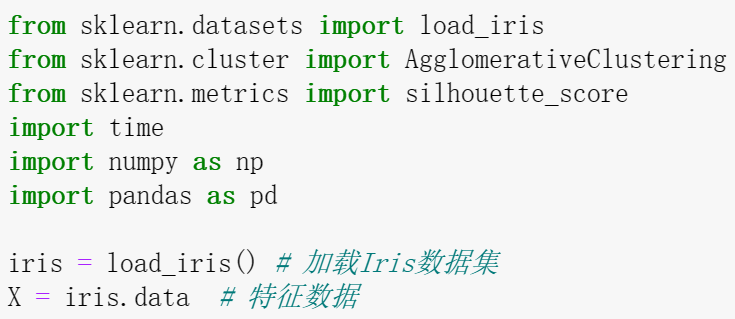
**凝聚层次聚类（Agglomerative Clustering）**

**一、加载和预处理数据**

首先，我选择了Iris数据集做聚类，可以从UCI Machine Learning Repository下载，我这里直接用scikit-learn来加载数据集。



Iris数据集包含了4个属性：Sepal.Length（花萼长度）、Sepal.Width（花萼宽度）、Petal.Length（花瓣长度）、Petal.Width（花瓣宽度）。同时有3个种类：Iris Setosa（山鸢尾）、Iris Versicolour（杂色鸢尾），以及Iris Virginica（维吉尼亚鸢尾）。

**二、先使用sklearn库中自带的AgglomerativeClustering实现算法**

AgglomerativeClustering的构造函数中比较重要的参数有：

1. n\_clusters参数，代表最终要分为多少个簇。

2、affinity参数，起到定义距离的作用，有以下选项：

euclidean：欧式距离，最常用的距离度量。

manhattan：曼哈顿距离，计算的是在标准坐标系上的点之间的距离总和。

cosine：余弦相似度，常用于文本或高维数据的相似性度量。

precomputed：允许用户提供一个预先计算好的距离矩阵。

3、linkage参数，用于计算两个簇之间的距离，共有4个选项:

ward：最小化合并聚类时的方差增加量。

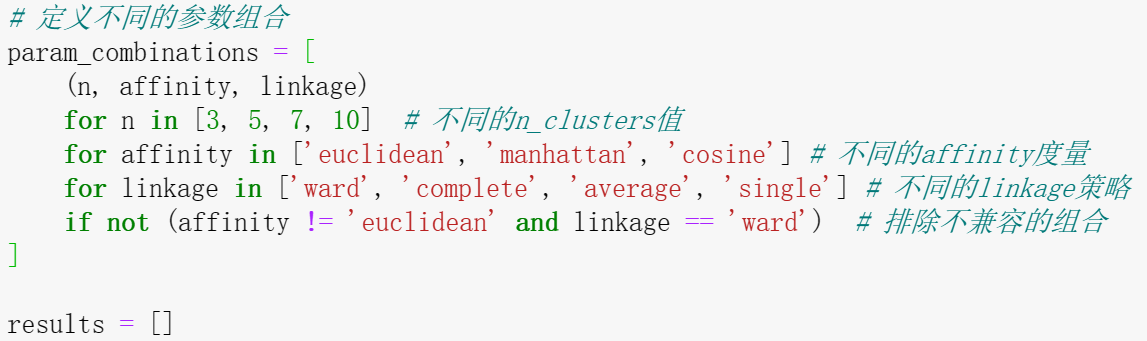
average：合并两个簇所有成员间平均距离最小的两个簇。

complete：计算每一对簇中距离最远的两个样本的距离，并合并距离最远的两个样本所属簇。

single：计算每一对簇中距离最近的两个样本的距离，并合并距离最近的两个样本所属簇。

**实验设定：**

我将在iris数据集上进行AgglomerativeClustering，并且尝试不同的n\_clusters，affinity和linkage参数组合，以此来比较不同的组合对于运行效率和结果的影响。



