六种常见聚类算法的分析与总结

常添

08/12/2024

1 常见聚类算法

1.1 K-Means 聚类算法

K-Means 是一种基于距离的聚类算法，通过迭代将数据集划分为 K 个簇。算法首先随机选择 K 个初始质心，然后将每个数据点分配到最近的质心，接着更新质心为簇内数据点的均值。这个过程重复进行，直到质心位置稳定或达到最大迭代次数为止。

K-Means 的目标是最小化簇内数据点到其质心的总距离平方和，具体表达为：

$$J = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

其中，x_i 表示数据点，μ_k 为第 k 个簇的质心，C_k 是簇 k 中的所有数据点。通过最小化 J 值，算法不断优化簇的划分。

K-Means 算法的优点在于简单易实现，且在处理大规模数据集时效率较高。然而，它需要预先指定簇的数量 K，且对初始质心选择较为敏感，不同的初始选择可能导致不同的聚类结果。此外，K-Means 更适用于凸形簇，对噪声和异常值的处理能力较弱。

K-Means 的平均时间复杂度为 $O(K \cdot n \cdot T)$，其中 K 为簇数，n 为样本数量，T 为迭代次数。最坏情况下，复杂度为 $O(n^{(K+2)/p})$，其中 p 为特征数量。处理高维数据时，算法的计算开销可能会增加，尤其是在初始质心选择不当的情况下。

1.2 Gaussian Mixture Model (GMM)

Gaussian Mixture Model (GMM) 是一种基于概率模型的聚类算法，它假设数据由多个高斯分布的混合组成。与 K-Means 不同，GMM 通过概率来表示每个数据点属于不同簇的可能性，而不仅仅是将数据点硬性地分配给某个簇。

GMM 使用期望最大化 (Expectation-Maximization, EM) 算法来估计每个高斯分布的参数，包括均值、协方差矩阵和混合系数。EM 算法的主要步骤包括：

- **期望步骤 (E-Step)**: 计算每个数据点属于每个高斯分布的后验概率（责任值）。
- **最大化步骤 (M-Step)**: 使用这些概率更新每个高斯分布的参数，包括均值、协方差矩阵和混合系数。

GMM 的目标是通过以下概率密度函数描述数据：

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

其中：

- K 是高斯分布的数量（即簇数）。
- π_k 是第 k 个高斯分布的权重，满足 $\sum_{k=1}^{K} \pi_k = 1$。
- $\mathcal{N}(x|\mu_k, \Sigma_k)$ 是第 k 个高斯分布的概率密度函数，μ_k 为均值，Σ_k 为协方差矩阵。

GMM 的优点在于其灵活性，可以处理不同形状和大小的簇，并能进行软聚类，即一个数据点可以部分属于多个簇。然而，GMM 的计算复杂度较高，尤其是在高维数据集上。此外，GMM 对初始参数的选择较为敏感，可能会陷入局部最优。

1.3 Affinity Propagation (AP) 聚类算法

Affinity Propagation (AP) 是一种基于消息传递的聚类算法，通过在数据点之间传递“责任”（Responsibility, $r(i,k)$）和“可用性”（Availability, $a(i,k)$）消息，自动确定簇的数量和中心点。算法首先计算数据点之间的相似度 $s(i,k)$，然后初始化责任和可用性矩阵。

AP 的目标是通过不断更新以下公式，使得责任和可用性值达到平衡：

$$r(i,k) = s(i,k) - \max_{k' \neq k} \{a(i,k') + s(i,k')\}$$

$$a(i,k) = \min \left(0, r(k,k) + \sum_{i' \notin \{i,k\}} \max(0, r(i',k)) \right)$$

其中，责任 $r(i,k)$ 表示数据点 i 作为簇中心 k 的适合度，可用性 $a(i,k)$ 表示数据点 i 选择 k 作为簇中心的适合度。最终具有较高“可用性”和“责任值”的点被选择为簇中心。

AP 的优点在于不需要预先指定簇的数量，且能够处理任意形状的簇。缺点在于对相似度度量和偏好值敏感，且计算复杂度较高。AP 算法的时间复杂度为 $O(n^2 \cdot \log(n))$，其中 n 为样本数量。

1.4 Hierarchical Clustering (HC) 聚类算法

层次聚类（Hierarchical Clustering, HC）是一种用于分析数据集内嵌层次结构的聚类算法。HC 通过反复地将数据点进行分割或合并来形成一个树状的簇结构，称为树状图（Dendrogram）。HC 的两种主要方法是自底向上（凝聚法）和自顶向下（分裂法）。

凝聚法（Agglomerative Method）：从每个数据点自身作为一个簇开始，逐步合并最近的簇，直到所有点都被合并到一个簇中。合并步骤通常基于以下几种距离度量：

- 最小距离法（Single Linkage）：两个簇之间的距离定义为它们之间最近点的距离：

$$d_{\text{single}}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \text{dist}(x, y)$$

- 最大距离法（Complete Linkage）：两个簇之间的距离定义为它们之间最远点的距离：

$$d_{\text{complete}}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \text{dist}(x, y)$$

