

# 模型

## 1. 模型的一些通用方法：

- `get_params([deep])`：返回模型的参数。
  - `deep`：如果为 `True`，则可以返回模型参数的子对象。
- `set_params(**params)`：设置模型的参数。
  - `params`：待设置的关键字参数。
- `fit(X,y[,sample_weight])`：训练模型。
  - `X`：训练集样本集合。通常是一个 `numpy array`，每行代表一个样本，每列代表一个特征。
  - `y`：训练样本的标签集合。它与 `X` 的每一行相对应。
  - `sample_weight`：每个样本的权重。它与 `X` 的每一行相对应。
- `predict(x)`：利用模型执行预测。返回一个预测结果序列。
  - `X`：测试集样本集合。通常是一个 `numpy array`，每行代表一个样本，每列代表一个特征。
- `score(X,y[,sample_weight])`：对模型进行评估，返回模型的性能评估结果。
  - `X`：验证集样本集合。通常是一个 `numpy array`，每行代表一个样本，每列代表一个特征。
  - `y`：验证集样本的标签集合。它与 `X` 的每一行相对应。
  - `sample_weight`：每个样本的权重。它与 `X` 的每一行相对应。

对于分类模型，其评估的是 `accuracy`；对于回归模型，其评估的是 `R2`。

如果希望有其它的评估指标，则可以执行 `predict()` 方法，然后把预测结果、真实标记作为参数来调用一些打分函数即可。

## 2. 模型的一些通用参数：

- `n_jobs`：一个正数，指定任务并行时指定的 `CPU` 数量。  
如果为 `-1` 则使用所有可用的 `CPU`。
- `verbose`：一个正数。用于开启/关闭迭代中间输出日志功能。
  - 数值越大，则日志越详细。
  - 数值为0或者 `None`，表示关闭日志输出。
- `warm_start`：一个布尔值。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。
- `max_iter`：一个整数，指定最大迭代次数。  
如果为 `None` 则为默认值（不同 `solver` 的默认值不同）。
- `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`，则使用默认随机数生成器。

## 3. 对于回归模型，其评估性能的指标为 $R^2$ 。

假设验证集为  $\mathbb{D}_{validate}$ ，真实标签记作  $\tilde{y}$ ，预测值记作  $\hat{y}$ ，则有：

$$R^2 = 1 - \frac{\sum_{\mathbb{D}_{\text{validate}}} (\tilde{y}_i - \hat{y}_i)^2}{(\tilde{y}_i - \bar{y})^2}$$

其中  $\bar{y}$  为所有真实标记的均值。

根据定义有：

- $R^2$  不超过 1，但是有可能小于 0。
- $R^2$  越大，模型的预测性能越好。

## 一、线性模型

### 1. 线性模型的一些通用参数：

- `fit_intercept`：一个布尔值，指定是否需要计算截距项。如果为 `False`，那么不会计算截距项。  
当  $\vec{w} = (w^{(1)}, w^{(2)}, \dots, w^{(n)}, b)^T = (\vec{w}^T, b)^T, \vec{x} = (x^{(1)}, x^{(2)}, \dots, x^{(n)}, 1)^T = (\vec{x}^T, 1)^T$  时，可以设置 `fit_intercept=False`。
- `intercept_scaling`：一个浮点数，用于缩放截距项的正则化项的影响。  
当采用 `fit_intercept` 时，相当于人造一个特征出来，该特征恒为 1，其权重为  $b$ 。  
在计算正则化项的时候，该人造特征也被考虑了。为了降低这个人造特征的影响，需要提供 `intercept_scaling`。
- `tol`：一个浮点数，指定判断迭代收敛与否的阈值。

## 1.1 LinearRegression

### 1. `LinearRegression` 是线性回归模型，它的原型为：

```
class sklearn.linear_model.LinearRegression(fit_intercept=True, normalize=False,
copy_X=True, n_jobs=1)
```

- `fit_intercept`：一个布尔值，指定是否需要计算截距项。
- `normalize`：一个布尔值。如果为 `True`，那么训练样本会在训练之前会被归一化。
- `copy_X`：一个布尔值。如果为 `True`，则会拷贝  $X$ 。
- `n_jobs`：一个整数，指定计算并行度。

### 2. 模型属性：

- `coef_`：权重向量。
- `intercept_`： $b$  值。

### 3. 模型方法：

- `fit(X, y[, sample_weight])`：训练模型。
- `predict(X)`：用模型进行预测，返回预测值。
- `score(X, y[, sample_weight])`：返回模型的预测性能得分。

## 1.2 Ridge

### 1. `Ridge` 类实现了岭回归模型。其原型为：

```
class sklearn.linear_model.Ridge(alpha=1.0, fit_intercept=True, normalize=False,
copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)
```

- `alpha` :  $\alpha$  值, 用于缓解过拟合。
- `max_iter` : 指定最大迭代次数。
- `tol` : 一个浮点数, 指定判断迭代收敛与否的阈值。
- `solver` : 一个字符串, 指定求解最优化问题的算法。可以为 :
  - `'auto'` : 根据数据集自动选择算法。
  - `'svd'` : 使用奇异值分解来计算回归系数。
  - `'cholesky'` : 使用 `scipy.linalg.solve` 函数来求解。
  - `'sparse_cg'` : 使用 `scipy.sparse.linalg.cg` 函数来求解。
  - `'lsqr'` : 使用 `scipy.sparse.linalg.lsqr` 函数求解。  
它运算速度最快, 但是可能老版本的 `scipy` 不支持。
  - `'sag'` : 使用 `Stochastic Average Gradient descent` 算法求解最优化问题。
- `random_state` : 用于设定随机数生成器, 它在 `solver=sag` 时使用。
- 其它参数参考 `LinearRegression` 。

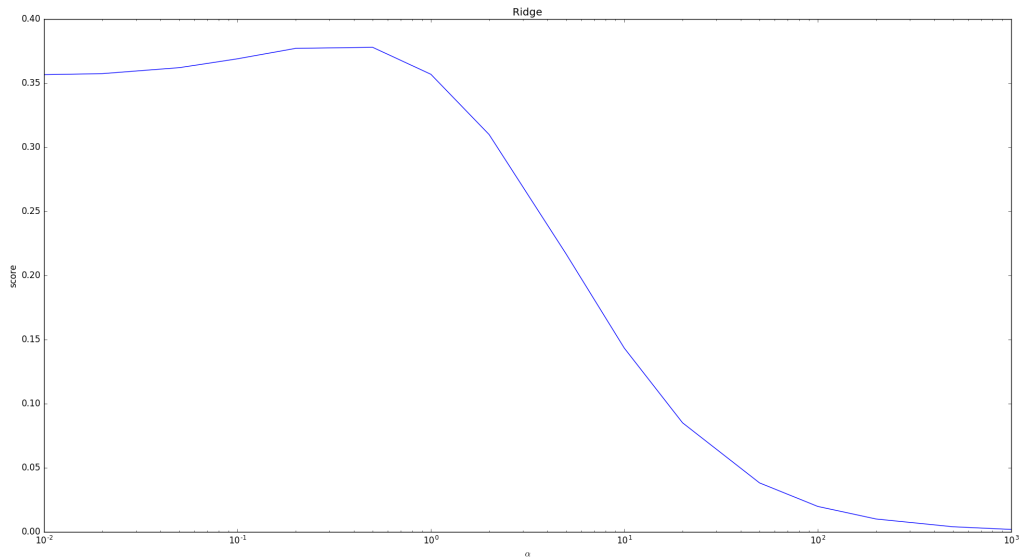
## 2. 模型属性 :

- `coef_` : 权重向量。
- `intercept_` :  $b$  值。
- `n_iter_` : 实际迭代次数。

## 3. 模型方法 : 参考 `LinearRegression` 。

## 4. 下面的示例给出了不同的 $\alpha$ 值对模型预测能力的影响。

- 当  $\alpha$  超过 1 之后, 随着  $\alpha$  的增长, 预测性能急剧下降。  
这是因为  $\alpha$  较大时, 正则化项  $\alpha \|\vec{w}\|_2^2$  影响较大, 模型趋向于简单。
- 极端情况下当  $\alpha \rightarrow \infty$  时,  $\vec{w} = \vec{0}$  从而使得正则化项  $\alpha \|\vec{w}\|_2^2 = 0$ , 此时的模型最简单。  
但是预测性能非常差, 因为对所有的未知样本, 模型都预测为同一个常数  $b$ 。

