Sklearn常用机器学习算法参数详解

更新日期：2019-08-17

讲师简介：菊安酱，CDA数据分析师讲师

线性回归

```
from sklearn.linear_model import LinearRegression

LinearRegression(fit_intercept=True,normalize=False,copy_X=True,n_jobs=1)

'''
参数含义：
1.fit_intercept:布尔值，指定是否需要计算线性回归中的截距，即b值。如果为False,那么不计算b值。
2.normalize:布尔值。如果为False，那么训练样本会进行归一化处理。
3.copy_X：布尔值。如果为True，会复制一份训练数据。
4.n_jobs:一个整数。任务并行时指定的CPU数量。如果取值为-1则使用所有可用的CPU。

属性
1.coef_:权重向量
2.intercept_:截距b值

方法：
1.fit(X,y)：训练模型。
2.predict(X)：用训练好的模型进行预测，并返回预测值。
3.score(X,y)：返回预测性能的得分。计算公式为：score=(1 - u/v)
其中u=((y_true - y_pred) ** 2).sum()，v=((y_true - y_true.mean()) ** 2).sum()
score最大值是1，但有可能是负值(预测效果太差)。score越大，预测性能越好。
'''
```

岭回归

```
# 加入L2正则化的线性回归
from sklearn.linear_model import Ridge

Ridge(alpha=1.0, fit_intercept=True, normalize=False,copy_X=True, max_iter=None,
tol=1e-3, solver="auto",random_state=None)

'''
参数含义：
1.alpha:正则项系数，值越大正则项占比越大。初始值建议一开始设置为0，这样先确定一个比较好的学习
率，学习率一旦确定，给alpha一个较小的值，然后根据验证集上的准确率，增大或减小10倍。10倍是粗调
节，当确定了合适的数量级后，再在同一个数量级内细调节。
2.fit_intercept：布尔值，指定是否需要计算截距b值。False则不计算b值。
3.normalize:布尔值。如果等于True，模型训练之前会把数据归一化。
    这里归一化有两个好处：
    (1):提升模型的收敛速度,减少寻找最优解的时间。
    (2)提升模型的精度
4.copy_X:布尔值。如果设置为True，则会复制一份训练数据。
5.max_iter：整数。指定了最大迭代次数。如果为None，则采用默认值。
6.tol:阈值。判断迭代是否收敛或者是否满足精度的要求。
7.solver:字符串。指定求解最优化问题的算法。
    (1).solver='auto',根据数据集自动选择算法。
    (2).solver='svd',采用奇异值分解的方法来计算
    (3).solver='cholesky',采用scipy.linalg.solve函数求解最优解。
    (4).solver='sparse_cg',采用scipy.sparse.linalg.cg函数来求取最优解。
    (5).solver='sag',采用Stochastic Average Gradient descent算法求解最优化问题。
8.random_state:一个整数或者一个RandomState实例，或者为None。它在solver="sag"时使用。
    (1).如果为整数，则它指定了随机数生成器的种子。
    (2).如果为RandomState实例，则指定了随机数生成器。
    (3).如果为None，则使用默认的随机数生成器。

属性：
1.coef_：权重向量。
2.intercept_：截距b的值。
3.n_iter_:实际迭代次数。

方法：
1.fit(X,y)：训练模型。
2.predict(X):用训练好的模型去预测，并且返回预测值。
3.score(X,y):返回预测性能的得分。计算公式为：score=(1 - u/v)
            其中u=((y_true - y_pred) ** 2).sum()，v=((y_true - y_true.mean()) **
2).sum()
            score最大值是1，但有可能是负值(预测效果太差)。score越大，预测性能越好。
'''
```

Lasso回归

```
# 加入L1正则化的线性回归
from sklearn.linear_model import Lasso
Lasso(alpha=1.0, fit_intercept=True, normalize=False, precompute=False,
copy_X=True, max_iter=1000,
      tol=1e-4, warm_start=False, positive=False,random_state=None,
selection='cyclic')
'''
1.alpha：正则化项系数
2.fit_intercept：布尔值，指定是否需要计算截距b值。False则不计算b值。
3.max_iter:指定最大迭代次数。
4.normalize：布尔值。如果等于True，模型训练之前会把数据归一化。
    这里归一化有两个好处：
     (1):提升模型的收敛速度,减少寻找最优解的时间。
     (2)提升模型的精度。
5.precompute:一个布尔值或者一个序列。它决定是否提前计算Gram矩阵来加速计算。
6.tol:阈值。判断迭代是否收敛或者是否满足精度的要求。
7.warm_start：布尔值。如果为True，那么使用前一次训练结果继续训练。否则从头开始训练。
8.positive：布尔值。如果为True，那么强制要求权重向量的分量都为正数。
9.selection:字符串，可以是"cyclic"或"random"。它指定了当每轮迭代的时候，选择权重向量的哪个
分量来更新。
    (1)"random":更新的时候，随机选择权重向量的一个分量来更新。
    (2)"cyclic":更新的时候，从前向后依次选择权重向量的一个分量来更新。
10.random_state：一个整数或者一个RandomState实例，或者None。
    (1):如果为整数，则它指定了随机数生成器的种子。
    (2):如果为RandomState实例，则它指定了随机数生成器。
    (3):如果为None，则使用默认的随机数生成器。

属性：
1.coef_：权重向量。
2.intercept_：截距b值。
3.n_iter_：实际迭代次数。

方法：
1.fit(X,y)：训练模型。
2.predict(X):用模型进行预测，返回预测值。
3.score(X,y):返回预测性能的得分。计算公式为：score=(1 - u/v)
            其中u=((y_true - y_pred) ** 2).sum()，v=((y_true - y_true.mean()) **
2).sum()
            score最大值是1，但有可能是负值(预测效果太差)。score越大，预测性能越好。
'''
```