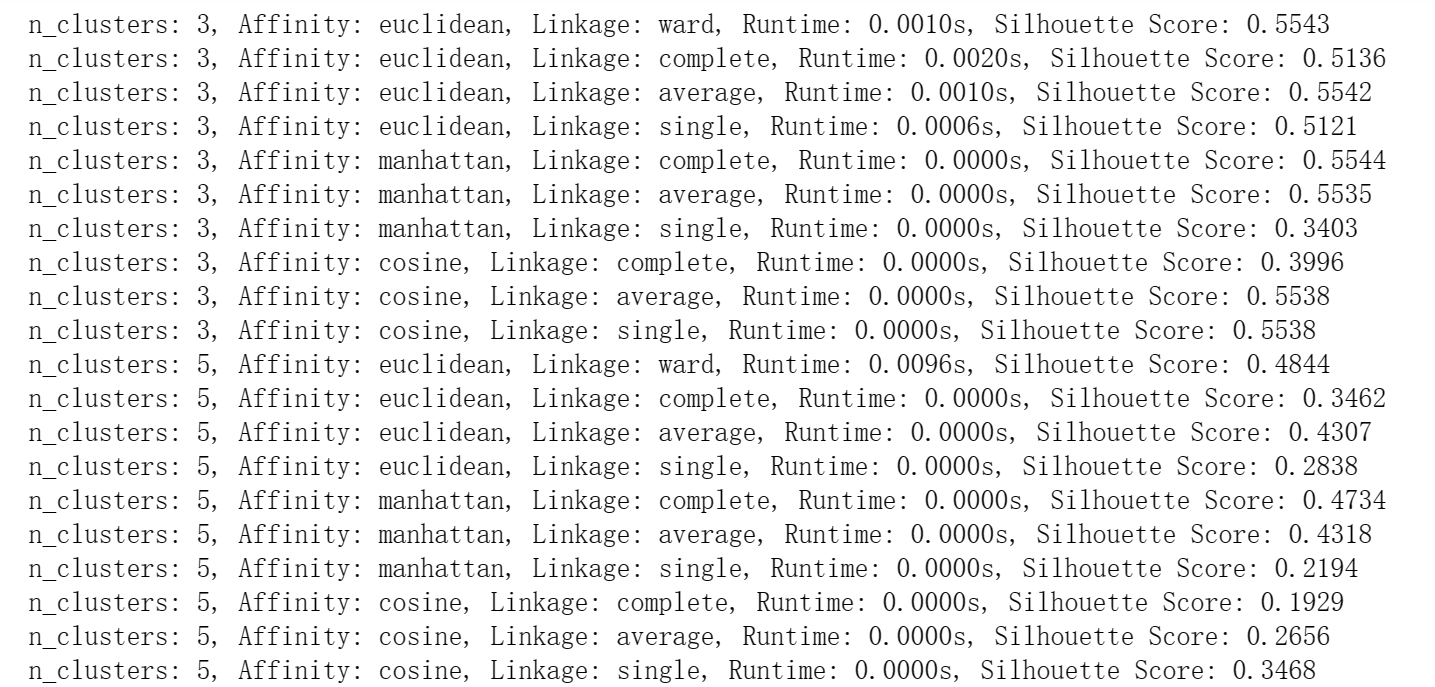
由于要事先定义，所以不使用precomputed距离；由于manhattan和cosine不能与ward链接策略一起使用，故略去；其他诸如切比雪夫距离和马氏距离因为不在选项内而忽略；对于结果评估，使用轮廓系数评估聚类效果的好坏，值越接近1表示聚类效果越好。

