KMeans算法及其修改

1.K-Means算法原理

K-Means算法的基本思想是：将N个对象划分到k个簇中，分类结果要使得相似度较高的对象划分到同一类簇中，而差异较大的对象存在于不同类簇中。

给定大小为n的数据集，设V={,,…，}，令I=1，将n个对象划分到K个不同的簇中，使用K-Means算法聚类的具体算法步骤为：

步骤1 在数据集中随机选取K个对象作为初始聚类中心 ,……；

步骤2 计算数据集中每个对象到聚类中心的距离，选取最小距离min|V- |,分配到聚类中,其中V={v1,v2,…，vn}，j=1,2……k；

步骤3 计算每个聚类中的所有对象均值，将此均值作为新的聚类中心，=

，为第j类中对象的个数，j=1,2，……k；

步骤4 如果每个簇的聚类中心不再发生变化，聚类准则函数=收敛，则算法结束。否则返回步骤2继续迭代。

### 2.优缺点

K-Means算法实现起来比较简单、运算速度较快，算法效率较高，能够处理大型数据集，空间复杂度和时间复杂度较低。但同时K-Means算法也有不足之处，包含以下几点：

(1) 聚类结果不确定

K-Means算法初始聚类中心是随机选择的，初始中心点的选择不同会导致最终聚类的效果不同。选取不同的初始聚类中心，会使得最终聚类得到的类簇发生变化。除此之外，K-Means算法一般采用准则函数为目标函数，准则函数中只存在一个全局最小值和N个极小值，这使得在进行计算时，使得算法陷入局部最小的情况，导致最终得到的不是全局最优解。

(2) 聚类个数不确定

K-Means算法中K表示聚类后簇的个数，K的取值决定着聚类的结果。K值的选取需要根据实际的需要来确定，但是通常情况下我们是不知道将原始数据集分为多少个类簇是合适的，所以需要针对不同的实验通过对比选取恰当的K值。

(3) 数据量大、算法时间复杂度较高

K-Means算法的计算过程是一个不断迭代的过程，为了寻找到合适的聚类中心，需要不断的计算和调整才能对数据对象进行有效的分类。这个过程中反复进行大量的对象间距离的计算，所以K-Means聚类算法过程会消耗大量的时间，降低聚类的效率。

3.改进点1：（融合k中心点算法）

普通kmeans算法是求出的聚簇之后对簇内的点取平均值，若这个簇形成的图像比较狭窄或其图像并不均匀，则使用平均值生成的点就极有可能偏离聚簇本身，所以本文希望通过求生成的簇的重心，以保证生成的中心点不会偏离聚簇本身并能够更好的代表整个簇。

4.改进点2:

普通kmeans算法在初始化K个中心点时使用数据集前K个点作为中心点或使用默认的随机化方法初始化中心点，本文采用高斯随机化方法从数据集中取K个点作为中心点。

5改进点3：（融合最小生成树）

由于kmeans选取的聚类中心是随机的，这使得算法容易陷入局部最优，若从不同的初始聚类中心出发，会得到不同的聚类结果。若使用Kruskal 最小生成树算法形成合理的初始聚类中心，再进行k-means 聚类的迭代步骤，从而能够进行有效的聚类。

最小生成树k-means 算法：

首先根据Kruskal 算法求此图的最小生成树( MST) ，然后按权值从小到大逐个删除最小生成树中的k － 1 条边，得到k 个互不相通的连通子图，接着分别计算出这k 个连通子图中所有对象的平均值，将其作为初始聚类中心，最后釆用原始的k-means算法做聚类分析。

算法具体步骤：

步骤1：

在有权重连通图中，首先根据最小生成树算法，如Kruskal 算法，求出图的最小生成树，并将边值按照从小到大的顺序排列，此时的最小生成树包含了n 个结点和n － 1 条边。

步骤2：

然后按照边的权值由小到大排列形成数据集E = { x1,x2,x3,…,xn}，对于距离最近的2 个数据点xi、xj，计算它们之间的中心点dij = Mid( xi，xj) 。

步骤3:

将dij替换xi、xj放入余下数据集中形成E = { x1,x2,…,dij，…,xn},重复步骤2，直到集合E中剩余k个结点为止。集合E中的元素即为初始聚类中心Centerj( I) ，j = 1，2，…，k，I 为循环次数。

步骤4:

计算每个数据对象与聚类中心的距离D( xi,Centerj( ) ) ，如果满足D( xi,Centerj( I) ) = min{ D( xj,Centerj( I) ) } ，则xi∈Ck。

步骤5:

重新计算k 个新的聚类中心Centerj( I+ 1) = (1/nj)\*Σnji = 1xji，j = 1,2,…,k。

步骤6:

判断Centerj( I + 1) ≠Centerj( I) ,j = 1,2,…,k 是否成立，若成立，则I = I +1，返回步骤3;否则算法结束。

6.实验效果：

|  |  |  |
| --- | --- | --- |
|  | 原始K-means算法求中心点（使用平均值） | 加入重心的K-means算法（即改进点1） |
| 初始化使用默认取数据集中前K个点信息 | 53.47% | 72.47% |
| 初始化使用默认的随机化方法 | 70.225% | 71.9% |
| 初始化使用高斯随机化方法（即改进点2） | 70.225% | 73．03% |

注：程序中

initCenterPointByFirstK(OriginalData means[])方法：

初始化使用默认取数据集中前K个点信息

initCenterPointByDefaultRandom(OriginalData means[])方法：

初始化使用默认的随机化方法

initCenterPointByGaussianRandom(OriginalData means[])方法：

初始化使用高斯随机化方法（即改进点2）

initCenterPointByMTK(OriginalData means[])方法：

使用Kruskal的最小生成时方法生成初始中心点

getMeans\_ByAvg(ArrayList<OriginalData> cluster,ArrayList<Double> dist, Map<Double, Integer> loc)方法：

原始K-means算法求中心点（使用平均值）

getMeans\_ByCore(ArrayList<OriginalData> cluster,ArrayList<Double> dist, Map<Double, Integer> loc)：

加入重心的K-means算法（即改进点1）