In [ ]:

```
dbscan.components_
```

In [ ]:

```
dbscan.fit_predict(iris.data)
```

In [ ]:

```
# 增大距离参数
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps = 1).fit(iris.data)
dbscan.labels_
```

10.7 实战练习

将iris数据集的案例顺序彻底随机化，然后重新使用BIRCH方法进行聚类。列出随机化以后聚类结果和真实类别间的交叉表，并且和按照原顺序得到的BIRCH聚类结果和真实类别的交叉表作比较，思考案例顺序随机化处理在BIRCH方法中的重要性。

> 提示：交叉表描述和案例随机化均可以使用Pandas中的功能完成。

将iris数据集的案例顺序彻底随机化，使用K-Means方法进行聚类操作，并比较随机化前后的聚类结果，思考为什么会和BIRCH方法存在这种差异。

进一步梳理一下，在聚类分析的各种方法中，哪些方法是必须要求案例事先随机化的。

11 最近邻分析

11.1 最近邻分析的基本原理

11.2 最近邻分类

11.2.1 KNeighborsClassifier

基于每个点的k个最近邻完成实现分类。

class sklearn.neighbors.KNeighborsClassifier(

```
    n_neighbors = 5 : 需要考虑的临近点数量
    weights = 'uniform' : 案例权重的设定方法
        'uniform' : 各案例等权重
        'distance' : 权重与距离成反比关系，此时越近的案例权重越大
        [callable] : 用户自定义函数
    algorithm = 'auto' : 具体使用的最近邻算法
        {'auto', 'ball_tree', 'kd_tree', 'brute'}

    leaf_size = 30 : BallTree或KDTree中使用的叶节点大小\
    p = 2 : minkowski距离使用的power参数，为2时等价于欧氏距离
    metric = 'minkowski' : 使用的距离测量方法

    metric_params = None, n_jobs = 1

)
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier

knc = KNeighborsClassifier()
knc.fit(iris.data, iris.target)
```

In [ ]:

```
knc.predict(iris.data)
```

In [ ]:

```
knc.score(iris.data, iris.target)
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier

knc = KNeighborsClassifier(n_neighbors = 10)
knc.fit(iris.data, iris.target)

knc.score(iris.data, iris.target)
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier

knc = KNeighborsClassifier(n_neighbors = 20)
knc.fit(iris.data, iris.target)

knc.score(iris.data, iris.target)
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier

knc = KNeighborsClassifier(n_neighbors = 30)
knc.fit(iris.data, iris.target)

knc.score(iris.data, iris.target)
```

11.2.2 RadiusNeighborsClassifier

基于每个点的固定半径r内的邻居数量实现分类。

如果数据分布不均匀，该方法可能更好一些

class sklearn.neighbors.RadiusNeighborsClassifier(

```
radius = 1.0 : 使用的相邻案例查询半径
weights = 'uniform', algorithm = 'auto', leaf_size = 30, p = 2
metric = 'minkowski', outlier_label = None, metric_params = None
```

)

In [ ]:

```
from sklearn.neighbors import RadiusNeighborsClassifier

knc = RadiusNeighborsClassifier()
knc.fit(iris.data, iris.target)

knc.score(iris.data, iris.target)
```

In [ ]:

```
from sklearn.neighbors import RadiusNeighborsClassifier

knc = RadiusNeighborsClassifier(radius = 3.0)
knc.fit(iris.data, iris.target)

knc.score(iris.data, iris.target)
```

In [ ]:

```
from sklearn.neighbors import RadiusNeighborsClassifier

knc = RadiusNeighborsClassifier(radius = 0.5)
knc.fit(iris.data, iris.target)

knc.score(iris.data, iris.target)
```

In [ ]:

```
from sklearn.neighbors import RadiusNeighborsClassifier

knc = RadiusNeighborsClassifier(radius = 0.1)
knc.fit(iris.data, iris.target)

knc.score(iris.data, iris.target)
```

11.3 最近邻回归

在sklearn中，相应案例的预测结果由它的最近邻标签的数值平均而来。

KNeighborsRegressor：基于每个案例点周围的k个最近邻加以计算
RadiusNeighborsRegressor：基于每个案例点的固定半径r内的最近邻加以计算

class sklearn.neighbors.KNeighborsRegressor(

```
n_neighbors = 5, weights = 'uniform', algorithm = 'auto'
leaf_size = 30, p = 2, metric = 'minkowski'
metric_params = None, n_jobs = 1
```

