**scikit-learn——Birch聚类**

**任务目标**

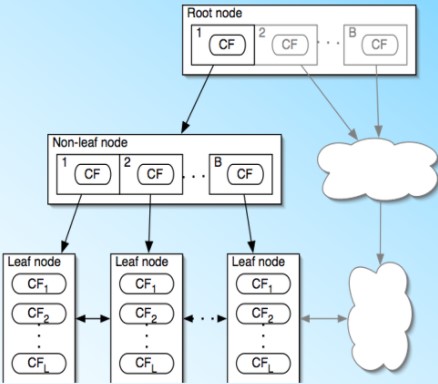
1.了解Birch聚类算法的原理

2.熟练掌握sklearn中Birch类进行聚类的操作

**相关知识**

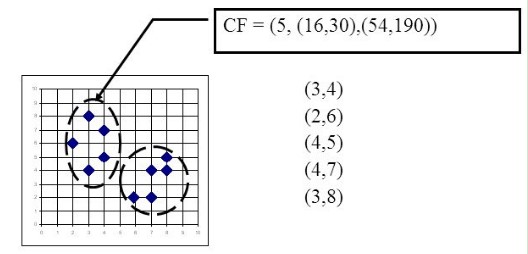
BIRCH的全称是利用层次方法的平衡迭代规约和聚类（Balanced Iterative Reducing and Clustering Using Hierarchies），它是用层次方法来聚类和规约数据。BIRCH只需要单遍扫描数据集就能进行聚类，那它是怎么做到的呢？

BIRCH算法利用了一个树结构来帮助我们快速的聚类，这个树结构类似于平衡B+树，一般将它称之为聚类特征树(Clustering Feature Tree，简称CF Tree)。这棵树的每一个节点是由若干个聚类特征(Clustering Feature，简称CF)组成。从下图我们可以看看聚类特征树是什么样子的：每个节点包括叶子节点都有若干个CF，而内部节点的CF有指向孩子节点的指针，所有的叶子节点用一个双向链表链接起来。

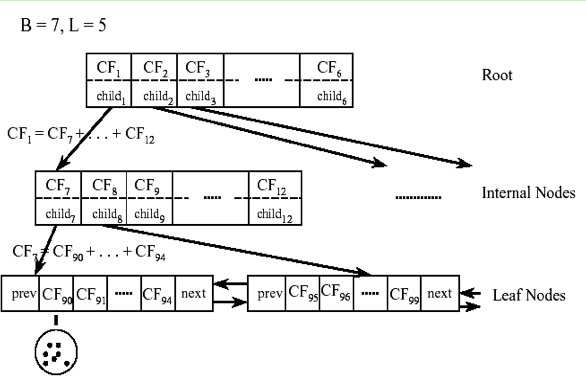
[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/46187821-5e87-4ecb-ba25-146489b474cb.jpg)

聚类特征CF与聚类特征树CF Tree

在聚类特征树中，一个聚类特征CF是这样定义的：每一个CF是一个三元组，可以用（N，LS，SS）表示。其中N代表了这个CF中拥有的样本点的数量，这个好理解；LS代表了这个CF中拥有的样本点各特征维度和向量，SS代表了这个CF中拥有的样本点各特征维度的平方和。举个例子如下图，在CF Tree中的某一个节点的某一个CF中，有下面5个样本(3,4), (2,6), (4,5), (4,7), (3,8)。则它对应的N=5， LS=(3+2+4+4+3,4+6+5+7+8)=(16,30)(3+2+4+4+3,4+6+5+7+8)=(16,30), SS =(32+22+42+42+32+42+62+52+72+82)=(54+190)=244

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/d646777a-89bf-4a20-9713-5c2ad1fa14a9.jpg)

CF有一个很好的性质，就是满足线性关系，也就是CF1+CF2=(N1+N2,LS1+LS2,SS1+SS2)。这个性质从定义也很好理解。如果把这个性质放在CF Tree上，也就是说，在CF Tree中，对于每个父节点中的CF节点，它的(N,LS,SS)三元组的值等于这个CF节点所指向的所有子节点的三元组之和。如下图所示：

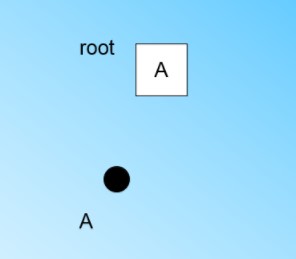
[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/6dc40b92-98d8-4053-a96a-7e7fae4eba62.png)

