人工智能基础算法 第四次作业

1

实现了K均值聚类算法`task1.py`，其中K作为用户可调参数。代码进行K均值聚类的核心函数为`k_means`，函数返回质心节点向量`centroids`、质心对应的样本在数据集中的序号`labels`和费用函数值`cost`，在主函数中进行储存，之后进行输出和可视化。

2

实验K=5，10，20，30，40，50六种情形下聚类结果，在每种情形下，随机选取初始化簇中心5次（为了让结果可重复，设置了随机数种子）。报告每次实验的K均值聚类费用函数值，计算时间，并可视化每个簇中心。

各次实验的费用函数值、计算时间如下表所示

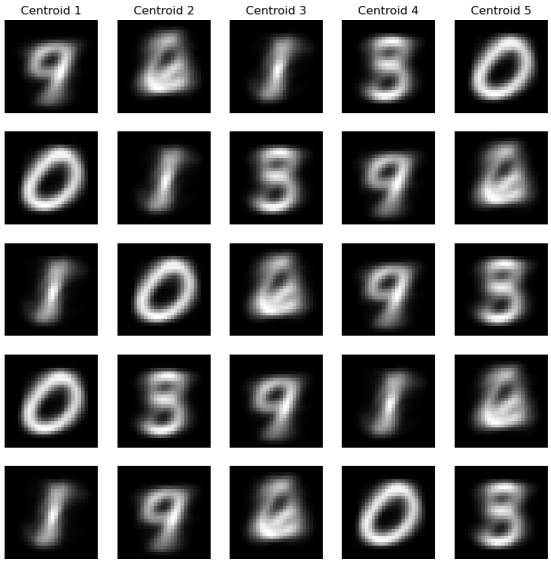
K	初始化次数	费用函数值	计算时间/s
5	1	19524.2	0.424
	2	19520.0	0.690
	3	19521.2	0.318
	4	19524.1	0.272
	5	19520.1	0.741
10	1	18442.2	0.854
	2	18433.4	0.786
	3	18546.7	0.765
	4	18541.8	0.592
	5	18536.6	1.014
20	1	17475.0	2.061
	2	17636.6	2.233
	3	17457.9	1.882
	4	17449.1	2.349
	5	17465.6	1.494
30	1	16842.8	3.429
	2	16912.6	3.958
	3	16970.2	2.110
	4	16850.6	4.281
	5	16830.0	3.379
40	1	16460.6	5.368
	2	16445.6	2.623
	3	16518.4	3.212
	4	16462.6	4.510
	5	16525.4	4.786
50	1	16218.5	7.738
	2	16241.4	3.213
	3	16183.5	3.130
	4	16187.1	3.126
	5	16169.7	3.432

六种情形下费用函数和计算时间的平均值为

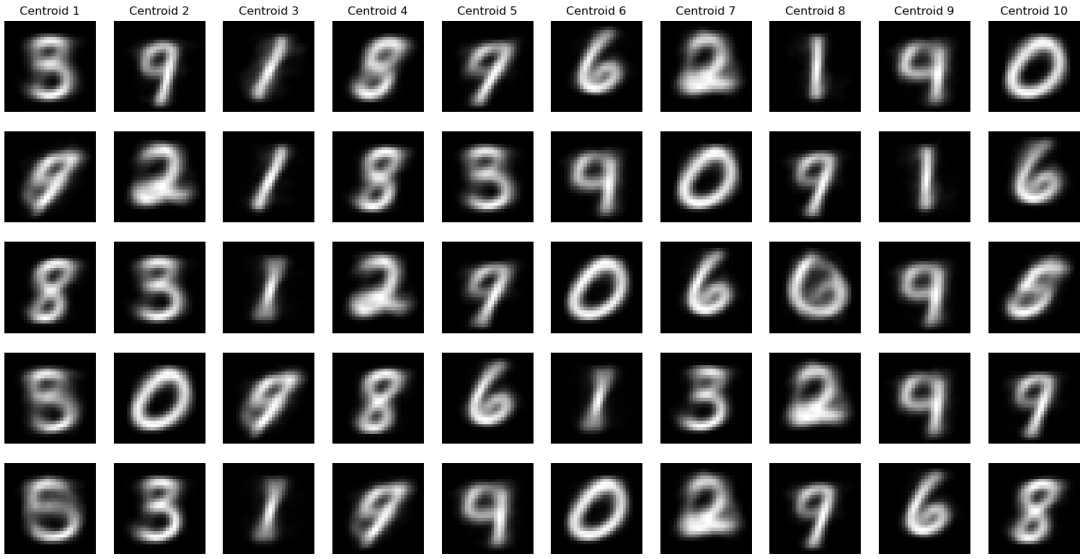
K	费用函数值	计算时间/s
5	19521.9	0.489
10	18500.2	0.802
20	17496.8	2.004
30	16881.2	3.431
40	16482.5	4.100
50	16200.0	4.128

