lec12.2 k-means++

选择初始点

选择初始质心对k-means算法的效果影响很大。以下是几种常见的方法：

1. 随机选择：直接随机选择k个数据点作为初始质心。这种方法简单但效果不稳定，可能会陷入局部最优解
2. 多次随机选择：重复随机选择初始质心多次，选择效果最好的结果
3. 采样加层次聚类：从数据集中抽样，然后使用层次聚类得到k个初始质心
4. 选择最远点：选择彼此距离最远的点作为初始质心，减少相似数据点被错误分配到同一个质心的问题
5. k-means++：一种改进的方法，通过更巧妙的选择初始质心，减少k-means算法的收敛时间和提高聚类效果

Sampling + Hierarchical Clustering 采样和层次聚类

1. 从数据集中抽样：从整个数据集 $\mathscr{D}$ 中抽取一个子集 $\mathscr{D}'$
2. 对子集进行层次聚类：使用层次聚类技术对子集 $\mathscr{D}'$ 进行聚类
3. 提取k个聚类：从层次聚类结果中提取出 k 个聚类
4. 计算这些聚类的均值：计算这 k 个聚类的均值，并将它们用作初始聚类代表
5. 使用这些聚类代表进行标准的k-means算法：用这些初始聚类代表作为初始质心，继续进行标准的k-means算法

这种方法的实用性条件

这种方法通常效果很好，但仅在以下条件下具有实用性：

1. 抽样的子集 $\mathscr{D}'$ 相对较小：因为层次聚类的计算成本较高，因此子集 $\mathscr{D}'$ 的大小应保持在几百到几千之间
2. k 值相对于子集 $\mathscr{D}'$ 的大小较小：k 值应该相对较小，以便于从子集中有效地提取出代表性的聚类

方法优势

这种方法结合了层次聚类和k-means的优势，通过层次聚类的初步聚类结果来选择更合理的初始质心，避免了传统k-means算法初始质心选择随机性带来的局部最优问题，同时也能提高聚类效果和计算效率

方法限制

由于层次聚类的计算复杂度较高，因此这种方法仅适用于较小的子集，在处理大规模数据集时需要注意计算资源的限制

Selecting Furthest Points 选择最远点

1. 随机选择第一个代表点：从数据集 $\mathscr{D}$ 中随机选择一个代表点 Y_1
2. 选择剩余的代表点：对于每一个 $i = 2, \ldots, k$：从数据集 $\mathscr{D}$ 中选择一个点作为代表点 Y_i，该点到已选代表点的距离 $R(\cdot)$ 最大

3. 使用这些初始代表点进行标准的 k-means 算法：使用 $Y-1,\ldots,Y_k$ 作为初始聚类代表，继续进行标准的 k-means 算法

主要缺点

该方法可能会选择离群点（outliers）作为代表点，而不是聚类中的点。这是因为离群点可能会距离其他已选择的代表点最远，从而被选择为新的代表点，这样会影响最终聚类的效果

方法概述

这种选择最远点的方法旨在通过选择距离已经选择的代表点最远的点来覆盖整个数据空间。然而，由于可能会选择离群点，所以这种方法并不总是能得到理想的初始代表点，从而影响最终的聚类质量

k-means++

1. 随机选择第一个代表点 Y_1：从数据集 $\mathscr{D}$ 中均匀随机选择一个代表点 Y_1
2. 选择剩余的代表点：对于每一个 $i=2,\ldots,k$：
    1. 计算数据集中每个点到当前已选择代表点集合的距离 $R(X)$
    2. 从数据集 $\mathscr{D}$ 中选择一个点作为代表点 Y_i，其被选择的概率为：

$$\frac{R(X)^2}{\sum_{X\in\mathscr{D}}R(X)^2}$$

3. 使用这些初始代表点进行标准的k-means算法：使用 $Y_1,\ldots,Y_k$ 作为初始聚类代表，继续进行标准的k-means算法

理论保证

聚类内平方和（Within-cluster sum of squares）：定义为：

$$\phi=\sum_{x\in X}\min_{c\in C}\|x-c\|^2$$

> *定理 3.1：如果聚类结果是通过 k-means++ 构造的，则相应的潜在函数 ϕ 满足：*
>
> $$E[\phi]\leq 8(\ln k+2)\phi_{\mathrm{OPT}}$$
>
> *其中，ϕ_{OPT} 表示最优聚类结果的聚类内平方和，$E[\phi]$ 表示通过 k-means++ 初始化的 k-means 算法的聚类内平方和的期望值*

