逻辑回归

Leon

2018 年 12 月 23 日

1 机器学习的概念

1.1 有监督学习

对训练集来说 X 对应着是确定的 f(x)，然后通过构建模型，进行超参的学习

1.2 无监督学习

大部分无监督学习都是没有确定的 f(x) 的，通过一些规则，让机器自己去判断，比如 knn 算法，用距离来做聚类。

1.3 泛化能力

在机器学习方法中，泛化能力通俗来讲就是指学习到的模型对未知数据的预测能力。在实际情况中，我们通常通过测试误差来评价学习方法的泛化能力。

1.4 过拟合

1.4.1 概念

先谈谈过拟合，所谓过拟合，指的是模型在训练集上表现的很好，但是在交叉验证集合测试集上表现一般，也就是说模型对未知样本的预测表现一般，泛化（generalization）能力较差。

1.4.2 解决办法

一般的方法有 early stopping、数据集扩增（Data augmentation）、正则化（Regularization）、Dropout 等。在机器学习算法中，我们常常将原始数据

集分为三部分：training data、validation data，testing data。这个 validation data 是什么？它其实就是用来避免过拟合的，在训练过程中，我们通常用它来确定一些超参数（比如根据 validation data 上的 accuracy 来确定 early stopping 的 epoch 大小、根据 validation data 确定 learning rate 等等）。那为啥不直接在 testing data 上做这些呢？因为如果在 testing data 做这些，那么随着训练的进行，我们的网络实际上就是在一点一点地 overfitting 我们的 testing data，导致最后得到的 testing accuracy 没有任何参考意义。

Early stopping: Early stopping 便是一种迭代次数截断的方法来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。对模型进行训练的过程即是对模型的参数进行学习更新的过程，这个参数学习的过程往往会用到一些迭代方法，如梯度下降（Gradient descent）学习算法。这样可以有效阻止过拟合的发生，因为过拟合本质上就是对自身特点过度地学习。

正则化: 指的是在目标函数后面添加一个正则化项，一般有 L1 正则化与 L2 正则化。L1 正则是基于 L1 范数，即在目标函数后面加上参数的 L1 范数和项，即参数绝对值和与参数的积项

$$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

L2 正则是基于 L2 范数，即在目标函数后面加上参数的 L2 范数和项，即参数的平方和与参数的积项：

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

1.5 交叉验证 (cross-validation)

交叉验证，是重复的使用数据，把得到的样本数据进行切分，组合为不同的训练集和测试集，用训练集来训练模型，用测试集来评估模型预测的好坏。在此基础上可以得到多组不同的训练集和测试集，某次训练集中的某样本在下次可能成为测试集中的样本，即所谓“交叉”。有简单交叉验证、S 折交叉验证、留一交叉验证。

1.6 线性回归的原理

1: 函数模型 (Model):

$$h_w(x^i) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + ... + \omega_n x_n = \sum \omega^T x_i = W^T X$$

$$X = \begin{bmatrix} 1 \\ x_1 \\ ... \\ x_n \end{bmatrix}, W = \begin{bmatrix} \omega_0 \\ \omega_2 \\ ... \\ \omega_n \end{bmatrix} \tag{1}$$

假设有训练数据

$$D = (X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)$$

那么方便我们写成矩阵的形式

$$X = \begin{bmatrix} 1, x_n^1, x_2^1, ..., x_n^1 \\ 1, x_1^2, x_2^2, ..., x_n^2 \\ ..... \\ 1, x_1^n, x_2^n, ...x_n^n \end{bmatrix}, XW = h_\omega(x^i)$$

2. 损失代价函数:

$$J(W) = \frac{1}{2M}\sum_{i=0}^{M}(h_\omega(x^i) - y^i)^2 = \frac{1}{2M}(XW - y)^T(XW - Y)$$

3. 算法 (algorithm): 求解使得损失函数最小。

1.7 优化方法

1.7.1 梯度下降法

梯度下降沿损失函数的导数方向下降，下降的步幅自己设置。

1.7.2 牛顿法

二阶下降，比梯度下降法更快，而且是求全局最优解，不是局部最优

1.7.3 拟牛顿法

没看懂，但知道适合非线性

1.8 sklearn 参数

Ordinary least squares Linear Regression.