各次实验得到的簇中心依次为

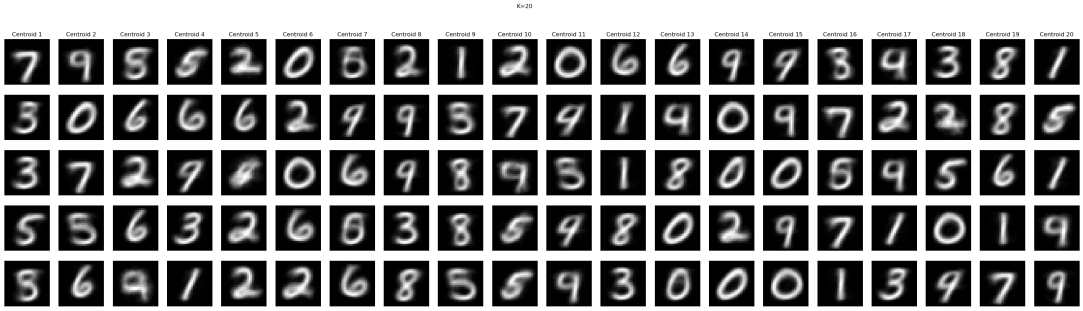
K=5

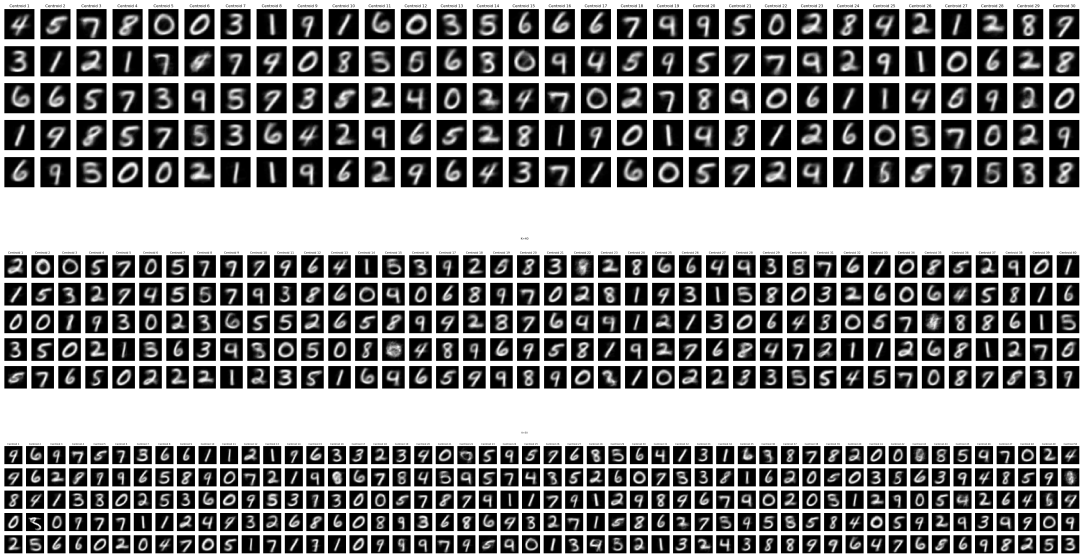

Centroid 1
Centroid 2
Centroid 3
Centroid 4
Centroid 5

K=10


Centroid 1
Centroid 2
Centroid 3
Centroid 4
Centroid 5
Centroid 6
Centroid 7
Centroid 8
Centroid 9
Centroid 10

