同濟大學
TONGJI UNIVERSITY

王睿智 ruizhiwang@tongji.edu.cn

人工智能技术与应用

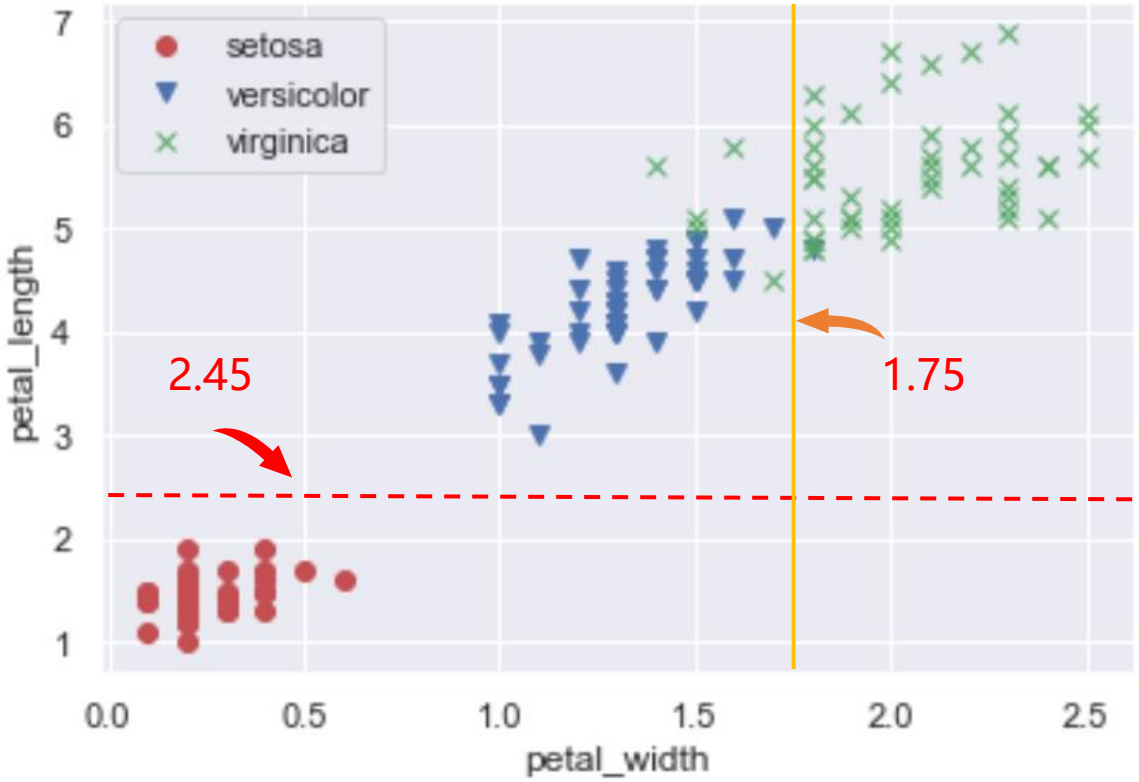
第三章 监督学习

思考：可直接根据特征对数据预测呢?

