层次聚类算法原理总结

原创 石头 机器学习算法那些事 2019-06-14

层次聚类(hierarchical clustering)基于簇间的相似度在不同层次上分析数据，从而形成树形的聚类结构，层次聚类一般有两种划分策略：自底向上的聚合（agglomerative）策略和自顶向下的分拆（divisive）策略，本文对层次聚类算法原理进行了详细总结。

目录

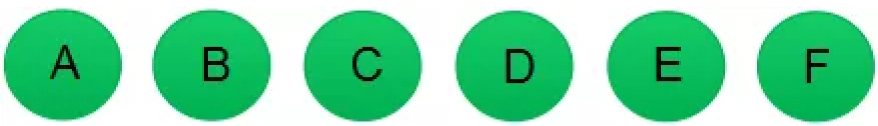
1. 层次聚类算法原理

层次聚类根据划分策略包括聚合层次聚类和拆分层次聚类，由于前者较后者有更广泛的应用且算法思想一致，因此本节重点介绍聚合层次聚类算法。

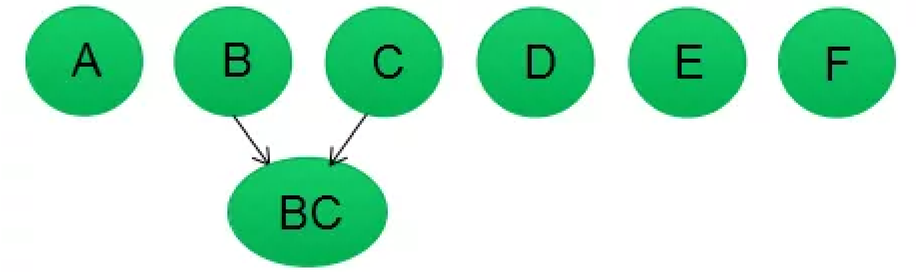
聚合层次聚类算法假设每个样本点都是单独的簇类，然后在算法运行的每一次迭代中找出相似度较高的簇类进行合并，该过程不断重复，直到达到预设的簇类个数K或只有一个簇类。

聚合层次聚类的基本思想：
1）计算数据集的相似矩阵；
2）假设每个样本点为一个簇类；
3）循环：合并相似度最高的两个簇类，然后更新相似矩阵；
4）当簇类个数为1时，循环终止；

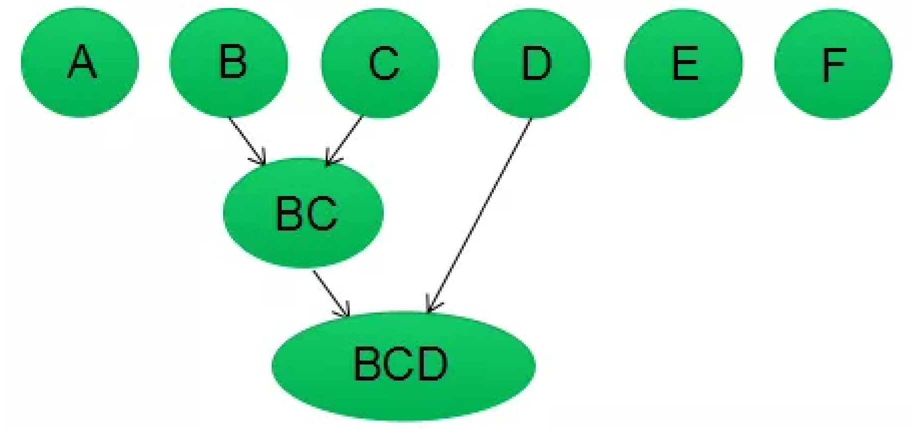
为了更好的理解，我们对算法进行图示说明，假设我们有6个样本点{A,B,C,D,E,F}。
第一步：我们假设每个样本点都为一个簇类（如下图），计算每个簇类间的相似度，得到相似矩阵；