K=20

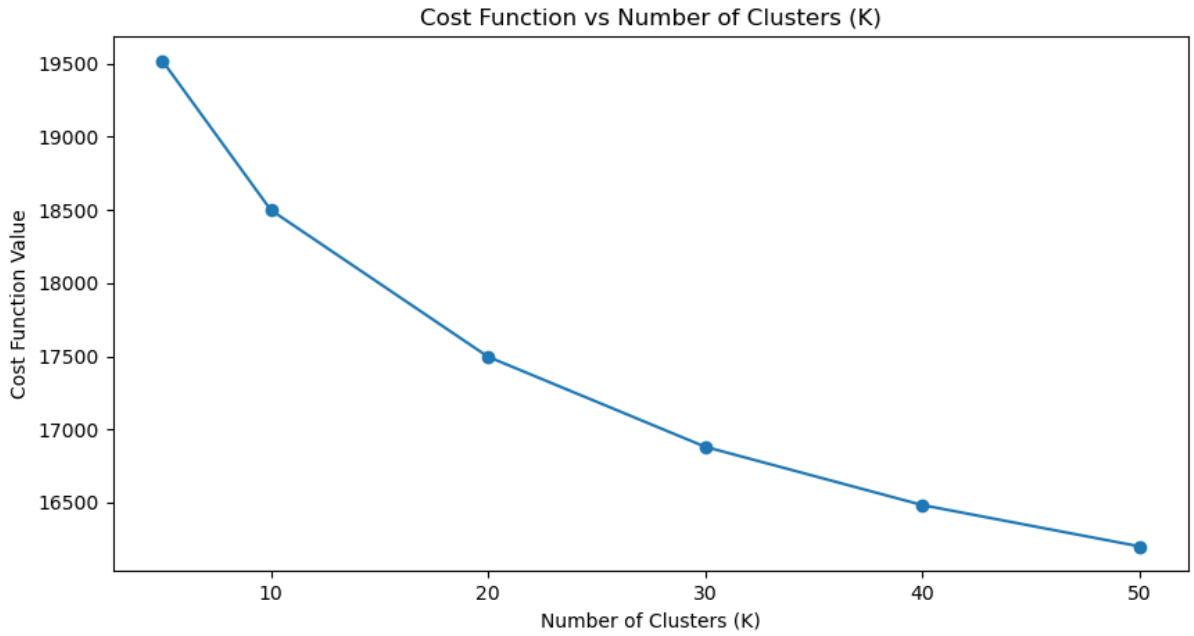

Centroid 1
Centroid 2
Centroid 3
Centroid 4
Centroid 5
Centroid 6
Centroid 7
Centroid 8
Centroid 9
Centroid 10
Centroid 11
Centroid 12
Centroid 13
Centroid 14
Centroid 15
Centroid 16
Centroid 17
Centroid 18
Centroid 19
Centroid 20



