决策树

课前准备

- 下载Anaconda软件，请点击这里进行下载。

本节要点

- 决策树算法原理。
- 不纯度度量标准。
- 三种常用的决策树算法。

决策树训练与预测

决策树概念

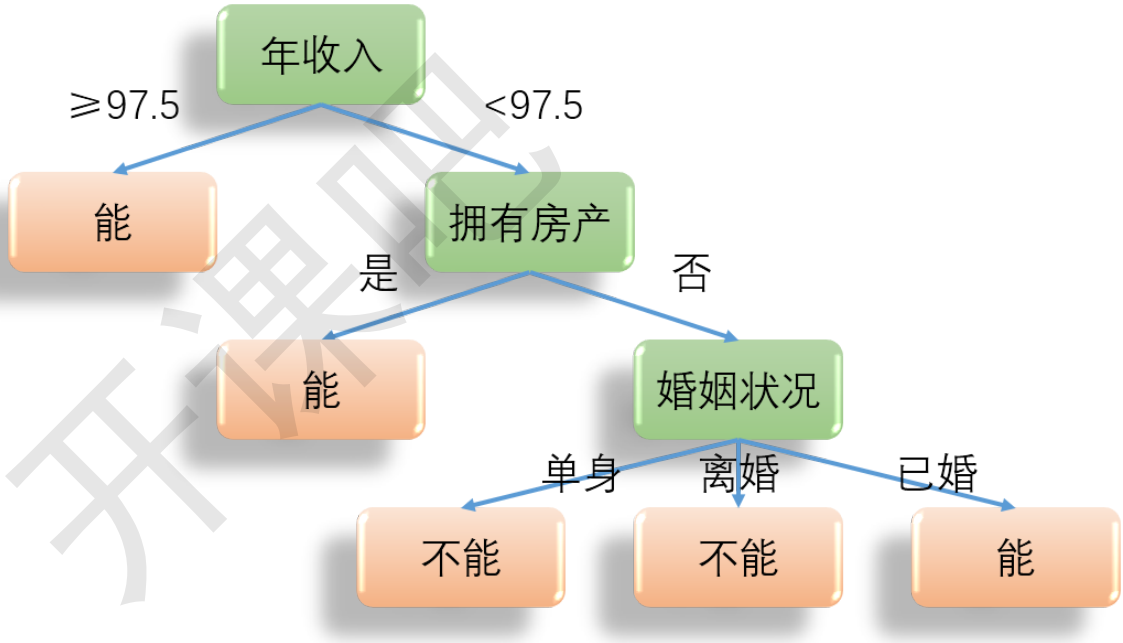
决策树是一种树形结构，通过特征的不同来将样本数据划分到不同的分支（子树）中，最终，每个样本一定会划分到一个叶子节点中。我们可以将每个特征视为一个问题（提问），特征值的不同，就视为样本给出的不同答案，然后，我们就可以根据一系列问题（特征），将样本划分到不同的叶子节点中。决策树可以用于分类与回归任务。

训练决策树

例如，给定如下的数据集：

序号	拥有房产（X_1）	婚姻状态（X_2）	年收入（X_3）	能否偿还债务（Y）
1	是	单身	125	能
2	否	已婚	100	能
3	否	单身	100	能
4	是	已婚	110	能
5	是	离婚	60	能
6	否	离婚	95	不能
7	否	单身	85	不能
8	否	已婚	75	能
9	否	单身	90	不能
10	是	离婚	220	能
11	否	已婚	94	?

我们就可以将三个特征作为三个问题，依次来“询问”数据集中的每个样本，经过每个样本依次“作答”之后，就可以将样本划分到不同的分支中，这样，决策树就训练完成。其实，决策树的训练，就是根据训练集去构建一颗决策树。结果如下图所示。



预测原理

当在训练集上构建决策树后，我们就可以对未知样本进行预测。预测的过程为：根据未知样本的特征，逐步进行分支选择（回答问题），直到叶子节点为止。那么，我们就可以使用该叶子节点中的已知样本来预测该未知样本。可是，我们预测的依据是什么呢?