1.8.1 fit_intercept:boolean, optional, default True

whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (e.g. data is expected to be already centered).

1.8.2 normalize : boolean, optional, default False

This parameter is ignored when “fit_intercept“ is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use :class:‘sklearn.preprocessing.StandardScaler‘ before calling “fit“ on an estimator with “normalize=False“.

1.8.3 copy_X : boolean, optional, default True

If True, X will be copied; else, it may be overwritten.

1.8.4 n_jobs : int or None, optional (default=None)

The number of jobs to use for the computation. This will only provide speedup for n_targets > 1 and sufficient large problems.“None“ means 1.

2 逻辑回归

2.1 逻辑回归和线性回归的联系与区别

2.1.1 联系

线性回归和逻辑回归都是广义线性模型，具体的说，都是从指数分布族导出的线性模型，线性回归假设 Y|X 服从高斯分布，逻辑回归假设 Y|X 服从伯努利分布，这两种分布都是属于指数分布族，我们可以通过指数分布族求解广义线性模型（GLM）的一般形式，导出这两种模型。

2.1.2 区别

区别是线性回归是用来做预测任务，逻辑回归是用来做分类任务，把每一个点映射到 0-1 之间，都是用最大似然概率去求解参数值。

2.2 逻辑回归的原理

线性回归的模型是求出输出特征向量 Y 和输入样本矩阵 X 之间的线性关系系数 ，满足 $Y = X\theta^T$。我们的 Y 是连续的，所以是回归模型。如果 Y 是离散的话，可以想到的办法是，我们对于这个 Y 再做一次函数转换，变为 g(Y)。如果我们令 g(Y) 的值在某个实数区间的时候是类别 A，在另一个实数区间的时候是类别 B，以此类推，就得到了一个分类模型。如果结果的类别只有两种，那么就是一个二元分类模型了。逻辑回归的出发点就是从

这来的。下面我们开始引入二元逻辑回归。也就是输入 X，输出的 Y，只有 1,0 两种情况, 而线性回归的 X* 系数矩阵得到的值如果直接通过比较得出 Y 为 1 或 0，是可以做，但是得到的结果无法进行再次学习，或者说很麻烦，优化方法跟导数有关，所以如果要优化，就得打造一个完美的连续函数，比如 sigmoid，方便求导，而且两个极限值是 0 和 1，但是有一个不好的地方，有可能求解到局部最优。

2.3 逻辑回归损失函数推导及优化

对线性回归的结果做一个函数 g 上的转换，可以变为逻辑回归，一般取为 sigmoid 函数, 形式如下:

$$g(z) = \frac{1}{1+e^{-z}}$$

它有一个非常好的导数性质:

$$g'(z) = g(z)(1-g(z))$$

这个通过函数对 g(z) 求导很容易得到如果我们令 g(z) 中的 z 为:$z = x\theta$, 这样就得到了二元逻辑回归模型的一般形式:

$$h_\theta(x) = \frac{1}{1+e^{-x\theta}}$$

其中 x 为样本输入，$h_\theta(x)$ 为模型输出，可以理解为某一分类的概率大小。而 θ 为分类模型的要求出的模型参数。对于模型输出 $h_\theta(x)$, 我们让它和我们的二元样本输出 y(假设 0 和 1) 有这样的对应关系, 如果 $h_\theta(x)$>0.5, 即 $x\theta$>0，则 y 为 1，反之亦然，y=0.5 是临界情况，此时无法确定分类,$h_\theta(x)$ 值越小，而分类为 0 的概率越高，反之，值越大的话分类为 1 的概率越高。如果靠近临界点，则分类准确率会下降。

此处将模型写成矩阵模式: $h_\theta(X) = \frac{1}{1+e^{-X\theta}}$

假设我们的样本输出是 0 或者 1 两类，那么我们有：

$$P(y=1|x,\theta) = h_\theta(x)$$

$$P(y=0|x,\theta) = 1-h_\theta(x)$$

把这两个式子写成一个式子，就是：

$$P(y|x,\theta) = h_\theta(x)^y(1-h_\theta(x))^{1-y}$$

其中 y 的取值只能是 0 或者 1。用矩阵法表示，即为

$$P(Y|X,\theta) = h_\theta(X)^Y(E - h_\theta(X))^{1-Y},\ \ E$$

得到了 y 的概率分布函数表达式，我们就可以用似然函数最大化来求解我们需要的模型系数 。为了方便求解，这里用对数似然函数最大化，对数似然函数取反即为我们的损失函数 J( )。其中：

$$L(\theta) = \prod_{i=1}^{m}(h_\theta(x^{(i)}))^{y(i)}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