从上图中可以看出，根节点的CF1的三元组的值，可以从它指向的6个子节点（CF7 - CF12）的值相加得到。这样我们在更新CF Tree的时候，可以很高效。

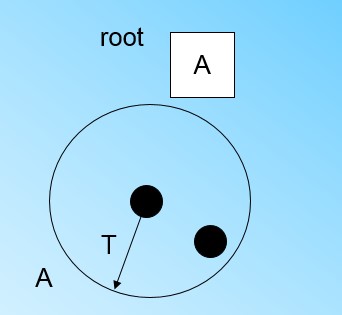
对于CF Tree，我们一般有几个重要参数，第一个参数是每个内部节点的最大CF数B，第二个参数是每个叶子节点的最大CF数L，第三个参数是针对叶子节点中某个CF中的样本点来说的，它是叶节点每个CF的最大样本半径阈值T，也就是说，在这个CF中的所有样本点一定要在半径小于T的一个超球体内。对于上图中的CF Tree，限定了B=7， L=5， 也就是说内部节点最多有7个CF，而叶子节点最多有5个CF。

聚类特征树CF Tree的生成下面我们看看怎么生成CF Tree。我们先定义好CF Tree的参数： 即内部节点的最大CF数B， 叶子节点的最大CF数L， 叶节点每个CF的最大样本半径阈值T

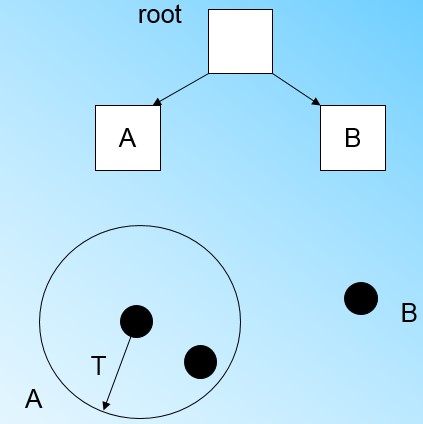
在最开始的时候，CF Tree是空的，没有任何样本，我们从训练集读入第一个样本点，将它放入一个新的CF三元组A，这个三元组的N=1，将这个新的CF放入根节点，此时的CF Tree如下图：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/6ff47742-6a21-403d-aad3-f7e551647139.jpg)

现在我们继续读入第二个样本点，我们发现这个样本点和第一个样本点A，在半径为T的超球体范围内，也就是说，他们属于一个CF，我们将第二个点也加入CF A,需要更新A的三元组的值，A的三元组中N=2。此时的CF Tree如下图：

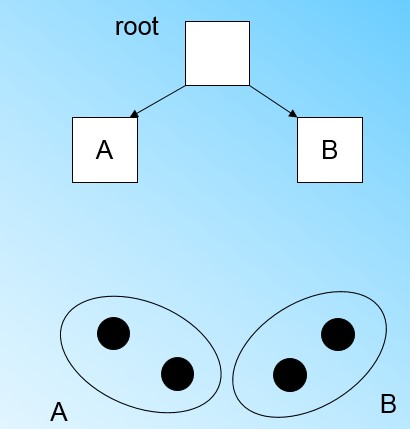
[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/9c8208ac-9bfa-4c30-b094-b82f2c207878.jpg)

继续读入第三个节点，结果我们发现这个节点不能融入刚才前面的节点形成的超球体内，也就是说，我们需要一个新的CF三元组B，来容纳这个新的值。此时根节点有两个CF三元组A和B，此时的CF Tree如下图：

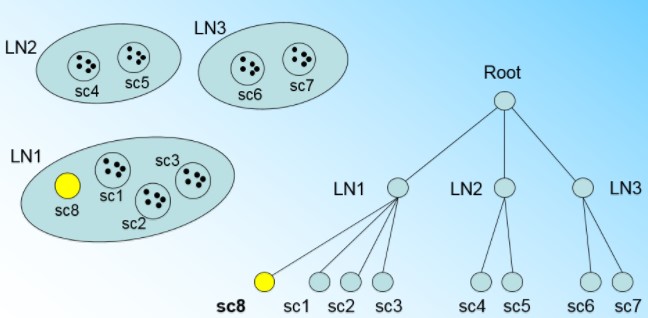
[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/fa25f1ac-8603-42c4-9c6f-8aaed16f425a.jpg)