我们可以想象，假设对A与B两个人进行性格测试，两个人回答一些相同的选择题，如果两个人的选项完全一致，则说明两个人的性格存在很大的相似性。同样，决策树是根据特征值来划分样本的（这类似于回答问题）。如果样本经过层层划分之后，分到了同一个叶子节点中，则表明这些样本应该也是非常相似的。因此，我们就可以使用在同一个叶子节点中的已知样本，去预测未知样本的标签了。

预测的方式为：

- 对于分类树，使用叶子节点中，数量最多的类别，作为未知样本的类别。
- 对于回归树，使用叶子节点中，所有样本标签（y）均值，作为未知样本的输出值（$\hat{y}$）。

例如，样本11预测的类别为：能。

决策树特征选择

训练的疑问

从刚才的介绍可知，决策树可以看做由若干个节点构成，其中，每个节点包含一定数量的样本（根节点包含所有样本数据）。决策树的训练过程就是根据特征值来分割样本数据。

然而，从刚才训练决策树的方式上，我们需要解决如下的问题：

- 当选择特征划分样本时，顺序是否是任意的？
    - 我们是否可以先根据“拥有房产”或“婚姻情况”特征进行划分？
- 对于连续变量类型的特征，年收入为什么选择以97.5来进行划分，而不是其他值？

直观解释

从最简单直观的角度讲，假设以分类任务为例，可以这样理解：

- 哪个特征能够将样本类别划分的效果更好，就选择哪个特征。
- 哪个取值能够将样本类别划分的效果更好，就选择哪个取值。

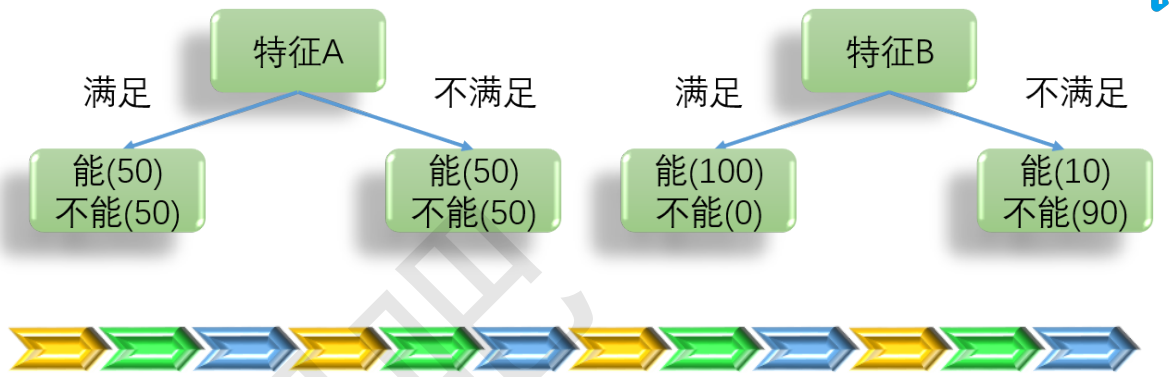
以之前能否偿还债务的场景为例，假设有200个样本，使用特征A与使用特征B划分的方式如下，我们应该优先选择哪个特征？

A 特征A

B 特征B

C 都可以。

D 无法判断。



然而，只通过主观上去识别肯定是不准确的，为了能够实现更精准的特征选择，我们需要取采取一定的量化方式去计算。因此，我们引出信息熵与信息增益的概念。

信息熵

概念

信息熵，在1948年由香农提出。用来描述系统信息量的不确定度。不确定性越大，则信息熵越大，反之，信息熵越小。

例如，4只猎豹参与赛跑，每只猎豹的能力都是旗鼓相当，平分秋色。我们很难确定哪只猎豹会获得胜利，因此，这种情况下，不确定性很大，信息熵就大。但是，假设让1只猎豹与3只蜗牛进行赛跑，则猎豹取胜便是毋容置疑的，因此，这种情况下，不确定性很小，信息熵就小。

计算方式

假设随机变量X具有m个值，分别为: V_1，V_2，...，V_m。并且各个值出现的概率如下:

$$\begin{cases} P(X=V_1)=p_1 \\ P(X=V_2)=p_2 \\ P(X=V_3)=p_3 \\ \cdots \\ P(X=V_m)=p_m \end{cases}$$