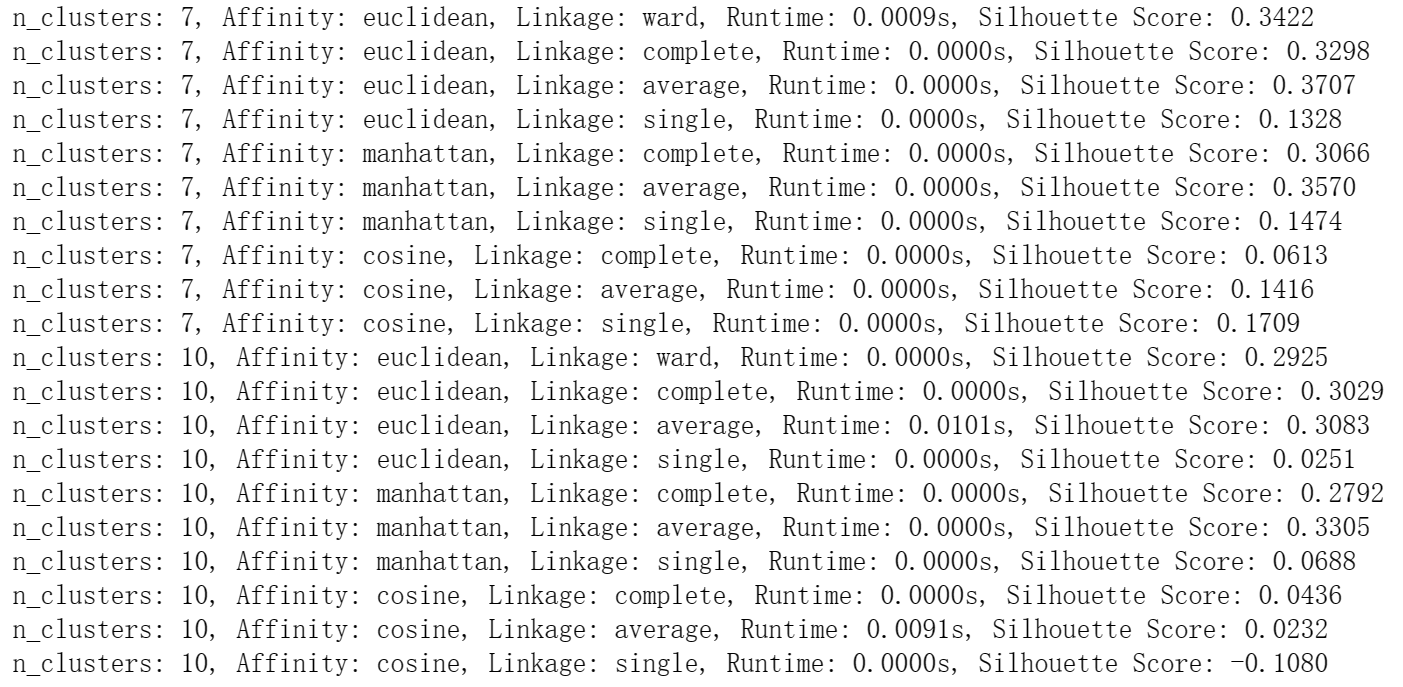
轮廓系数的计算方法如下：

* 对于每个样本，计算与同簇其他样本的平均距离a。
* 对于每个样本，计算与最近簇内样本所在簇的平均距离b。
* 计算轮廓系数：s= (b- a) / max(a, b)

对于每一个组合，打印出它的runtime和轮廓系数。







**通过结果可以观察到：**

1、轮廓系数随着聚类簇数量的增加而整体下降。

2、euclidean 和 manhattan 距离在大多数情况下，提供了较高的轮廓系数；cosine距离度量在聚类数增加时，其轮廓系数显著下降，特别是当聚类数为10时，部分情况下轮廓系数甚至为负值。

3、single策略在大多数情况下提供了较低的轮廓系数；使用euclidean距离和ward策略似乎是一个较好的选择；complete和average策略的表现较为一致，没有明显的优势，但average的时间复杂度可能会低一点。

由于runtime会受到多种因素的影响（如CPU 负载、系统资源分配等），每次实际运行的结果可能会有所不同。时间复杂度可能可以提供一个更稳定的理解框架。

基础的凝聚层次聚类的时间复杂度为O(n3)，如果某个簇到其他所有簇的距离存放在一个有序表或堆中,层次聚类所需要的时间复杂度将为O(n2logn)

对于不同的linkage策略，时间复杂度大致如下：

* Ward, Average, Complete linkage: 这些策略的时间复杂度通常为O(n2logn)到O(n3)，具体取决于实现的细节。Ward和Average链接在实践中通常表现为接近O(n2logn)，而Complete链接在某些情况下可能会接近O(n3)。
* Single linkage: 对于single链接策略，有一些特别高效的算法实现（如基于MST的算法），其时间复杂度可以达到O(n2)。

