果拓展后的子节点内只包含一个聚类中心时, 则该子节点为叶子节点, 并将该子节点的 $cluster$ 设置为该聚类中心所对应的簇号.

算法 4. cal_mistakes 算法

输入: 状态 $state$
输出: 状态 $state$ 产生的错误点数 $state\_mis$

1) *state_mis*=0
2) for each *node* ∈ *state*
3) if (|*node.centers*|≥2)
4) *state_mis*+= *node.mistakes*
5) end for
6) return *state_mis*

如算法 4 所示, 在算法 2 的步骤 (16) 和步骤 (21) 中的 cal_mistakes 算法则描述了计算状态产生的错误点数的过程. 其中, 状态产生的错误点数即为该状态中各个内部节点的 *mistakes* 之和.

3 实验与分析

如表 1 所示, 实验阶段使用 10 个 UCI 数据集, 分别是 Robot、Pendigits、Anuran、Vowel、Waveform、Mice、Digits、Avila 、Yeast 和 Ecoli. Robot 表示 Wall-Following Robot Navigation Data 数据集, Pendigits 表示 Pen-Based Recognition of Handwritten Digits 数据集, Anuran 表示 Anuran Calls (MFCCs) 数据集, Vowel 表示 Connectionist Bench (Vowel Recognition-Deterding Data) 数据集, Digits 表示 Optical Recognition of Handwritten Digits 数据集. 在每个数据集中, 对可解释聚类算法中的初始 K-means 算法进行 10 次随机种子迭代, 并将每次迭代生成的 K-means 分区作为可解释聚类算法的初始参考分区, 最后取可解释聚类算法结果的平均值作为最终结果. 实验的评价指标采用标准化分区成本 (normalized partition cost, NPC) 和标准化互信息 (normalized mutual information, NMI). 前者描述了可解释聚类算法与初始 K-means 算法之间关于分区成本的比例, 即式 (3) 与式 (1) 之比. 当该比例越低且越接近 1 时, 说明可解释聚类算法的分区成本越贴近初始 K-means 算法的分区成本; 而后者则用来比较预测标签与参考簇标签的吻合程度.

实验中加入了 IMM 算法、ExGreedy 算法[16] 以及 ExShallow 算法[17] 进行对比, 这些常用的可解释聚类算法都是生成 K-means 分区之后, 选择一个分割代表树的内部节点从而自上而下地构建树. 这些方法的区别在于选择分割的策略不同: IMM 算法选择错误点数最少的分割; ExGreedy 算法选择分区成本最少的分割; ExShallow 算法则是在 ExGreedy 算法的基础上考虑了树的深度. 本文中涉及的算法的聚类数量 k 均采用与数据集相关联的分类数. 根据文献 [17] 所给建议, ExShallow 算法中用来权衡分区成本和树的深度的参数 λ 设置为 0.03. 接下来考虑 BSIMM 算法中集束宽 B 和节点考虑的分割数量 C 的设置. 首先, 当 B 越大时, BSIMM 算法考虑了更多种解决方案, 可能会产生更好的结果, 但同时也会伴随着更长的运行时间. 而 C 的大小也会影响到具体分割的选择. 经过性能权衡以及实验比较, BSIMM 算法中集束宽 B 设置为 40, 节点考虑的分割数量 C 设置为 10. 实验结果如表 2 所示.

表 1 数据集信息

数据集	实例个数	特征个数	聚类数
Robot	5 456	24	4
Pendigits	10 992	16	10
Anuran	7 195	22	10
Vowel	990	10	11
Waveform	5 000	40	3
Mice	1 077	69	8
Digits	1 797	64	10
Avila	20 867	10	12
Yeast	1 484	8	10
Ecoli	336	7	8

根据表 2, 我们可以得到集束宽 B 为 40 的 BSIMM 算法、IMM 算法、ExGreedy 算法以及 ExShallow 算法关于 NPC 和 NMI 的对比实验结果. 其中, 我们可以看到 BSIMM 算法在大部分数据集中的 NMI 都优于其余 3 个算法, 意味着 BSIMM 算法在聚类任务当中效果显著. 在 Vowel 和 Avila 数据集上, 尽管 BSIMM 算法的 NMI 最高, 但是 ExShallow 算法的 NPC 比 BSIMM 算法更低, 这是由于 ExShallow 算法在选择分割时考虑到了分区成本而忽略了预测标签与参考簇标签的吻合程度. 结果表明, 相较于其他 3 种算法在阈值树的每一轮划分期间只考虑一个分割, 本文所提出的 BSIMM 算法则是通过根据启发式值存储预定数量的状态来拓展搜索范围, 考虑了更多在之后划分中表现良好的当前次优状态, 有效地减缓了局部最优问题. 总而言之, 该实验证明了在大部分数据集上 BSIMM 算法能够在保证分区成本的情况下提供与初始聚类划分更加一致的聚类结果.

4 结论与展望

针对 IMM 算法所采用的贪心策略容易陷入局部最优的问题, 提出了一种基于集束搜索的可解释阈值树构造算法. 该算法在阈值树的每一轮划分中根据启发式值存储预定数量的状态, 而并非仅是一个状态. 通过保留更多的状态来拓展搜索范围, 获得更好的解决方案, 使得阈值树提供的聚类划分更加贴近初始聚类

划分. 最后, 通过在 UCI 数据集上进行实验, 对比了 IMM 算法、ExGreedy 算法、ExShallow 算法和 BSIMM 算法的标准化分区成本和标准化互信息. 实验结果表明, BSIMM 算法的结果明显优于其余 3 个算法, 验证了 BSIMM 算法的有效性. 未来工作将致力于如何进一步减少贪心策略带来的影响.