解释

- 随机性和概率选择：通过随机选择第一个点，并根据与现有代表点距离的平方比例选择后续点，k-means++能够更好地分布初始中心点，从而减少初始点选择带来的不确定性和偏差
- 理论保证：k-means++的理论保证意味着其期望聚类结果会接近最优结果的一个常数倍，这显著提升了聚类质量和稳定性

k-means++ 算法和传统 k-means 算法在数据集上的表现

合成数据集

- 合成数据集：使用了两个合成数据集，分别称为 Norm-10 和 Norm-25
    - Norm-10：选择了 10 个“真实”中心，这些中心是从边长为 500 的超立方体中均匀随机选取的

    - Norm-25：选择了 25 个“真实”中心，同样是从边长为 500 的超立方体中均匀随机选取的
- 数据生成：
    - 在每个“真实”中心点周围生成数据点，这些点是从方差为 1 的高斯分布中采样的，中心位置即为“真实”中心
    - 这样，就得到了多个相互分离的高斯分布，且这些高斯分布的真实中心点提供了一个较好的最优聚类近似

- Synthetic datasets Norm-10.

k	Average ϕ		Minimum ϕ	
	k-means	k-means++	k-means	k-means++
10	10898	5.122	2526.9	5.122
25	787.992	4.46809	4.40205	4.41158
50	3.47662	3.35897	3.40053	3.26072

Table 1: Experimental results on the *Norm-10* dataset (n = 10000, d = 5)

- Synthetic datasets Norm-25.

k	Average ϕ		Minimum ϕ	
	k-means	k-means++	k-means	k-means++
10	135512	126433	119201	111611
25	48050.5	15.8313	25734.6	15.8313
50	5466.02	14.76	14.79	14.73

Table 2: Experimental results on the *Norm-25* dataset (n = 10000, d = 15)

实验结果对比

k-means算法在合成数据集上的表现不佳，原因如下：

1. 随机初始中心选择的问题：
    - 由于 k-means 算法在选择初始中心时采用随机方法，这会导致聚类中心初始位置较差，可能会将一些实际不属于同一类的点归为同一类
    - 随机初始中心选择可能会使得聚类结果出现聚类间重叠的情况，算法无法将它们正确地分开
2. 聚类中心重叠：
   - 由于初始中心选择的不合理，随机种子方法不可避免地会将一些本应分开的簇合并在一起
   - 一旦这种情况发生，k-means 算法将无法将这些簇正确地分离开

   k-means++算法的优势：
3. 更精细的初始中心选择：
    - k-means++ 算法通过更谨慎地选择初始聚类中心，避免了 k-means 算法的上述问题
    - 这种方法确保了初始中心更加分散，减少了聚类中心重叠的可能性
4. 接近最优结果：
    - 通过这种优化的初始中心选择方法，k-means++ 算法几乎总能在合成数据集上达到最优结果
    - 这使得 k-means++ 在处理类似的聚类任务时表现更好，更可靠

真实数据集

- Real datasets: Cloud dataset

k	Average ϕ		Minimum ϕ	
	k-means	k-means++	k-means	k-means++
10	7553.5	6151.2	6139.45	5631.99
25	3626.1	2064.9	2568.2	1988.76
50	2004.2	1133.7	1344	1088

Table 3: Experimental results on the *Cloud* dataset (n = 1024, d = 10)

- Real datasets: Intrusion dataset

k	Average ϕ		Minimum ϕ	
	k-means	k-means++	k-means	k-means++
10	$3.45 \cdot 10^8$	$2.31 \cdot 10^7$	$3.25 \cdot 10^8$	$1.79 \cdot 10^7$
25	$3.15 \cdot 10^8$	$2.53 \cdot 10^6$	$3.1 \cdot 10^8$	$2.06 \cdot 10^6$
50	$3.08 \cdot 10^8$	$4.67 \cdot 10^5$	$3.08 \cdot 10^8$	$3.98 \cdot 10^5$

Table 4: Experimental results on the *Intrusion* dataset (n = 494019, d = 35)

实验结果对比

1. Cloud数据集：
   - 速度：k-means++ 算法收敛速度几乎是传统 k-means 算法的两倍
   - 效果：在潜在函数（簇内平方和）值方面，k-means++ 算法比 k-means 算法好约 20%
2. Intrusion数据集：
   - 效果：在较大的 Intrusion 数据集中，k-means++ 算法获得的潜在值比 k-means 算法高出 10 到 1000 倍
   - 速度：k-means++ 算法的计算速度提高了多达 70%