而不同的affinity对算法的总体时间复杂度影响较小，主要是影响单次距离计算的成本。

**三、用代码实现凝聚层次聚类算法**

我将使用在iris模型上表现较好的euclidean距离和average策略来实现该算法。

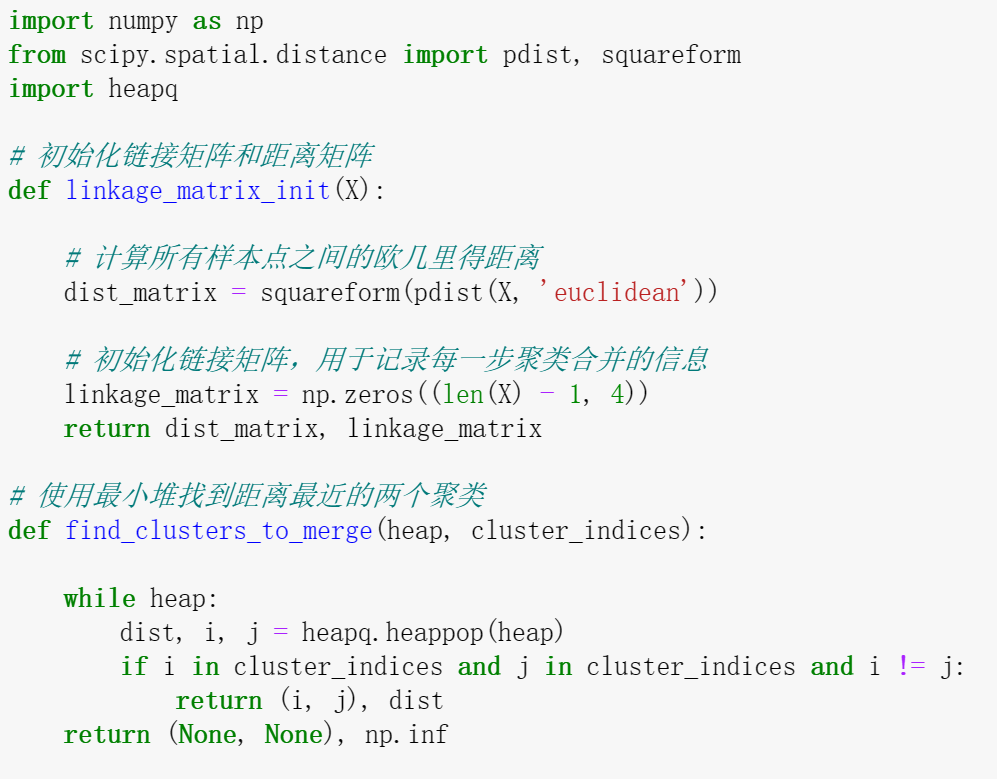
由于sklearn中的算法经过了合适的优化和聚类合并策略，运用连通性约束矩阵，减少了重复计算，所以本算法效果可能不如官方库好。