Elastic Net

```
'''
ElasticNet回归是对Lasso回归和岭回归的融合，其正则化项是L1范数和L2范数的一个权衡。
正则化项为: alpha * l1_ratio * ||w||_1 + 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
'''
from sklearn.linear_model import ElasticNet

ElasticNet(alpha=1.0, l1_ratio=0.5, fit_intercept=True,
normalize=False,precompute=False, max_iter=1000,copy_X=True, tol=1e-4,
warm_start=False, positive=False,random_state=None, selection='cyclic')

'''
参数含义：
1.alpha:正则化项中alpha值。
2.l1_ratio:正则化项中的l1_ratio值。
3.fit_intercept:布尔值，指定是否需要计算截距b值。False则不计算b值。
4.max_iter：指定最大迭代次数。
5.normalize：布尔值。如果等于True，模型训练之前会把数据归一化。
    这里归一化有两个好处：
    (1):提升模型的收敛速度,减少寻找最优解的时间。
    (2)提升模型的精度。
6.copy_X:布尔值。如果设置为True，则会复制一份训练数据。
7.precompute:一个布尔值或者一个序列。它决定是否提前计算Gram矩阵来加速计算。
8.tol:阈值。判断迭代是否收敛或者是否满足精度的要求。
9.warm_start:布尔值。如果为True，那么使用前一次训练结果继续训练。否则从头开始训练。
10.positive:布尔值。如果为True，那么强制要求权重向量的分量都为正数。
11.selection:字符串，可以是"cyclic"或"random"。它指定了当每轮迭代的时候，选择权重向量的哪个分量来更新。
    (1)"random":更新的时候，随机选择权重向量的一个分量来更新。
    (2)"cyclic":更新的时候，从前向后依次选择权重向量的一个分量来更新。
12.random_state:一个整数或者一个RandomState实例，或者None。
    (1):如果为整数，则它指定了随机数生成器的种子。
    (2):如果为RandomState实例，则它指定了随机数生成器。
    (3):如果为None，则使用默认的随机数生成器。

属性：
1.coef_：权重向量。
2.intercept_：截距b值。
3.n_iter_：实际迭代次数。

方法：
1.fit(X,y)：训练模型。
2.predict(X):用模型进行预测，返回预测值。
3.score(X,y):返回预测性能的得分。计算公式为：score=(1 - u/v)
            其中u=((y_true - y_pred) ** 2).sum()，v=((y_true - y_true.mean()) **
2).sum()
            score最大值是1，但有可能是负值(预测效果太差)。score越大，预测性能越好。
'''
```

逻辑回归

```
from sklearn.linear_model import LogisticRegression

LogisticRegression(penalty='l2',dual=False,tol=1e-4,C=1.0,
                   fit_intercept=True,intercept_scaling=1,class_weight=None,
                   random_state=None,solver='liblinear',max_iter=100,
                   multi_class='ovr',verbose=0,warm_start=False,n_jobs=1)


'''
参数含义：
1.penalty:字符串，指定了正则化策略。默认为"l2"
    (1)如果为"l2",则优化的目标函数为：0.5*||w||^2_2+C*L(w),C>0， L(w)为极大似然函数。
    (2)如果为"l1",则优化的目标函数为||w||_1+C*L(w),C>0， L(w)为极大似然函数。
2.dual:布尔值。默认为False。如果等于True，则求解其对偶形式。
  只有在penalty="l2"并且solver="liblinear"时才有对偶形式。如果为False，则求解原始形式。
  当n_samples > n_features，偏向于dual=False。
3.tol:阈值。判断迭代是否收敛或者是否满足精度的要求。
4.C:float,默认为1.0.指定了正则化项系数的倒数。必须是一个正的浮点数。C值越小,正则化项就越大。
5.fit_intercept:bool值。默认为True。如果为False,就不会计算b值。
6.intercept_scaling：float, default 1。
  只有当solver="liblinear"并且  fit_intercept=True时，才有意义。
  在这种情况下，相当于在训练数据最后一列增加一个特征，该特征恒为1。其对应的权重为b。
7.class_weight：dict or 'balanced', default: None。
    (1)如果是字典，则给出每个分类的权重。按照{class_label: weight}这种形式。
    (2)如果是"balanced"：则每个分类的权重与该分类在样本集中出现的频率成反比。
       n_samples / (n_classes * np.bincount(y))
    (3)如果未指定，则每个分类的权重都为1。
8.random_state: int, RandomState instance or None, default: None
    (1):如果为整数，则它指定了随机数生成器的种子。
    (2):如果为RandomState实例，则它指定了随机数生成器。
    (3):如果为None，则使用默认的随机数生成器。
9.solver: 字符串，指定求解最优化问题的算法。
{'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'},default: 'liblinear'
   (1)solver='liblinear',对于小数据集，'liblinear'是很好的选择。
       对于大规模数据集，'sag'和'saga'处理起来速度更快。
   (2)solver='newton-cg',采用牛顿法
   (3)solver='lbfgs',采用L-BFGS拟牛顿法。
   (4)solver='sag',采用Stochastic Average Gradient descent算法。
   (5)对于多分类问题，只有'newton-cg'，'sag'，'saga'和'lbfgs'处理多项损失;
      'liblinear'仅限于'ovr'方案。
   (6)newton-cg', 'lbfgs' and 'sag' 只能处理 L2 penalty,
      'liblinear' and 'saga' 能处理 L1 penalty。
10.max_iter: 指定最大迭代次数。default: 100。只对'newton-cg', 'sag' and 'lbfgs'适用。
11.multi_class：{'ovr', 'multinomial'}, default: 'ovr'。指定对分类问题的策略。
    (1)multi_class='ovr',采用'one_vs_rest'策略。
    (2)multi_class='multinomal',直接采用多分类逻辑回归策略。
12.verbose: 用于开启或者关闭迭代中间输出日志功能。
13.warm_start: 布尔值。如果为True，那么使用前一次训练结果继续训练。否则从头开始训练。
14.n_jobs: int, default: 1。指定任务并行时的CPU数量。如果为-1，则使用所有可用的CPU。
```

```
属性：
1.coef_：权重向量。
2.intercept_：截距b值。
3.n_iter_：实际迭代次数。

方法：
1.fit(X,y)：训练模型。
2.predict(X)：用训练好的模型进行预测，并返回预测值。
3.predict_log_proba(X)：返回一个数组，数组元素依次是X预测为各个类别的概率的对数值。
4.predict_proba(X)：返回一个数组，数组元素依次是X预测为各个类别的概率值。
5.score(X,y)：返回预测的准确率。
'''
```