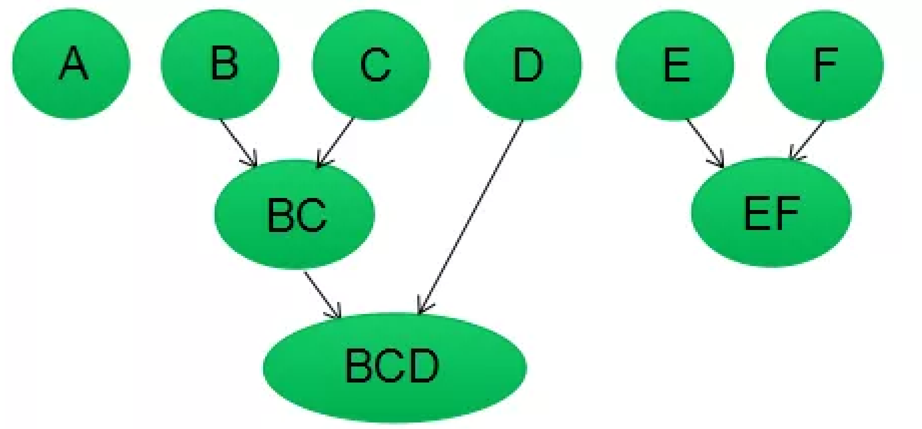
第二步：若B和C的相似度最高，合并簇类B和C为一个簇类。现在我们还有五个簇类，分别为A，BC，D，E，F。



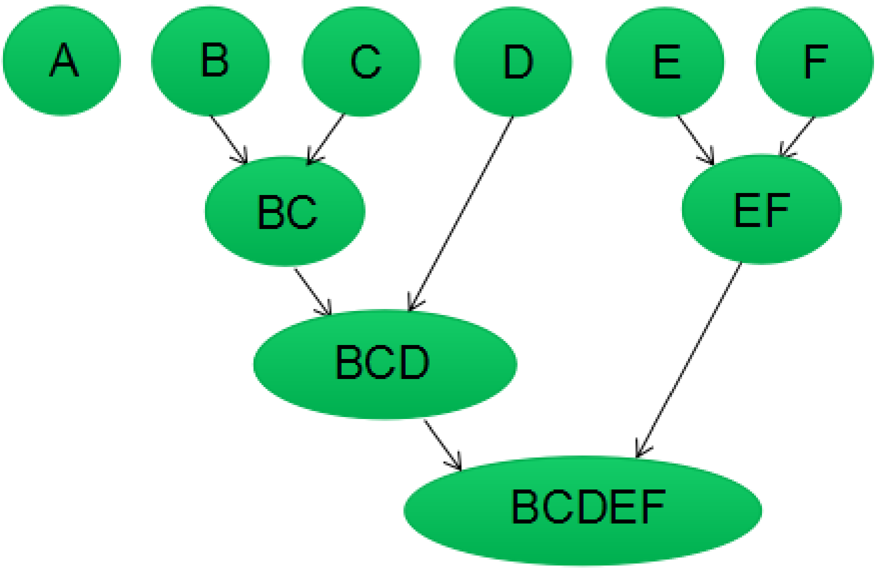
第三步：更新簇类间的相似矩阵，相似矩阵的大小为5行5列；若簇类BC和D的相似度最高，合并簇类BC和D为一个簇类。现在我们还有四个簇类，分别为A，BCD，E，F。