3.5 决策树 | 引例 决策树预测鸢尾花类别

取iris中petal width 、petal length两特征，画出散点图。有什么发现？

iris-两个属性



（1）观测所有样本在**petal length**上的取值，

setosa的petal length： 1.0~1.9；

非setosa的petal length： 3.0~6.9。

在1.9和3.0之间设个阈值，如(1.9+3.0)/2=2.45，就能将setosa和非setosa样本正确分开。

（2）再观察非setosa样本在**petal width**上取值，

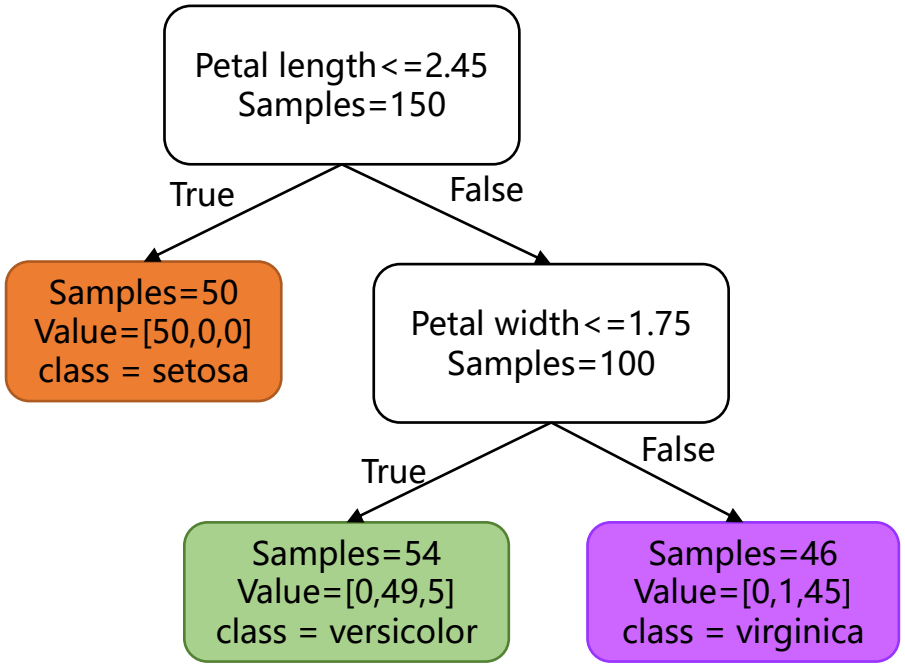
versicolor的petal width： 1.0~1.8，有1个>1.7；

virginica的petal width： 1.4~2.5，有5个<1.8。

在1.7和1.8之间设个阈值，如 1.75，也能将versicolor和virginica基本分开。（错分到versicolor中5个样本，错分到virginica中1个样本）

3.5 决策树 | 引例 决策树预测鸢尾花类别

将上述决策过程用一棵树表达，就得到如下所示的结果。



预测一朵新鸢尾花的类别：

从根节点开始，先看其“petal_length≤2.45?”。如果是，则进入根的左子节点。该节点是叶节点，它不再提问，查看该节点给出的预测类别，为setosa。

如果花瓣长大于2.45cm，则进入根的右子节点，该节点不是叶节点，它提问“petal_width ≤1.75?”，如果“是”则预测为versicolor，否则预测为virginica。

3.5.1 决策树

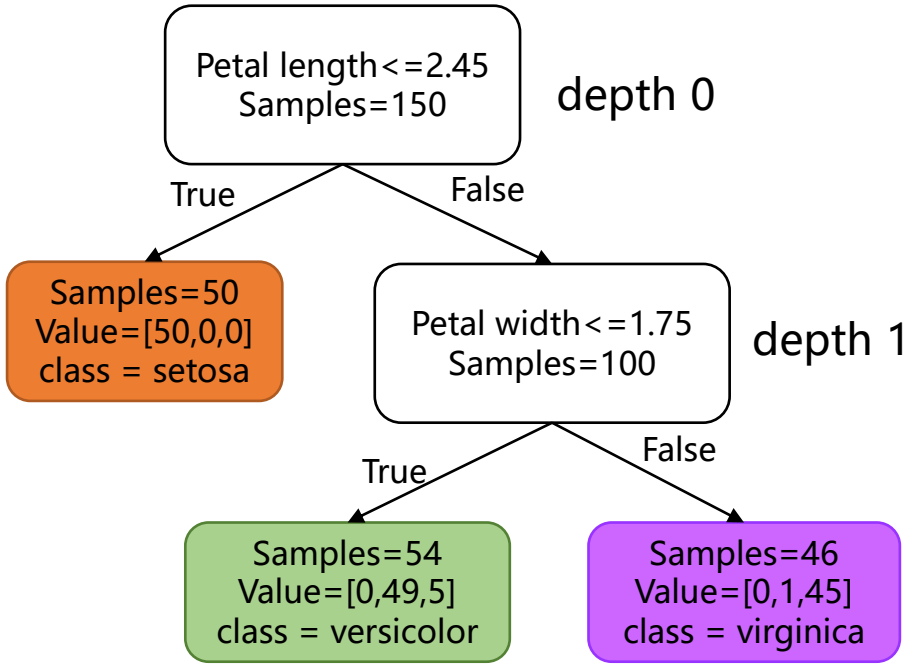
一棵决策树(Decision Tree)由一系列节点和有向边组成。

- 根节点包含训练样本全集。
- 内部节点（包括根节点）：有一个关于已知特征的提问。每个内部节点所包含的样本集合根据特征检测结果被划分到子节点中。
- 叶节点：对应于决策结果。
- 从根节点到每个叶节点的路径，对应一个判定规则。