svm.LinearSVC

```
# LinearSVC实现了线性分类支持向量机
from sklearn.svm import LinearSVC

LinearSVC(penalty='l2',loss='squared_hinge',dual=True,tol=0.0001,C=1.0,multi_cla
ss='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,verbose=0,
random_state=None, max_iter=1000)

'''
参数：
1.C：一个浮点数，罚项系数。C值越大对误分类的惩罚越大。
2.loss：字符串，表示损失函数，可以为如下值：
   'hinge'：此时为合页损失(标准的SVM损失函数)，
   'squared_hinge'：合页损失函数的平方。
3.penalty：字符串，指定'l1'或者'l2'，罚项范数。默认为'l2'(他是标准的SVM范数)。
4.dual：布尔值，如果为True，则解决对偶问题，如果是False，则解决原始问题。
  当n_samples>n_features是，倾向于采用False。
5.tol：浮点数，指定终止迭代的阈值。
6.multi_class：字符串，指定多分类的分类策略。
   'ovr'：采用one-vs-rest策略。
   'crammer_singer'：多类联合分类，很少用，因为他计算量大，而且精度不会更佳，此时忽略
loss,penalty,dual等参数。
7.fit_intecept：布尔值，如果为True，则计算截距，即决策树中的常数项，否则忽略截距。
8.intercept_scaling：浮点值，若提供了，则实例X变成向量[X,intercept_scaling]。
  此时相当于添加一个人工特征，该特征对所有实例都是常数值。
9.class_weight：可以是个字典或者字符串'balanced'。指定个各类的权重，若未提供，则认为类的权
重为1。
  如果是字典，则指定每个类标签的权重。如果是'balanced'，则每个累的权重是它出现频数的倒数。
10.verbose：一个整数，表示是否开启verbose输出。
11.random_state：一个整数或者一个RandomState实例，或者None。
    如果为整数，指定随机数生成器的种子。
    如果为RandomState，指定随机数生成器。
    如果为None，指定使用默认的随机数生成器。
12.max_iter：一个整数，指定最大迭代数。

属性：
1.coef_：一个数组，它给出了各个特征的权重。
2.intercept_：一个数组，它给出了截距。

方法
1.fit(X,y)：训练模型。
2.predict(X)：用模型进行预测，返回预测值。
3.score(X,y)：返回在(X,y)上的预测准确率。
'''
```

svm.SVC

```
# SVC实现了非线性分类支持向量机
from sklearn.svm import SVC

SVC(C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0,
shrinking=True,
    probability=False, tol=0.001, cache_size=200, class_weight=None,
verbose=False,
    max_iter=-1, decision_function_shape='ovr', random_state=None)
'''
参数：
1.C：惩罚参数，默认值是1.0
  C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，
  这样对训练集测试时准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，允许容错，将他们当成噪声点，泛化能力较强。
2.kernel ：核函数，默认是rbf，可以是‘linear’，‘poly’，‘rbf’，‘sigmoid’，
    ‘precomputed’(表示使用自定义的核函数矩阵)
3.degree ：多项式poly函数的阶数，默认是3，选择其他核函数时会被忽略。
4.gamma ： ‘rbf’,‘poly’ 和‘sigmoid’的核函数参数。默认是’auto’，则会选择1/n_features
5.coef0 ：核函数的常数项。对于‘poly’和 ‘sigmoid’有用。
6.probability ：是否采用概率估计.这必须在调用fit()之前启用，并且会fit()方法速度变慢。默认为False
7.shrinking ：是否采用启发式收缩方式方法，默认为True
8.tol ：停止训练的误差值大小，默认为1e-3
9.cache_size ：指定训练所需要的内存，以MB为单位，默认为200MB。
10.class_weight ：类别的权重，字典形式传递。设置第几类的参数C为weight*C(SVC中的C)
11.verbose ：用于开启或者关闭迭代中间输出日志功能。
12.max_iter ：最大迭代次数。-1为无限制。
13.decision_function_shape ：‘ovo’，‘ovr’ or None，default=‘ovr’
14.random_state ：数据洗牌时的种子值，int值

属性：
1.support_ ： 支持向量的索引
2.support_vectors_ ： 支持向量
3.n_support_ ： 每个类别的支持向量的数目
4.dual_coef_ ： 一个数组，形状为[n_class-1,n_SV]。对偶问题中，在分类决策函数中每一个支持向量的系数。
5.coef_ ：一个数组，形状为[n_class-1,n_features]。原始问题中，每个特征的系数。只有在linear kernel中有效。
6.intercept_ ：一个数组，形状为[n_class*(n_class-1)/2]。决策函数中的常数项。
7.fit_status_ ：整型，表示拟合的效果，如果正确拟合为0，否则为1
8.probA_ : array, shape = [n_class * (n_class-1) / 2]
9.probB_ : array, shape = [n_class * (n_class-1) / 2]

方法：
1.decision_function(X)：样本X到决策平面的距离
2.fit(X, y[, sample_weight])：训练模型
3.get_params([deep])：获取参数
4.predict(X)：预测
5.score(X, y[, sample_weight])：返回预测的平均准确率
6.set_params(**params)：设定参数
'''
```

svm.LinearSVR

```
from sklearn.svm import LinearSVR

LinearSVR(epsilon=0.0, tol=0.0001, C=1.0, loss='epsilon_insensitive',
fit_intercept=True,       intercept_scaling=1.0, dual=True, verbose=0,
random_state=None, max_iter=1000)

'''
参数：
1.C: 一个浮点值，罚项系数。
2.loss:字符串，表示损失函数，可以为：
    ‘epsilon_insensitive’:此时损失函数为L_ε(标准的SVR)
    ‘squared_epsilon_insensitive’:此时损失函数为LεLε
3.epsilon: 浮点数，用于lose中的εε参数。
4.dual: 布尔值。如果为True，则解决对偶问题，如果是False则解决原始问题。
5.tol: 浮点数，指定终止迭代的阈值。
6.fit_intercept: 布尔值。如果为True，则计算截距，否则忽略截距。
7.intercept_scaling: 浮点值。如果提供了，则实例X变成向量[X,intercept_scaling]。此时相当
于添加了一个人工特征，
  该特征对所有实例都是常数值。
8.verbose: 是否输出中间的迭代信息。
9.random_state: 指定随机数生成器的种子。
10.max_iter: 一个整数，指定最大迭代次数。

属性：
1.coef_: 一个数组，他给出了各个特征的权重。
2.intercept_: 一个数组，他给出了截距，及决策函数中的常数项。

方法：
1.fit(X,y): 训练模型。
2.predict(X): 用模型进行预测，返回预测值。
3.score(X,y): 返回性能得分。
'''
```