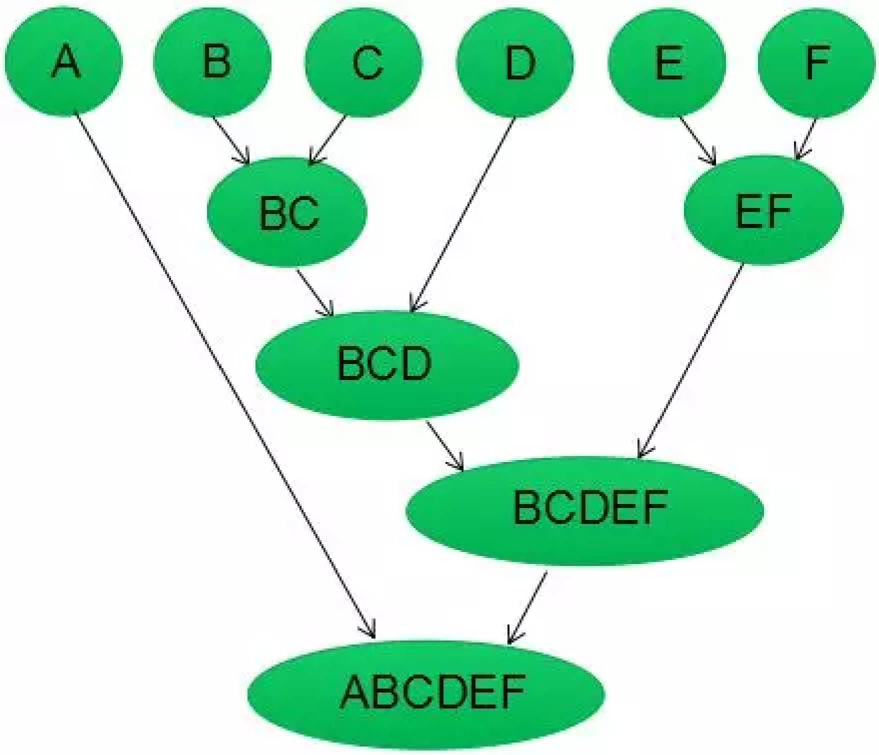
第四步：更新簇类间的相似矩阵，相似矩阵的大小为4行4列；若簇类E和F的相似度最高，合并簇类E和F为一个簇类。现在我们还有3个簇类，分别为A，BCD，EF。



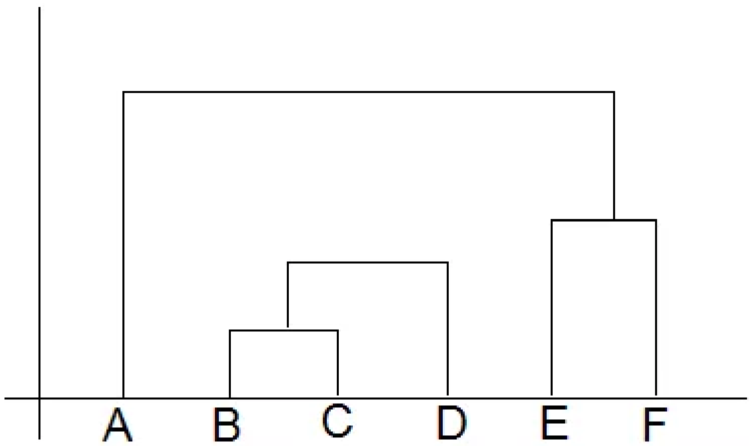
第五步：重复第四步，簇类BCD和簇类EF的相似度最高，合并该两个簇类；现在我们还有2个簇类，分别为A，BCDEF。



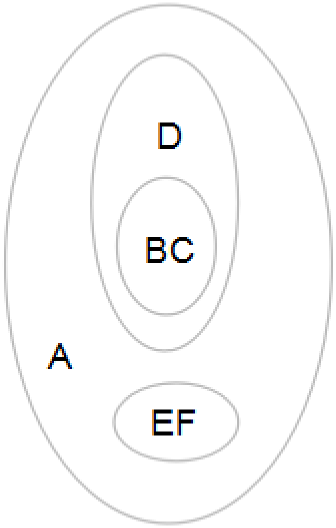
第六步：最后合并簇类A和BCDEF为一个簇类，层次聚类算法结束。



树状图是类似树（tree-like）的图表，记录了簇类聚合和拆分的顺序。我们根据上面的步骤，使用树状图对聚合层次聚类算法进行可视化：



也可用下面的图记录簇类聚合和拆分的顺序：



拆分层次聚类算法假设所有数据集归为一类，然后在算法运行的每一次迭代中拆分相似度最低的样本，该过程不断重复，最终每个样本对应一个簇类。简单点说，拆分层次聚类是聚合层次聚类的反向算法，读者可通过树状图去加强理解，一个是自底向上的划分，一个是自顶向下的划分。