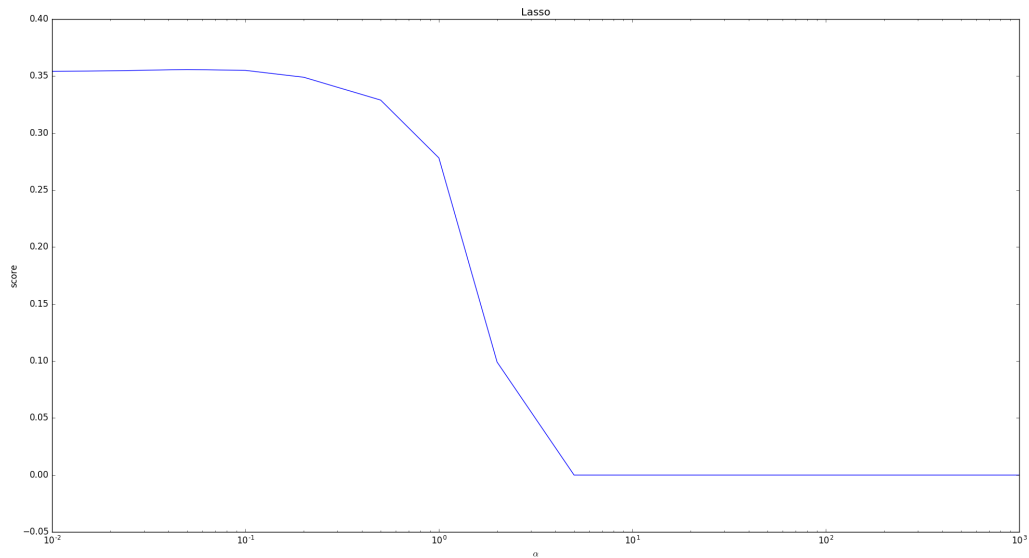


## 1.3 Lasso

1. `Lasso` 类实现了 `Lasso` 回归模型。其原型为：

```
lass sklearn.linear_model.Lasso(alpha=1.0, fit_intercept=True, normalize=False,
precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False,
positive=False, random_state=None, selection='cyclic')
```

- `alpha`： $\alpha$  值，用于缓解过拟合。
  - `precompute`：一个布尔值或者一个序列。是否提前计算 `Gram` 矩阵来加速计算。
  - `warm_start`：是否从头开始训练。
  - `positive`：一个布尔值。如果为 `True`，那么强制要求权重向量的分量都为正数。
  - `selection`：一个字符串，可以为 `'cyclic'` 或者 `'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。
    - `'random'`：更新的时候，随机选择权重向量的一个分量来更新
    - `'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
  - 其它参数参考 `Ridge`。
2. 模型属性：参考 `Ridge`。
  3. 模型方法：参考 `LinearRegression`。
  4. 下面的示例给出了不同的  $\alpha$  值对模型预测能力的影响。  
当  $\alpha$  超过 1 之后，随着  $\alpha$  的增长，预测性能急剧下降。原因同 `Ridge` 中的分析。



## 1.4 ElasticNet

1. `ElasticNet` 类实现了 `ElasticNet` 回归模型。其原型为：

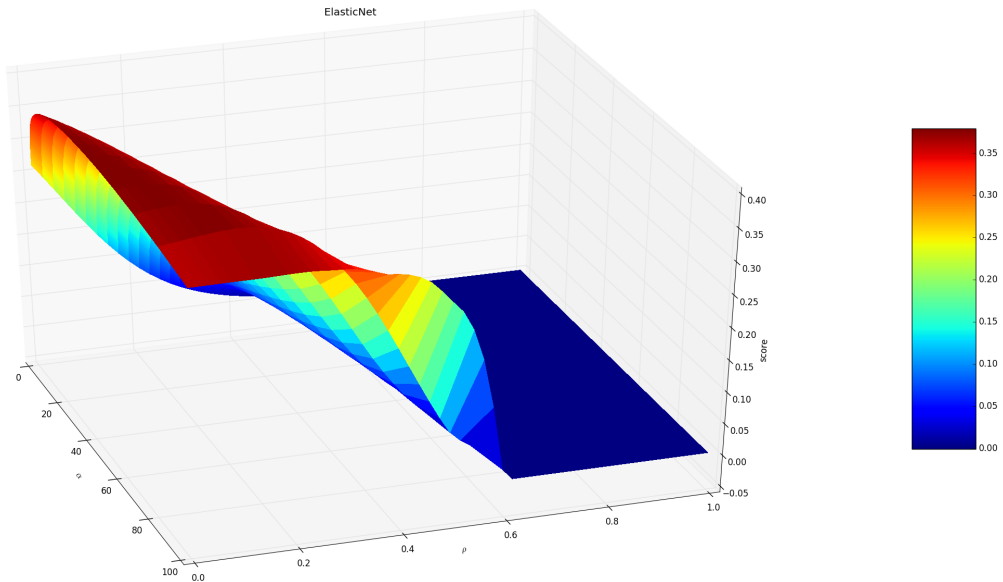
```
class sklearn.linear_model.ElasticNet(alpha=1.0, l1_ratio=0.5, fit_intercept=True,
normalization=False, precompute=False, max_iter=1000, copy_X=True, tol=0.0001,
warm_start=False, positive=False, random_state=None, selection='cyclic')
```

- `alpha` :  $\alpha$  值。
  - `l1_ratio` :  $\rho$  值。
  - 其它参数参考 `Lasso`。
2. 模型属性：参考 `Lasso`。
  3. 模型方法：参考 `Lasso`。
  4. 下面的示例给出了不同的  $\alpha$  值和  $\rho$  值对模型预测能力的影响。

- 随着  $\alpha$  的增大，预测性能下降。因为正则化项为：

$$\alpha \rho \|\vec{w}\|_1 + \frac{\alpha(1-\rho)}{2} \|\vec{w}\|_2^2, \alpha \geq 0, 1 \geq \rho \geq 0$$

- $\rho$  影响的是性能下降的速度，因为这个参数控制着  $\|\vec{w}\|_1$ ， $\|\vec{w}\|_2^2$  之间的比例。



## 1.4 LogisticRegression

1. `LogisticRegression` 实现了对数几率回归模型。其原型为：

```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001,
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)
```

- `penalty`：一个字符串，指定了正则化策略。
  - 如果为 `'l2'`，则为  $L_2$  正则化。
  - 如果为 `'l1'`，则为  $L_1$  正则化。
- `dual`：一个布尔值。
  - 如果为 `True`，则求解对偶形式（只在 `penalty='l2'` 且 `solver='liblinear'` 有对偶形式）。
  - 如果为 `False`，则求解原始形式。
- `C`：一个浮点数。它指定了罚项系数的倒数。如果它的值越小，则正则化项越大。
- `class_weight`：一个字典或者字符串 `'balanced'`，指定每个类别的权重。
  - 如果为字典：则字典给出了每个分类的权重。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中出现的频率成反比。
  - 如果未指定，则每个分类的权重都为 `1`。
- `solver`：一个字符串，指定了求解最优化问题的算法。可以为下列的值：
  - `'newton-cg'`：使用牛顿法。
  - `'lbfgs'`：使用 L-BFGS 拟牛顿法。
  - `'liblinear'`：使用 `liblinear`。
  - `'sag'`：使用 Stochastic Average Gradient descent 算法。

注意：

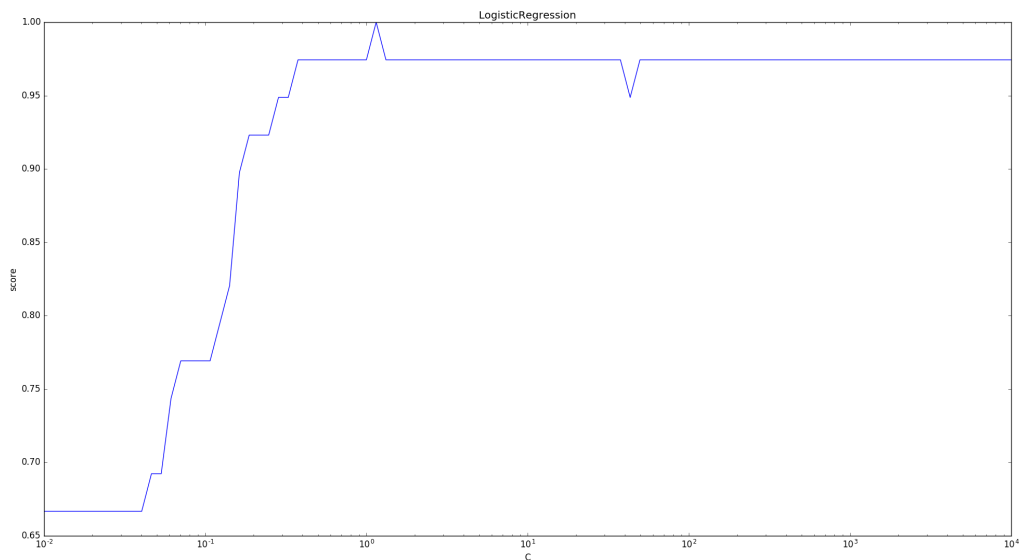
- 对于规模小的数据集，`'liblinear'` 比较适用；对于规模大的数据集，`'sag'` 比较适用。