svm.SVR

```
from sklearn.svm import SVR

SVR(kernel='rbf', degree=3, gamma='auto', coef0=0.0, tol=0.001, C=1.0,
epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)

'''
参数：
1.C：一个浮点值，罚项系数。
2.epsilon：浮点数，用于lose中的ϵ参数。
3.kernel：一个字符串，指定核函数。
   'linear' ：线性核
   'poly'：多项式核
   'rbf'：默认值，高斯核函数
   'sigmoid'：Sigmoid核函数
   'precomputed'：表示支持自定义核函数
4.degree：一个整数，指定当核函数是多项式核函数时，多项式的系数。对于其它核函数该参数无效。
5.gamma：一个浮点数。当核函数是'rbf','poly','sigmoid'时，核函数的系数。如果为'auto'，则表示系数为1/n_features。
6.coef0：浮点数，用于指定核函数中的自由项。只有当核函数是'poly'和'sigmoid'时有效。
7.shrinking：布尔值。如果为True，则使用启发式收缩。
8.tol：浮点数，指定终止迭代的阈值。
9.cache_size：浮点值，指定了kernel cache的大小，单位为MB。
10.verbose：指定是否开启verbose输出。
11.max_iter：一个整数，指定最大迭代步数。

属性：
1.support_：一个数组，形状为[n_SV]，支持向量的下标。
2.support_vectors_：一个数组，形状为[n_SV,n_features]，支持向量。
3.n_support_：一个数组，形状为[n_class]，每一个分类的支持向量个数。
4.dual_coef_：一个数组，形状为[n_class-1,n_SV]。对偶问题中，在分类决策函数中每一个支持向量的系数。
5.coef_：一个数组，形状为[n_class-1,n_features]。原始问题中，每个特征的系数。只有在linear kernel中有效。
5.intercept_：一个数组，形状为[n_class*(n_class-1)/2]。决策函数中的常数项。

方法：
1.fit(X,y)：训练模型。
2.predict(X)：用模型进行预测，返回预测值。
3.score(X,y)：返回性能得分。
'''
```

K近邻分类器

```
from sklearn.neighbors import KNeighborsClassifier

KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

'''
参数:
1.n_neighbors: 一个整数，指定k值。
2.weights: 一字符串或者可调用对象，指定投票权重类型。即这些邻居投票权可以为相同或者不同。
   uniform: 本节点的所有邻居节点的投票权重都相等。
   distance: 本节点的所有邻居节点的投票权重与距离成反比，即越近节点，其投票权重越大。
   [callable]: 一个可调用对象，它传入距离的数组，返回同样形状的权重数组。
3.algorithm: 一个字符串，指定最近邻的算法，可以为下:
   ball_tree: 使用BallTree算法。
   kd_tree: 使用KDTree算法。
   brute: 使用暴力搜索算法。
   auto: 自动决定最合适算法。
4.leaf_size: 一个整数，指定BallTree或者KDTree叶节点的规模。它影响树的构建和查询速度。
5.metric: 一个字符串，指定距离度量。默认为'minkowski'(闵可夫斯基)距离。
6.p: 整数值。
   p=1:  对应曼哈顿距离。
   p=2: 对应欧氏距离。
7.n_jobs: 并行性。默认为-1表示派发任务到所有计算机的CPU上。

方法:
1.fit(X,y): 训练模型。
2.predict(X): 预测模型。
3.score(X,y): 返回在(X,y)上预测的准确率(accuracy)。
4.predict_proba(X): 返回样本为每种标记的概率。
5.kneighbors([X,n_neighbors,return_distace]): 返回样本点的k邻近点。
  如果return_distance=True，同时还返回到这些近邻点的距离。
6.kneighbors_graph([X,n_neighbors,mode]): 返回样本点的连接图。
'''
```

K近邻回归

```
from sklearn.neighbors import KNeighborsRegressor

KNeighborsRegressor(n_neighbors=5, weights='uniform', algorithm='auto',
leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

'''
参数:
1.n_neighbors: 一个整数，指定k值。
2.weights: 一字符串或者可调用对象，指定投票权重类型。即这些邻居投票权可以为相同或者不同。
   uniform: 本节点的所有邻居节点的投票权重都相等。
   distance: 本节点的所有邻居节点的投票权重与距离成反比，即越近节点，其投票权重越大。
   [callable]: 一个可调用对象，它传入距离的数组，返回同样形状的权重数组。
3.algorithm: 一个字符串，指定最近邻的算法，可以为下：
   ball_tree: 使用BallTree算法。
   kd_tree: 使用KDTree算法。
   brute: 使用暴力搜索算法。
   auto: 自动决定最合适算法。
4.leaf_size: 一个整数，指定BallTree或者KDTree叶节点的规模。它影响树的构建和查询速度。
5.metric: 一个字符串，指定距离度量。默认为'minkowski'(闵可夫斯基)距离。
6.p: 整数值。
   p=1:  对应曼哈顿距离。
   p=2: 对应欧氏距离。
7.n_jobs: 并行性。默认为-1表示派发任务到所有计算机的CPU上。

方法:
1.fit(X,y): 训练模型。
2.predict(X): 预测模型。
3.score(X,y): 返回在(X,y)上预测的准确率(accuracy)。
4.predict_proba(X): 返回样本为每种标记的概率。
5.kneighbors([X,n_neighbors,return_distace]): 返回样本点的k邻近点。
  如果return_distance=True，同时还返回到这些近邻点的距离。
6.kneighbors_graph([X,n_neighbors,mode]): 返回样本点的连接图。
'''
```