3.5.1 决策树 | 决策树训练算法

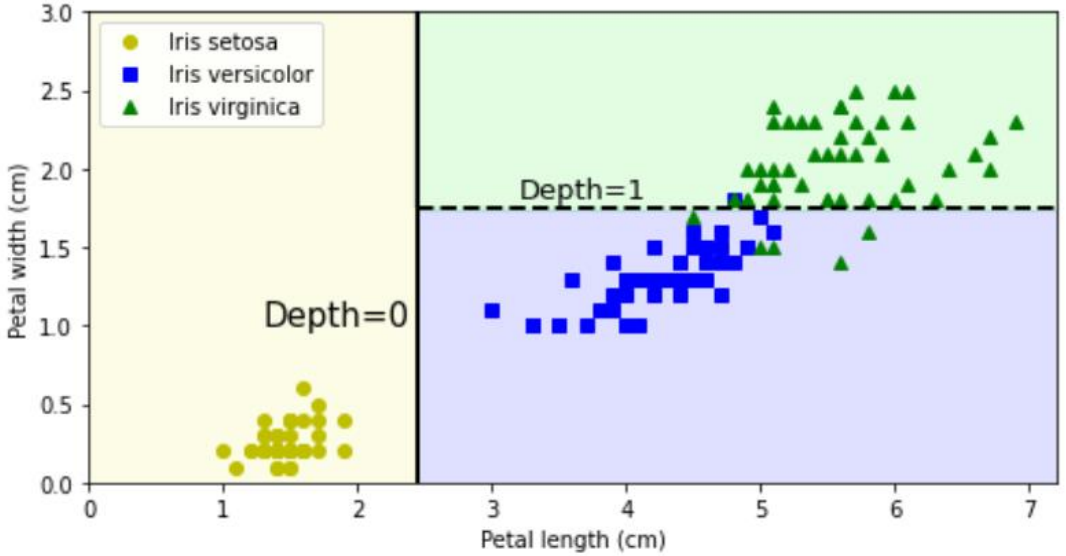
- 决策树是基于特征进行预测的方法，既可用于分类又可用于回归任务。
  - 用决策树分类：从根节点开始，将新样本与遇到的每个内部节点的提问特征相比较，根据结果进入下一节点，最后到达预测的叶节点，预测结果为该叶节点的类别。
- 决策树训练(也称决策树学习)的目的是生成一棵泛化能力强的决策树。
- 决策树训练算法：
  - CART算法（Classification and Regression Tree）
  - ID3算法、C4.5算法、C5.0算法