2. 簇间相似度的计算方法

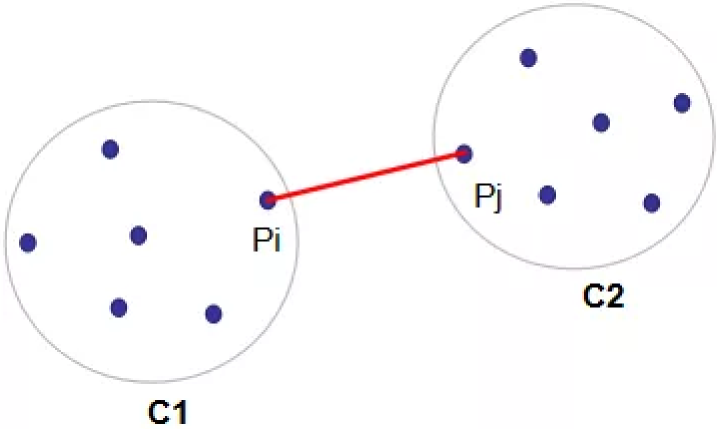
由上节知道，合并或拆分层次聚类算法都是基于簇间相似度进行的，每个簇类包含了一个或多个样本点，通常用距离评价簇间或样本间的相似度，即距离越小相似度越高，距离越大相似度越低。因此我们首先假设样本间的距离为：dist(Pi,Pj)，其中Pi，Pj为任意两个样本，下面介绍常用的簇间相似度计算方法：

- 最小距离
- 最大距离
- 平均距离
- 中心距离
- 最小方差法

最小距离：也称为单链接算法（single linkage algorithm），含义为簇类C1和C2的距离由该两个簇的最近样本决定，数学表达式写为：

$$dist(C1,C2) = \min_{P_i \in C1, P_j \in C2} dist(P_i, P_j)$$

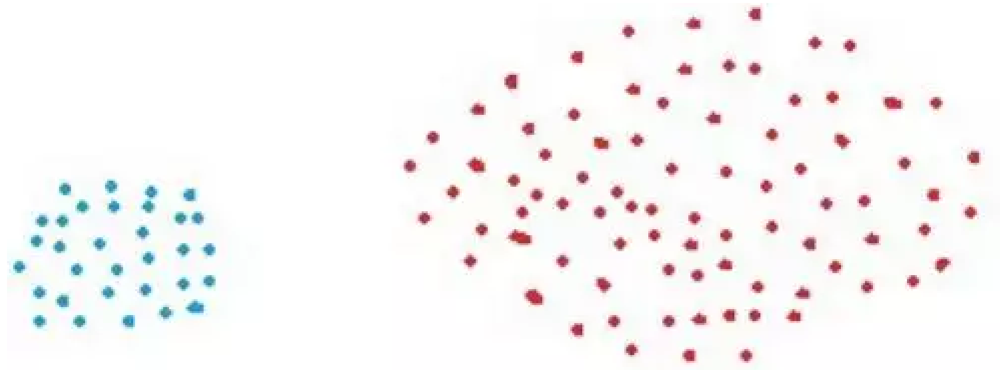
算法也可用下图表示，其中红色线表示簇类C1和C2的距离。



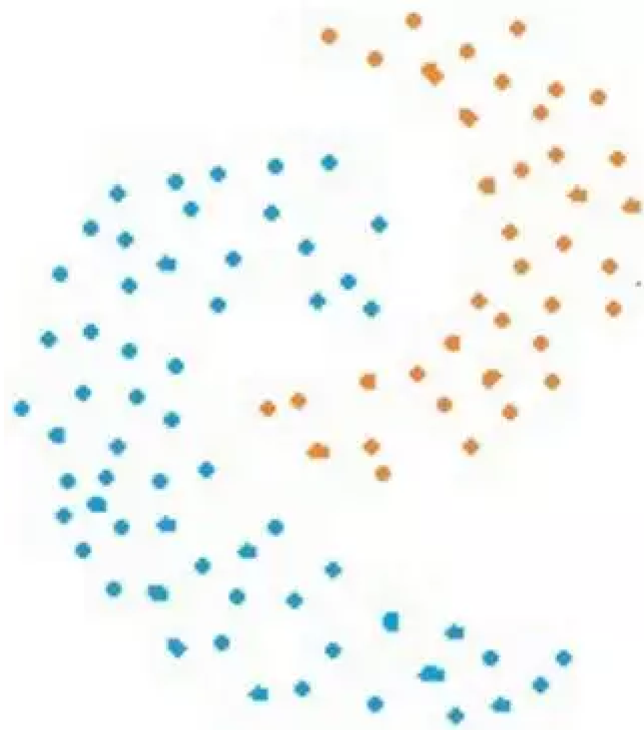
优点:

只要两个簇类的间隔不是很小，单链接算法可以很好的分离非椭圆形状的样本分布。

如下图的两个聚类例子，其中不同颜色表示不同的簇类:



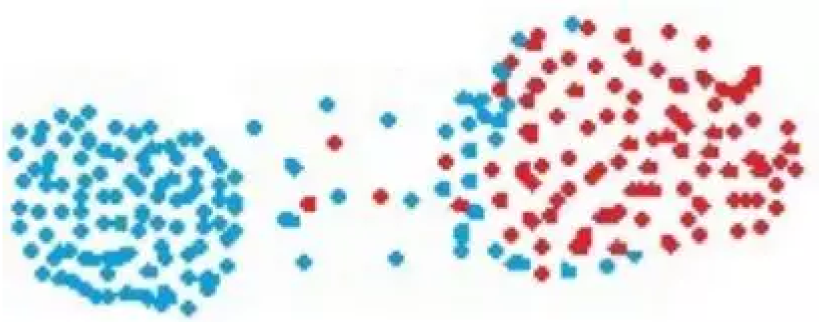
例1



例2

缺点:

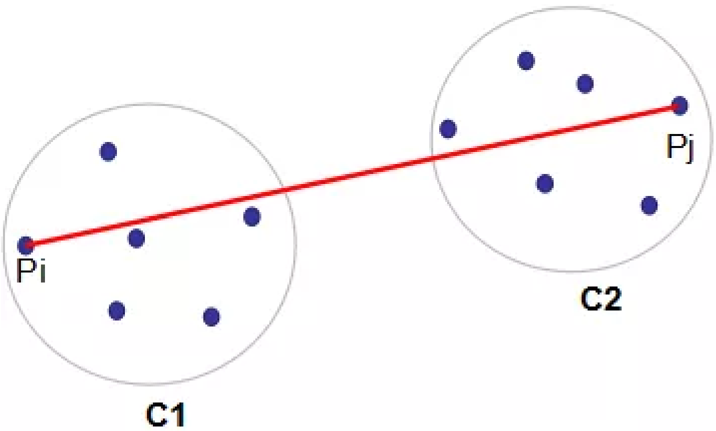
单链接算法不能很好的分离簇类间含有噪声的数据集，如下图:



最大距离：也称为全链接算法（complete linkage algorithm），含义为簇类C1和C2的距离由该两个簇的最远样本决定，与单链接算法的含义相反，数学表达式写为：

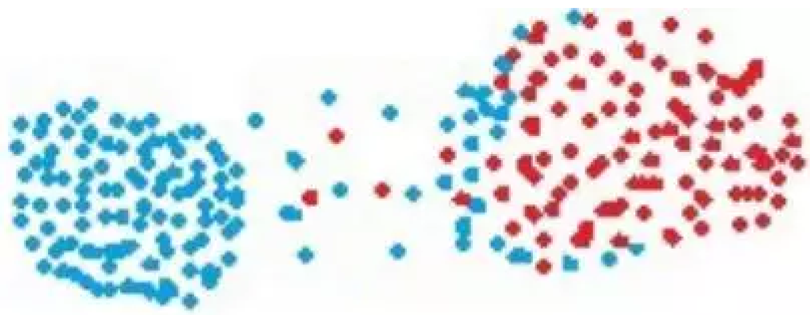
$$dist(C1,C2)=\max_{P_i\in C1,P_j\in C2} dist(P_i,P_j)$$