并且:

$$p_1+p_2+\cdots+p_m=1$$

则变量X的信息期望值（信息熵）为:

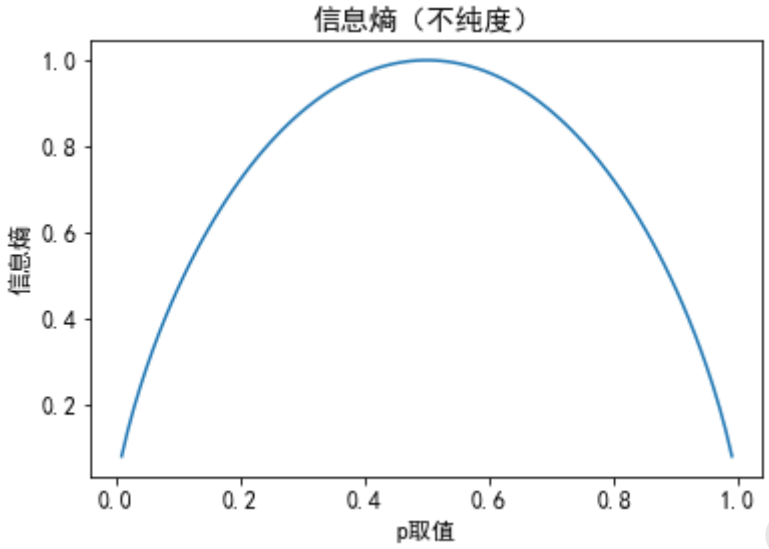
$$H(X)=-p_1*log_2p_1-p_2*log_2p_2-\cdots-p_m*log_2p_m$$
$$=-\sum_{i=1}^{m}p_ilog_2p_i$$

不纯度

从数据集的角度来讲，信息熵是样本**不纯度**的度量。

- 样本中类别比例越均衡，则不纯度越大，信息熵越大。
- 样本中类别比例越失衡，则不纯度越小，信息熵越小。

```
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams["font.family"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False
plt.rcParams["font.size"] = 12

# 设数据集X中含有两个类别，一个类别比例为p，则另外一个类别的比例为1 - p。
# 定义p的值，不断调整两个类别的比例。
p = np.linspace(0.01, 0.99, 100)
# 计算在不同比例下的信息熵。
h = -p * np.log2(p) - (1 - p) * np.log2(1 - p)
plt.plot(p, h)
plt.xlabel("p取值")
plt.ylabel("信息熵")
plt.title("信息熵（不纯度）")
plt.show()
```



课堂练习

不通过计算，以下哪个数据集的信息熵最大?

A 箱子中有9个红球，1个白球。

B 班级中有30个男生，25个女生。

C 饮料店成交30单，其中10单苹果味，10单哈密瓜味，10单葡萄味。

D 路口经过55辆车，其中50辆本地车，5辆外地车。

信息增益

信息增益（IG-Information gain）定义如下：

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{n} \frac{N_j}{N_p} I(D_j)$$

- f：划分的特征。
- D_p：父节点，即使用特征f分割之前的节点。
- $IG(D_p, f)$：父节点D_p使用特征f划分下，获得的信息增益。
- D_j：父节点D_p经过分割之后，会产生n个子节点，D_j为第j个子节点。
- N_p：父节点D_p包含样本的数量。
- N_j：第j个子节点D_j包含样本的数量。
- I：不纯度度量标准。例如，之前介绍的信息熵，就是标准之一。

出于简化与缩小组合搜索空间的考虑，很多库（包括scikit-learn）实现的都是二叉决策树，即每个父节点最多含有两个子节点（左子树节点与右子树节点），此时，信息增益定义为：

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

通过定义我们可知，信息增益就是父节点的不纯度减去所有子节点不纯度（加权）。

假设你是总经理助理，目前有10份文件，其中4个与M公司有关，6个与N公司有关。我们需要对文件进行整理，目前X与Y两个盒子，我们会选择（ ）。
A 4个与M公司相关的文件放入X，6个与N公司相关的文件放入Y。
B 3个与M公司相关的文件放入X，另外1个与6个与N公司相关的文件放入Y。
C 1个与M公司相关的文件放入X，另外3个与6个与N公司相关的文件放入Y。
D 所有文件放入X或Y中，另外一个盒子空闲。