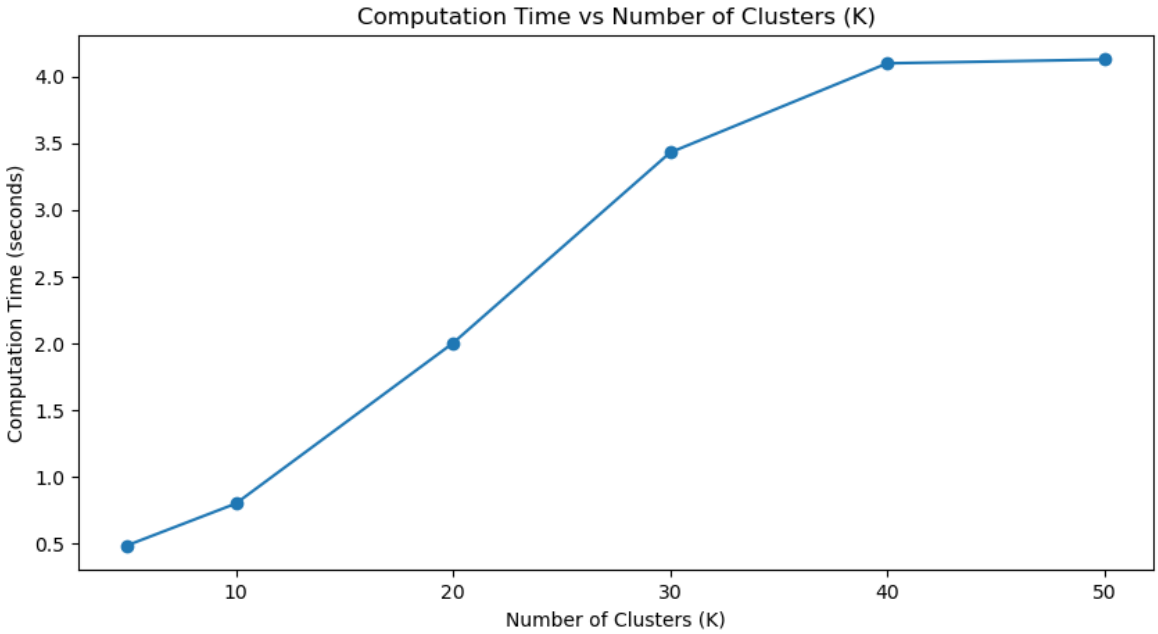
3

为了评估如何选取最佳K值，首先分别做出费用函数、计算时间的均值与K值的关系曲线

平均费用函数值与K值的关系为



平均计算时间与K值的关系为

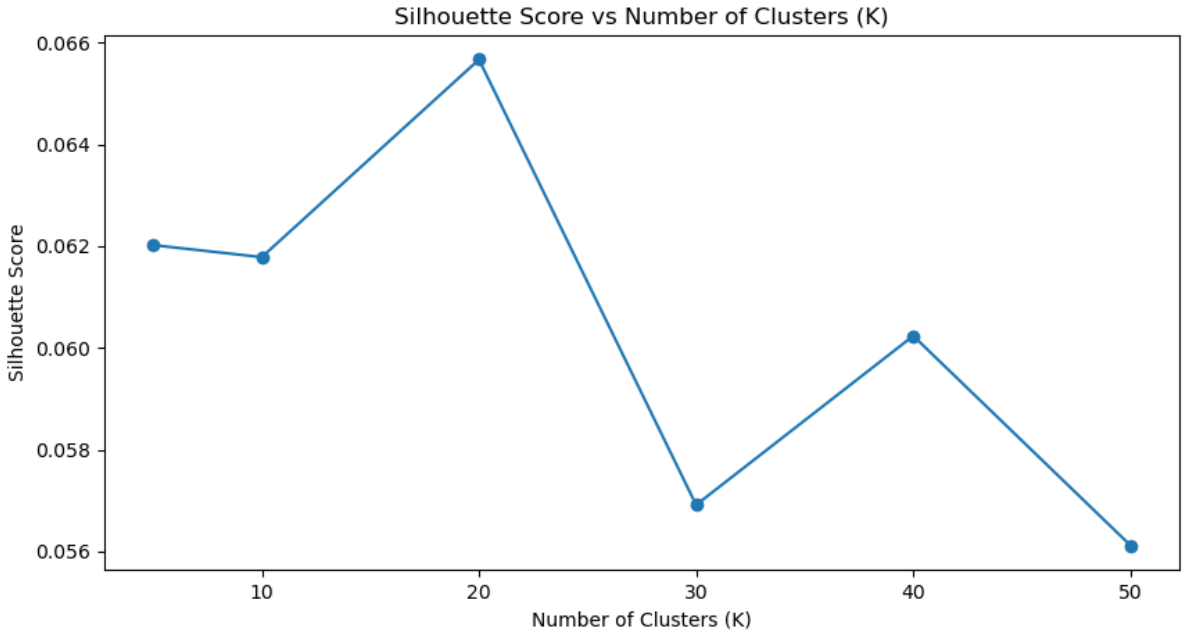


费用函数计算的是各样本到其所属簇中心的距离之和，一般来说，它显然随着K的增大而减小，且在不同K值之间，费用函数越小也不能说明聚类效果更好，该对比只在相同、相近的K值之间才有意义。因而对于上述结果，我们应该选择费用函数随K值变化的较明显的拐点，同时平衡计算时间，作为最佳K值。

按照上述原则进行判定，可以发现损失函数在K值在10～20处下降趋势有比较明显的变缓情况，且计算时间在该段上升非常明显，因此推测选择K=10或20为比较好的均值。但同时，上述结果仅为简单的定性推测，且特征不够明显，因而需要引入其他的评判方式进行判定。

引入轮廓系数