决策树（回归树）

```
from sklearn.tree import DecisionTreeRegressor

DecisionTreeRegressor(criterion='mse',splitter='best',max_depth=None,min_samples_split=2,min_samples_leaf=1,min_weight_fraction_leaf=0.0,max_features=None,random_state=None,max_leaf_nodes=None,min_impurity_split=1e-07,presort=False)
'''
参数：
1.criterion ： 一个字符串，指定切分质量的评价标准。默认为‘mse’，且只支持该字符串，表示均方误差。
2.splitter ： 一个字符串，指定切分原则，可以为：
    best ： 表示选择最优的切分。
    random ： 表示随机切分。
3.max_features ： 可以为整数、浮点、字符或者None，指定寻找best split时考虑的特征数量。
    如果是整数，则每次切分只考虑max_features个特征。
    如果是浮点数，则每次切分只考虑max_features*n_features个特征(max_features指定了百分比)。
    如果是字符串‘auto’，则max_features等于n_features。
    如果是字符串‘sqrt’，则max_features等于sqrt(n_features)。
    如果是字符串‘log2’，则max_features等于log2(n_features)。
    如果是字符串None，则max_features等于n_features。
4.max_depth ： 可以为整数或者None，指定树的最大深度。
  如果为None，表示树的深度不限(知道每个叶子都是纯的，即叶子结点中的所有样本点都属于一个类，
  或者叶子中包含小于min_sanples_split个样本点)。
如果max_leaf_nodes参数非None，则忽略此项。
5.min_samples_split ： 为整数，指定每个内部节点(非叶子节点)包含的最少的样本数。
6.min_samples_leaf ： 为整数，指定每个叶子结点包含的最少的样本数。
7.min_weight_fraction_leaf ： 为浮点数，叶子节点中样本的最小权重系数。
8.max_leaf_nodes ： 为整数或None，指定叶子结点的最大数量。
    如果为None，此时叶子节点数不限。如果非None，则max_depth被忽略。
9.class_weight ： 一个字典、字典的列表、字符串‘balanced’或者None，它指定了分类的权重。
  权重形式为：{class_label:weight} 如果为None，则每个分类权重都为1.
  字符串‘balanced’表示每个分类的权重是各分类在样本出现的频率的反比。
10.random_state ： 一个整数或者一个RandomState实例，或者None。
    如果为整数，则它指定了随机数生成器的种子。
    如果为RandomState实例，则指定了随机数生成器。
    如果为None，则使用默认的随机数生成器。
11.presort ： 一个布尔值，指定了是否要提前排序数据从而加速寻找最优切分的过程。
  设置为True时，对于大数据集会减慢总体训练过程，但对于小数据集或者设定了最大深度的情况下，则会加速训练过程。

属性：
1.feature_importances_ ： 给出了特征的重要程度。该值越高，则特征越重要(也称为Gini importance)。
2.max_features_ ： max_feature的推断值。
3.n_features_ ： 当执行fit后，特征的数量。
4.n_outputs_ ： 当执行fit后，输出的数量。
5.tree_ ： 一个Tree对象，即底层的决策树。

方法：
1.fit(X,y) ： 训练模型。
2.predict(X) ： 用模型预测，返回预测值。
3.score(X,y) ： 返回性能得分
'''
```

决策树（分类树）

```
from sklearn.tree import DecisionTreeClassifier

DecisionTreeClassifier(criterion='gini',splitter='best',max_depth=None,
                       min_samples_split=2,min_samples_leaf=1,
                       min_weight_fraction_leaf=0.0,max_features=None,
                       random_state=None,max_leaf_nodes=None,
                       min_impurity_decrease=0.0,min_impurity_split=1e-07,
                       class_weight=None, presort=False)


'''
参数:
1.criterion : 一个字符串，指定切分质量的评价标准。可以为:
   'gini' :表示切分标准是Gini系数。切分时选取基尼系数小的属性切分。
   'entropy' :  表示切分标准是熵。
2.splitter : 一个字符串，指定切分原则，可以为:
   best : 表示选择最优的切分。
   random :  表示随机切分。
   默认的"best"适合样本量不大的时候，而如果样本数据量非常大，此时决策树构建推荐"random"。
3.max_features : 可以为整数、浮点、字符或者None，指定寻找best split时考虑的特征数量。
   如果是整数，则每次切分只考虑max_features个特征。
   如果是浮点数，每次切分只考虑max_features*n_features个特征(max_features指定百分比)。
   如果是字符串'auto'，则max_features等于n_features。
   如果是字符串'sqrt'，则max_features等于sqrt(n_features)。
   如果是字符串'log2'，则max_features等于log2(n_features)。
   如果是字符串None，则max_features等于n_features。
4.max_depth : 可以为整数或者None，指定树的最大深度，防止过拟合
   如果为None，表示树的深度不限(知道每个叶子都是纯的，即叶子结点中的所有样本点都属于一个类，
   或者叶子中包含小于min_sanples_split个样本点)。
   如果max_leaf_nodes参数非None，则忽略此项。
5.min_samples_split : 为整数，指定每个内部节点(非叶子节点)包含的最少的样本数。
6.min_samples_leaf : 为整数，指定每个叶子结点包含的最少的样本数。
7.min_weight_fraction_leaf : 为浮点数，叶子节点中样本的最小权重系数。
8.max_leaf_nodes : 为整数或None，指定叶子结点的最大数量。
   如果为None，此时叶子节点数不限。如果非None，则max_depth被忽略。
9.min_impurity_decrease=0.0 如果该分裂导致不纯度的减少大于或等于该值，则将分裂节点。
10.min_impurity_split=1e-07, 限制决策树的增长，
11.class_weight : 一个字典、字典的列表、字符串'balanced'或者None，他指定了分类的权重。
   权重形式为:{class_label:weight} 如果为None，则每个分类权重都为1.
   字符串'balanced'表示每个分类的权重是各分类在样本出现的频率的反比。
12.random_state : 一个整数或者一个RandomState实例，或者None。
   如果为整数，则它指定了随机数生成器的种子。
   如果为RandomState实例，则指定了随机数生成器。
   如果为None，则使用默认的随机数生成器。
13.presort : 一个布尔值，指定了是否要提前排序数据从而加速寻找最优切分的过程。
  设置为True时，对于大数据集会减慢总体训练过程，但对于小数据集或者设定了最大深度的情况下，则会
加速训练过程。
```

```
属性:
1.classes_ : 分类的标签值。
2.feature_importances_ : 给出了特征的重要程度。该值越高，则特征越重要(也称为Gini
importance)。
3.max_features_ : max_feature的推断值。
4.n_classes_ : 给出了分类的数量。
5.n_features_ : 当执行fit后，特征的数量。
6.n_outputs_ : 当执行fit后，输出的数量。
7.tree_ : 一个Tree对象，即底层的决策树。


方法:
1.fit(X,y) : 训练模型。
2.predict(X) : 用模型预测，返回预测值。
3.predict_log_proba(X) : 返回一个数组，数组元素依次为X预测为各个类别的概率值的对数值。
4.predict_proba(X) : 返回一个数组，数组元素依次为X预测为各个类别的概率值。
5.score(X,y) : 返回在(X,y)上预测的准确率(accuracy)。
'''
```