当来到第四个样本点的时候，我们发现和B在半径小于T的超球体，这样更新

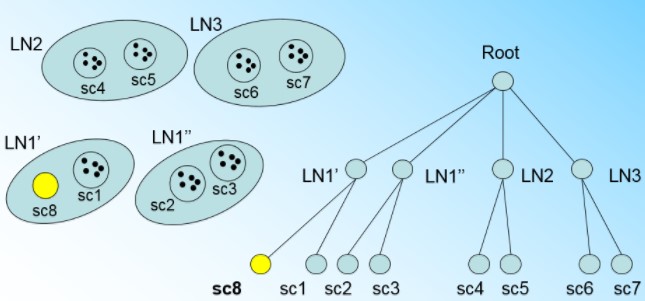
后的CF Tree如下图：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/01.jpg)

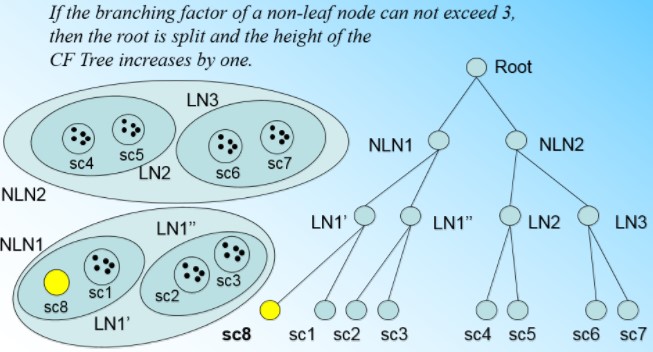
那个什么时候CF Tree的节点需要分裂呢？假设我们现在的CF Tree 如下图， 叶子节点LN1有三个CF， LN2和LN3各有两个CF。我们的叶子节点的最大CF数L=3。此时一个新的样本点来了，我们发现它离LN1节点最近，因此开始判断它是否在sc1,sc2,sc3这3个CF对应的超球体之内，但是很不幸，它不在，因此它需要建立一个新的CF，即sc8来容纳它。问题是我们的L=3，也就是说LN1的CF个数已经达到最大值了，不能再创建新的CF了，怎么办？此时就要将LN1叶子节点一分为二了。

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/2e0ea2f3-4a9c-4c03-b63a-db61ec2cfde8.jpg)

我们将LN1里所有CF元组中，找到两个最远的CF做这两个新叶子节点的种子CF，然后将LN1节点里所有CF sc1, sc2, sc3，以及新样本点的新元组sc8划分到两个新的叶子节点上。将LN1节点划分后的CF Tree如下图：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/14ac701d-9385-4c0d-92a3-876a337401c8.jpg)

如果我们的内部节点的最大CF数B=3，则此时叶子节点一分为二会导致根节点的最大CF数超了，也就是说，我们的根节点现在也要分裂，分裂的方法和叶子节点分裂一样，分裂后的CF Tree如下图：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/7fa646f9-1dd5-47da-b02c-667475b14975.jpg)

上面讲了半天的CF Tree，终于我们可以步入正题BIRCH算法，其实将所有的训练集样本建立了CF Tree，一个基本的BIRCH算法就完成了，对应的输出就是若干个CF节点，每个节点里的样本点就是一个聚类的簇。也就是说BIRCH算法的主要过程，就是建立CF Tree的过程。

当然，真实的BIRCH算法除了建立CF Tree来聚类，其实还有一些可选的算法步骤的，现在我们就来看看 BIRCH算法的流程。

1） 将所有的样本依次读入，在内存中建立一颗CF Tree, 建立的方法参考上一节。

2）（可选）将第一步建立的CF Tree进行筛选，去除一些异常CF节点，这些节点一般里面的样本点很少。对于一些超球体距离非常近的元组进行合并

3）（可选）利用其它的一些聚类算法比如K-Means对所有的CF元组进行聚类，得到一颗比较好的CF Tree.这一步的主要目的是消除由于样本读入顺序导致的不合理的树结构，以及一些由于节点CF个数限制导致的树结构分裂。

4）（可选）利用第三步生成的CF Tree的所有CF节点的质心，作为初始质心点，对所有的样本点按距离远近进行聚类。这样进一步减少了由于CF Tree的一些限制导致的聚类不合理的情况。