)

class sklearn.neighbors.RadiusNeighborsRegressor(

```
radius = 1.0, weights = 'uniform', algorithm = 'auto'
leaf_size = 30, p = 2, metric = 'minkowski', metric_params = None
```

)

In [ ]:

```
from sklearn.neighbors import KNeighborsRegressor

knr = KNeighborsRegressor()
knr.fit(boston.data, boston.target)
```

In [ ]:

```
knr.score(boston.data, boston.target)
```

In [ ]:

```
KNeighborsRegressor(n_neighbors = 10).fit(boston.data,
                          boston.target).score(boston.data, boston.target)
```

In [ ]:

```
KNeighborsRegressor(n_neighbors = 2).fit(boston.data,
                          boston.target).score(boston.data, boston.target)
```

In [ ]:

```
# 通过数据的标准化消除量纲不同的影响
from sklearn import preprocessing

X_scaled = preprocessing.scale(boston.data)
```

In [ ]:

```
KNeighborsRegressor().fit(X_scaled,
                          boston.target).score(X_scaled, boston.target)
```

In [ ]:

```
KNeighborsRegressor(n_neighbors = 10).fit(X_scaled,
                          boston.target).score(X_scaled, boston.target)
```

11.4 无监督最近邻分析

无监督knn返回的实际上是案例所对应的所有近邻案例的列表。

class sklearn.neighbors.NearestNeighbors(

```
    n_neighbors = 5 : kneighbors方法中需要考虑的临近点数量
    radius = 1.0 : radius_neighbors方法中使用的范围查询半径
    algorithm = 'auto' : 具体使用的最近邻算法
        {'auto', 'ball_tree', 'kd_tree', 'brute'}
    leaf_size = 30, metric = 'minkowski' : 使用的距离测量方法

    p = 2, metric_params = None, n_jobs = 1
```

)

sklearn.neighbors.NearestNeighbors类的方法：

```
    fit(X[, y])     Fit the model using X as training data
    get_params([deep])     Get parameters for this estimator.
    kneighbors([X, n_neighbors, return_distance]) : 案例的K个近邻
    kneighbors_graph([X, n_neighbors, mode]) : 案例K个近邻的矩阵
    radius_neighbors([X, radius, return_distance]) : 案例半径r内的近邻
    radius_neighbors_graph([X, radius, mode]) : 案例半径r内近邻的矩阵
    set_params(**params)     Set the parameters of this estimator.
```

In [ ]:

```
from sklearn.neighbors import NearestNeighbors

knn = NearestNeighbors()
knn.fit(iris.data)
```

In [ ]:

```
knn.kneighbors(iris.data[:1])
```

In [ ]:

```
knn.kneighbors_graph(iris.data[:1]).toarray()
```

In [ ]:

```
knn.radius_neighbors(iris.data[:1])
```

In [ ]:

```
knn.radius_neighbors(iris.data[:1], radius = 0.2)
```

11.5 实战练习

对iris数据进行标化，然后重新拟合俩终最近邻分类方法，观察其分析结果的变化，并思考原因。

对boston数据进行KNN回归，并进行参数调优，找到最优模型。

提示：需要将数据拆分为训练集和验证集进行结果验证。

尝试使用SVM分类方法对logit表单数据进行建模分析。

12 生存分析

12.1 生存分析的基本概念

12.2 计算生存概率

12.2.1 生存曲线的计算

class statsmodels.duration.survfunc.SurvfuncRight(

```
time : 时间变量
status : 生存结局变量，1代表事件发生，0代表截尾
entry = None : 进入时间变量，在该时点之前案例并未暴露在风险中
title = None : 生存分析图表中使用的标题
freq_weights = None
exog = None : 生存状态的影响因素
bw_factor = 1.0 : Band-width multiplier for kernel-based estimation
```

)

statsmodels.duration.survfunc.SurvfuncRight类的方法：

```
plot([ax]) : 生存分析曲线
quantile(p) : 指定生存概率所对应的生存时间
quantile_ci(p[, alpha, method]) : 对应生存时间的可信区间
simultaneous_cb([alpha, method, transform]) : 生存函数的置信带
summary() : 模型分析结果的汇总
```