GBDT分类器

```
from sklearn.ensemble import GradientBoostingClassifier

GradientBoostingClassifier(loss='deviance',learning_rate=0.1,n_estimators=100,subsample=1.0,criterion='friedman_mse',min_samples_split=2,min_samples_leaf=1,min_weight_fraction_leaf=0.0,max_depth=3,min_impurity_decrease=0.0,min_impurity_split=None,init=None,random_state=None,max_features=None,verbose=0,max_leaf_nodes=None,warm_start=False,presort='auto',
validation_fraction=0.1,n_iter_no_change=None, tol=0.0001)

'''
1.loss: 损失函数，'deviance'：对数损失函数，'exponential'：指数损失函数，只能用于二分类。
2.learning_rate：学习率
3.n_ estimators: 基学习器的个数，这里是树的颗数
4.subsample: 取值在(0, 1)之间，取原始训练集中的一个子集用于训练基础决策树
5.criterion: 'friedman_mse'改进型的均方误差;'mse'标准的均方误差; 'mae'平均绝对误差。
6.min_samples_split:一个整数，指定了每个基础决策树模型分裂所需最小样本数。
7.min_samples_leaf:一个整数，指定了每个基础决策树模型叶节点所包含的最小样本数。
8.min_weight_fraction_leaf:一个浮点数。叶节点的最小加权权重。当不提供sample_weight时，
  样本的权重是相等的。
9.max_depth:一个整数或者None，指定每一个基础决策树模型的最大深度。
    如果max_leaf_noeds不是None，则忽略此参数。
10.max_features:一个整数，浮点数或者None。代表节点分裂是参与判断的最大特征数。
    整数为个数，浮点数为所占比重。
11.max_leaf_nodes:为整数或者None，指定了每个基础决策树模型的最大叶节点数量。
12.min_impurity_split:一个浮点数，指定在树生长的过程中，节点分裂的阈值，默认为1e-7。
13.init：一个基础分类器对象或者None
14.verbose：如果为0则不输出日志，如果为1，则每隔一段时间输出日志
15.warm_start：热启动，当你训练GBM到一定程度停止时，如果你想在这个基础上接着训练，就需要用到该参数减少重复训练

属性
1.n_estimators_ : 基学习器的个数，这里是树的颗数
2.feature_importance_:一个数组，给出了每个特征的重要性（值越高重要性越大）。
3.oob_improvement_:一个数组,给出每增加一棵基础决策树,在包外估计(即测试集)的损失函数的减少值
4.train_score_:一个数组，给出每增加一棵基础决策树，在训练集上的损失函数的值。
5.loss:具体损失函数对象。
6.init:初始预测使用的分类器。
7.estimators_:一个数组，给出了每个基础决策树。

方法
1.apply(X)  Apply trees in the ensemble to X, return leaf indices.
2.decision_function(X)  Compute the decision function of X.
3.fit(X,y):训练模型。
4.get_params([deep])    Get parameters for this estimator.
5.predict(X):用模型进行预测，返回预测值。
6.predict_log_proba(X):返回一个数组，数组的元素依次是X预测为各个类别的概率的对数值。
7.predict_proba(X):返回一个数组，数组的元素依次是X预测为各个类别的概率值。
8.score(X,y):返回在（X,y）上预测的准确率。
9.set_params(**params)  设定参数.
10.staged_predict(X):返回一个数组,数组元素依次是每一轮迭代结束时尚未完成的集成分类器的预测值
11.staged_predict_proba(X):返回一个二维数组，数组元素依次是每一轮迭代结束时尚未完成的集成分类器预测X为各个类别的概率值。
'''
```

GBDT回归器

```
from sklearn.ensemble import GradientBoostingRegressor

GradientBoostingRegressor(loss='ls',learning_rate=0.1,n_estimators=100,subsample
=1.0,criterion='friedman_mse',min_samples_split=2,min_samples_leaf=1,min_weight_
fraction_leaf=0.0,
max_depth=3,min_impurity_decrease=0.0,min_impurity_split=None,init=None,random_s
tate=None, max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None,
warm_start=False, presort='auto', validation_fraction=0.1,
n_iter_no_change=None, tol=0.0001)

'''
1.loss: 损失函数，‘ls’：此时损失函数为平方损失函数。
- ‘lad’：此时使用指数绝对值损失函数。
- ‘quantile’：分位数回归（分位数指的是百分之几），采用绝对值损失。
- ‘huber’：此时损失函数为上述两者的综合，即误差较小时，采用平方损失，在误差较大时，采用绝对值损失。
2.learning_rate：学习率
3.n_ estimators: 基学习器的个数，这里是树的颗数
4.subsample: 取值在(0, 1)之间，取原始训练集中的一个子集用于训练基础决策树
5.criterion: 'friedman_mse'改进型的均方误差;'mse'标准的均方误差; 'mae'平均绝对误差。
6.min_samples_split:一个整数，指定了每个基础决策树模型分裂所需最小样本数。
7.min_samples_leaf:一个整数，指定了每个基础决策树模型叶节点所包含的最小样本数。
8.min_weight_fraction_leaf:一个浮点数。叶节点的最小加权权重。当不提供sample_weight时，样本的权重是相等的
9.max_depth:一个整数或者None，指定每一个基础决策树模型的最大深度。如果max_leaf_noeds不是None，则忽略此参数
10.max_features:一个整数，浮点数或者None。代表节点分裂是参与判断的最大特征数。整数为个数，浮点数为所占比重
11.max_leaf_nodes:为整数或者None，指定了每个基础决策树模型的最大叶节点数量。
12.min_impurity_split:一个浮点数，指定在树生长的过程中，节点分裂的阈值，默认为1e-7。
13.init：一个基础分类器对象或者None
14.alpha:一个浮点数，只有当loss=‘huber’或者loss=‘quantile’时才有效。
15.verbose：如果为0则不输出日志，如果为1，则每隔一段时间输出日志
16.warm_start：热启动，当你训练GBM到一定程度停止时，如果你想在这个基础上接着训练，就需要用到该参数减少重复训练
属性
1.feature_importance_:一个数组，给出了每个特征的重要性（值越高重要性越大）。
2.oob_improvement_:一个数组,给出每增加一棵基础决策树,在包外估计(测试集)的损失函数的减少值
3.train_score_:一个数组，给出每增加一棵基础决策树，在训练集上的损失函数的值。
4.loss:具体损失函数对象。
5.init:初始预测使用的分类器。
6.estimators_:一个数组，给出了每个基础决策树。
方法
1.apply(X)  Apply trees in the ensemble to X, return leaf indices.
2.fit(X,y):训练模型。
3.get_params([deep]).获得参数
4.predict(X):用模型进行预测，返回预测值。
5.score(X,y):返回在（X,y）上预测的准确率。
6.set_params(**params)  Set the parameters of this estimator.
7.staged_predict(X):返回一个数组,数组元素依次是每一轮迭代结束时尚未完成的集成分类器的预测值
'''
```