表 2 IMM、ExGreedy、ExShallow 和 BSIMM 算法对比

数据集	IMM		ExGreedy		ExShallow		BSIMM	
	NPC	NMI	NPC	NMI	NPC	NMI	NPC	NMI
Robot	1.0758	0.5322	1.0634	0.5214	1.0634	0.5214	**1.0621**	**0.5641**
Pendigits	1.2465	0.7187	1.1405	**0.7733**	1.1346	0.7709	**1.1324**	0.7594
Anuran	1.2857	0.6978	1.1533	0.7119	1.1591	0.6959	**1.1504**	**0.7278**
Vowel	1.3534	0.5531	1.2458	0.5715	**1.2138**	0.6203	1.2265	**0.6297**
Waveform	1.1064	0.4645	1.1090	0.4678	1.1090	0.4678	**1.0924**	**0.4756**
Mice	1.1305	0.6465	1.0908	0.6578	1.0864	0.6593	**1.0813**	**0.6721**
Digits	1.2440	0.5364	1.2132	0.5487	1.1883	0.5841	**1.1810**	**0.5969**
Avila	1.0721	0.7331	1.0577	0.7226	**1.0496**	0.7345	1.0623	**0.7448**
Yeast	1.0984	0.6457	1.0762	0.6379	1.0988	0.6227	**1.0711**	**0.6553**
Ecoli	1.0820	0.8436	1.0305	0.8546	1.0390	0.8410	**1.0212**	**0.8640**

参考文献

1 Saxena A, Prasad M, Gupta A, *et al*. A review of clustering techniques and developments. Neurocomputing, 2017, 267: 664–681. [doi: 10.1016/j.neucom.2017.06.053]

2 Moshkovitz M, Dasgupta S, Rashtchian C, *et al*. Explainable *k*-means and *k*-medians clustering. Proceedings of the 37th International Conference on Machine Learning. PMLR, 2020. 7055–7065.

3 Makarychev K, Shan LR. Near-optimal algorithms for explainable K-medians and K-means. Proceedings of the 38th International Conference on Machine Learning. PMLR, 2021. 7358–7367.

4 Esfandiari H, Mirrokni V, Narayanan S. Almost tight approximation algorithms for explainable clustering. Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Alexandria: SIAM, 2022. 2641–2663. [doi: 10.1137/1.9781611977073.103]

5 Jiao LM, Yang HY, Liu ZG, *et al*. Interpretable fuzzy clustering using unsupervised fuzzy decision trees. Information Sciences, 2022, 611: 540–563. [doi: 10.1016/j.ins.2022.08.077]

6 Liu B, Xia YY, Yu PS. Clustering through decision tree construction. Proceedings of the 9th International Conference on Information and Knowledge Management. McLean: ACM, 2000. 20–29. [doi: 10.1145/354756.354775]

7 Fraiman R, Ghattas B, Svarc M. Interpretable clustering using unsupervised binary trees. Advances in Data Analysis and Classification, 2013, 7(2): 125–145. [doi: 10.1007/s11634-013-0129-3]

8 Loh WY. Classification and regression trees. WIREs Data Mining and Knowledge Discovery, 2011, 1(1): 14–23. [doi: 10.1002/widm.8]

9 Ghattas B, Michel P, Boyer L. Clustering nominal data using unsupervised binary decision trees: Comparisons with the state of the art methods. Pattern Recognition, 2017, 67: 177–185. [doi: 10.1016/j.patcog.2017.01.031]

10 Bertsimas D, Orfanoudaki A, Wiberg H. Interpretable clustering: An optimization approach. Machine Learning, 2021, 110(1): 89–138. [doi: 10.1007/s10994-020-05896-2]

11 Bertsimas D, Dunn J. Optimal classification trees. Machine Learning, 2017, 106(7): 1039–1082. [doi: 10.1007/s10994-017-5633-9]

12 Meister CI, Amini A, Vieira T, *et al*. Conditional Poisson stochastic beams. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Punta Cana: ACL, 2021. 664–681. [doi: 10.3929/ethz-b-000518994]

13 Libralesso L, Fontan F. An anytime tree search algorithm for the 2018 ROADEF/EURO challenge glass cutting problem. European Journal of Operational Research, 2021, 291(3): 883–893. [doi: 10.1016/j.ejor.2020.10.050]

14 Basgalupp MP, Barros RC, de Carvalho ACPLF, *et al*. A beam search based decision tree induction algorithm. Machine Learning Algorithms for Problem Solving in Computational Applications: Intelligent Techniques. Hershey: Information Science Reference, 2012. 357–370. [doi: 10.4018/978-1-4666-1833-6.ch020]

15 Zhuo JW, Xu ZR, Dai W, *et al*. Learning optimal tree models under beam search. Proceedings of the 37th International Conference on Machine Learning. PMLR, 2020. 11650–11659.

16 Laber ES, Murtinho L. On the price of explainability for some clustering problems. Proceedings of the 38th International Conference on Machine Learning. PMLR, 2021. 5915–5925.

17 Laber E, Murtinho L, Oliveira F. Shallow decision trees for explainable K-means clustering. Pattern Recognition, 2023, 137: 109239. [doi: 10.1016/j.patcog.2022.109239]

(校对责编: 牛欣悦)