轮廓系数是一种评估聚类效果的方法，它可以衡量每个样本与其所属聚类的相似度，以及与其他聚类的不相似度。轮廓系数的取值范围在-1到1之间，值越接近1表示聚类效果越好。本例中，对每次计算的结果计算轮廓系数，并做出平均轮廓系数关于K值的关系图



可以看到，K=20的轮廓系数最大，因而选择K=20为最佳K值，该结果也基本符合上述推测。

4

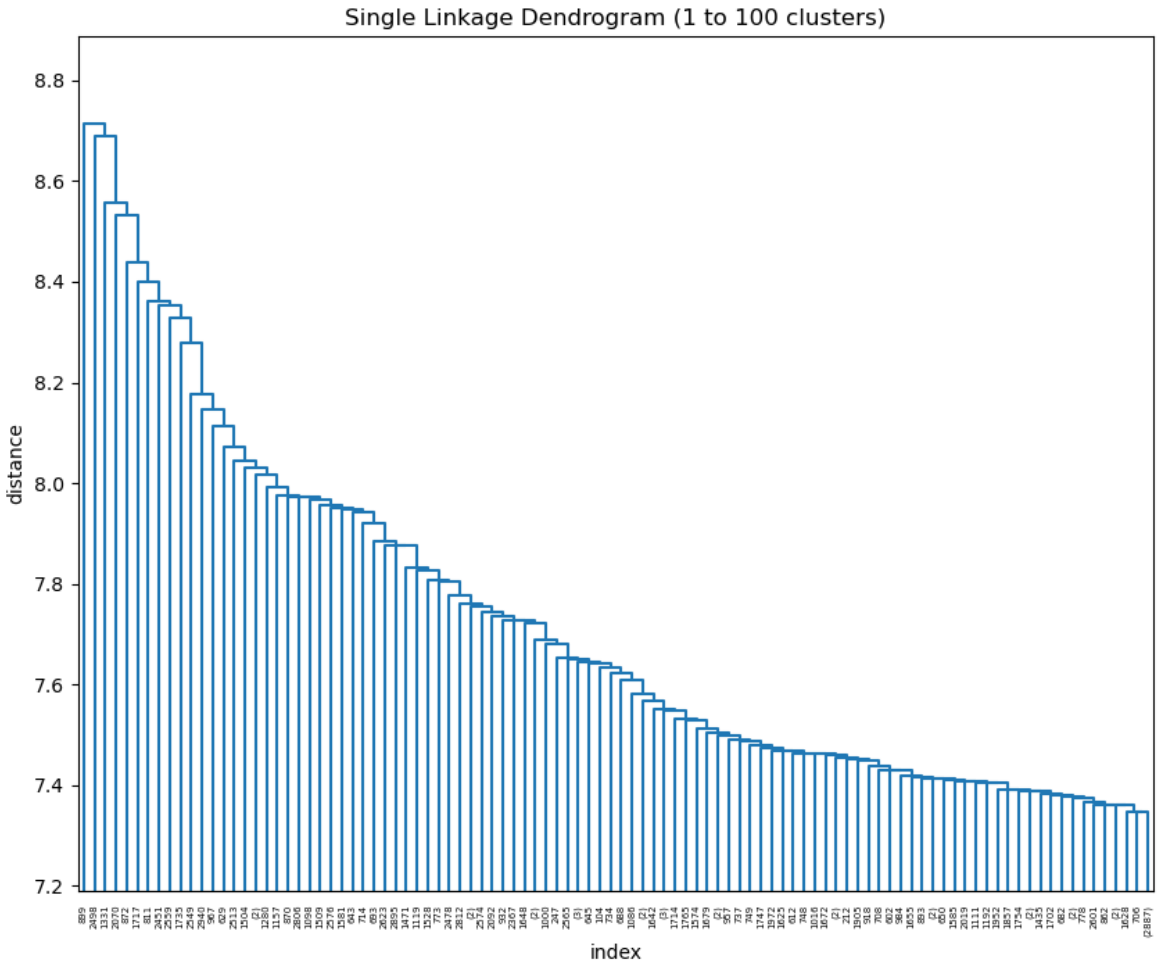
实现了分级聚类算法`task2.py`，并以三种簇与簇之间距离定义（Single Link，Complete Link和Average Link）作为用户可选项。程序中进行分级聚类的核心函数为`hierarchical_clustering`，函数首先将每个样本单独作为一个簇，计算出各个簇之间的距离，储存在一个二维矩阵中，之后进行簇合并，每次删除原有的两个簇，并添加一个新簇，再更新距离矩阵，同时用一个伴随的数组来记录每个簇的序号，用于后续绘制树状图。函数将记录合并过程的数组`linkage_info`返回，用于绘制树状图。

