```
decision_function(X) : 各样本对各类别的决策函数值
densify() : 将系数矩阵转换为标准格式
sparsify() : 将系数矩阵转换为稀疏格式
fit(X, y[, coef_init, intercept_init, …])
get_params([deep])
partial_fit(X, y[, classes, sample_weight])
predict(X)
score(X, y[, sample_weight])
set_params(*args, **kwargs)
```

In [ ]:

```
from sklearn.preprocessing import scale

ZX = scale(iris.data)
```

In [ ]:

```
from sklearn.linear_model import SGDClassifier

sgdcls = SGDClassifier(max_iter = 100)
sgdcls.fit(ZX, iris.target)
```

In [ ]:

```
sgdcls.score(ZX, iris.target)
```

In [ ]:

```
sgdcls.predict(ZX)[:10]
```

In [ ]:

```
sgdcls.decision_function(ZX)[:5]
```

5.6 实战练习

尝试在使用MLP回归分析bosston数据时，将连接函数改为'identity'，并且限制神经元数量，找到神经元数量和最终模型准确率（score）之间的关系，并和上一章中的线性回归模型作比较，思考其中的原因。

使用各种不同的随机种子对iris数据拟合树模型，汇总得到的feature_*importances*以及模型评分情况，思考树模型对样本量的需求是怎样的。

6 聚类模型的训练

6.1 聚类模型概述

6.2 K-均值聚类

class sklearn.cluster.KMeans(

```
    n_clusters = 8
    init = 'k-means++' : 'k-means++'/'random'/ndarray，初始类中心位置
        'k-means++' : 采用优化后的算法确定类中心
        'random' : 随机选取k个案例作为初始类中心
        ndarray : (n_clusters, n_features)格式提供的初始类中心位置

    n_init = 10, max_iter = 300, tol = 0.0001
    precompute_distances = 'auto' : {'auto', True, False}
        是否预先计算距离，分析速度更快，但需要更多内存
        'auto' : 如果n_samples*n_clusters > 12 million，则不事先计算距离
    verbose = 0, random_state = None, copy_x = True, n_jobs = 1
    algorithm = 'auto' : 'auto', 'full' or 'elkan'，具体使用的算法
        'full' : 经典的EM风格算法
        'elkan' : 使用三角不等式，速度更快，但不支持稀疏数据
        'auto' : 基于数据类型自动选择

)
```

KMeans类的属性：

```
    cluster_centers_ : array, [n_clusters, n_features]
    labels_ :
    inertia_ : float，各样本和其最近的类中心距离之和
```

注意：对于样本量超过1万的情形，建议使用MiniBatchKMeans，算法改进为在线增量，速度会更快

In [ ]:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3, random_state = 0).fit(iris.data)
kmeans.labels_
```

In [ ]:

```
kmeans.cluster_centers_
```

In [ ]:

```
kmeans.predict([iris.data[0],
                iris.data[100]])
```

6.3 Birch聚类

class sklearn.cluster.Birch(

```
    threshold = 0.5 : float, 每个子类的辐射半径
    branching_factor = 50 : int, 每个节点容许纳入的最大子类数
    n_clusters = 3 : int, 最终的类别数
    compute_labels = True : 是否每次拟合时都计算类别标签
    copy = True

)
```

Birch类的属性：

```
root_ : _CFNode，CF树的根节点
dummy_leaf_ : _CFNode，所有叶节点的起点
subcluster_centers_ : ndarray，所有子类的中心位置
subcluster_labels_ : ndarray，所有子类的最终标签
labels_ : ndarray，shape (n_samples,) 所有案例的最终标签
```

In [ ]:

```
from sklearn.cluster import Birch

birch = Birch(n_clusters = 3).fit(iris.data)
birch.labels_
```

In [ ]:

```
birch.subcluster_centers_
```

In [ ]:

```
birch.subcluster_labels_
```

In [ ]:

```
birch.predict([iris.data[0],
               iris.data[100]])
```

In [ ]:

```
dbscan.components_
```

6.4 DBSCAN聚类

class sklearn.cluster.DBSCAN(

```
eps = 0.5 : float，两个案例被归为一类的最大距离
min_samples = 5 : int，案例被考虑为核心案例的最小数量
metric = 'euclidean', : string, or callable，距离的具体算法
metric_params = None : dict，距离计算方法所需的其他参数
algorithm = 'auto' : 具体使用的最近邻算法
    {'auto', 'ball_tree', 'kd_tree', 'brute'}
leaf_size = 30 : int，BallTree或cKDTree中的最大叶子数量
p = None : float，Minkowski距离中的指数
n_jobs = 1
```

)

DBSCAN类的属性：

```
core_sample_indices_ : array, shape = [n_core_samples]
components_ : array, shape = [n_core_samples, n_features]，核心样本
labels_ : array, shape = [n_samples]，各类别标签，噪声样本为-1
```

注意：DBSCAN无predict方法，只有fit_predict方法

In [ ]:

```
from sklearn.cluster import DBSCAN

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

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_
```

6.5 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：交叉表描述可使用Pandas中的功能完成，也可以使用7.1.1中将要介绍的混淆矩阵完成。案例随机化可以使用Pandas中的功能完成，如果对Pandas不熟悉，也可以参考8.2.1节中的程序实现方式。

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。

7 评估模型效果