其中 m 为样本的个数对似然函数对数化取反的表达式，即损失函数表达式为：

$$J(\theta) = -lnL(\theta) = -\sum_{i=1}^{m}(y^{(i)}log(h_\theta(x^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)})))$$

损失函数用矩阵法表达更加简洁：
J( )=−Y logh (X)−(E−Y) log(E−h (X))
其中 E 为全 1 向量, 为哈达马乘积 (对应位置相乘)。
对于 J( )=−Y logh (X)−(E−Y) log(E−h (X))，我们用 J( ) 对
向量求导可得：

$$\frac{\partial J(\theta)}{\partial \theta} = -Y \bigodot X^T \frac{1}{h_\theta(X)} \bigodot h_\theta(X) \bigodot (E - h_\theta(X)) + (E - Y) \bigodot X^T \frac{1}{1 - h_\theta(X)} \bigodot h_\theta(X) \bigodot (E - h_\theta(X))$$

简化得到
$\frac{\partial}{\partial \theta} J(\theta) = X^T(h_\theta(X) - Y)$
从而在梯度下降法中每一步向量 θ 的迭代公式如下：
$\theta = \theta - \alpha X^T(h_\theta(X) - Y)$
其中，α 为梯度下降法的步长。

2.4 正则化与模型评估指标

在最小化残差平方和的基础上加上 L1 范数或者 L2 范数的惩罚项，如果 L2 正则正则化就是-岭回归 Ridge Regression,L1 正则化是 lasso 回归。回归模型评估指标有
1. 解释方差
$Explained_variance(y, y_hat) = 1 - Var(y - y_hat)/Var(y)$ 2. 绝对平均误差
3. 均方误差
4. 决定系数 (R^2 score)
5.AIC
6.BIC

2.5 逻辑回归的优缺点

优点：

1）预测结果是界于 0 和 1 之间的概率；

2）可以适用于连续性和类别性自变量；

3）容易使用和解释；

缺点：

1）对模型中自变量多重共线性较为敏感，例如两个高度相关自变量同时放入模型，可能导致较弱的一个自变量回归符号不符合预期，符号被扭转。 需要利用因子分析或者变量聚类分析等手段来选择代表性的自变量，以减少候选变量之间的相关性；

2）预测结果呈“S”型，因此从 log(odds) 向概率转化的过程是非线性的，在两端随着 log(odds) 值的变化，概率变化很小，边际值太小，slope 太小，而中间概率的变化很大，很敏感。导致很多区间的变量变化对目标概率的影响没有区分度，无法确定阀值。

2.6 样本不均衡问题的解决办法

分类时，由于训练集合中各样本数量不均衡，导致模型训偏在测试集合上的泛化性不好。解决样本不均衡的方法主要包括两类：（1）数据层面，修改各类别的分布；（2）分类器层面，修改训练算法或目标函数进行改进。还有方法是将上述两类进行融合。

2.6.1 数据层面

过采样

1. 基础版本的过采样：随机过采样训练样本中数量比较少的数据；缺点，容易过拟合；

2. 改进版本的过采样：SMOTE，通过插值的方式加入近邻的数据点；

3. 基于聚类的过采样：先对数据进行聚类，然后对聚类后的数据分别进行过采样。这种方法能够降低类间和类内的不平衡。

4. 神经网络中的过采样：SGD 训练时，保证每个 batch 内部样本均衡。

欠采样

与过采样方法相对立的是欠采样方法，主要是移除数据量较多类别中的部分数据。这个方法的问题在于，丢失数据带来的信息缺失。为克服这一缺点，可以丢掉一些类别边界部分的数据。