- 'newton-cg'、'lbfgs'、'sag' 只处理 `penalty='l2'` 的情况。
  - `multi_class`：一个字符串，指定对于多分类问题的策略。可以为：
    - 'ovr'：采用 one-vs-rest 策略。
    - 'multinomial'：直接采用多分类 logistic 回归策略。
  - 其它参数参考 `ElasticNet`。
2. 模型属性：参考 `ElasticNet`。
3. 模型方法：
- `fit(X,y[,sample_weight])`：训练模型。
  - `predict(X)`：用模型进行预测，返回预测值。
  - `score(X,y[,sample_weight])`：返回模型的预测性能得分。
  - `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。
  - `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。
4. 下面的示例给出了不同的 `C` 值对模型预测能力的影响。

`C` 是正则化项系数的倒数，它越小则正则化项的权重越大。

- 随着 `C` 的增大（即正则化项的减小），`LogisticRegression` 的预测准确率上升。
- 当 `C` 增大到一定程度（即正则化项减小到一定程度），`LogisticRegression` 的预测准确率维持在较高的水准保持不变。

事实上，当 `C` 太大时，正则化项接近于0，此时容易发生过拟合，预测准确率会下降。



## 1.5 LinearDiscriminantAnalysis

1. 类 `LinearDiscriminantAnalysis` 实现了线性判别分析模型。其原型为：

```
class sklearn.discriminant_analysis.LinearDiscriminantAnalysis(solver='svd',
    shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001)
```

- `solver`：一个字符串，指定求解最优化问题的算法。可以为：

- 'svd' : 奇异值分解。对于有大规模特征的数据, 推荐用这种算法。
- 'lsqr' : 最小平方差算法, 可以结合 shrinkage 参数。
- 'eigen' : 特征值分解算法, 可以结合 shrinkage 参数。
- shrinkage : 字符串 'auto' 或者浮点数或者 None 。

该参数只有在 solver='lsqr' 或者 'eigen' 下才有意义。当矩阵求逆时, 它会在对角线上增加一个小的数  $\lambda$ , 防止矩阵为奇异的。其作用相当于正则化。

- 字符串 'auto' : 根据 Ledoit-Wolf 引理来自动决定  $\lambda$  的大小。
  - None : 不使用 shrinkage 参数。
  - 一个 0 到 1 之间的浮点数: 指定  $\lambda$  的值。
  - priors : 一个数组, 数组中的元素依次指定了每个类别的先验概率。
- 如果为 None 则认为每个类的先验概率都是等可能的。
- n\_components : 一个整数, 指定了数据降维后的维度 (该值必须小于 n\_classes-1) 。
  - store\_covariance : 一个布尔值。如果为 True, 则需要额外计算每个类别的协方差矩阵  $\Sigma_i$  。
  - tol : 一个浮点值。它指定了用于 SVD 算法中评判迭代收敛的阈值。

## 2. 模型属性:

- coef\_ : 权重向量。
- intercept\_ :  $b$  值。
- covariance\_ : 一个数组, 依次给出了每个类别的协方差矩阵。
- means\_ : 一个数组, 依次给出了每个类别的均值向量。
- xbar\_ : 给出了整体样本的均值向量。
- n\_iter\_ : 实际迭代次数。

## 3. 模型方法: 参考 LogisticRegression 。

# 二、支持向量机

## 1. SVM 的通用参数:

- tol : 浮点数, 指定终止迭代的阈值。
  - fit\_intercept : 一个布尔值, 指定是否需要计算截距项。如果为 False, 那么不会计算截距项。
- 当  $\vec{w} = (w^{(1)}, w^{(2)}, \dots, w^{(n)}, b)^T = (\vec{w}^T, b)^T, \vec{x} = (x^{(1)}, x^{(2)}, \dots, x^{(n)}, 1)^T = (\vec{x}^T, 1)^T$  时, 可以设置 fit\_intercept=False 。
- intercept\_scaling : 一个浮点数, 用于缩放截距项的正则化项的影响。
- 当采用 fit\_intercept 时, 相当于人造一个特征出来, 该特征恒为 1, 其权重为  $b$ 。
- 在计算正则化项的时候, 该人造特征也被考虑了。为了降低这个人造特征的影响, 需要提供 intercept\_scaling 。
- class\_weight : 一个字典或者字符串 'balanced', 指定每个类别的权重。
- 如果为字典: 则字典给出了每个分类的权重。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中出现的频率成反比。
  - 如果未指定, 则每个分类的权重都为 1 。

## 2.1 LinearSVC



1. `LinearSVC` 是根据 `liblinear` 实现的，它可以用于二类分类，也可以用于多类分类问题（此时是根据 `one-vs-rest` 原则来分类）。

2. 线性支持向量机 `LinearSVC`：

```
sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0,
multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None,
verbose=0, random_state=None, max_iter=1000)
```

- `penalty`：字符串，指定 'l1' 或者 'l2'，罚项的范数。默认为 'l2'（它是标准SVC采用的）。
- `loss`：一个字符串，表示损失函数。可以为：
  - 'hinge'：此时为合页损失函数（它是标准 SVM 的损失函数）。
  - 'squared\_hinge'：合页损失函数的平方。
- `dual`：一个布尔值。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` 时，倾向于采用 `False`。
- `tol`：一个浮点数，指定终止迭代的阈值。
- `C`：一个浮点数，罚项系数。
- `multi_class`：一个字符串，指定多类分类问题的策略。
  - 'ovr'：采用 `one-vs-rest` 分类策略。
  - 'crammer\_singer'：多类联合分类，很少用。因为它计算量大，而且精度不会更佳。此时忽略 `loss, penalty, dual` 项。
- `fit_intercept`：一个布尔值，指定是否需要计算截距项。
- `intercept_scaling`：一个浮点数，用于缩放截距项的正则化项的影响。
- `class_weight`：一个字典或者字符串 'balanced'，指定每个类别的权重。
- `verbose`：一个正数。用于开启/关闭迭代中间输出日志功能。
- `random_state`：指定随机数种子。
- `max_iter`：一个整数，指定最大迭代次数。

3. 模型属性：

- `coef_`：权重向量。
- `intercept_`：截距值。

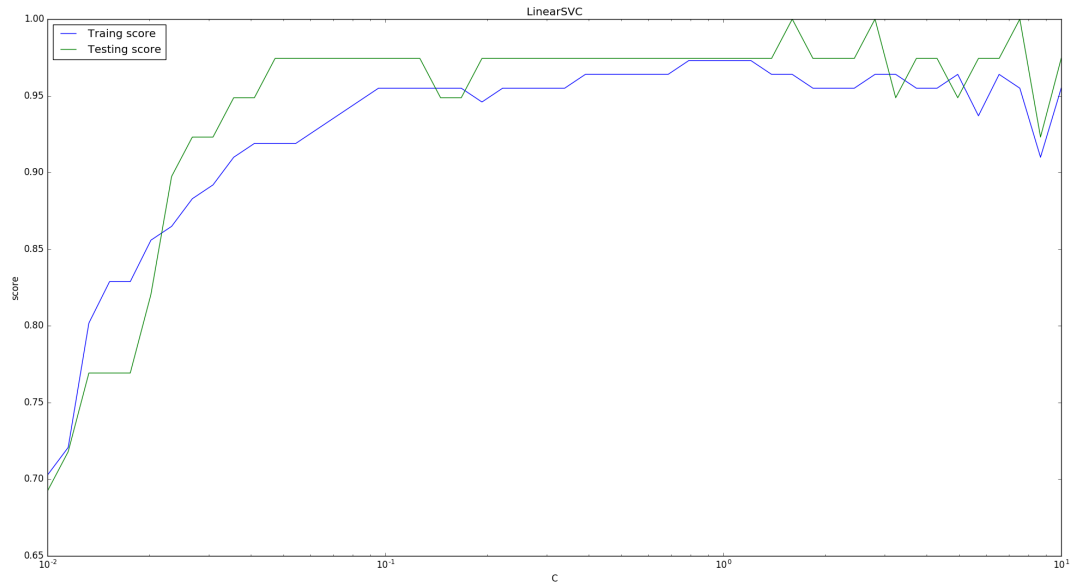
4. 模型方法：

- `fit(X, y)`：训练模型。
- `predict(X)`：用模型进行预测，返回预测值。
- `score(X, y[, sample_weight])`：返回模型的预测性能得分。