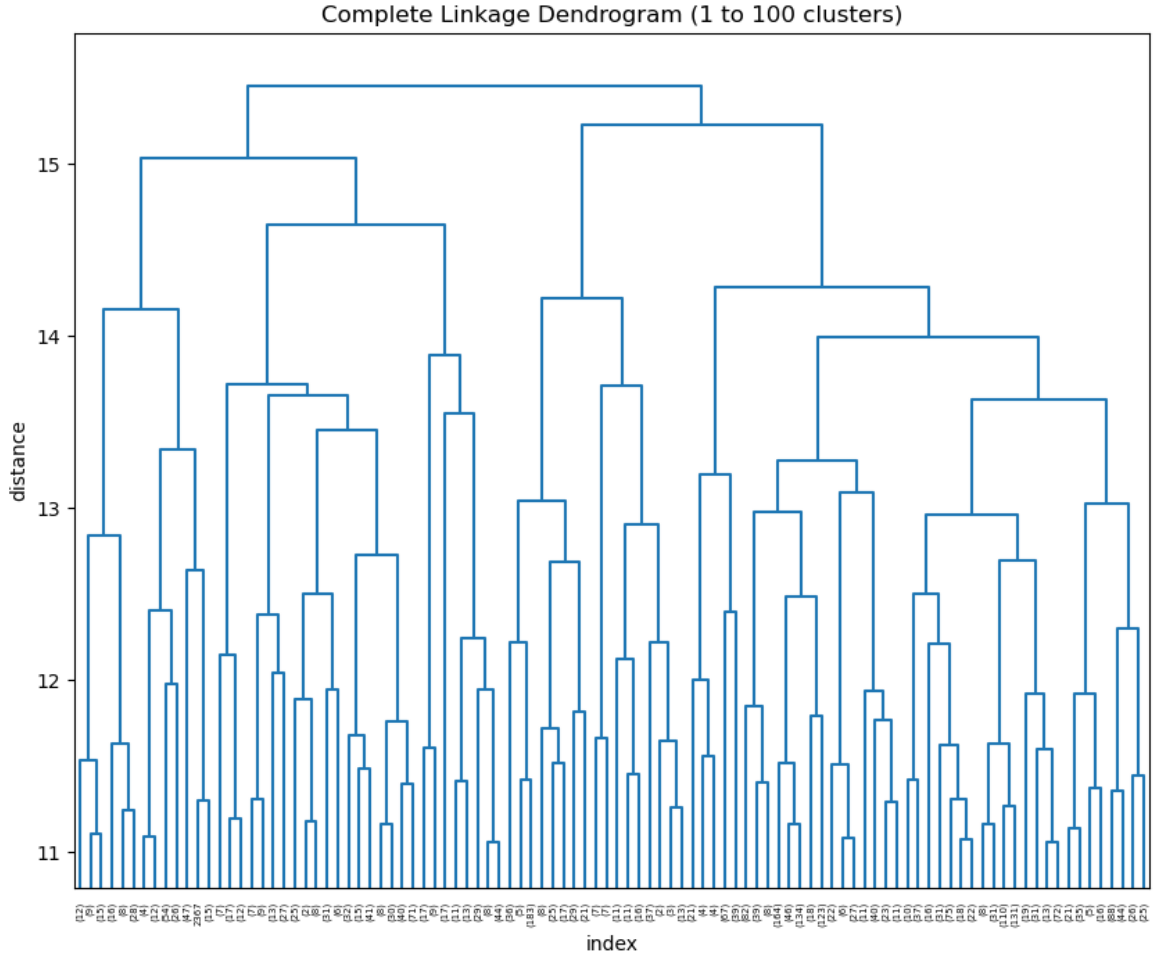
5

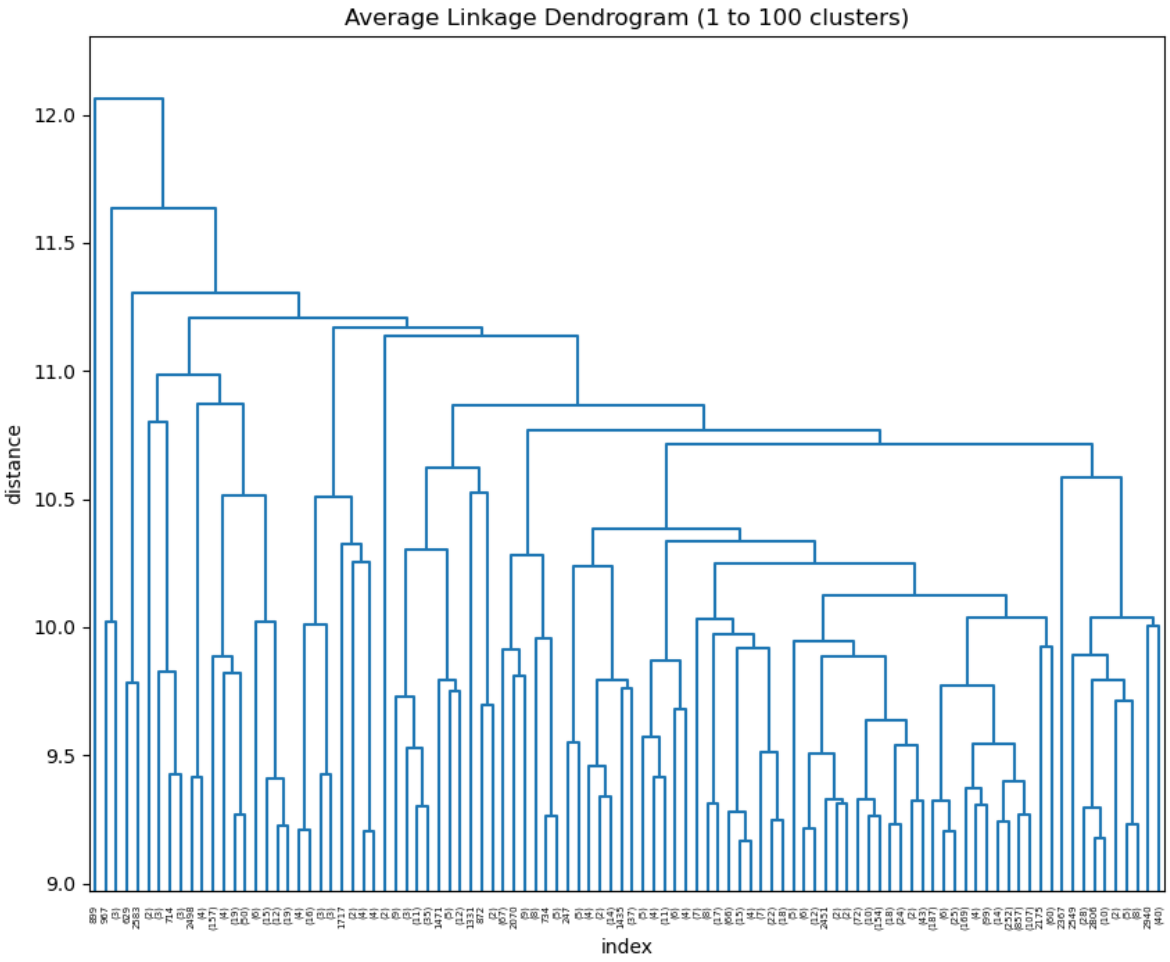
在给定规模为3000的数据集上依次进行了Single Link，Complete Link和Average Link的计算，计算时间如下所示