2.6.2 分类器层面

过采样，欠采样，都存在相应的问题。

过采样：可能会存在过拟合问题。（可以使用 SMOTE 算法，增加随机的噪声的方式来改善这个问题）

欠采样：可能会存在信息减少的问题。因为只是利用了一部分数据，所以模型只是学习到了一部分模型。

有以下两种方法可以解决欠采样所带来的问题。

方法一：模型融合（bagging 的思想）

思路：从丰富类样本中随机的选取（有放回的选取）和稀有类等量样本的数据。和稀有类样本组合成新的训练集。这样我们就产生了多个训练集，并且是互相独立的，然后训练得到多个分类器。

若是分类问题，就把多个分类器投票的结果（少数服从多数）作为分类结果。

若是回归问题，就将均值作为最后结果。

方法二：增量模型（boosting 的思想）

思路：使用全部的样本作为训练集，得到分类器 L1

从 L1 正确分类的样本中和错误分类的样本中各抽取 50

从 L1 和 L2 分类结果中，选取结果不一致的样本作为训练集得到分类器 L3.

最后投票 L1,L2,L3 结果得到最后的分类结果。

3 决策树

3.1 信息论基础

3.1.1 信息

引用香农的话，信息是用来消除随机不确定性的东西，则某个类（xi）的信息定义如下:

$$I(X = xi) = -log_2 p(x_i)$$

3.1.2 信息熵

信息熵便是信息的期望值，可以记作：

$$H(X) = \sum_{i=1}^{n} p(x_i) I(x_i) = -\sum_{i=1}^{n} p(x_i) log_b p(x_i)$$

熵只依赖 X 的分布，和 X 的取值没有关系，熵是用来度量不确定性，当熵越大，概率说 X=xi 的不确定性越大，反之越小，在机器学期中分类中说，熵越大即这个类别的不确定性更大，反之越小。

当 p=0 或 p=1 时，H(p)=0, 随机变量完全没有不确定性，当 p=0.5 时，H(p)=1, 此时随机变量的不确定性最大

3.1.3 条件熵

X 给定条件下 Y 的条件分布的熵对 X 的数学期望，在机器学习中为选定某个特征后的熵，公式如下：

$$H(Y|X) = \sum_{x} p(x) H(Y|X = x)$$

一个特征对应着多个类别 Y，因此在此的多个分类即为 X 的取值 x。

3.1.4 信息增益

信息增益在决策树算法中是用来选择特征的指标，信息增益越大，则这个特征的选择性越好，在概率中定义为：待分类的集合的熵和选定某个特征的条件熵之差（这里只的是经验熵或经验条件熵，由于真正的熵并不知道，是根据样本计算出来的），公式如下：

$$IG(Y|X) = H(Y) - H(Y|X)$$

3.1.5 基尼不纯度

为了构造决策树，算法首先创建一个根节点，然后评估表中的所有观测变量，从中选出最合适的变量对数据进行拆分。为了选择合适的变量，我们需要一种方法来衡量数据集合中各种因素的混合情况。

基尼不纯度：将来自集合中的某种结果随机应用于集合中某一数据项的预期误差率。

维基上的公式是这样：

$$I_G(f) = \sum_{i=1}^{m} f_i(1-f_i) = \sum_{i=1}^{m}(f_i - f_i^2) = \sum_{i=1}^{m} f_i - \sum_{i=1}^{m} f_i^2 = 1 - \sum_{i=1}^{m} f_i^2$$

3.2 决策树的不同分类算法的原理及应用场景

3.2.1 ID3 决策树

信息熵是度量样本集合纯度最常用的一种指标。假设样本集合 D 中第 k 类样本所占的比重为 pk，那么信息熵的计算则为下面的计算方式

$$Ent(D) = -\sum_{k=1}^{|y|} p_k log_2 p_k$$

当这个 Ent(D) 的值越小，说明样本集合 D 的纯度就越高

有了信息熵，当我选择用样本的某一个属性 a 来划分样本集合 D 时，就可以得出用属性 a 对样本 D 进行划分所带来的“信息增益”

$$Gain(D,a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v)$$

一般来讲，信息增益越大，说明如果用属性 a 来划分样本集合 D，那么纯度会提升，因为我们分别对样本的所有属性计算增益情况，选择最大的来作为决策树的一个结点，或者可以说那些信息增益大的属性往往离根结点越近，因为我们会优先用能区分度大的也就是信息增益大的属性来进行划分。当一个属性已经作为划分的依据，在下面就不在参与竞选了，我们刚才说过根结点代表全部样本，而经过根结点下面属性各个取值后样本又可以按照相应属性值进行划分，并且在当前的样本下利用剩下的属性再次计算信息增益来进一步选择划分的结点，ID3 决策树就是这样建立起来的。