就像我们整理文件一样，在选择特征分裂样本时，我们应该让子节点的不纯度尽可能的低，这样就可以更快的完成训练（更少的分割次数），同时，在预测未知样本时，也会具有更高的准确度。

因此，在我们选择特征进行划分时，特征的顺序不是任意的，而是应该选择在分割样本集后，能够使得所有子节点不纯度最低（加权）的特征。由于父节点的不纯度是不变的，因此，能够让所有子节点不纯度最小的特征，实际上也就是能够所得信息增益最大的特征。而这，也正是训练分类决策树时，选择特征顺序的依据。

训练规则

训练分类决策树的具体规则如下：

1. 将每一个特征看成是一种分裂可能。特征可以分为离散型与连续性。
    - 对于离散型特征，每一个类别可以划分为一个子节点（多叉树），或者属于类别A与不属于类别A（二叉树）。
    - 对于连续型特征，可以划分为大于等于A与小于A。

2. 从根节点开始，选择可获得最大信息增益的特征进行分裂（实现信息增益最大化）。
3. 对子节点继续选择能够获得最大信息增益的特征进行分裂，直到满足如下条件之一，停止分裂。
    - 所有叶子节点中的样本属于同一个类别。
    - 树达到指定的最大深度（max_depth），每次分裂视为一层。
    - 节点包含的样本数量小于指定的最小分裂样本数量（min_samples_split）。
    - 如果节点分裂后，叶子节点包含的样本数量小于指定的叶子最小样本数量（min_samples_leaf）。

直觉上，我们应该只将条件1作为训练结束的依据。然而，将条件2 ~ 4作为训练的结束条件，目的是什么呢?
A 在训练集较大时，可以具有更快的训练速度。
B 为了防止欠拟合。
C 为了防止过拟合。
D 以上都不是。

分类决策树示例

以不纯度衡量使用信息熵为例，父节点的信息熵为：

$$I_H(D_p) = -0.7 * log_2 0.7 - 0.3 * log_2 0.3 = 0.88$$

如果以特征“拥有房产”作为分裂特征，则：

$$I_H(D_{有房产}) = -4/4 * log_2 4/4 - 0 * log_2 0 = 0$$
$$I_H(D_{无房产}) = -0.5 * log_2 0.5 - 0.5 * log_2 0.5 = 1$$

因此，特征“拥有房产”的信息增益为：

$$IG_H(拥有房产) = 0.88 - 0.4 * 0 - 0.6 * 1 = 0.28$$

同理：

$$IG_H(婚姻) = 0.205$$

对于收入来说，为连续型变量，我们这里将收入的取值进行排序，然后选择“否”与“是”的分界点（75与85之间，95与100之间），取平均值进行分割：

$$I_H(D_{收入<80}) = -1 * log_2 1 - 0 * log_2 0 = 0$$
$$I_H(D_{收入>=80}) = -3/8 * log_2 3/8 - 5/8 * log_2 5/8 = 0.954$$
$$IG_H(收入 = 80) = 0.88 - 0.2 * 0 - 0.8 * 0.954 = 0.117$$

同样的方式：

$$I_H(D_{收入<97.5}) = -0.4 * log_2 0.4 - 0.6 * log_2 0.6 = 0.97$$
$$I_H(D_{收入>=97.5}) = -1 * log_2 1 - 0 * log_2 0 = 0$$
$$IG_H(收入 = 97.5) = 0.88 - 0.5 * 0.97 - 0.5 * 0 = 0.395$$