算法也可用如下图表示，其中红色线表示簇类C1和C2的距离。



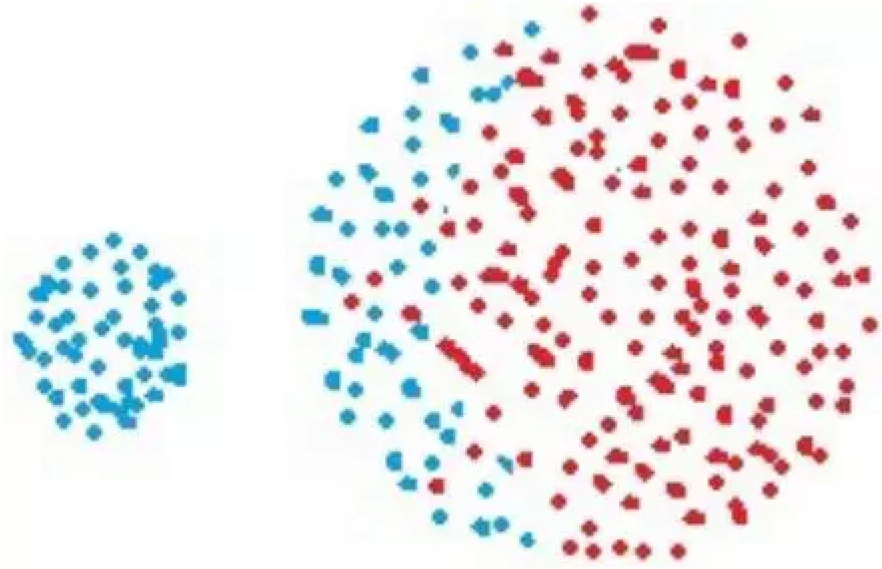
优点：

全链接算法可以很好的分离簇类间含有噪声的数据集，如下图：



缺点：

全链接算法对球形数据集的分离会产生偏差，如下图：

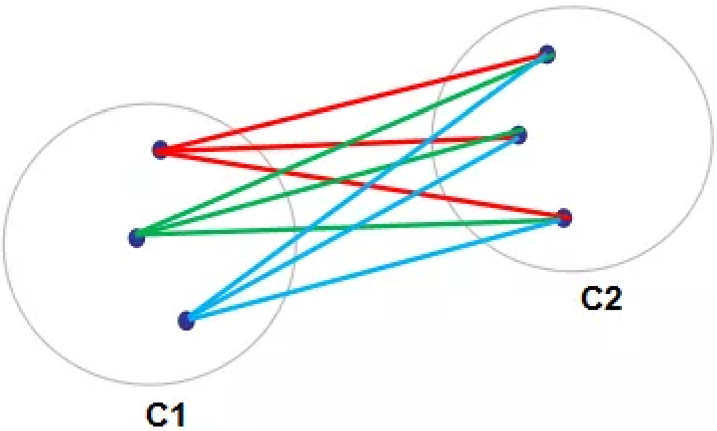


平均距离：也称为均链接算法（average-linkage algorithm），含义为簇类C1和C2的距离等于两个簇类所有样本对的距离平均，数学表达式为：

$$dist(C1,C2)=\frac{1}{|C1|\cdot|C2|}\sum_{P_i\in C1,P_j\in C2}dist(P_i,P_j)$$

其中|C1|，|C2|分别表示簇类的样本个数。

均链接算法也可用如下图表示：



所有连线的距离求和平均即为簇类C1和C2的距离。

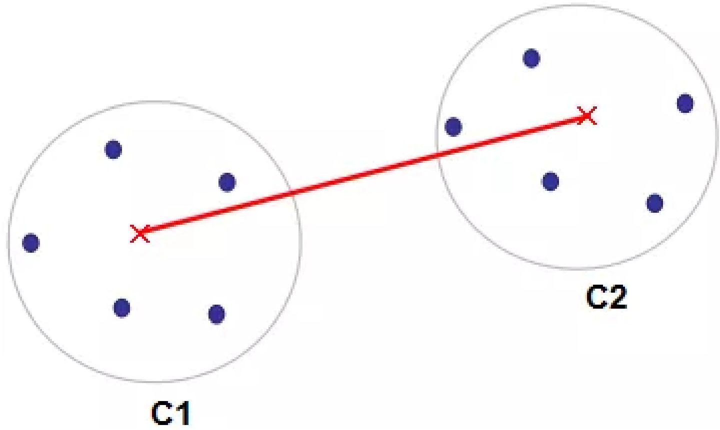
优点：

均链接算法可以很好的分离簇类间有噪声的数据集。

缺点：

均链接算法对球形数据集的分离会产生偏差。

中心距离：簇类C1和C2的距离等于该两个簇类中心间的距离，如下图：



其中红色点表示簇类的中心，红色线表示簇类C1和C2的距离。这种计算簇间距离的方法非常少用，个人不建议使用这一算法。

离差平方和：簇类C1和C2的距离等于两个簇类所有样本对距离平方和的平均，与均链接算法很相似，数学表达式为：

$$dist(C1,C2)=\frac{1}{|C1|\cdot|C2|}\sum_{Pi\in C1,Pj\in C2}(dist(Pi,Pj))^2$$

优点：

离差平方和可以很好的分离簇间有噪声的数据集。

缺点：

离差平方和对球形数据集的分离会产生偏差。

我们已经知道了如何通过样本间的距离来评估簇间的距离，本节只剩下最后一个问题了，如何计算样本间的距离，假设样本是n维，常用的距离计算方法有：

1）欧拉距离（Euclidean distance）：

$$dist(P_i,P_j)=\sqrt{\sum_{k=1}^{n}(P_{ik}-P_{jk})^2}$$

2）平方欧式距离（Squared Euclidean distance）：

$$dist(P_i,P_j)=\sum_{k=1}^{n}(P_{ik}-P_{jk})^2$$