5. 下面的示例给出了不同的 `C` 值对模型预测能力的影响。

`C` 衡量了误分类点的重要性，`C` 越大则误分类点越重要。

为了便于观察将 `x` 轴以对数表示。可以看到当 `C` 较小时，误分类点重要性较低，此时误分类点较多，分类器性能较差。



## 2.2 SVC

1. SVC 是根据 `libsvm` 实现的，其训练的时间复杂度是采样点数量的平方。

它可以用于二类分类，也可以用于多类分类问题（此时默认是根据 `one-vs-rest` 原则来分类）。

2. 支持向量机 SVC：

```
sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True,
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False,
max_iter=-1, decision_function_shape=None, random_state=None)
```

- `C`：一个浮点数，罚项系数。
- `kernel`：一个字符串，指定核函数。
  - `'linear'`：线性核： $K(\vec{x}, \vec{z}) = \vec{x} \cdot \vec{z}$ 。
  - `'poly'`：多项式核： $K(\vec{x}, \vec{z}) = (\gamma(\vec{x} \cdot \vec{z} + 1) + r)^p$ 。其中：
    - $p$  由 `degree` 参数决定。
    - $\gamma$  由 `gamma` 参数决定。
    - $r$  由 `coef0` 参数决定。
  - `'rbf'`（默认值）：高斯核函数： $K(\vec{x}, \vec{z}) = \exp(-\gamma \|\vec{x} - \vec{z}\|^2)$ 。  
其中  $\gamma$  由 `gamma` 参数决定。
  - `'sigmoid'`： $K(\vec{x}, \vec{z}) = \tanh(\gamma(\vec{x} \cdot \vec{z}) + r)$ 。其中：
    - $\gamma$  由 `gamma` 参数决定。
    - $r$  由 `coef0` 参数指定。
  - `'precomputed'`：表示提供了 `kernel matrix`。
  - 或者提供一个可调用对象，该对象用于计算 `kernel matrix`。

- `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。对于其他核函数, 该参数无效。
- `gamma` : 一个浮点数。当核函数是 'rbf', 'poly', 'sigmoid' 时, 核函数的系数。如果 'auto', 则表示系数为  $1/n\_features$ 。
- `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' 是有效。
- `probability` : 布尔值。如果为 True 则会进行概率估计。它必须在训练之前设置好, 且概率估计会拖慢训练速度。
- `shrinking` : 布尔值。如果为 True, 则使用启发式( `shrinking heuristic` )。
- `tol` : 浮点数, 指定终止迭代的阈值。
- `cache_size` : 浮点值, 指定了 `kernel cache` 的大小, 单位为 MB。
- `class_weight` : 指定各类别的权重。
- `decision_function_shape` : 为字符串或者 None, 指定决策函数的形状。
  - 'ovr' : 则使用 `one-vs-rest` 准则。那么决策函数形状是  $(n\_samples, n\_classes)$ 。  
此时对每个分类定义了一个二类 SVM, 一共  $n\_classes$  个二类 SVM。
  - 'ovo' : 则使用 `one-vs-one` 准则。那么决策函数形状是  $(n\_samples, n\_classes * (n\_classes - 1) / 2)$   
此时对每一对分类直接定义了一个二类 SVM, 一共  $n\_classes * (n\_classes - 1) / 2$  个二类 SVM。
  - None : 默认值。采用该值时, 目前会使用 'ovo', 但是在 `scikit v0.18` 之后切换成 'ovr'。
- 其它参数参考 `LinearSVC`。

### 3. 模型属性 :

- `support_` : 一个数组, 形状为  $[n\_SV]$ , 给出了支持向量的下标。
- `support_vectors_` : 一个数组, 形状为  $[n\_SV, n\_features]$ , 给出了支持向量。
- `n_support_` : 一个数组, 形状为  $[n\_class]$ , 给出了每一个分类的支持向量的个数。
- `dual_coef_` : 一个数组, 形状为  $[n\_class-1, n\_SV]$ 。给出了对偶问题中, 每个支持向量的系数。
- `coef_` : 一个数组, 形状为  $[n\_class-1, n\_features]$ 。给出了原始问题中, 每个特征的系数。
  - 它只有在 `linear kernel` 中有效。
  - 它是个只读的属性。它是从 `dual_coef_` 和 `support_vectors_` 计算而来。
- `intercept_` : 一个数组, 形状为  $[n\_class * (n\_class-1) / 2]$ , 给出了决策函数中的常数项。

### 4. 模型方法 :

- `fit(X, y[, sample_weight])` : 训练模型。
- `predict(X)` : 用模型进行预测, 返回预测值。
- `score(X, y[, sample_weight])` : 返回模型的预测性能得分。
- `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。
- `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。

## 2.3 NuSVC

1. `NuSVC`: Nu-Support Vector Classificatio 与 `SVC` 相似，但是用一个参数来控制了支持向量的个数。它是基于 `libsvm` 来实现的。
2. `NuSVC` 支持向量机：

```
sklearn.svm.NuSVC(nu=0.5, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
verbose=False, max_iter=-1, decision_function_shape=None, random_state=None)
```

- `nu`：一个浮点数，取值范围为  $(0,1]$ ，默认为0.5。它控制训练误差与支持向量的比值，间接控制了支持向量的个数。
  - 其它参数参考 `SVC`。
3. 模型属性：参考 `SVC`。
  4. 模型方法：参考 `SVC`。

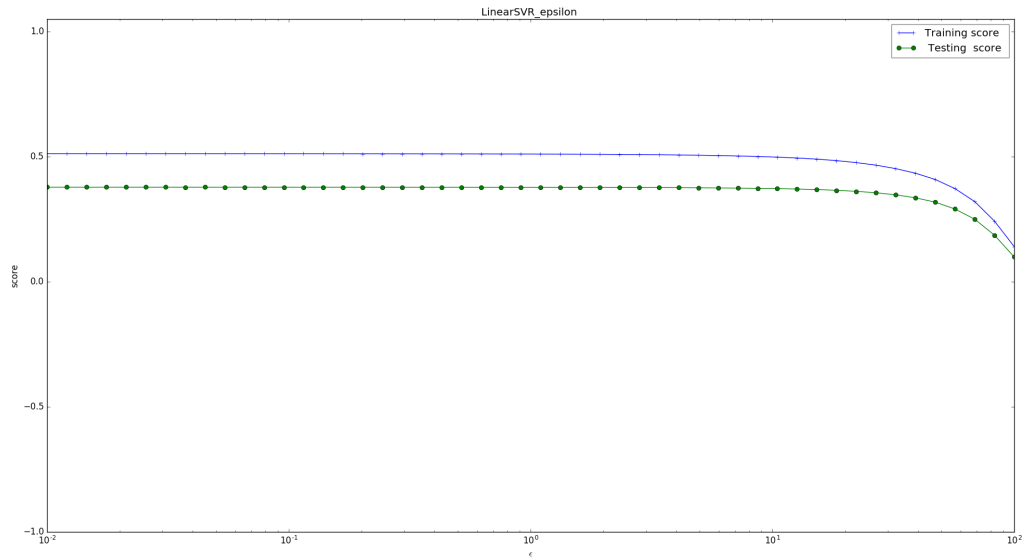
## 2.4 LinearSVR

1. `LinearSVR` 是根据 `liblinear` 实现的。
2. 线性支持向量回归 `LinearSVR`：

```
class sklearn.svm.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)
```

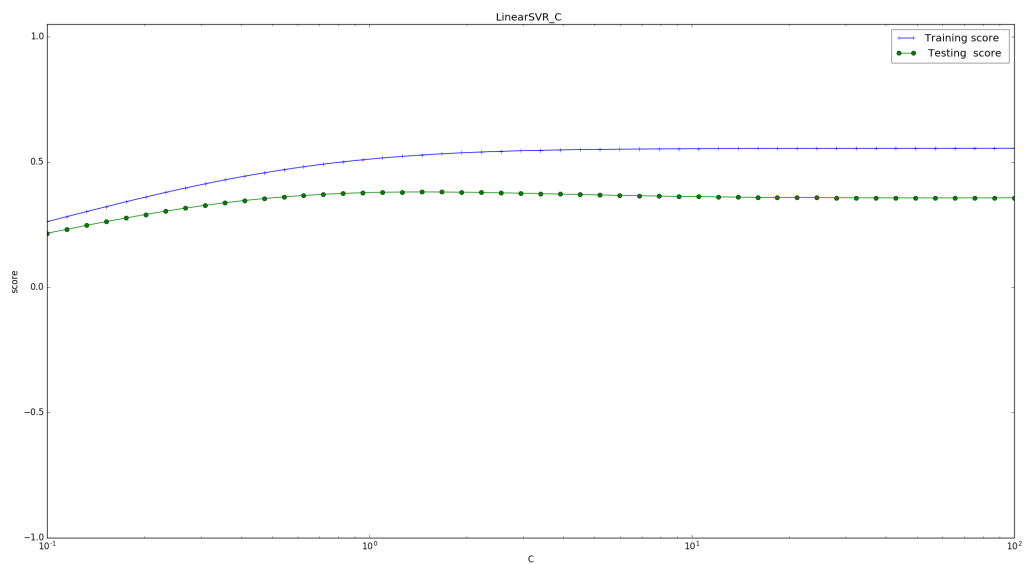
- `epsilon`：一个浮点数，表示  $\epsilon$  值。
  - `loss`：字符串。表示损失函数。可以为：
    - `'epsilon_insensitive'`：此时损失函数为  $L_\epsilon$ （标准的 `SVR`）
    - `'squared_epsilon_insensitive'`：此时损失函数为  $L_\epsilon^2$
  - 其它参数参考 `LinearSVC`。
3. 模型属性：参考 `LinearSVC`。
  4. 模型方法：参考 `LinearSVC`。
  5. 下面的示例给出了不同的  $\epsilon$  值对模型预测能力的影响。