3.2.2 C4.5 决策树

C4.5 决策树的提出完全是为了解决 ID3 决策树的一个缺点，当一个属性的可取值数目较多时，那么可能在这个属性对应的可取值下的样本只有一个或者是很少个，那么这个时候它的信息增益是非常高的，这个时候纯度

很高，ID3 决策树会认为这个属性很适合划分，但是较多取值的属性来进行划分带来的问题是它的泛化能力比较弱，不能够对新样本进行有效的预测。

而 C4.5 决策树则不直接使用信息增益来作为划分样本的主要依据，而提出了另外一个概念，增益率

$$Gainratio(D,a) = \frac{Gain(D,a)}{IV(a)}$$

$$IV(a) = -\sum_{v=1}^{V} \frac{|D^v|}{|D|} log_2 \frac{|D^v|}{|D|}$$

但是同样的这个增益率对可取值数目较少的属性有所偏好，因此 C4.5 决策树先从候选划分属性中找出信息增益高于平均水平的属性，在从中选择增益率最高的。

3.2.3 CART 决策树

CART 决策树的全称为 Classification and Regression Tree, 可以应用于分类和回归。

采用基尼系数来划分属性

3.3 回归树原理

回归树总体流程类似于分类树，区别在于，回归树的每一个节点都会得一个预测值，以年龄为例，该预测值等于属于这个节点的所有人年龄的平均值。分枝时穷举每一个 feature 的每个阈值找最好的分割点，但衡量最好的标准不再是最大熵，而是最小化平方误差。也就是被预测出错的人数越多，错的越离谱，平方误差就越大，通过最小化平方误差能够找到最可靠的分枝依据。分枝直到每个叶子节点上人的年龄都唯一或者达到预设的终止条件(如叶子个数上限)，若最终叶子节点上人的年龄不唯一，则以该节点上所有人的平均年龄做为该叶子节点的预测年龄。

3.4 决策树防止过拟合

3.4.1 先剪枝

通过提前停止树的构建而对树“剪枝”，一旦停止，节点就成为树叶。该树叶可以持有子集元组中最频繁的类；

3.4.2 后剪枝

它首先构造完整的决策树，允许树过度拟合训练数据，然后对那些置信度不够的结点子树用叶子结点来代替，该叶子的类标号用该结点子树中最频繁的类标记。后剪枝的剪枝过程是删除一些子树，然后用其叶子节点代替，这个叶子节点所标识的类别通过大多数原则 (majority class criterion) 确定。所谓大多数原则，是指剪枝过程中, 将一些子树删除而用叶节点代替, 这个叶节点所标识的类别用这棵子树中大多数训练样本所属的类别来标识, 所标识的类称为 majority class . 相比于先剪枝，这种方法更常用，正是因为在先剪枝方法中精确地估计何时停止树增长很困难。

3.5 模型评估

3.5.1 自助法

训练集是对于原数据集的有放回抽样，如果原始数据集 N，可以证明，大小为 N 的自助样本大约包含原数据 63.2

3.5.2 准确的区间估计

将分类问题看做二项分布，则有：令 X 为模型正确分类，p 为准确率，X 服从均值 Np、方差 Np（1-p）的二项分布。acc=X/N 为均值 p，方差 p（1-p）/N 的二项分布。