3）曼哈顿距离（Manhattan distance）：

$$dist(P_i,P_j)=\sum_{k=1}^{n}|P_{ik}-P_{jk}|$$

4）切比雪夫距离（Chebyshev distance）：

$$dist(P_i,P_j)=\max_{k=1,2...,n}|P_{ik}-P_{jk}|$$

5）马氏距离（Mahalanobis distance）：

$$dist(P_i,P_j)=\sqrt{(P_i-P_j)^T S^{-1}(P_i-P_j)}$$

其中S为协方差矩阵。

对于文本或非数值型的数据，我们常用汉明距离（Hamming distance）和编辑距离（Levenshtein distance）表示样本间的距离。

不同的距离度量会影响簇类的形状，因为样本距离因距离度量的不同而不同，如点（1.1）和（0,0）的曼哈顿距离是2，欧式距离是sqrt(2)，切比雪夫距离是1。

3. 层次聚类算法的复杂度计算

空间复杂度：当样本点数目很大时，层次聚类需要较大的空间存储相似矩阵，相似矩阵的大小是样本个数的平方，因此空间复杂度是n的平方阶次，即空间复杂度=O(n^2)。

时间复杂度：层次聚类算法需要进行n次迭代，每次迭代都需要更新并存储相似矩阵，所以时间复杂度是样本个数n的立方阶次，即时间复杂度=O(n^3)。

4.层次聚类算法的优化方法

上节介绍数据量较大时，层次聚类算法的空间复杂度和时间复杂度较高，我们可以通过连通性约束（connectivity constraint）降低算法复杂度，甚至提高聚类结果。

连通性约束是通过连通性矩阵（connectivity matrix）实现的，连通性矩阵的元素只有1和0两种结果，1表示两个样本是连通的，0表示两个样本不连通，我们只对连通性的样本进行距离计算并融合，这一过程大大降低了计算量，常采用sklearn.neighbors.kneighbors_graph来构建连接矩阵。

我们构建了容量为15000的瑞士卷（swiss roll dataset）数据集：

```
# 生成瑞士卷数据集，容量为15000
n_samples = 15000
noise = 0.05
X, _ = make_swiss_roll(n_samples, noise)
# 减小瑞士卷的厚度
X[:, 1] *= .5
```