理解决策树工作原理



提出一串问题

对应关系



划分输入空间

决策树训练算法是一种不断地选出特征，根据特征取值将输入空间分成不同区域，而每个区域尽可能**纯**（样本属于同一类）的算法。

3.5.2 CART训练算法

CART训练算法是最常用的决策树训练算法，也是Scikit-learn中使用的算法。它生成的是一棵二叉树。

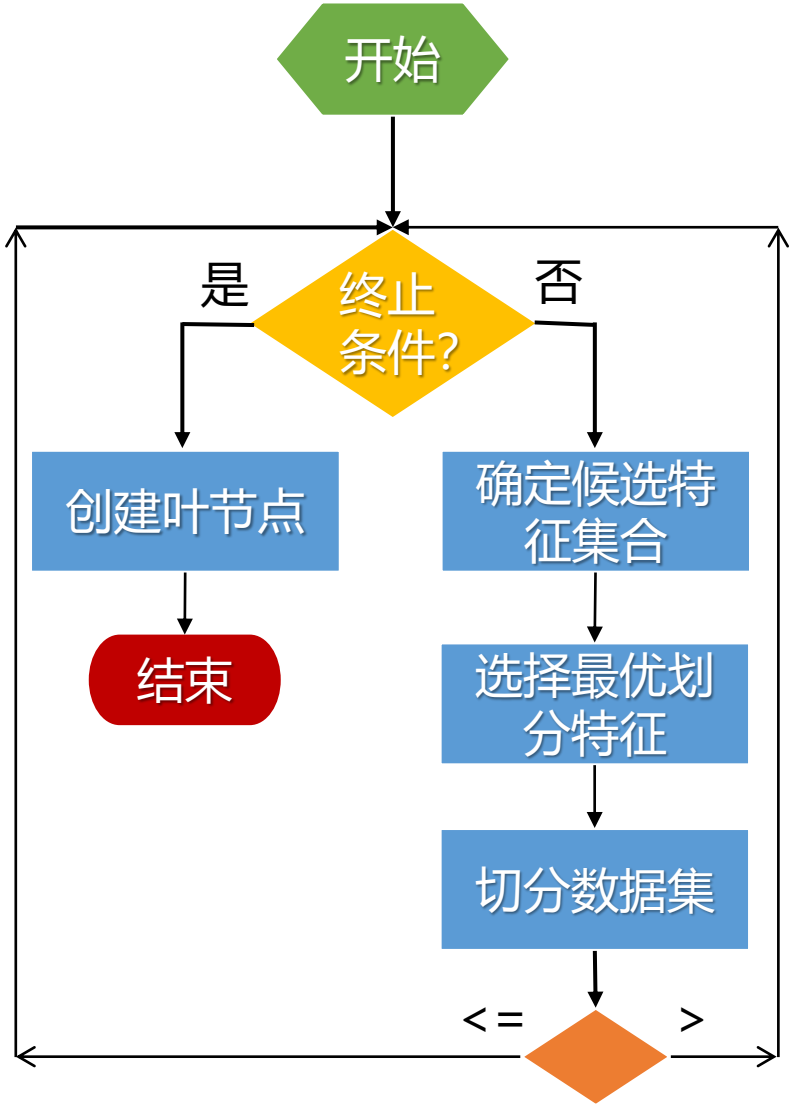
基本思想：

该算法首先选用一个**特征k**和**阈值t_k**（如,petal length≤2.45cm）将训练集**划分**成两个子集，**使得划分得到的两个子集尽可能纯**。然后使用上述逻辑对子集进行划分，递归地进行，直至到达终止条件。

关键： ① 怎样选择划分训练集的特征 k 和阈值 t_k 呢？

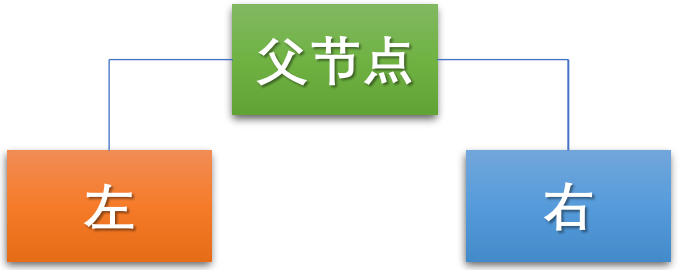
② 终止条件怎么设定？

划分特征选择



3.5.2 CART训练算法 | 划分特征选择

CART分类算法的划分目标：



最小化 $J(k,t_k)=\frac{m_{\text{left}}}{m}G_{\text{left}}+\frac{m_{\text{right}}}{m}G_{\text{right}}$

其中，k是特征和t_k是其阈值，

$G_{\text{left/right}}$ 测量左/右子集的不纯度

$m_{\text{left/right}}$ 左/右子集中样本数目

m 是待划分的集合中样本数目

CART分类算法通过**最小化加权平均不纯度**来划分数据集，寻找能产生最纯子集的一个特征k和阈值t_k对(k,t_k)。

怎样衡量一个样本集的不纯度?