从上面可以看出，BIRCH算法的关键就是步骤1，也就是CF Tree的生成，其他步骤都是为了优化最后的聚类结果。

总结下CF Tree的插入：

1. 从根节点向下寻找和新样本距离最近的叶子节点和叶子节点里最近的CF节点

2. 如果新样本加入后，这个CF节点对应的超球体半径仍然满足小于阈值T，则更新路径上所有的CF三元组，插入结束。否则转入3。

3. 如果当前叶子节点的CF节点个数小于阈值L，则创建一个新的CF节点，放入新样本，将新的CF节点放入这个叶子节点，更新路径上所有的CF三元组，插入结束。否则转入4。

4.将当前叶子节点划分为两个新叶子节点，选择旧叶子节点中所有CF元组里超球体距离最远的两个CF元组，作为两个新叶子节点的第一个CF节点。将其他元组和新样本元组按照距离远近原则放入对应的叶子节点。依次向上检查父节点是否也要分裂，如果需要按和叶子节点相同的分裂方式。

BIRCH算法可以不用输入类别数K值，这点和K-Means，Mini Batch K-Means不同。如果不输入K值，则最后的CF元组的组数即为最终的K，否则会按照输入的K值对CF元组按距离大小进行合并。

一般来说，BIRCH算法适用于样本量较大的情况，这点和Mini Batch K-Means类似，但是BIRCH适用于类别数比较大的情况，而Mini Batch K-Means一般用于类别数适中或者较少的时候。BIRCH除了聚类还可以额外做一些异常点检测和数据初步按类别规约的预处理。但是如果数据特征的维度非常大，比如大于20，则BIRCH不太适合，此时Mini Batch K-Means的表现较好。

对于调参，BIRCH要比K-Means，Mini Batch K-Means复杂，因为它需要对CF Tree的几个关键的参数进行调参，这几个参数对CF Tree的最终形式影响很大。

最后总结下BIRCH算法的优缺点：

BIRCH算法的主要优点有：

1) 节约内存，所有的样本都在磁盘上，CF Tree仅仅存了CF节点和对应的指针。

2) 聚类速度快，只需要一遍扫描训练集就可以建立CF Tree，CF Tree的增删改都很快。

3) 可以识别噪音点，还可以对数据集进行初步分类的预处理

BIRCH算法的主要缺点有

1) 由于CF Tree对每个节点的CF个数有限制，导致聚类的结果可能和真实的类别分布不同.

2) 对高维特征的数据聚类效果不好。此时可以选择Mini Batch K-Means

3) 如果数据集的分布簇不是类似于超球体，或者说不是凸的，则聚类效果不好。

BIRCH类参数

在scikit-learn中BIRCH聚类算法模型：

class sklearn.cluster.Birch(threshold=0.5, branching\_factor=50, n\_clusters=3, compute\_labels=True, copy=True)

参数说明：

1) threshold:即叶节点每个CF的最大样本半径阈值T，它决定了每个CF里所有样本形成的超球体的半径阈值。一般来说threshold越小，则CF Tree的建立阶段的规模会越大，即BIRCH算法第一阶段所花的时间和内存会越多。但是选择多大以达到聚类效果则需要通过调参决定。默认值是0.5.如果样本的方差较大，则一般需要增大这个默认值。

2) branching\_factor：即CF Tree内部节点的最大CF数B，以及叶子节点的最大CF数L。这里scikit-learn对这两个参数进行了统一取值。也就是说，branching\_factor决定了CF Tree里所有节点的最大CF数。默认是50。如果样本量非常大，比如大于10万，则一般需要增大这个默认值。选择多大的branching\_factor以达到聚类效果则需要通过和threshold一起调参决定

3）n\_clusters：即类别数K，在BIRCH算法是可选的，如果类别数非常多，我们也没有先验知识，则一般输入None，此时BIRCH算法第4阶段不会运行。但是如果我们有类别的先验知识，则推荐输入这个可选的类别值。默认是3，即最终聚为3类。

4）compute\_labels：布尔值，表示是否标示类别输出，默认是True。一般使用默认值挺好，这样可以看到聚类效果。

在评估各个参数组合的聚类效果时，还是推荐使用Calinski-Harabasz Index，Calinski-Harabasz Index在scikit-learn中对应的方法是metrics.calinski\_harabaz\_score.