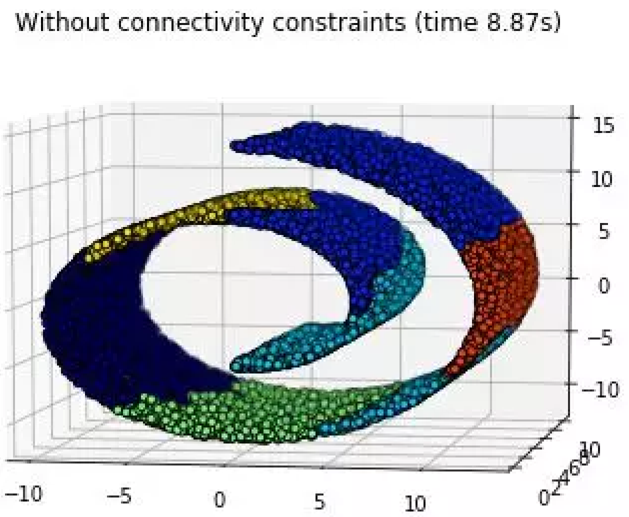
用离差平方和的层次聚类算法建模，可视化聚类结果并输出算法运行时间。

```
print("Compute unstructured hierarchical clustering...")
st = time.time()
ward = AgglomerativeClustering(n_clusters=6, linkage='ward').fit(X)
elapsed_time = time.time() - st
label = ward.labels_
# 运行时间
print("Elapsed time: %.2fs" % elapsed_time)
```

```
print("Number of points: %i" % label.size)

# #############################################################################
# 可视化结果
fig = plt.figure()
ax = p3.Axes3D(fig)
ax.view_init(7, -80)
for l in np.unique(label):
    ax.scatter(X[label == l, 0], X[label == l, 1], X[label == l, 2],
               color=plt.cm.jet(np.float(l) / np.max(label + 1)),
               s=20, edgecolor='k')
plt.title('Without connectivity constraints (time %.2fs)' % elapsed_time)
```

结果：



我们在构建层次聚类算法模型前，先定义数据集的连通矩阵：

```
# 定义不包含样本点在内的10个最近邻的连通样本点
from sklearn.neighbors import kneighbors_graph
connectivity = kneighbors_graph(X, n_neighbors=10, include_self=False)
```

用离差平方和的层次聚类算法建模，可视化聚类结果并输出算法运行时间：

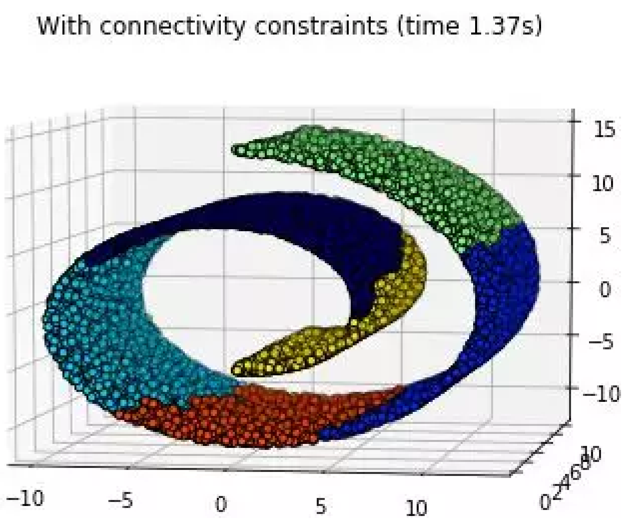
```
print("Compute structured hierarchical clustering...")
st = time.time()
ward = AgglomerativeClustering(n_clusters=6, connectivity=connectivity,
                               linkage='ward').fit(X)
elapsed_time = time.time() - st
label = ward.labels_
print("Elapsed time: %.2fs" % elapsed_time)
print("Number of points: %i" % label.size)

# #############################################################################
# Plot result
fig = plt.figure()
ax = p3.Axes3D(fig)
ax.view_init(7, -80)
for l in np.unique(label):
    ax.scatter(X[label == l, 0], X[label == l, 1], X[label == l, 2],
               color=plt.cm.jet(float(l) / np.max(label + 1)),
               s=20, edgecolor='k')
```

```
plt.title('With connectivity constraints (time %.2fs)' % elapsed_time)

plt.show()
```

结果：



由上面例子可知：大数据量的情况下，增加连通性约束矩阵可以降低算法的运行时间，若只关注数据集的局部信息，连通性约束也能提高算法性能。

6. 层次聚类算法的优缺点

优点：

算法简单，易于理解

树状图包含了整个算法过程的信息；

缺点：

选择合适的距离度量与簇类的链接准则较难；

高的时间复杂度和空间复杂度；

参考：

https://towardsdatascience.com

https://scikit-learn.org/stable/

推荐阅读

k-means聚类算法原理总结

聚类 | 超详细的性能度量和相似度方法总结