从上面的结果中可知，相比于收入=80来说，收入=97.5可以获得更大的信息增益。

不纯度度量标准

不纯度可以采用如下方式度量：

- 信息熵（Entropy）
- 基尼系数（Gini Index）
- 错误率（classification error）

信息熵

$$I_H(D) = -\sum_{i=1}^{m} p(i \mid D) log_2 p(i \mid D)$$

- m： 节点D中含有样本的类别数量。
- $p(i \mid D)$： 节点D中，属于类别i的样本占节点D中样本总数的比例（概率）。

基尼系数

$$I_G(D) = 1 - \sum_{i=1}^{m} p(i \mid D)^2$$

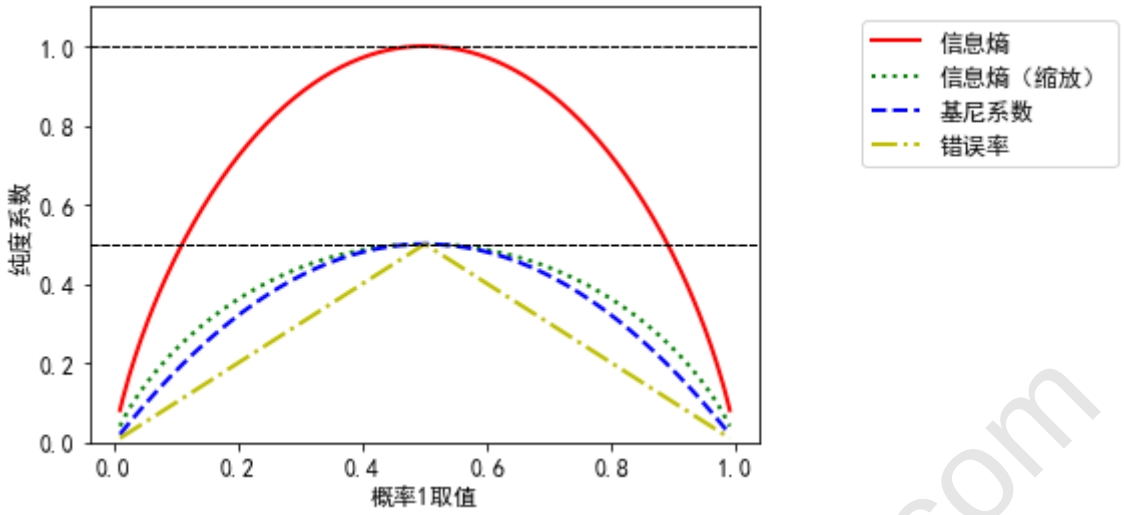
错误率

$$I_E(D) = 1 - max\{p(i \mid D)\}$$

无论哪种度量标准，都有一个特性：如果样本以相同的比例分布于不同的类别时，度量值最大，不纯度最高。如果所有的样本都属于同一个类别，则度量值为0，不纯度最低。

```
def gini(p):
    """计算指定概率组合下的基尼系数。

    Parameters
    ----------
    p : array-like
        概率数组。

    Returns
    -------
    v : float
        基尼系数值。

    """
    return 1 - np.sum(p ** 2, axis=1)


def entropy(p):
    """计算指定概率组合下的信息熵。

    Parameters
    ----------
    p : array-like
        概率数组。

    Returns
    -------
    v : float
        信息熵值。

    """
    return -np.sum(p * np.log2(p), axis=1)


def error(p):
    """计算指定概率组合下的错误率。

    Parameters
    ----------
    p : array-like
        概率数组。

    Returns
    -------
    v : float
        错误率值。

    """
    return 1 - np.max(p, axis=1)
```

```
50
51
52  # 定义概率的取值范围。
53  p = np.linspace(0.01, 0.99, 200)
54  # 计算概率组合。
55  parray = np.array([p, 1 - p]).T
56  # 计算信息熵。
57  en = entropy(parray)
58  # 计算缩放的信息熵。
59  en2 = en * 0.5
60  # 计算错误率。
61  err = error(parray)
62  # 计算基尼系数。
63  g = gini(parray)
64  fig = plt.figure()
65  for i, lab, ls, c, in zip([en, en2, g, err], ["信息熵", "信息熵（缩放）", "基尼
    系数", "错误率"],
66          ["-", ":", "--", "-."], ["r", "g", "b", "y"]):
67      plt.plot(p, i, label=lab, linestyle=ls, lw=2, color=c)
68      plt.legend(loc="right", bbox_to_anchor=(1.55, 0.8))
69      plt.axhline(y=0.5, linewidth=1, color='k', linestyle="--")
70      plt.axhline(y=1.0, linewidth=1, color='k', linestyle="--")
71      plt.ylim([0, 1.1])
72      plt.xlabel("概率1取值")
73      plt.ylabel("纯度系数")
74  plt.show()
```