为了方便观看将 `x` 轴转换成对数坐标。可以看到预测准确率随着  $\epsilon$  下降。



6. 下面的示例给出了不同的  $C$  值对模型预测能力的影响。

为了方便观看将  $x$  轴转换成对数坐标。可以看到预测准确率随着  $C$  增大而上升。说明越看重误分类点，则预测的越准确。



## 2.5 SVR

1. SVR 是根据 `libsvm` 实现的。
2. 支持向量回归 `SVR` :

```
class sklearn.svm.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：参考 `SVC` 。

3. 模型属性：参考 `SVC` 。

- 模型方法：参考 `SVC`。

## 2.6 NuSVR

- `NuSVR` 是根据 `libsvm` 实现的。
- 支持向量回归 `NuSVR`：

```
class sklearn.svm.NuSVR(nu=0.5, C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
shrinking=True, tol=0.001, cache_size=200, verbose=False, max_iter=-1)
```

- `C`：一个浮点数，罚项系数。
  - 其它参数参考 `NuSVC`。
- 模型属性：参考 `NuSVC`。
  - 模型方法：参考 `NuSVC`。

## 2.7 OneClassSVM

- `OneClassSVM` 是根据 `libsvm` 实现的。
- 支持向量描述 `OneClassSVM`：

```
class sklearn.svm.OneClassSVM(kernel='rbf', degree=3, gamma='auto', coef0=0.0, tol=0.001,
nu=0.5, shrinking=True, cache_size=200, verbose=False, max_iter=-1, random_state=None)
```

参数：参考 `NuSVC`。

- 模型属性：参考 `NuSVC`。
- 模型方法：
  - `fit(X[, y, sample_weight])`：训练模型。
  - `predict(X)`：用模型进行预测，返回预测值。每个预测值要么是 `+1` 要么是 `-1`。

## 三、贝叶斯模型

- 在 `scikit` 中有多种不同的朴素贝叶斯分类器。他们的区别就在于它们假设了不同的  $p(x_j | y)$  分布。

### 3.1 GaussianNB

- 高斯贝叶斯分类器 `GaussianNB`：它假设特征  $x_j$  的条件概率分布满足高斯分布：

$$p(x_j | y = c_k) = \frac{1}{\sqrt{2\pi\sigma_{k,j}^2}} \exp\left(-\frac{(x_j - \mu_{k,j})^2}{2\sigma_{k,j}^2}\right)$$

其中： $\mu_{k,j}$  为第  $j$  个特征的条件概率分布的均值， $\sigma_{k,j}$  为第  $j$  个特征的条件概率分布的方差。

- `GaussianNB` 的原型为：

```
class sklearn.naive_bayes.GaussianNB()
```

### 3. 模型属性：

- `class_prior_`：一个数组，形状为 `(n_classes,)`，是每个类别的概率。
- `class_count_`：一个数组，形状为 `(n_classes,)`，是每个类别包含的训练样本数量。
- `theta_`：一个数组，形状为 `(n_classes, n_features)`，是每个类别上，每个特征的均值。
- `sigma_`：一个数组，形状为 `(n_classes, n_features)`，是每个类别上，每个特征的标准差。

### 4. 模型方法：

- `fit(X, y[, sample_weight])`：训练模型。
- `partial_fit(X, y[, classes, sample_weight])`：分批训练模型。

该方法主要用于大规模数据集的训练。此时可以将大数据集划分成若干个小数据集，然后在这些小数据集上连续调用 `partial_fit` 方法来训练模型。

- `predict(X)`：用模型进行预测，返回预测值。
- `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。
- `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。
- `score(X, y[, sample_weight])`：返回模型的预测性能得分。

## 3.2 MultinomialNB

### 1. 多项式贝叶斯分类器 `MultinomialNB`：它假设特征的条件概率分布满足多项式分布：

$$p(x_j = a_{j,t} \mid y = c_k) = \frac{N_{k,j,t} + \alpha}{N_k + \alpha n}$$

其中：

- $N_k = \sum_{i=1}^N I(\tilde{y}_i = c_k)$ ，表示属于类别  $c_k$  的样本的数量。
- $N_{k,j,t} = \sum_{i=1}^N I(\tilde{y}_i = c_k, x_j = a_{j,t})$ ，表示属于类别  $c_k$  且第  $j$  个特征取值为  $x_j = a_{j,t}$  的样本的数量。

### 2. `MultinomialNB` 的原型为：

```
class sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
```

- `alpha`：一个浮点数，指定  $\alpha$  值。
- `fit_prior`：一个布尔值。
  - 如果为 `True`，则不去学习  $p(y)$ ，替代以均匀分布。
  - 如果为 `False`，则去学习  $p(y)$ 。
- `class_prior`：一个数组。它指定了每个分类的先验概率  $p(y)$ 。

如果指定了该参数，则每个分类的先验概率不再从数据集中学得

### 3. 模型属性：

- `class_log_prior_`：一个数组对象，形状为 `(n_classes,)`。给出了每个类别的调整后的的经验概率分布的对数值。

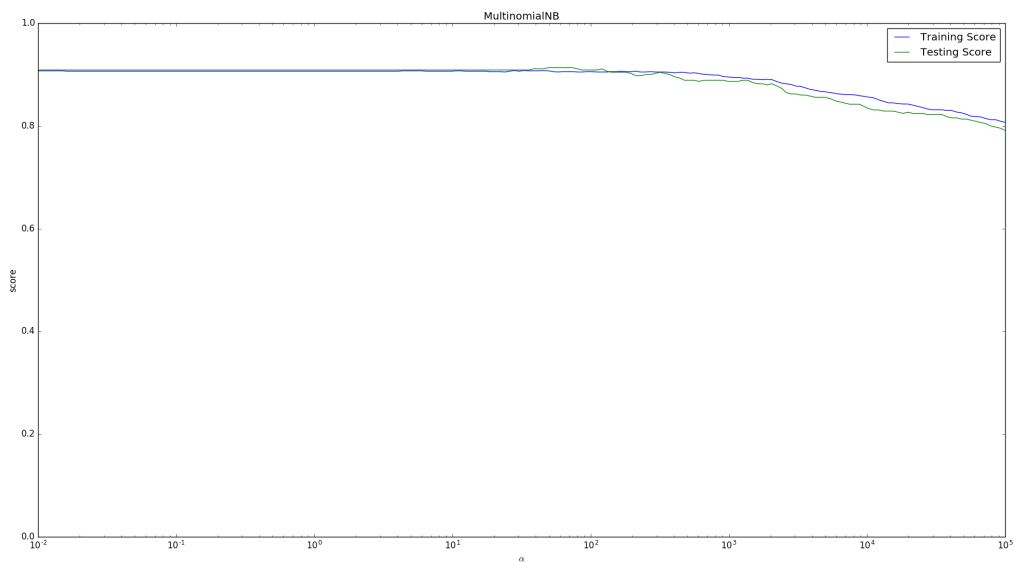
- `feature_log_prob_`: 一个数组对象, 形状为 `(n_classes, n_features)`。给出了  $p(x_j | y)$  的经验概率分布的对数值。
- `class_count_`: 一个数组, 形状为 `(n_classes,)`, 是每个类别包含的训练样本数量。
- `feature_count_`: 一个数组, 形状为 `(n_classes, n_features)`。训练过程中, 每个类别每个特征遇到的样本数。

4. 模型方法: 参考 `GaussianNB`。

5. 下面的示例给出了不同的  $\alpha$  值对模型预测能力的影响。运行结果如下。

为了便于观察将 `x` 轴设置为对数坐标。可以看到随着  $\alpha > 100$  之后, 随着  $\alpha$  的增长, 预测准确率在下降。

这是因为, 当  $\alpha \rightarrow \infty$  时,  $p(x_j = a_{j,t} | y = c_k) = \frac{N_{k,j,t} + \alpha}{N_k + \alpha n} \rightarrow \frac{1}{n}$ 。即对任何类型的特征、该类型特征的任意取值, 出现的概率都是  $\frac{1}{n}$ 。它完全忽略了各个特征之间的差别, 也忽略了每个特征内部的分布。



### 3.3 BernoulliNB

1. 伯努利贝叶斯分类器 `BernoulliNB`: 它假设特征的条件概率分布满足二项分布:

$$p(x_j | y) = p \times x_j + (1 - p)(1 - x_j)$$

其中  $p = p(x_j = 1 | y)$ , 且要求特征的取值为  $x_j \in \{0, 1\}$ 。

2. `BernoulliNB` 的原型为:

```
class sklearn.naive_bayes.BernoulliNB(alpha=1.0, binarize=0.0, fit_prior=True,
class_prior=None)
```

- `binarize`: 一个浮点数或者 `None`。
  - 如果为 `None`, 那么会假定原始数据已经是二元化的。
  - 如果是浮点数, 则执行二元化策略: 以该数值为界:
    - 特征取值大于它的作为 1。
    - 特征取值小于它的作为 0。
- 其它参数参考 `MultinomialNB`。



3. 模型属性：参考 `MultinomialNB` 。

4. 模型方法：参考 `MultinomialNB` 。

## 四、决策树

### 4.1 DecisionTreeRegressor

1. `DecisionTreeRegressor` 是回归决策树，其原型为：

```
class sklearn.tree.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, presort=False)
```

- `criterion`：一个字符串，指定切分质量的评价准则。  
默认为 `'mse'`，且只支持该字符串，表示均方误差。
- `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。
- `max_features`：可以为整数、浮点、字符串或者 `None`，指定寻找最优拆分时考虑的特征数量。
  - 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 `'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 `None` 或者 `'auto'`，则 `max_features` 等于 `n_features`。

注意：如果已经考虑了 `max_features` 个特征，但是还没有找到一个有效的切分，那么还会继续寻找下一个特征，直到找到一个有效的切分为止。
- `max_depth`：可以为整数或者 `None`，指定树的最大深度。
  - 如果为 `None`，则表示树的深度不限。  
分裂子结点，直到每个叶子都是纯的（即：叶结点中所有样本点都属于一个类），或者叶结点中包含小于 `min_samples_split` 个样点。
  - 如果 `max_leaf_nodes` 参数非 `None`，则忽略此选项。
- `min_samples_split`：为整数，指定每个内部结点包含的最少的样本数。
- `min_samples_leaf`：为整数，指定每个叶结点包含的最少的样本数。
- `min_weight_fraction_leaf`：为浮点数，叶结点中样本的最小权重系数。
- `max_leaf_nodes`：为整数或者 `None`，指定最大的叶结点数量。
  - 如果为 `None`，此时叶结点数量不限。
  - 如果非 `None`，则 `max_depth` 被忽略。
- `class_weight`：为一个字典、字符串 `'balanced'`、或者 `None`。它指定了分类的权重。

- 如果为字典，则权重的形式为：`{class_label:weight}`。
- 如果为字符串 `'balanced'`，则表示分类的权重是样本中各分类出现的频率的反比。
- 如果为 `None`，则每个分类的权重都为1。

注意：如果提供了 `sample_weight` 参数（由 `fit` 方法提供），则这些权重都会乘以 `sample_weight`。

- `random_state`：指定随机数种子。
- `presort`：一个布尔值，指定是否要提前排序数据从而加速寻找最优切分的过程。
  - 对于大数据集，设置为 `True` 会减慢总体的训练过程。
  - 对于一个小数据集或者设定了最大深度的情况下，设置为 `True` 会加速训练过程。

## 2. 模型属性：

- `feature_importances_`：给出了特征的重要程度。该值越高，则该特征越重要。
- `max_features_`：`max_features` 的推断值。
- `n_features_`：当执行 `fit` 之后，特征的数量。
- `n_outputs_`：当执行 `fit` 之后，输出的数量。
- `tree_`：一个 `Tree` 对象，即底层的决策树。

## 3. 模型方法：

- `fit(X, y[, sample_weight, check_input, ...])`：训练模型。
- `predict(X[, check_input])`：用模型进行预测，返回预测值。
- `score(X,y[,sample_weight])`：返回模型的预测性能得分。

## 4.2 DecisionTreeClassifier

### 1. `DecisionTreeClassifier` 是分类决策树，其原型为：

```
sklearn.tree.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, class_weight=None,
presort=False)
```

- `criterion`：一个字符串，指定切分质量的评价准则。可以为：
  - `'gini'`：表示切分时评价准则是 Gini 系数
  - `'entropy'`：表示切分时评价准则是熵
- 其它参数参考 `DecisionTreeRegressor`。

### 2. 模型属性：

- `classes_`：分类的标签值。
- `n_classes_`：给出了分类的数量。
- 其它属性参考 `DecisionTreeRegressor`。

### 3. 模型方法：

- `fit(X, y[, sample_weight, check_input, ...])`：训练模型。
- `predict(X[, check_input])`：用模型进行预测，返回预测值。
- `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。
- `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。

- `score(X,y[,sample_weight])` : 返回模型的预测性能得分。

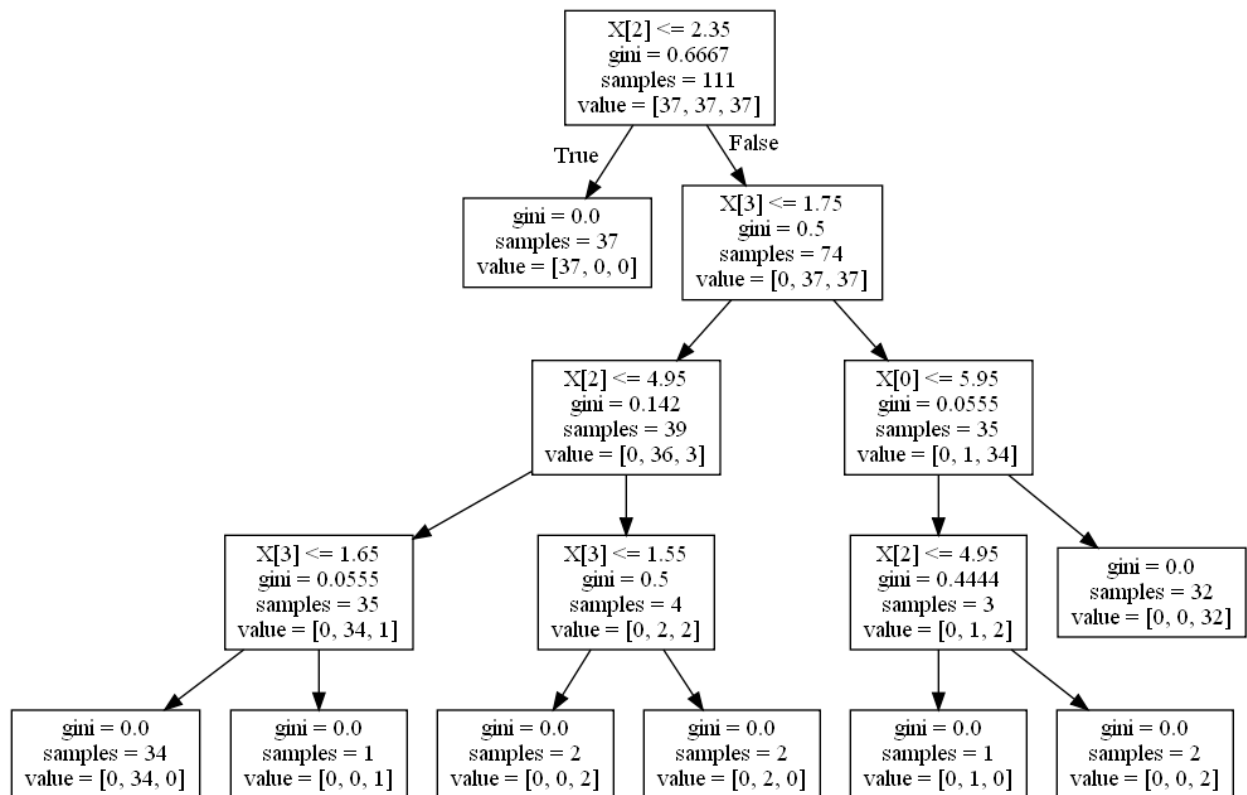
## 4.3 决策图

1. 当训练完毕一棵决策树的时候, 可以通过 `sklearn.tree.export_graphviz(classifier,out_file)` 来将决策树转化成 `Graphviz` 格式的文件。

这里要求安装 `Graphviz` 程序。`Graphviz` 是贝尔实验室开发的一个开源的工具包, 用于绘制结构化的图形网络, 支持多种格式输出如常用的图片格式、SVG、PDF格式等, 且支持 `Linux\mid Windows` 操作系统。

2. 然后通过 `Graphviz` 的 `dot` 工具, 在命令行中运行命令 `dot.exe -Tpdf F:\mid out -o F:\mid out.pdf` 生成 `pdf` 格式的决策图; 或者执行 `dot.exe -Tpng F:\mid out -o F:\mid out.png` 来生成 `png` 格式的决策图。

其中: `-T` 选项指定了输出文件的格式, `-o` 选项指定了输出文件名。



## 五、KNN

### 5.1 KNeighborsClassifier

1. `KNeighborsClassifier` 是 `knn` 分类模型, 其原型为:

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

- `n_neighbors` : 一个整数, 指定  $k$  值。
- `weights` : 一字符串或者可调用对象, 指定投票权重策略。

- 'uniform' : 本结点的所有邻居结点的投票权重都相等。
- 'distance' : 本结点的所有邻居结点的投票权重与距离成反比。即越近的结点, 其投票权重越大。
- 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。
- algorithm : 一字符串, 指定计算最近邻的算法。可以为:
  - 'ball\_tree' : 使用 BallTree 算法。
  - 'kd\_tree' : 使用 KDTree 算法。
  - 'brute' : 使用暴力搜索法。
  - 'auto' : 自动决定最合适的算法。
- leaf\_size : 一个整数, 指定 BallTree 或者 KDTree 叶结点规模。它影响了树的构建和查询速度。
- metric : 一个字符串, 指定距离度量。默认为 'minkowski' 距离。
- p : 一个整数, 指定在 'Minkowski' 度量上的指数。  
如果 p=1, 对应于曼哈顿距离; p=2 对应于欧拉距离。
- n\_jobs : 并行度。

## 2. 模型方法:

- fit(X,y) : 训练模型。
- predict(X) : 用模型进行预测, 返回预测值。
- score(X,y) : 返回模型的预测性能得分。
- 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。
- kneighbors([X, n\_neighbors, return\_distance]) : 返回样本点的  $k$  近邻点。如果 return\_distance=True, 同时还返回到这些近邻点的距离。
- kneighbors\_graph([X, n\_neighbors, mode]) : 返回样本点的  $k$  近邻点的连接图。

## 5.2 KNeighborsRegressor

1. KNeighborsRegressor 是 knn 回归模型, 其原型为:

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

参数: 参考 KNeighborsClassifier。

## 2. 模型方法:

- fit(X,y) : 训练模型。
- predict(X) : 用模型进行预测, 返回预测值。
- score(X,y) : 返回模型的预测性能得分。
- kneighbors([X, n\_neighbors, return\_distance]) : 返回样本点的  $k$  近邻点。如果 return\_distance=True, 同时还返回到这些近邻点的距离。
- kneighbors\_graph([X, n\_neighbors, mode]) : 返回样本点的  $k$  近邻点的连接图。

# 六、AdaBoost

## 6.1 AdaBoostClassifier

1. `AdaBoostClassifier` 是 `AdaBoost` 分类器，其原型为：