- 平均距离法（Average Linkage）：两个簇之间的距离定义为它们之间所有点的平均距离：

$$d_{\text{average}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} \text{dist}(x, y)$$

- 质心法（Centroid Method）：两个簇之间的距离定义为它们质心之间的距离：

$$d_{\text{centroid}}(C_i, C_j) = \text{dist}(\mu_i, \mu_j)$$

Ward 法：Ward 法通过最小化每次合并后簇内方差的增加来决定簇的合并顺序：

$$d_{\text{ward}}(C_i, C_j) = \frac{|C_k|}{|C_k| + |C_j|} \|\mu_k - \mu\|^2 + \frac{|C_j|}{|C_k| + |C_j|} \|\mu_j - \mu\|^2$$

层次聚类的结果通常以树状图的形式展示，树状图展示了数据点合并或分裂的过程。通过剪切树状图可以得到不同数量的簇。层次聚类的优点在于它不需要预先指定簇的数量，并且能够生成一个多层次的聚类结果。缺点在于算法的计算复杂度较高，特别是在处理大规模数据集时。

HC 算法的时间复杂度通常为 $O(n^2 \log n)$，其中 n 为样本数量。在某些情况下，复杂度可以达到 $O(n^3)$。

1.5 OPTICS 聚类算法

OPTICS（Ordering Points To Identify the Clustering Structure）是一种基于密度的聚类算法，用于识别任意形状和密度变化的簇。OPTICS 算法通过计算每个数据点的可达距离（Reachability Distance）和核心距离（Core Distance），并按照可达距离对数据点排序，从而构建簇的层次结构。首先，算法计算每个数据点的核心距离，即它到其邻域内满足最小点数要求的最远距离。然后，计算数据点之间的可达距离，即到达某个点所需的最小核心距离。根据可达距离对数据点进行排序，生成簇的层次结构。

核心距离的计算公式为：

$$\text{core\_dist}(p) = \min_{p' \in \text{Neighbors}(p)} \text{dist}(p, p')$$

可达距离的计算公式为：

$$\text{reachability\_dist}(o, p) = \max(\text{core\_dist}(p), \text{dist}(o, p))$$

其中，$\text{dist}(p, p')$ 是数据点 p 和 p' 之间的距离，$\text{Neighbors}(p)$ 是点 p 的邻域。

OPTICS 算法的优点在于它能够处理任意形状和密度的簇，并且无需预先指定簇的数量。缺点是算法在处理大规模数据集时计算复杂度较高，特别是在高维空间中。OPTICS 算法的时间复杂度为 $O(n \log n)$，其中 n 为样本数量。虽然与 DBSCAN 类似，但 OPTICS 能更好地处理具有密度变化的复杂数据集。

1.6 BIRCH 聚类算法

BIRCH（Balanced Iterative Reducing and Clustering using Hierarchies）是一种适用于大规模数据集的聚类算法。BIRCH 算法通过构建和维护一个簇特征树（Clustering Feature Tree, CF Tree）来有效地对大规模数据进行聚类。CF Tree 是一种高度压缩的树状数据结构，能够通过增量式学习动态地调整簇。

BIRCH 算法的核心在于簇特征（Clustering Feature, CF）的计算和使用。簇特征 CF 是一个三元组 $(N, \vec{LS}, SS)$，用于有效地描述簇中的数据点。具体计算如下：

$$\text{CF} = (N, \vec{LS}, SS)$$

其中，N 是簇中的点数，$\vec{LS}$ 是簇内所有数据点的线性和：

$$\vec{LS} = \sum_{i=1}^{N} \vec{x}_i$$

SS 是簇内所有数据点的平方和：

$$SS = \sum_{i=1}^{N} \|\vec{x}_i\|^2$$

通过簇特征，簇的质心和半径可以计算为：

$$\vec{C} = \frac{\vec{LS}}{N}, \quad \text{Radius} = \sqrt{\frac{SS}{N} - \left(\frac{\vec{LS}}{N}\right)^2}$$

两个簇 CF_1 和 CF_2 之间的距离可以通过以下公式计算：

$$D(CF_1, CF_2) = \sqrt{\frac{N_1 \cdot N_2}{N_1 + N_2}} \cdot \|\vec{C}_1 - \vec{C}_2\|$$

BIRCH 算法的主要步骤如下：

- 构建 CF Tree：通过遍历数据集，BIRCH 算法将每个数据点插入到 CF Tree 中，并根据簇特征进行调整。
- 聚类阶段：在 CF Tree 构建完成后，BIRCH 可以使用凝聚层次聚类或其他算法对叶节点进行进一步的聚类，以生成最终的簇。

BIRCH 的优点在于它能够有效地处理大规模数据，并且能够在内存受限的情况下进行聚类。缺点是对于非球形簇的识别能力有限，并且可能对簇的初始构建顺序较为敏感。

BIRCH 算法的时间复杂度通常为 $O(n)$，其中 n 为样本数量，适合大规模数据集的聚类任务。