决策树算法

决策树主要包含以下三种算法：

- ID3
- C4.5
- CART（Classification And Regression Tree）

ID3

ID3（Iterative Dichotomiser3-迭代二分法）算法是非常经典的决策树算法，该算法描述如下：

- 使用多叉树结构。
- 使用信息熵作为不纯度度量标准，选择信息增益最大的特征分割数据。

ID3算法简单，训练较快。但该算法具有一些局限，如下：

- 不支持连续特征。
- 不支持缺失值。
- 仅支持分类，不支持回归。
- 在选择特征时，会倾向于选择类别多的特征。

C4.5

C4.5算法是在ID3算法上改进而来，该算法描述如下：

- 使用多叉树结构。
- 仅支持分类，不支持回归。

不过，C4.5在ID3算法上，进行了一些优化，包括：

- 支持对缺失值的处理。
- 支持将连续值进行离散化处理。
- 使用信息熵作为不纯度度量标准，但选择信息增益率（而不是信息增益）最大的特征分裂节点。

信息增益率的定义方式为：

$$IG_{Ratio}(D_p, f) = \frac{IG_H(D_p, f)}{I_H(f)}$$

- $I_H(f)$：根据特征f的不同类别值比例（概率），计算得到的信息熵。

之所以从信息增益改为信息增益率，是因为在ID3算法中，倾向于选择类别多的特征，因此，经过这样的调整，在C4.5中就可以得到缓解。因为类别多的特征在计算信息熵$I_H(f)$时，往往会比类别少的特征信息熵大。这样，就可以在分母上进行一定的惩罚。

```
# 计算在不同类别数量时，信息熵的对比。每个元素代表一个类别所占的比例。
a1 = np.array([[0.4, 0.6]])
a2 = np.array([[0.3, 0.3, 0.2, 0.2]])
a3 = np.array([[0.1] * 10])
print(entropy(a1))
print(entropy(a2))
print(entropy(a3))
```

```
[0.97095059]
[1.97095059]
[3.32192809]
```

CART

CART（Classification And Regression Tree），分类与回归树。该算法描述如下：

- 使用二叉树结构。
- 支持连续值与缺失值处理。
- 既支持分类，也支持回归。
    - 使用基尼系数作为不纯度度量标准，选择基尼增益最大的特征分裂节点。（分类）
    - 使用MSE或MAE最小的特征分类节点。（回归）

回归决策树

当使用回归决策树时，与分类决策树会有所不同。回归任务的标签（y值）是连续的，故之前以分类为基础的不纯度度量标准（信息熵，基尼系数与错误率）都不适用于回归树，因此，在回归树中，自然也就没有信息增益，信息增益率或基尼增益等概念了。可以说，分类决策树选择特征的方式，完全不适用于回归决策树。

对于回归决策树，会使用叶子节点的均值来预测未知样本。同时，回归决策树使用MSE或MAE作为评估指标，用来选择特征。也就是说，回归决策树在选择特征上，每次选择能够使得MSE或MAE最小的特征，用来分裂节点。