```
class sklearn.ensemble.AdaBoostClassifier(base_estimator=None, n_estimators=50,
learning_rate=1.0, algorithm='SAMME.R', random_state=None)
```

- `base_estimator`：一个基础分类器对象，该基础分类器必须支持带样本权重的学习。默认为 `DecisionTreeClassifier`。
- `n_estimators`：一个整数，指定基础分类器的数量（默认为50）。  
当然如果训练集已经完美的训练好了，可能算法会提前停止，此时基础分类器数量少于该值。
- `learning_rate`：一个浮点数，表示学习率，默认为1。它就是下式中的  $\nu$ ：  

$$H_m(\vec{x}) = H_{m-1}(\vec{x}) + \nu \alpha_m h_m(\vec{x})。$$
  - 它用于减少每一步的步长，防止步长太大而跨过了极值点。
  - 通常学习率越小，则需要的基础分类器数量会越多，因此在 `learning_rate` 和 `n_estimators` 之间会有所折中。
- `algorithm`：一个字符串，指定用于多类分类问题的算法，默认为 `'SAMME.R'`。
  - `'SAMME.R'`：使用 `SAMME.R` 算法。基础分类器对象必须支持计算类别的概率。  
通常 `'SAMME.R'` 收敛更快，且误差更小、迭代数量更少。
  - `'SAMME'`：使用 `SAMME` 算法。
- `random_state`：指定随机数种子。

2. 模型属性：

- `estimators_`：所有训练过的基础分类器。
- `classes_`：所有的类别标签。
- `n_classes_`：类别数量。
- `estimator_weights_`：每个基础分类器的权重。
- `estimator_errors_`：每个基础分类器的分类误差。
- `feature_importances_`：每个特征的重要性。

3. 模型方法：

- `fit(X, y[, sample_weight])`：训练模型。
- `predict(X)`：用模型进行预测，返回预测值。
- `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。
- `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。
- `score(X, y[, sample_weight])`：返回模型的预测性能得分。
- `staged_predict(X)`：返回一个数组，数组元素依次是：集成分类器在每一轮迭代结束时的预测值。
- `staged_predict_proba(X)`：返回一个二维数组，数组元素依次是：集成分类器在每一轮迭代结束时，预测 `x` 为各个类别的概率值。
- `staged_score(X, y[, sample_weight])`：返回一个数组，数组元素依次是：集成分类器在每一轮迭代结束时，该集成分类器的预测性能得分。

## 6.1 AdaBoostRegressor

1. `AdaBoostRegressor` 是 `AdaBoost` 回归器，其原型为：

```
class sklearn.ensemble.AdaBoostRegressor(base_estimator=None, n_estimators=50,
learning_rate=1.0, loss='linear', random_state=None)
```

- `base_estimator` : 一个基础回归器对象，该基础回归器必须支持带样本权重的学习。默认为 `DecisionTreeRegressor`。
- `loss` : 一个字符串。指定了损失函数。可以为：
  - `'linear'` : 线性损失函数（默认）。
  - `'square'` : 平方损失函数。
  - `'exponential'` : 指数损失函数。
- 其它参数参考 `AdaBoostClassifier`。

## 2. 模型属性：

- `estimators_` : 所有训练过的基础回归器。
- `estimator_weights_` : 每个基础回归器的权重。
- `estimator_errors_` : 每个基础回归器的回归误差。
- `feature_importances_` : 每个特征的重要性。

## 3. 模型方法：

- `fit(X, y[, sample_weight])` : 训练模型。
- `predict(X)` : 用模型进行预测，返回预测值。
- `score(X, y[, sample_weight])` : 返回模型的预测性能得分。
- `staged_predict(X)` : 返回一个数组，数组元素依次是：集成回归器在每一轮迭代结束时的预测值。
- `staged_score(X, y[, sample_weight])` : 返回一个数组，数组元素依次是：集成回归器在每一轮迭代结束时，该集成回归器的预测性能得分。

# 七、梯度提升树

## 7.1 GradientBoostingClassifier

1. `GradientBoostingClassifier` 是 `GBDT` 分类模型，其原型为：