3.6 sklearn 参数解析

```
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.,
                      max_features=None,
                      random_state=None,
                      max_leaf_nodes=None,
                      min_impurity_decrease=0.,
                      min_impurity_split=None,
                      presort=False)
```

```
,,,
1.criterion:string, optional (default="mse")
        它指定了切分质量的评价准则。默认为'mse'(mean squared error)
2.splitter:string, optional (default="best")
        它指定了在每个节点切分的策略。有两种切分策略:
        (1).splitter='best':表示选择最优的切分特征和切分点。
        (2).splitter='random':表示随机切分。
3.max_depth:int or None, optional (default=None)
        指定树的最大深度。如果为None,则表示树的深度不限,直到
        每个叶子都是纯净的,即叶节点中所有样本都属于同一个类别,
        或者叶子节点中包含小于min_samples_split个样本。
4.min_samples_split:int, float, optional (default=2)
        整数或者浮点数,默认为2。它指定了分裂一个内部节点(非叶子节
        需要的最小样本数。如果为浮点数(0到1之间),最少样本分割数为
5.min_samples_leaf:int, float, optional (default=1)
        整数或者浮点数,默认为1。它指定了每个叶子节点包含的最少样本
        如果为浮点数(0到1之间),每个叶子节点包含的最少样本数为ceil
6.min_weight_fraction_leaf:float, optional (default=0.)
        它指定了叶子节点中样本的最小权重系数。默认情况下样本有相同
7.max_feature:int, float, string or None, optional (default=None)
        可以是整数,浮点数,字符串或者None。默认为None。
        (1).如果是整数,则每次节点分裂只考虑max_feature个特征。
        (2).如果是浮点数(0到1之间),则每次分裂节点的时候只考虑int(
        (3).如果是字符串'auto',max_features=n_features。
        (4).如果是字符串'sqrt',max_features=sqrt(n_features)。
        (5).如果是字符串'log2',max_features=log2(n_features)。
        (6).如果是None,max_feature=n_feature。
8.random_state:int, RandomState instance or None, optional (default=Non
        (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。
9.max_leaf_nodes:int or None, optional (default=None)
        (1).如果为None,则叶子节点数量不限。
        (2).如果不为None,则max_depth被忽略。
10.min_impurity_decrease:float, optional (default=0.)
        如果节点的分裂导致不纯度的减少(分裂后样本比分裂前更加纯净)
```

个人理解这个参数应该是针对分类问题时才有意义。这里的不纯度
回归生成树采用的是平方误差最小化策略。分类生成树采用的是基
加权不纯度的减少量计算公式为：
*min_impurity_decrease=N_t / N * (impurity − N_t_R / N_t *
− N_t_L / N_t * left_impurity)*
其中*N*是样本的总数，*N_t*是当前节点的样本数，*N_t_L*是分裂后左
*N_t_R*是分裂后右子节点的样本数。*impurity*指当前节点的基尼指
分裂后右子节点的基尼指数。*left_impurity*指分裂后左子节点的

11.min_impurity_split:float
树生长过程中早停止的阈值。如果当前节点的不纯度高于阈值，节
这个参数已经被弃用。用*min_impurity_decrease*代替了*min_impu*

12.presort: *bool, optional (default=False)*
指定是否需要提前排序数据从而加速寻找最优切分的过程。设置为
会减慢总体的训练过程；但是对于一个小数据集或者设定了最大深

属性：

1.feature_importances_ : array of shape = [n_features]
特征重要性。该值越高，该特征越重要。
特征的重要性为该特征导致的评价准则的（标准化的）总减少量。

2.max_feature_:int
*max_features*推断值。

*3.n_features_*: *int*
执行*fit*的时候，特征的数量。

4.n_outputs_ : int
执行*fit*的时候，输出的数量。

5.tree_ : 底层的*Tree*对象。

Notes:
控制树大小的参数的默认值（例如‘‘*max_depth*‘‘，‘‘*min_samples_leaf*‘‘等）导
这些树在某些数据集上可能表现很好。为减少内存消耗，应通过设置这些参数值
方法：

*1.fit(X, y):*训练模型。

*2.predict(X):*预测。

’’’

```
from sklearn.tree import DecisionTreeClassifier
```

’’’

分类决策树

```
'''
DecisionTreeClassifier(criterion="gini",
                       splitter="best",
                       max_depth=None,
                       min_samples_split=2,
                       min_samples_leaf=1,
                       min_weight_fraction_leaf=0.,
                       max_features=None,
                       random_state=None,
                       max_leaf_nodes=None,
                       min_impurity_decrease=0.,
                       min_impurity_split=None,
                       class_weight=None,
                       presort=False)
'''
```