3.5.2 CART训练算法 | 不纯度度量

度量数据集纯与否的常用指标有两个：

- **Gini不纯度**（Gini impurity）：$G_i = 1 - \sum_{k=1}^{K} {P_{i,k}}^2$

  当一个集合所有实例均属同一类别时，它是纯的，它的基尼值为零。

  $P_{i,k}$是第i个节点中属于类别k的样本占该节点中样本总数的比重。

- **熵**（entropy）：$H_i = -\sum_{k=1}^{K} P_{i,k} \log_2\left(P_{i,k}\right)$，其中$P_{i,k} \neq 0$

  当一个集合只包含一个类的实例时，它的熵为零。

例3.16　购车数据集的划分特征选择

id	x_0	x_1	y
0	7000	24	0
1	8000	24	0
2	10000	24	1
3	13000	30	1

一组顾客购车数据，有两个特征：x_0 (收入)，x_1 (年龄)。根据是否买车，将顾客分两类：是、否，分别对应 $y=1$ 和 $y=0$。

① 计算根节点的gini值和熵。

② 利用gini不纯度，选出根节点的划分特征对。

③ 利用entropy，选出根节点的划分特征对。

例3.16　购车数据集的划分特征选择

id	x_0	x_1	y
0	7000	24	0
1	8000	24	0
2	10000	24	1
3	13000	30	1

① 计算根节点的gini值和熵。

解答：

决策树的根节点包含初始数据集，共4个样本，分属两类，两类样本数目列表：value=[2,2]

gini=?
samples= 4
value=[2,2]

根节点的gini值：$1-(2/4)^2-(2/4)^2=0.5$

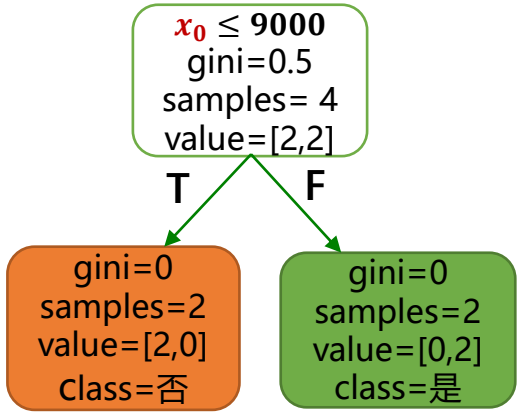
entropy=?
samples= 4
value=[2,2]

根节点的熵：$-2/4\log_2(2/4)-2/4\log_2\left(\frac{2}{4}\right)=1$

② 利用gini不纯度，选出根节点的划分（特征，阈值）对。

若以x_0为划分特征，阈值可能取值有7500、9000、11500。

id	x_0	x_1	y
0	7000	24	0
1	8000	24	0
2	10000	24	1
3	13000	30	1



- 当以$(x_0, 7500)$划分初始数据集，左节点中两类样本数[1,0]，右节点中两类样本数[1,2]。

  $G_{\text{left}} = 1-(1/1)^2 = 0$，$G_{\text{right}} = 1-(1/3)^2-(2/3)^2 = 4/9$

  得 $J(x_0, 7500) = \frac{1}{4}G_{\text{left}} + \frac{3}{4}G_{\text{right}} = \frac{1}{3}$

选择$(x_0, 9000)$划分

- 当以$(x_0, 9000)$划分初始数据集，左节点中两类样本数[2,0]，右节点中两类样本数[0,2]，$G_{\text{left}} = G_{\text{right}} = 0$，得 $J(x_0, 9000) = 0$
- 当以$(x_0, 11500)$划分初始数据集，得 $J(x_0, 11500) = \frac{1}{3}$

若以x_1为划分特征，阈值为27。

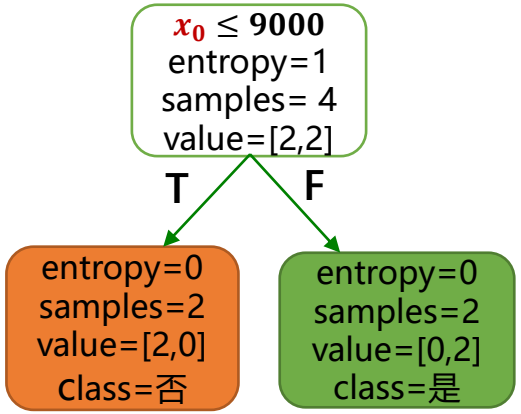
- 当以$(x_1, 27)$划分初始数据集，左节点中两类样本数[2,1]，右节点中两类样本数目[0,1]。