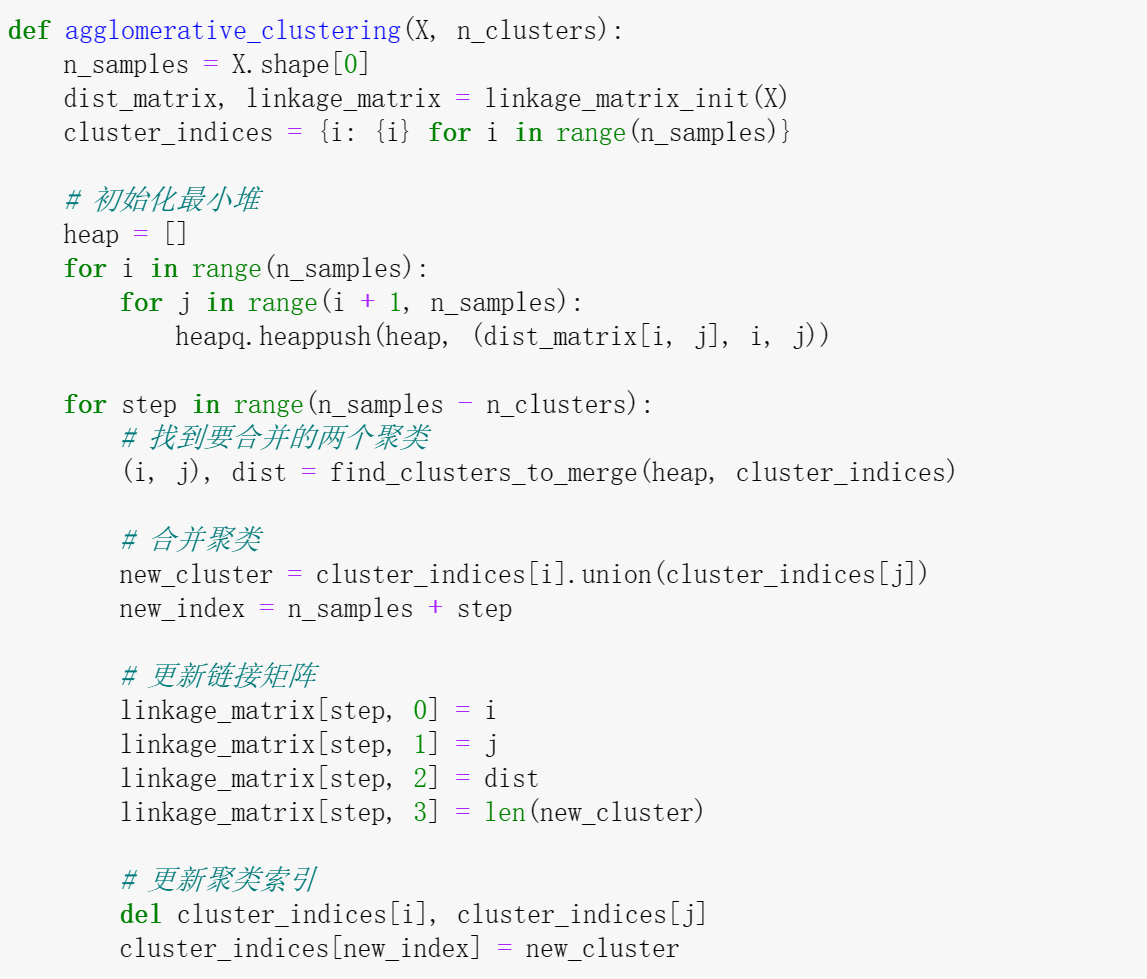
步骤：

1、初始距离计算、堆初始化：首先计算所有样本点之间的距离，并使用这些距离初始化一个最小堆。最小堆中的每个元素包括一个距离值和这个距离对应的两个样本点（或聚类）的索引。

2、聚类合并、距离更新：在每一步聚类过程中，算法从最小堆中弹出最小的距离及其对应的样本点（或聚类）对，作为要合并的聚类对。然后，根据这两个聚类的合并结果更新聚类索引，并重新计算新聚类与其他所有聚类之间的平均距离，将这些新的距离信息加入到最小堆中。

3、迭代：重复上述合并与距离更新步骤，直到聚类的数量减少到指定的目标聚类数目。







算法思想：通过引入最小堆来维护距离，每次从堆中取出最小的距离对应的聚类对进行合并，从而减少每次查找最近聚类对的时间复杂度。当合并两个聚类后，需要更新这两个聚类与其他所有聚类之间的平均距离，并将这些距离加入到最小堆中，以保证下一次能够正确地找到最近的聚类对。

时间复杂度优化：

使用最小堆来管理聚类间的距离可以显著提高查找最近聚类对的效率。在未优化的情况下，每次查找最近聚类对的时间复杂度为O(n2)，其中n为当前聚类的数量。而最小堆使得该操作的时间复杂度降低到了O(log n)，因为堆的插入和弹出操作的时间复杂度为O(log n)。