程序实现

在scikit-learn中，使用优化的CART算法来实现决策树。

分类

scikit-learn中，提供DecisionTreeClassifier类，用来实现决策树分类。

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X, y = load_iris(return_X_y=True)
# 为了后续的可视化方便，这里选择两个特征。
X = X[:, :2]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)
# criterion：不纯度度量标准，默认为gini。
#    gini：基尼系数  entropy：信息熵
# splitter：选择分裂节点的方式。默认为best。
#    best：在最好的位置分裂节点。  random：在随机的位置分裂节点。
# max_depth：树的最大深度，默认为None（不限制深度）。
# min_samples_split：分裂节点的最小样本数，默认为2。
# min_samples_leaf：分裂节点后，叶子节点最少的样本数量，默认为1。
# max_features：分裂节点时，考虑的最大特征数量，默认为None（考虑所有特征）。
# random_state：随机种子。
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
print(tree.score(X_train, y_train))
print(tree.score(X_test, y_test))
```

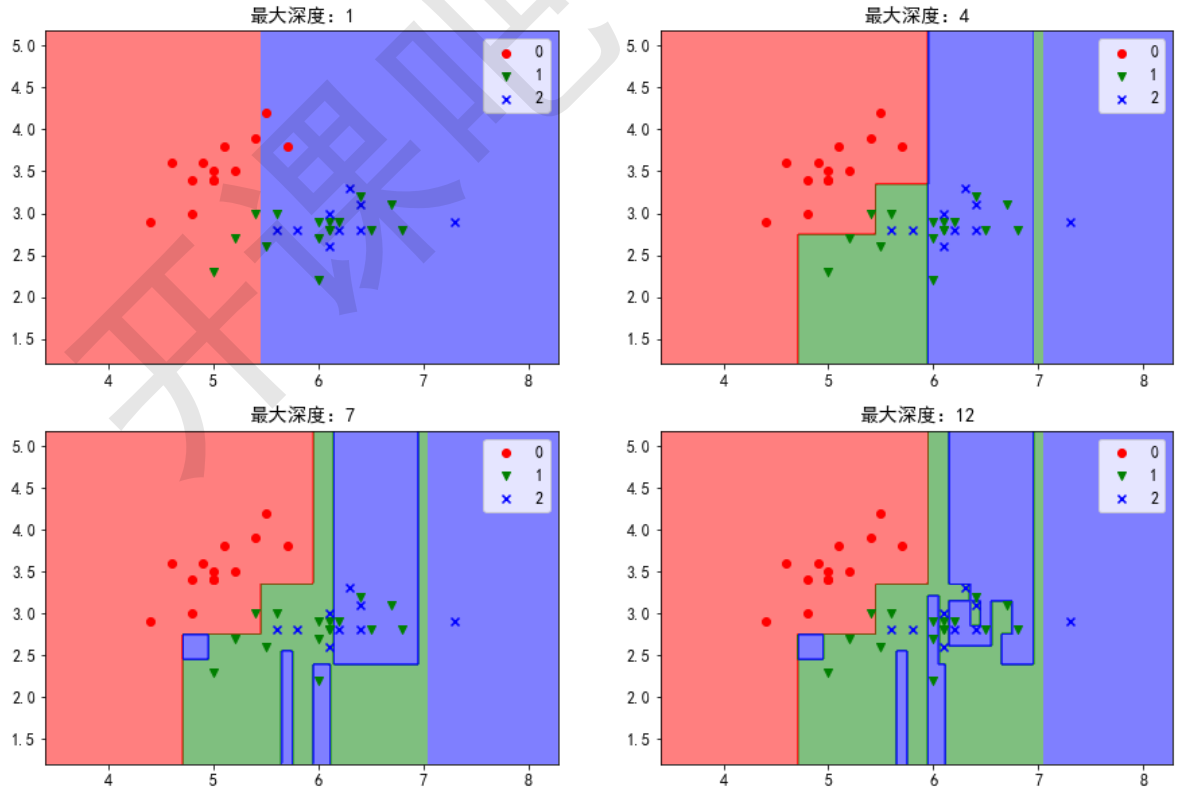
```
0.9375
0.6578947368421053
```

我们发现，模型存在严重的过拟合倾向，原因在于，如果没有指定树的深度，则默认会训练一颗完全生长的决策树（不限深度），这会容易导致模型复杂化，从而过分依赖于训练集数据的特性，造成过拟合。

我们可以从不同深度树的决策边界，来证实这一点。

```
from matplotlib.colors import ListedColormap

def plot_decision_boundary(model, X, y):
    color = ["r", "g", "b"]
    marker = ["o", "v", "x"]
    class_label = np.unique(y)
    cmap = ListedColormap(color[: len(class_label)])
    x1_min, x2_min = np.min(X, axis=0)
    x1_max, x2_max = np.max(X, axis=0)
    x1 = np.arange(x1_min - 1, x1_max + 1, 0.02)
    x2 = np.arange(x2_min - 1, x2_max + 1, 0.02)
    X1, X2 = np.meshgrid(x1, x2)
    Z = model.predict(np.c_[X1.ravel(), X2.ravel()])
    Z = Z.reshape(X1.shape)
    plt.contourf(X1, X2, Z, cmap=cmap, alpha=0.5)
    for i, class_ in enumerate(class_label):
        plt.scatter(x=X[y == class_, 0], y=X[y == class_, 1],
                c=cmap.colors[i], label=class_, marker=marker[i])
    plt.legend()