**系统环境**

Linux Ubuntu 16.04

Python 3.6.1

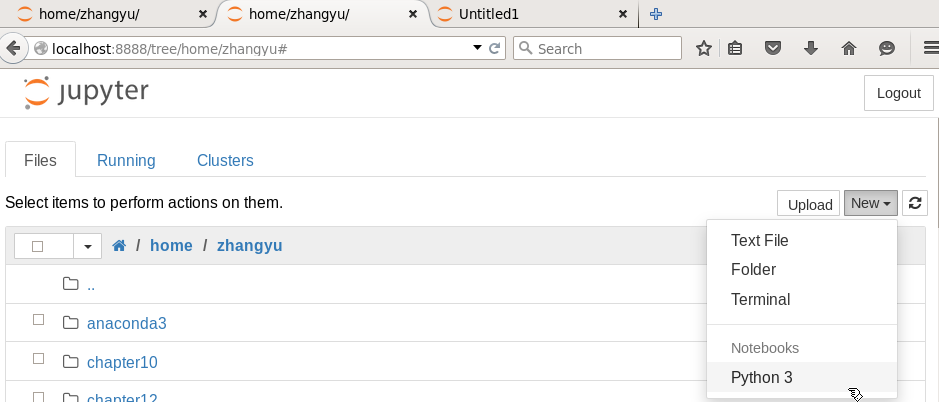
Jupyter

**任务内容**

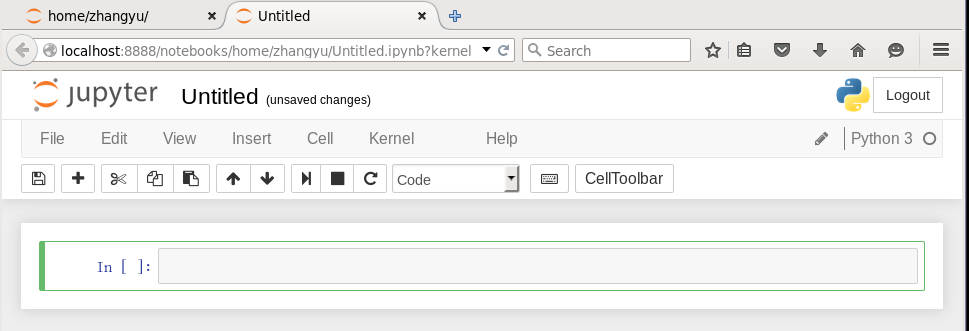
练习sklearn中Birch聚类的操作过程。

**任务步骤**

1.在终端模拟器的命令行输入ipython notebook --ip='127.0.0.1'，在浏览器中会打开下面界面，切换到/home/zhangyu目录下，点击New，在其下拉框中选择Python3.

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/205dfc16-99a9-4daf-93e2-970a85696662.png)

2.新建一个ipynb文件，用于编写并执行代码。

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/64175004-973f-479d-8cda-edd07a97f45b.png)

3.导入实验所需要的模块

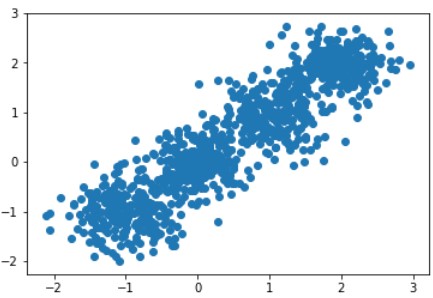
[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. %matplotlib inline
4. from sklearn.datasets.samples\_generator **import** make\_blobs

4.使用make\_blobs函数创建X为样本特征，Y为样本簇类别的1000个样本，设置每个样本2个特征，共4个簇，簇中心在[-1,-1], [0,0],[1,1], [2,2]，并看看数据的分布图。

[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

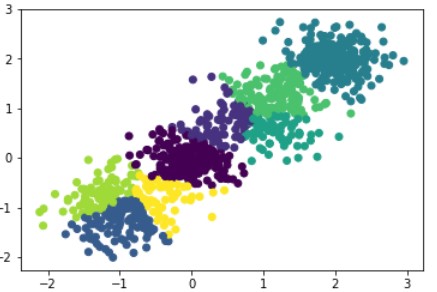
1. X, y = make\_blobs(n\_samples=1000, n\_features=2, centers=[[-1,-1], [0,0], [1,1], [2,2]], cluster\_std=[0.4, 0.3, 0.4, 0.3], random\_state =9)
2. plt.scatter(X[:, 0], X[:, 1], marker='o')
3. plt.show()

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/6d5598bf-3c5b-4352-8171-6058a2c095cc.jpg)