```
class sklearn.ensemble.GradientBoostingClassifier(loss='deviance', learning_rate=0.1,
n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None,
max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto')
```

- `loss` : 一个字符串，指定损失函数。可以为：
  - `'deviance'`（默认值）：此时损失函数为对数损失函数： $L(\hat{y}, \hat{y}) = -\log p(\hat{y})$ 。
  - `'exponential'`：此时使用指数损失函数。
- `n_estimators` : 一个整数，指定基础决策树的数量（默认为100），值越大越好。
- `learning_rate` : 一个浮点数，表示学习率，默认为1。它就是下式中的  $\nu$ ：
 
$$H_m(\vec{x}) = H_{m-1}(\vec{x}) + \nu \alpha_m h_m(\vec{x})。$$
  - 它用于减少每一步的步长，防止步长太大而跨过了极值点。

- 通常学习率越小，则需要的基础分类器数量会越多，因此在 `learning_rate` 和 `n_estimators` 之间会有所折中。
- `max_depth`：一个整数或者 `None`，指定了每个基础决策树模型的 `max_depth` 参数。
  - 调整该参数可以获得最佳性能。
  - 如果 `max_leaf_nodes` 不是 `None`，则忽略本参数。
- `min_samples_split`：一个整数，指定了每个基础决策树模型的 `min_samples_split` 参数。
- `min_samples_leaf`：一个整数，指定了每个基础决策树模型的 `min_samples_leaf` 参数。
- `min_weight_fraction_leaf`：一个浮点数，指定了每个基础决策树模型的 `min_weight_fraction_leaf` 参数。
- `subsample`：一个大于 0 小于等于 1.0 的浮点数，指定了提取原始训练集中多大比例的一个子集用于训练基础决策树。
  - 如果 `subsample` 小于 1.0，则梯度提升决策树模型就是随机梯度提升决策树。此时会减少方差但是提高了偏差。
  - 它会影响 `n_estimators` 参数。
- `max_features`：一个整数或者浮点数或者字符串或者 `None`，指定了每个基础决策树模型的 `max_features` 参数。
 

如果 `max_features < n_features`，则会减少方差但是提高了偏差。
- `max_leaf_nodes`：为整数或者 `None`，指定了每个基础决策树模型的 `max_leaf_nodes` 参数。
- `init`：一个基础分类器对象或者 `None`，该分类器对象用于执行初始的预测。
 

如果为 `None`，则使用 `loss.init_estimator`。
- `verbose`：一个正数。用于开启/关闭迭代中间输出日志功能。
- `warm_start`：一个布尔值。用于指定是否继续使用上一次训练的结果。
- `random_state`：一个随机数种子。
- `presort`：一个布尔值或者 'auto'。指定了每个基础决策树模型的 `presort` 参数。

## 2. 模型属性：

- `feature_importances_`：每个特征的重要性。
- `oob_improvement_`：给出训练过程中，每增加一个基础决策树，在测试集上损失函数的改善情况（即：损失函数的减少值）。
- `train_score_`：给出训练过程中，每增加一个基础决策树，在训练集上的损失函数的值。
- `init`：初始预测使用的分类器。
- `estimators_`：所有训练过的基础决策树。

## 3. 模型方法：

- `fit(X, y[, sample_weight, monitor])`：训练模型。
 

其中 `monitor` 是一个可调对象，它在当前迭代过程结束时调用。如果它返回 `True`，则训练过程提前终止。
- `predict(X)`：用模型进行预测，返回预测值。
- `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。
- `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。



- `score(X, y[, sample_weight])` : 返回模型的预测性能得分。
- `staged_predict(X)` : 返回一个数组，数组元素依次是：GBDT 在每一轮迭代结束时的预测值。
- `staged_predict_proba(X)` : 返回一个二维数组，数组元素依次是：GBDT 在每一轮迭代结束时，预测 `X` 为各个类别的概率值。
- `staged_score(X, y[, sample_weight])` : 返回一个数组，数组元素依次是：GBDT 在每一轮迭代结束时，该 GBDT 的预测性能得分。

## 7.2 GradientBoostingRegressor

1. `GradientBoostingRegressor` 是 GBRT 回归模型，其原型为：

```
class sklearn.ensemble.GradientBoostingRegressor(loss='ls', learning_rate=0.1,
n_estimators=100, subsample=1.0, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_depth=3, init=None, random_state=None,
max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False,
presort='auto')
```

- `loss` : 一个字符串，指定损失函数。可以为：
  - `'ls'` : 损失函数为平方损失函数。
  - `'lad'` : 损失函数为绝对值损失函数。
  - `'huber'` : 损失函数为上述两者的结合，通过 `alpha` 参数指定比例，该损失函数的定义为：

$$L_{Huber} = \begin{cases} \frac{1}{2}(y - f(x))^2, & \text{if } |y - f(x)| \leq \alpha \\ \alpha|y - f(x)| - \frac{1}{2}\alpha^2, & \text{else} \end{cases}$$

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。

- `'quantile'` : 分位数回归（分位数指得是百分之几），通过 `alpha` 参数指定分位数。
- `alpha` : 一个浮点数，只有当 `loss='huber'` 或者 `loss='quantile'` 时才有效。
- `n_estimators` : 其它参数参考 `GradientBoostingClassifier`。

2. 模型属性：

- `feature_importances_` : 每个特征的重要性。
- `oob_improvement_` : 给出训练过程中，每增加一个基础决策树，在测试集上损失函数的改善情况（即：损失函数的减少值）。
- `train_score_` : 给出训练过程中，每增加一个基础决策树，在训练集上的损失函数的值。
- `init` : 初始预测使用的回归器。
- `estimators_` : 所有训练过的基础决策树。

3. 模型方法：

- `fit(X, y[, sample_weight, monitor])` : 训练模型。

其中 `monitor` 是一个可调对象，它在当前迭代过程结束时调用。如果它返回 `True`，则训练过程提前终止。

- `predict(X)` : 用模型进行预测，返回预测值。



- `score(X,y[,sample_weight])` : 返回模型的预测性能得分。
- `staged_predict(X)` : 返回一个数组，数组元素依次是：`GBRT` 在每一轮迭代结束时的预测值。
- `staged_score(X, y[, sample_weight])` : 返回一个数组，数组元素依次是：`GBRT` 在每一轮迭代结束时，该 `GBRT` 的预测性能得分。

## 八、Random Forest

1. `scikit-learn` 基于随机森林算法提供了两个模型：

- `RandomForestClassifier` 用于分类问题
- `RandomForestRegressor` 用于回归问题

### 8.1 RandomForestClassifier

1. `GradientBoostingClassifier` 是随机森林分类模型，其原型为：

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=10, 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, bootstrap=True, oob_score=False, n_jobs=1,
random_state=None, verbose=0, warm_start=False, class_weight=None)
```

- `n_estimators` : 一个整数，指定了随机森林中决策树的数量。
- `criterion` : 一个字符串，指定了每个决策树的 `criterion` 参数。
- `max_features` : 一个整数或者浮点数或者字符串或者 `None`，指定了每个决策树的 `max_features` 参数。
- `max_depth` : 一个整数或者 `None`，指定了每个决策树的 `max_depth` 参数。  
如果 `max_leaf_nodes` 不是 `None`，则忽略本参数。
- `min_samples_split` : 一个整数，指定了每个决策树的 `min_samples_split` 参数。
- `min_samples_leaf` : 一个整数，指定了每个决策树的 `min_samples_leaf` 参数。
- `min_weight_fraction_leaf` : 一个浮点数，指定了每个决策树的 `min_weight_fraction_leaf` 参数。
- `max_leaf_nodes` : 为整数或者 `None`，指定了每个基础决策树模型的 `max_leaf_nodes` 参数。
- `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集。
- `oob_score` : 为布尔值。如果为 `True`，则使用包外样本来计算泛化误差。
- `n_jobs` : 指定并行性。
- `random_state` : 指定随机数种子。
- `verbose` : 一个正数。用于开启/关闭迭代中间输出日志功能。
- `warm_start` : 一个布尔值。用于指定是否继续使用上一次训练的结果。
- `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 `None`：

- 如果为字典：则字典给出了每个分类的权重，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中出现的频率成反比。
- 如果为字符串 `'balanced_subsample'`：则样本集为采样法 `bootstrap sampling` 产生的决策树的训练数据集，每个分类的权重与该分类在采用生成的样本集中出现的频率成反比。
- 如果为 `None`：则每个分类的权重都为 `1`。

## 2. 模型属性：

- `estimators_`：所有训练过的基础决策树。
- `classes_`：所有的类别标签。
- `n_classes_`：类别数量。
- `n_features_`：训练时使用的特征数量。
- `n_outputs_`：训练时输出的数量。
- `feature_importances_`：每个特征的重要性。
- `oob_score_`：训练数据使用包外估计时的得分。

## 3. 模型方法：

- `fit(X, y[, sample_weight])`：训练模型。
- `predict(X)`：用模型进行预测，返回预测值。
- `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。
- `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。
- `score(X, y[, sample_weight])`：返回模型的预测性能得分。

# 8.2 RandomForestRegressor

## 1. `RandomForestRegressor` 是随机森林回归模型，其原型为：

```
class sklearn.ensemble.RandomForestRegressor(n_estimators=10, 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, bootstrap=True, oob_score=False, n_jobs=1,
random_state=None, verbose=0, warm_start=False)
```

参数：参考 `GradientBoostingClassifier`。

## 2. 模型属性：

- `estimators_`：所有训练过的基础决策树。
- `n_features_`：训练时使用的特征数量。
- `n_outputs_`：训练时输出的数量。
- `feature_importances_`：每个特征的重要性。
- `oob_score_`：训练数据使用包外估计时的得分。
- `oob_prediction_`：训练数据使用包外估计时的预测值。

## 3. 模型方法：

- `fit(X, y[, sample_weight])`：训练模型。
- `predict(X)`：用模型进行预测，返回预测值。
- `score(X, y[, sample_weight])`：返回模型的预测性能得分。