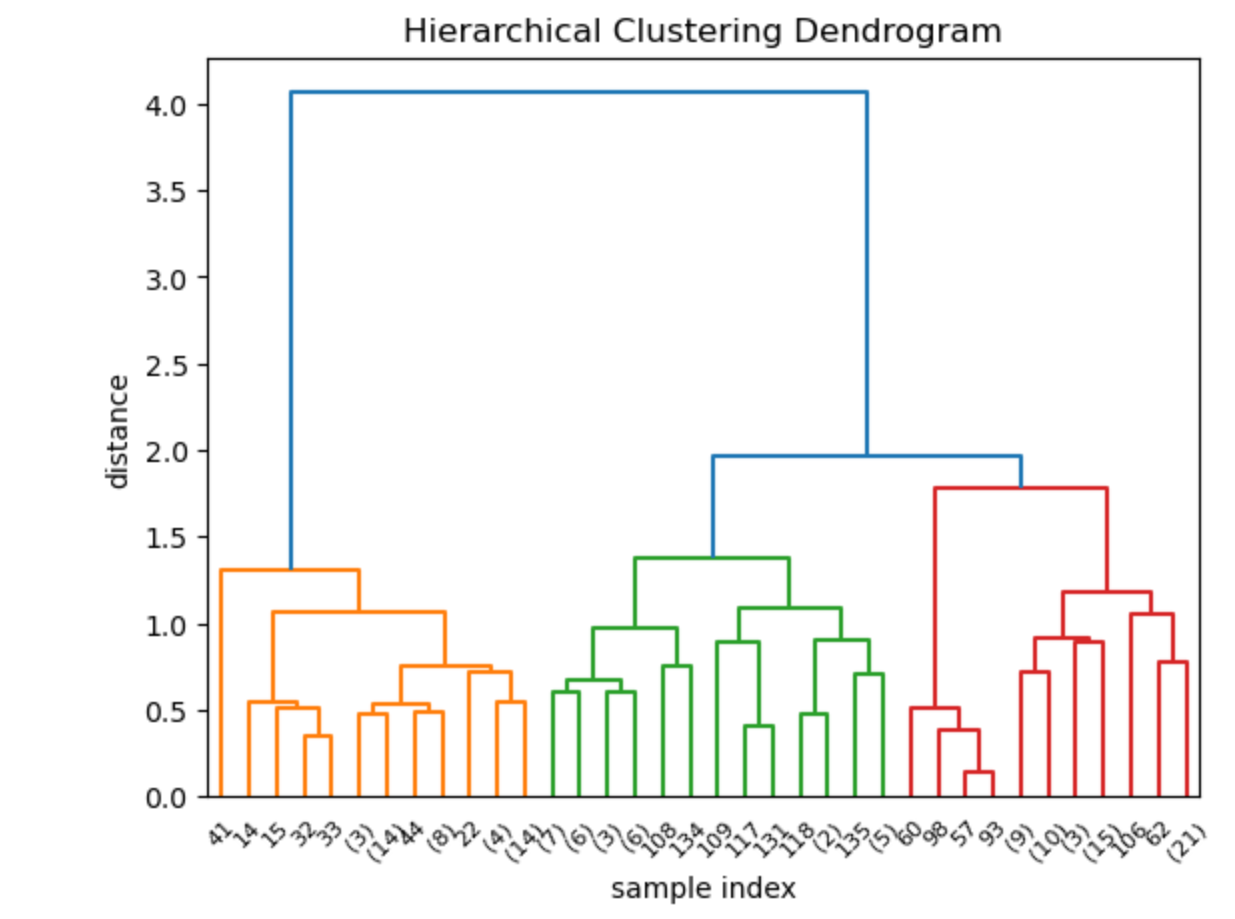
空间复杂度优化：

主要体现在对距离矩阵的处理上。在初始的距离计算之后，算法不需要存储完整的距离矩阵，因为聚类间的距离被动态计算并即时更新在最小堆中。虽然最小堆需要额外的空间来存储距离和索引，但这比维护完整的距离矩阵要小得多，特别是随着聚类合并的进行，聚类数量减少，相关的距离条目也相应减少。

结果与使用官方库得到的结果相同（n\_clusters=5，affinity=euclidean，linkage=average）：



树状图：



可以看出有3个簇的区别。

**四、凝聚层次聚类的优缺点**

1、优点

1）一次性地得到了整个聚类的过程，改变cluster数目不需要再次计算数据点的归属。

2) 适用于任意形状的聚类，并且对样本的输入顺序不敏感。

2、缺点

1）时间复杂度大。

2）由于运用贪心算法，得到的是局部最优，不一定就是全局最优。