5.用BIRCH算法来聚类，首先我们选择不输入可选的类别数K，做出聚类效果，并使用Calinski-Harabasz分数评估模型好坏。

[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

1. from sklearn.cluster **import** Birch
2. y\_pred=Birch(n\_clusters=None).fit\_predict(X)
3. plt.scatter(X[:,0],X[:,1],c=y\_pred)
4. plt.show()
5. from sklearn **import** metrics
6. print("Calinski-Harabasz Score",metrics.calinski\_harabaz\_score(X,y\_pred))

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/e033f858-2091-40dc-9f0a-9a8fd039510e.jpg)

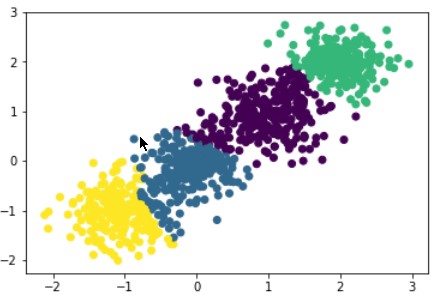
对应的Calinski-Harabasz 分数输出为：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/f776818c-a353-4183-b452-06f87c2233ac.png)

6.由于数据是4个簇随机产生的，因此我们可以通过输入可选的类别数4来看看BIRCH聚类的输出。代码如下：

[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

1. y\_pred=Birch(n\_clusters=4).fit\_predict(X)
2. plt.scatter(X[:,0],X[:,1],c=y\_pred)
3. plt.show()
4. print("Calinski-Harabasz Score:",metrics.calinski\_harabaz\_score(X,y\_pred))

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/93ecef20-563a-4f12-9395-c8f61281c810.jpg)

对应的Calinski-Harabasz 分数输出为：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/1685d97e-9713-4a01-8428-a6d3aa21aca4.png)

结论：由于分数值越大则聚类效果越好，可见如果我们不输入类别数的话，在某些时候BIRCH算法的聚类效果并不一定好，因此这个可选的类别数K一般还是需要调参的。

7.上面我们的参数threshold与参数branching\_factor都是使用的默认值（默认的threshold值0.5和默认的branching\_factor值50.）现在我们来设置参数threshold和参数branching\_factor的值，将threshold从0.5降低为0.3，让BIRCH算法第一阶段的CF Tree规模变大，并观察Calinski-Harabasz 分数。

[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

1. y\_pred = Birch(n\_clusters = 4, threshold = 0.3).fit\_predict(X)
2. print("Calinski-Harabasz Score", metrics.calinski\_harabaz\_score(X, y\_pred))

对应的Calinski-Harabasz 分数输出为：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/f9b40b41-95ff-41ae-82a8-48b892ffb603.png)

结论：可见此时的聚类效果有了进一步的提升

8.那么是不是threshold越小越好呢？我们看看threshold从0.3降低为0.1时的情况。

[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

1. y\_pred = Birch(n\_clusters = 4, threshold = 0.1).fit\_predict(X)
2. print("Calinski-Harabasz Score", metrics.calinski\_harabaz\_score(X, y\_pred))

对应的Calinski-Harabasz 分数输出为：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/cf19452c-a382-4dd3-aa94-337d5f5db45e.png)

结论：也就是说threshold不是越小聚类效果越好。

9.我们基于threshold为0.3的情况，调试branching\_factor，将branching\_factor从50降低为20，让BIRCH算法第一阶段的CF Tree规模变大。

[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

1. y\_pred = Birch(n\_clusters = 4, threshold = 0.3, branching\_factor = 20).fit\_predict(X)
2. print("Calinski-Harabasz Score", metrics.calinski\_harabaz\_score(X, y\_pred))

对应的Calinski-Harabasz 分数输出为：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/aa167f87-242e-4471-97ba-eb24b4d77c33.png)

结论：可见调试branching\_factor也可以让聚类分数提高。

11.那么和threshold类似，是不是branching\_factor越小越好呢？我们将branching\_factor从20降低为10，观察聚类分数：

[view plain](https://www.ipieuvre.com/e/407/311/9539) [copy](https://www.ipieuvre.com/e/407/311/9539)

1. y\_pred = Birch(n\_clusters = 4, threshold = 0.3, branching\_factor = 10).fit\_predict(X)
2. print("Calinski-Harabasz Score", metrics.calinski\_harabaz\_score(X, y\_pred))

对应的Calinski-Harabasz 分数输出为：

[](https://www.ipieuvre.com/doc/exper/3670b15b-91ad-11e9-beeb-00215ec892f4/img/991cbf83-97e3-487d-b2f3-766f13985b55.png)

结论：也就是说和threshold类似，branching\_factor不是越小聚类效果越好，需要调参。