plt.figure(figsize=(15, 10))
```

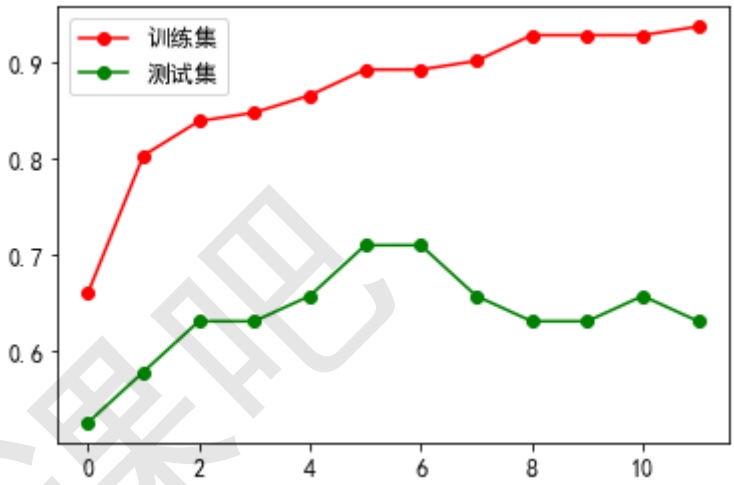
```
for index, depth in enumerate([1, 4, 7, 12], start=1):
    plt.subplot(2, 2, index)
    plt.title(f"最大深度：{depth}")
    tree = DecisionTreeClassifier(random_state=0, max_depth=depth)
    tree.fit(X_train, y_train)
    plot_decision_boundary(tree, X_test, y_test)
```



对于决策树来说，最大深度对模型有着较重要的影响，如果最大深度很小，意味着仅进行少数的切分，容易欠拟合，但是，如果最大深度很大，则意味着可能进行较多次切分，容易过拟合。

```
# 定义列表，用来存储在不同深度下，模型的分值。
train_score = []
test_score = []
for depth in range(1, 13):
    tree = DecisionTreeClassifier(random_state=0, max_depth=depth)
    tree.fit(X_train, y_train)
    train_score.append(tree.score(X_train, y_train))
    test_score.append(tree.score(X_test, y_test))

plt.plot(train_score, marker="o", c="red", label="训练集")
plt.plot(test_score, marker="o", c="green", label="测试集")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x208f1992dc8>
```



从运行结果中，我们可知，随着最大深度的增加，训练集的表现越来越好，但是测试集的表现，是先增加后减少，这说明，在树深度较小时，模型是欠拟合的，因此，增加树深度，能够提升预测效果，但随着深度的增加，模型越来越依赖于训练集，这反而降低预测效果，造成过拟合。

回归

scikit-learn中，提供DecisionTreeRegressor类，用来实现决策树回归。

```
from sklearn.datasets import load_boston
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split

X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

# 回归决策树的参数，可以参考分类决策树的参数。
tree = DecisionTreeRegressor(max_depth=3)
tree.fit(X_train, y_train)
print(tree.score(X_train, y_train))
print(tree.score(X_test, y_test))
```

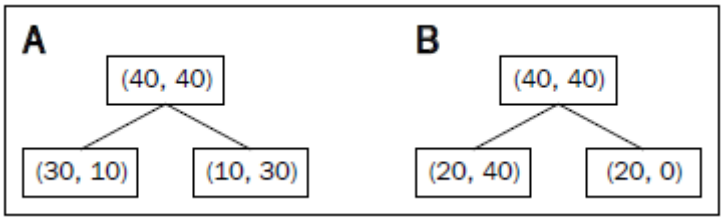
```
0.8290972700366354
0.6354364289453208
```

我们可以对决策树实现可视化，通过可视化，就能够知道决策树内部是如何依次选择特征，并且又是如何选择值来进行划分的。关于决策树的可视化，老梁提供辅助视频，供大家学习。

拓展点

- 决策树的可视化。

作业



1. 三种不纯度度量标准，是否在实际应用中，效果是差不多的？为什么在sklearn库中，没有错误率的衡量方式？
2. 在上图中，分别计算节点在A与B两种不同分裂方式下的信息熵，基尼系数与错误率，比较结果是否存在差异。
3. 决策树是否需要对数据进行标准化处理，为什么？