  得 $J(x_1, 27) = \frac{3}{4}\left(1-(2/3)^2-(1/3)^2\right) + \frac{1}{4}\left(1-(1/1)^2\right) = \frac{1}{3}$

③ 利用entropy，选出根节点的划分特征对。

若以x_0为划分特征，阈值可能取值有7500、9000、11500。

id	x_0	x_1	y
0	7000	24	0
1	8000	24	0
2	10000	24	1
3	13000	30	1



- 当以$(x_0, 7500)$划分初始数据集，左节点中两类样本数[1,0]，右节点中两类样本数[1,2]。 $H_{\text{left}} = -1 \times \log_2 1 = 0$，$H_{\text{right}} = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}$

  得 $J(x_0, 7500) = \frac{1}{4}H_{\text{left}} + \frac{3}{4}H_{\text{right}} \approx 0.6887$

- 当以$(x_0, 11500)$划分初始数据集，得$J(x_0, 11500) \approx 0.6887$

- 当以$(x_0, 9000)$划分初始数据集，左节点中两类样本数[2,0]，右节点中两类样本数[0,2]。 $H_{\text{left}} = -1 \times \log_2 1 = 0$，$H_{\text{right}} = -1 \times \log_2 1 = 0$

  得 $J(x_0, 9000) = \frac{2}{4}H_{\text{left}} + \frac{2}{4}H_{\text{right}} = 0$

选择$(x_0, 9000)$划分

若以x_1为划分特征，阈值为27。

- 当以$(x_1, 27)$划分初始数据集，左节点中两类样本数[2,1]，右节点中两类样本数[0,1]。 $H_{\text{left}} = -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}$，$H_{\text{right}} = -1 \times \log_2 1 = 0$

  得 $J(x_1, 27) = \frac{3}{4}H_{\text{left}} + \frac{1}{4}H_{\text{right}} \approx 0.6887$