随机森林分类器

```
from sklearn.ensemble import RandomForestClassifier

RandomForestClassifier(n_estimators='warn', criterion='gini', max_depth=None,
min_samples_split=2,  min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None,  bootstrap=True, oob_score=False, n_jobs=None,
random_state=None, verbose=0,  warm_start=False, class_weight=None)
'''
参数:
1.n_estimators :一个整数，指定基础决策树的数量（默认为10）.
2.criterion:字符串。指定分裂的标准，可以为‘entory’或者‘gini’。
3.max_depth:一个整数或者None，指定每一个基础决策树模型的最大深度。如果max_leaf_nodes不是
None，则忽略此参数。
4.min_samples_split:一个整数，指定了每个基础决策树模型分裂所需最小样本数。
5.min_samples_leaf:一个整数，指定了每个基础决策树模型叶节点所包含的最小样本数。
6.min_weight_fraction_leaf:一个浮点数。叶节点的最小加权权重。当不提供sample_weight时，样
本的权重是相等的。
7.max_features:一个整数，浮点数或者None。代表节点分裂是参与判断的最大特征数。整数为个数，浮点
数为所占比重。
8.max_leaf_nodes:为整数或者None，指定了每个基础决策树模型的最大叶节点数量。
9.bootstrap:为布尔值。如果为True，则使用采样法bootstrap sampling来产生决策树的训练数据。
10.oob_score：为布尔值。如果为True，则使用包外样本来计算泛化误差。
11.n_jobs：一个整数，指定并行性。如果为-1，则表示将训练和预测任务派发到所有CPU上。
12.verbose:一个整数,如果为0则不输出日志,如果为1,则每隔一段时间输出日志,大于1输出日志会更频繁
13.warm_start:布尔值。当为True是，则继续使用上一次训练结果。否则重新开始训练。
14.random_state:一个整数或者一个RandomState实例，或者None。
  如果为整数，指定随机数生成器的种子。
  如果为RandomState，指定随机数生成器。
  如果为None，指定使用默认的随机数生成器。
15.class_weight:一个字典，或者字典的列表，或者字符串‘balanced’，或者字符串
  ‘balanced_subsample’，或者为None。
  如果为字典，则字典给出每个分类的权重，如{class_label：weight}
  如果为字符串‘balanced’，则每个分类的权重与该分类在样本集合中出现的频率成反比。
  如果为字符串‘balanced_subsample’，则样本为采样法bootstrap sampling产生的决策树的训练数
据，每个分类的权重与该分类在样本集合中出现的频率成反比。
  如果为None，则每个分类的权重都为1。

属性
1.estimators_:一个数组，存放所有训练过的决策树。
2.classes_:一个数组，形状为[n_classes]，为类别标签。
3.n_classes_:一个整数，为类别数量。
4.n_features_:一个整数，在训练时使用的特征数量。
5.n_outputs_:一个整数，在训练时输出的数量。
6.feature_importances_:一个数组，形状为[n_features]。如果base_estimator支持，
  则他给出每个特征的重要性。
7.oob_score_:一个浮点数，训练数据使用包外估计时的得分。
方法
1.fit(X,y):训练模型。
2.predict(X):用模型进行预测，返回预测值。
3.predict_log_proba(X):返回一个数组，数组的元素依次是X预测为各个类别的概率的对数值。
4.predict_proba(X):返回一个数组，数组的元素依次是X预测为各个类别的概率值。
5.score(X,y):返回在（X,y）上预测的准确度。
'''
```

随机森林回归器

```
from sklearn.ensemble import RandomForestRegressor

RandomForestRegressor(n_estimators='warn', criterion='mse', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto',  max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None,  bootstrap=True, oob_score=False, n_jobs=None,
random_state=None, verbose=0,  warm_start=False)

'''
参数:
1.n_estimators :一个整数，指定基础决策树的数量（默认为10）.
2.criterion:字符串。指定分裂的标准，默认为sse
3.max_depth:一个整数或者None，指定每一个基础决策树模型的最大深度。如果
  max_leaf_nodes不是None，则忽略此参数。
4.min_samples_split:一个整数，指定了每个基础决策树模型分裂所需最小样本数。
5.min_samples_leaf:一个整数，指定了每个基础决策树模型叶节点所包含的最小样本数。
6.min_weight_fraction_leaf:一个浮点数。叶节点的最小加权权重。当不提供
  sample_weight时，样本的权重是相等的。
7.max_features:一个整数，浮点数或者None。代表节点分裂是参与判断的最大特征数。
8.max_leaf_nodes:为整数或者None，指定了每个基础决策树模型的最大叶节点数量。
9.bootstrap:为布尔值。如果为True，则使用采样法bootstrap sampling来产生决策树的训练数据。
10.oob_score：为布尔值。如果为True，则使用包外样本来计算泛化误差。
11.n_jobs：一个整数，指定并行性。如果为-1，则表示将训练和预测任务派发到所有CPU上。
12.verbose:一个整数，如果为0则不输出日志，如果为1，则每隔一段时间输出日志，大于1输出日志会更频
繁。
13.warm_start:布尔值。当为True是，则继续使用上一次训练结果。否则重新开始训练。
14.random_state:一个整数或者一个RandomState实例，或者None。
  如果为整数，指定随机数生成器的种子。
  如果为RandomState，指定随机数生成器。
  如果为None，指定使用默认的随机数生成器。

属性
1.estimators_:一个数组，存放所有训练过的决策树。
2.oob_prediction_:一个数组，训练数据使用包外估计时的预测值。
3.n_features_:一个整数，在训练时使用的特征数量。
4.n_outputs_:一个整数，在训练时输出的数量。
5.feature_importances_:一个数组，形状为[n_features]。如果base_estimator支持，
  则他给出每个特征的重要性。
6.oob_score_:一个浮点数，训练数据使用包外估计时的得分。

方法
1.fit(X,y):训练模型。
2.predict(X):用模型进行预测，返回预测值。
3.score(X,y):返回在（X,y）上预测的准确度。
'''
```