参数含义：

1. *criterion:string, optional (default="gini")*
   (1). *criterion='gini'*, 分裂节点时评价准则是*Gini*指数。
   (2). *criterion='entropy'*, 分裂节点时的评价指标是信息增益。
2. *max_depth:int or None, optional (default=None)*。指定树的最大深度。
   如果为*None*，表示树的深度不限。直到所有的叶子节点都是纯净的，
   中所有的样本点都属于同一个类别。或者每个叶子节点包含的样本数
3. *splitter:string, optional (default="best")*。指定分裂节点时的策略。
   (1). *splitter='best'*, 表示选择最优的分裂策略。
   (2). *splitter='random'*, 表示选择最好的随机切分策略。
4. *min_samples_split:int, float, optional (default=2)*。表示分裂一个内部节
   (1). 如果为整数，则*min_samples_split*就是最少样本数。
   (2). 如果为浮点数(*0*到*1*之间)，则每次分裂最少样本数为*ceil(min_*
5. *min_samples_leaf: int, float, optional (default=1)*。指定每个叶子节点需
   (1). 如果为整数，则*min_samples_split*就是最少样本数。
   (2). 如果为浮点数(*0*到*1*之间)，则每个叶子节点最少样本数为*ceil(n*
6. *min_weight_fraction_leaf:float, optional (default=0.)*
   指定叶子节点中样本的最小权重。
7. *max_features:int, float, string or None, optional (default=None).*
   搜寻最佳划分的时候考虑的特征数量。
   (1). 如果为整数，每次分裂只考虑*max_features*个特征。

(2). 如果为浮点数 (0到1之间), 每次切分只考虑 int(max_features *
(3). 如果为 'auto' 或者 'sqrt', 则每次切分只考虑 sqrt(n_features)
(4). 如果为 'log2', 则每次切分只考虑 log2(n_features) 个特征。
(5). 如果为 None, 则每次切分考虑 n_features 个特征。
(6). 如果已经考虑了 max_features 个特征, 但还是没有找到一个有效
下一个特征, 直到找到一个有效的切分为止。

8. random_state: int, RandomState instance or None, optional (default=Non
(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。

9. max_leaf_nodes: int or None, optional (default=None)。指定了叶子节点的
(1). 如果为 None, 叶子节点数量不限。
(2). 如果为整数, 则 max_depth 被忽略。

10. min_impurity_decrease: float, optional (default=0.)
如果节点的分裂导致不纯度的减少 (分裂后样本比分裂前更加纯净) 大于
加权不纯度的减少量计算公式为:

```
min_impurity_decrease=N_t / N * (impurity - N_t_R / N_t * righ
                    - N_t_L / N_t * left_impurity)
```

其中 N 是样本的总数, N_t 是当前节点的样本数, N_t_L 是分裂后左子节点
N_t_R 是分裂后右子节点的样本数。impurity 指当前节点的基尼指数, r
分裂后右子节点的基尼指数。left_impurity 指分裂后左子节点的基尼指

11. min_impurity_split: float
树生长过程中早停止的阈值。如果当前节点的不纯度高于阈值, 节点将
这个参数已经被弃用。用 min_impurity_decrease 代替了 min_impurity

12. class_weight: dict, list of dicts, "balanced" or None, default=None
类别权重的形式为 {class_label: weight}
(1). 如果没有给出每个类别的权重, 则每个类别的权重都为 1。
(2). 如果 class_weight='balanced', 则分类的权重与样本中每个类别出
计算公式为: n_samples / (n_classes * np.bincount(y))
(3). 如果 sample_weight 提供了样本权重 (由 fit 方法提供), 则这些权重

13. presort: bool, optional (default=False)
指定是否需要提前排序数据从而加速训练中寻找最优切分的过程。设置为
会减慢总体的训练过程; 但是对于一个小数据集或者设定了最大深度的情

属性:

1. classes_: array of shape = [n_classes] or a list of such arrays
类别的标签值。

2.feature_importances_ : array of shape = [n_features]
特征重要性。越高，特征越重要。
特征的重要性为该特征导致的评价准则的（标准化的）总减少量。它也

3.max_features_ : int
max_features的推断值。

4.n_classes_ : int or list
类别的数量

5.n_features_ : int
执行fit后，特征的数量

6.n_outputs_ : int
执行fit后，输出的数量

7.tree_ : Tree object
树对象，即底层的决策树。

方法：

1.fit(X,y)：训练模型。

2.predict(X)：预测

3.predict_log_poba(X)：预测X为各个类别的概率对数值。

4.predict_proba(X)：预测X为各个类别的概率值。

'''