	Single Link	Complete Link	Average Link
计算时间/s	6220.41	6741.62	7471.46

三种距离定义对应的树状图（只画出从1簇到100簇）依次为

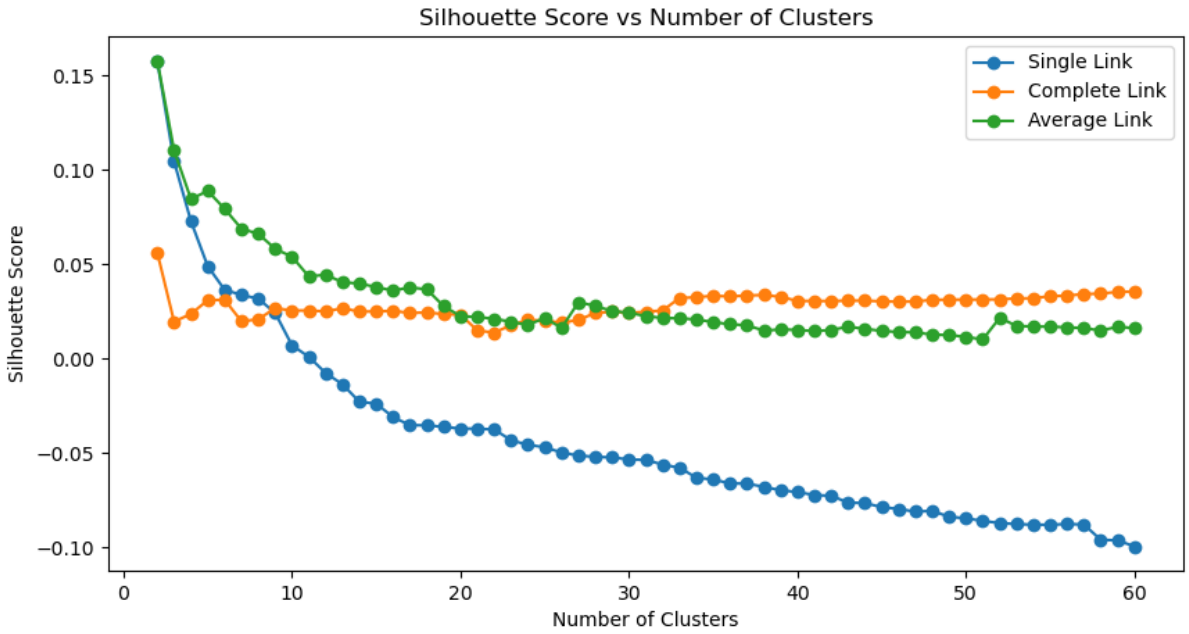



Complete Linkage Dendrogram (1 to 100 clusters)
distance
15
14
13
12
11
index

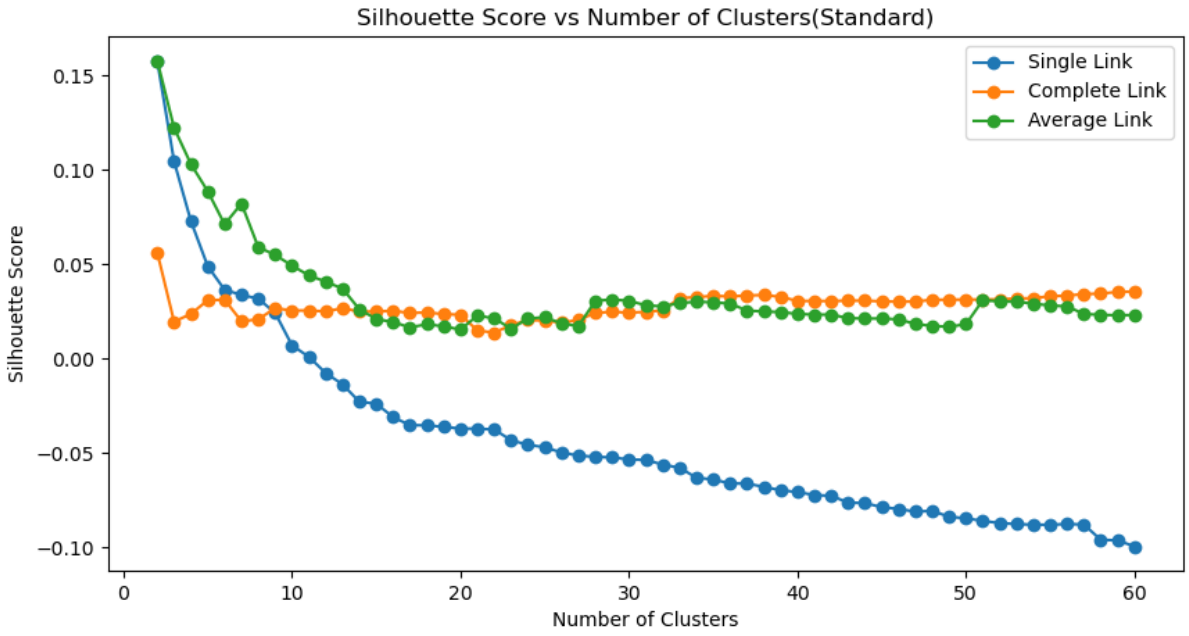


6

为了比较不同方法、不同层数的分级聚类效果，计算了相应的轮廓系数，并绘制成曲线图，如图所示



为了验证该聚类过程的正确性，另直接通过`scipy`中的分级聚类函数进行计算，并计算轮廓函数，绘制成曲线图如下所示，所得到的曲线图与本例的结果基本相同，验证了本例分级聚类计算过程的正确性。