xgboost分类器

```
from xgboost import XGBClassifier

XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1,
colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
base_score=0.5,
random_state=0, seed=None, missing=None, **kwargs)

'''
三种参数
General parameters：参数控制在提升（boosting）过程中使用哪种booster，
常用的booster有树模型（tree）和线性模型（linear model）。
Booster parameters：这取决于使用哪种booster。
Task parameters：控制学习的场景，例如在回归问题中会使用不同的参数控制排序。

通用参数
这些参数用来控制XGBoost的宏观功能。
1、booster[默认gbtree]
选择每次迭代的模型，有两种选择：gbtree：基于树的模型和gbliner：线性模型
2、silent[默认0]
当这个参数值为1时，静默模式开启，不会输出任何信息。
3、nthread[默认值为最大可能的线程数] 用来进行多线程控制，应当输入系统的核数。

booster参数
尽管有两种booster可供选择，这里只介绍tree booster，因为它的表现远远胜过linear booster，所以linear booster很少用到。
1、learning_rate[默认0.1]
    和GBM中的learning rate参数类似。通过减少每一步的权重，可以提高模型的鲁棒性。典型值为0.01-0.2。
2、min_child_weight[默认1]
    决定最小叶子节点样本权重和。和GBM的 min_child_leaf 参数类似，但不完全一样。
    XGBoost的这个参数是最小样本权重的和，而GBM参数是最小样本总数。
    值越大，越容易欠拟合；值越小，越容易过拟合（值较大时，避免模型学习到局部的特殊样本）。
3、max_depth[默认6]
    为树的最大深度。值越大，越容易过拟合；值越小，越容易欠拟合。典型值：3-10
4、max_leaf_nodes
    树上最大的节点或叶子的数量。
    可以替代max_depth的作用。因为如果生成的是二叉树，一个深度为n的树最多生成n2个叶子。
    如果定义了这个参数，GBM会忽略max_depth参数。
5、gamma[默认0]
    在节点分裂时，只有分裂后损失函数的值下降了，才会分裂这个节点。
    Gamma指定了节点分裂所需的最小损失函数下降值。
    这个参数的值越大，算法越保守。这个参数的值和损失函数息息相关，所以是需要调整的。
6、max_delta_step[默认0]
    这参数限制每棵树权重改变的最大步长。如果这个参数的值为0，那就意味着没有约束。如果它被赋予了某个正值，
    那么它会让这个算法更加保守。
    通常，这个参数不需要设置。但是当各类别的样本十分不平衡时，它对逻辑回归是很有帮助的。
    这个参数一般用不到，但是你可以挖掘出来它更多的用处。
7、subsample[默认1]
    训练每棵树时，使用的数据占全部训练集的比例。默认值为1，典型值为0.5-1。
    减小这个参数的值，算法会更加保守，避免过拟合。但是，如果这个值设置得过小，它可能会导致欠拟合。
```

```
8、colsample_bytree[默认1]
    和GBM里面的max_features参数类似。训练每棵树时，使用的特征占全部特征的比例。
    默认值为1，典型值为0.5-1。
9、colsample_bylevel[默认1]
    用来控制树的每一级的每一次分裂，对列数的采样的占比。
10、reg_lambda[默认1]
    权重的L2正则化项。
11、reg_alpha[默认1]
    权重的L1正则化项。可以应用在很高维度的情况下，使得算法的速度更快。
12、scale_pos_weight[默认1]
    在各类别样本十分不平衡时，把这个参数设定为一个正值，可以使算法更快收敛。

学习目标参数
这个参数用来控制理想的优化目标和每一步结果的度量方法。
1、objective[默认binary:logistic]
    这个参数定义需要被最小化的损失函数。最常用的值有：
    binary:logistic 二分类的逻辑回归，返回预测的概率(不是类别)。
    multi:softmax 使用softmax的多分类器，返回预测的类别(不是概率)。
        在这种情况下，你还需要多设一个参数：num_class(类别数目)。
    multi:softprob 和multi:softmax参数一样，但是返回的是每个数据属于各个类别的概率。
2、eval_metric[默认值取决于objective参数的取值]
    对于有效数据的度量方法。
    对于回归问题，默认值是rmse，对于分类问题，默认值是error。
    典型值有：
    rmse、mae、logloss 负对数似然函数值、error 二分类错误率(阈值为0.5)
    merror 多分类错误率、mlogloss 多分类logloss损失函数、auc 曲线下面积
3、seed(默认0)
    随机数的种子。设置它可以复现随机数据的结果，也可以用于调整参数

属性
1.feature_importances_  给出每个特征的重要性。

方法
1.apply(X[, ntree_limit]):  返回每个样本的每棵树的预测叶子索引
2.evals_result(): 返回评估结果.
3.fit(X[, y]): 训练模型
4.get_booster(): 获取此模型的基础xgboost Booster.
5.get_params([deep]): 获取参数.
6.get_xgb_params(): 获取xgboost类型参数
7.predict(X): 返回样本预测结果
8.predict_proba(data[, ntree_limit]): 预测每个数据属于给定类的概率
9.score(X, y[, sample_weight]): 模型准确率
10.set_params(**params) :设定参数.
'''
```

xgboost回归器

```
from xgboost import XGBRegressor

XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
objective='reg:linear',  booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1,
colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)


'''
属性
1.feature_importances_  给出每个特征的重要性。

用法
1.apply(X[, ntree_limit]):  返回每个样本的每棵树的预测叶子索引
2.evals_result(): 返回评估结果.
3.fit(X[, y]): 训练模型
4.get_booster(): 获取此模型的基础xgboost Booster.
5.get_params([deep]): 获取参数.
6.get_xgb_params(): 获取xgboost类型参数
7.predict(X): 返回样本预测结果
8.score(X, y[, sample_weight]): 模型准确率
0.set_params(**params)  :设定参数.
'''
```