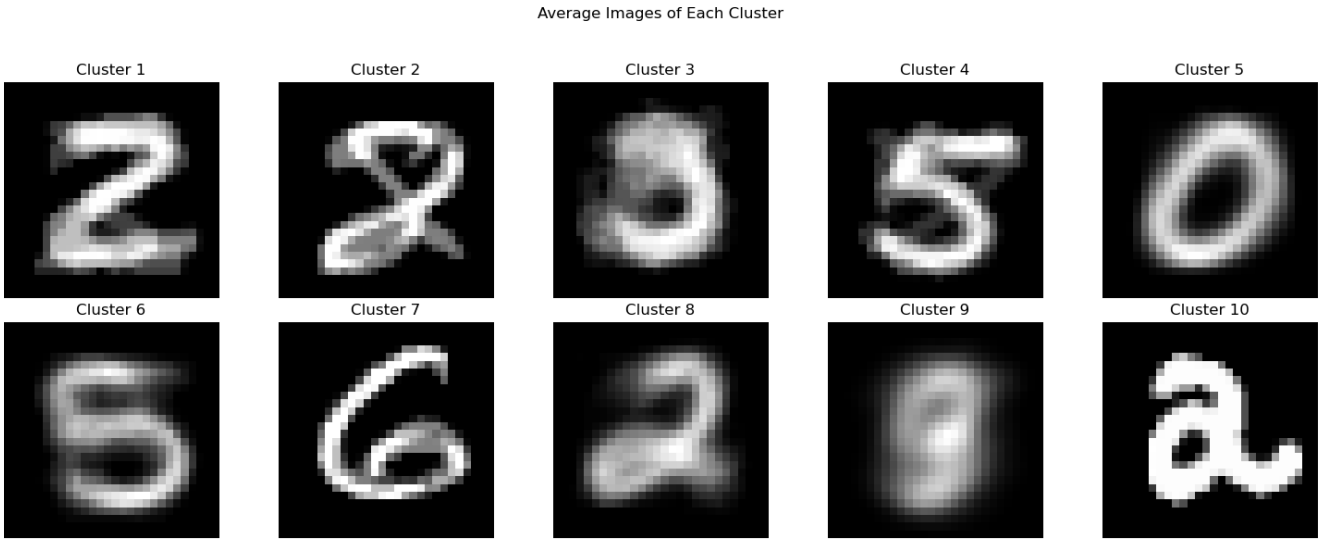


以下对该结果进行分析。

对于Single Link，在层数较少的情况下轮廓函数值较高，但随着层数的增加，轮廓函数值快速下降至负值；对于Complete Link，其轮廓函数值相对比较稳定，且始终大于0；对于Average Link，在层数较少的情况下轮廓函数值较高，随后逐渐下降并在某个点之后趋于平稳，其相比于Single Link更为平稳，且始终大于0。

一方面，基于本例中数据集的特点，过小的层数（如5以下的层数）不符合对数据集进行聚类的要求，也不应选择过大的层数，造成过拟合；另一方面，基于轮廓函数值的正负代表了是否将数据集中的样本分在了与其具有接近特点、距离较近的聚类中这一含义，应选择对应轮廓函数值为正的层数、方法。基于上述原则，首先排除了Single Link，因为其轮廓函数值在层数较小时较大之后，迅速下降至负值，不符合要求；而对于剩下的两种方法，在排除了很小的层数之后，二者在剩余区域轮廓函数值均为正值且比较接近，其中Average Link在层数为10左右有较高的轮廓函数值。

同时，三种方法的计算时间总体相差不大，因而主要基于轮廓函数值，选择层数为10的Average Link方法。该方法对应结果的每个簇的平均结果如下图所示



7

首先对二者的计算时间进行比较。本例中，分级聚类算法的计算时间明显长于K均值聚类算法，但需要指出的是，本例的自行实现分级聚类算法有明显的优化方法，如只在开始计算一次各个样本之间的距离并存储在矩阵中，后续需要进行比较只需查询即可，而不需要如本例合并簇每次都进行距离计算，耗费大量的时间；事实上，在利用已有库进行验证时，进行三种方法的分级聚类的总用时在3s以内；此外，对算法进行粗略的时间复杂度分析，K均值聚类算法在计算距离这一点上的复杂度一定不高于分级聚类算法（分级聚类算法的第一步一定需要计算所有点之间的距离，但K均值聚类算法不一定需要，且其若计算得到该全排列距离矩阵后便一定不需要再进行距离计算，即其总共需要计算距离的次数一定小于这一全排列的数量）。

但另一方面来说，若要进行不同聚类层数的效果分析，分级聚类算法只需计算一次，而K均值聚类算法则需进行多次重复计算（但在计算距离这一点上，其次数仍一定不高于分级聚类算法）。此外，K均值聚类是一个贪心算法，其初值选择可能会影响最终结果，但分级聚类问题则不受这一因素影响。

总结：随着单个样本数据量的增大，距离计算越来越在总计算时间中占据主要部分，因而我们有理由认为：单次计算中K均值算法在时间复杂度上相比分级聚类算法更具有优势，但这需要基于解决如何评估选择合理的层数值、如何减弱初值对结果的影响这两个问题的基础之上。

而对于两种方法的聚类结果，对于两种方法的最佳方案（K=10、层数为10的Average Link），前者的分类基本能对7～8个数字进行分类，而后者只能较明显地分类出6个数字，因而本例中K均值聚类的效果更好。但需要指出的是，仅凭借轮廓函数对结果进行评价，效果可能不够好，还需要引入其他的评价方法，或利用部分有标签的数据集进行评价，才能找到不同聚类算法内更好的方案，实现效果更好的聚类，因而上述比较结果也不能说明在其它情况下或者本例的计算情景中两种算法的真实优劣程度。