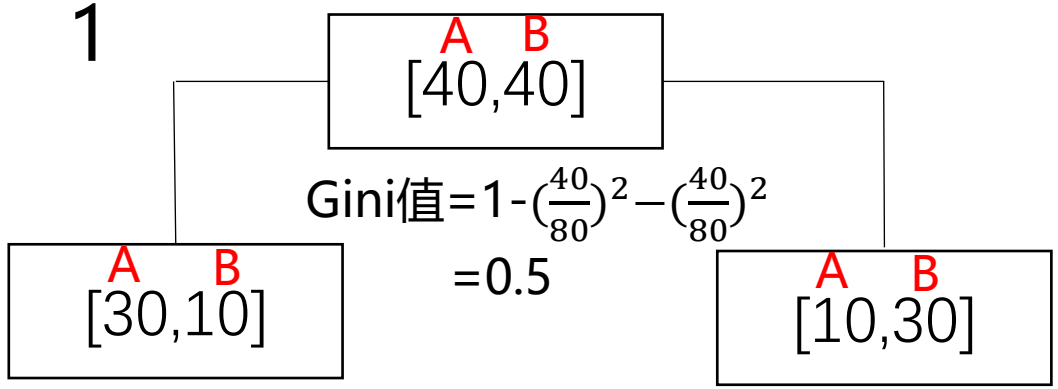
举例：CART算法的划分特征选择原理

$$G = 1 - \sum_{k=1}^{K} P_k{}^2$$

1和2两种划分哪种好？利用Gini值

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

1



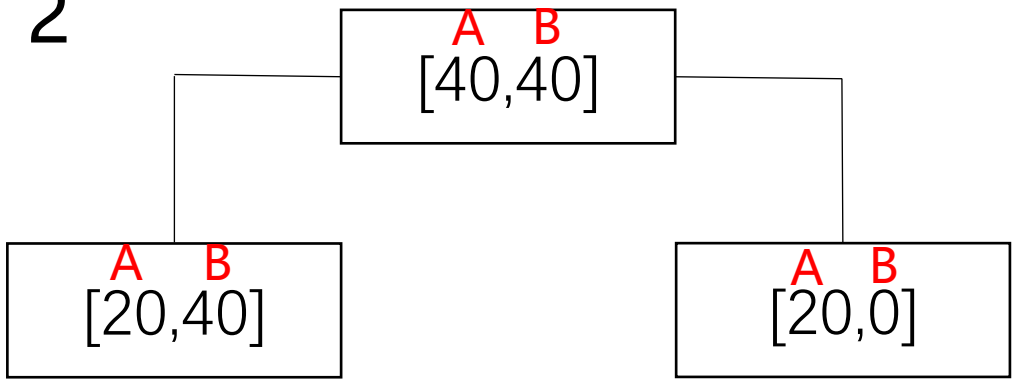
Gini值=1-$(\frac{40}{80})^2-(\frac{40}{80})^2$
=0.5

Gini值=1-$(\frac{30}{40})^2-(\frac{10}{40})^2$
=0.375

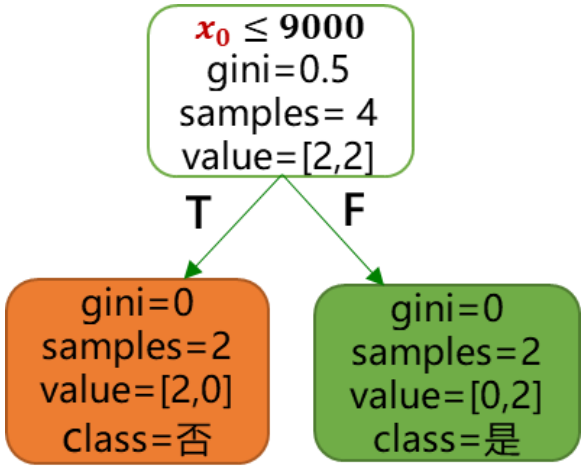
Gini值=1-$(\frac{10}{40})^2-(\frac{30}{40})^2$
=0.375

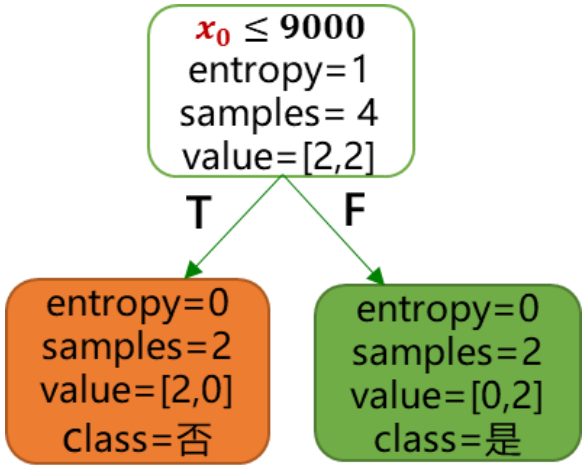
$$J_1 = \frac{40}{80} \times 0.375 + \frac{40}{80} \times 0.375 = 0.375$$

2



3.5.2 CART训练算法 | 不纯度度量





树的属性：

- Sample：显示该节点的训练样本数目。
- value：显示该节点上的各个类别训练样本数目。
- gini：节点上样本集合的Gini不纯度。
- entropy：节点上样本集合的熵。

- 基尼不纯度和熵在多数情况下差别不大，它们会生成类似的决策树。
  基尼不纯度速度稍快。
- CART算法默认使用Gini不纯度，但可替换为entropy。

特征的阈值点选取

如，一个数据集中有“温度”特征，出现过的值有[10,-15,0,-9]

$$\begin{bmatrix} \cdots & 10 & \cdots \\ \cdots & -15 & \cdots \\ \cdots & 0 & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & -9 & \cdots \\ \cdots & 10 & \cdots \\ \cdots & 0 & \cdots \end{bmatrix}$$

CART算法做如下处理：

(1) 特征值排序，　　得 [-15,-9,0,10]　（4个值）

(2) 依次取[-15,-9,0,10]中相邻两值的中点作为阈值点，得到阈值列表

　[-12,-4.5,5]　（3个值）

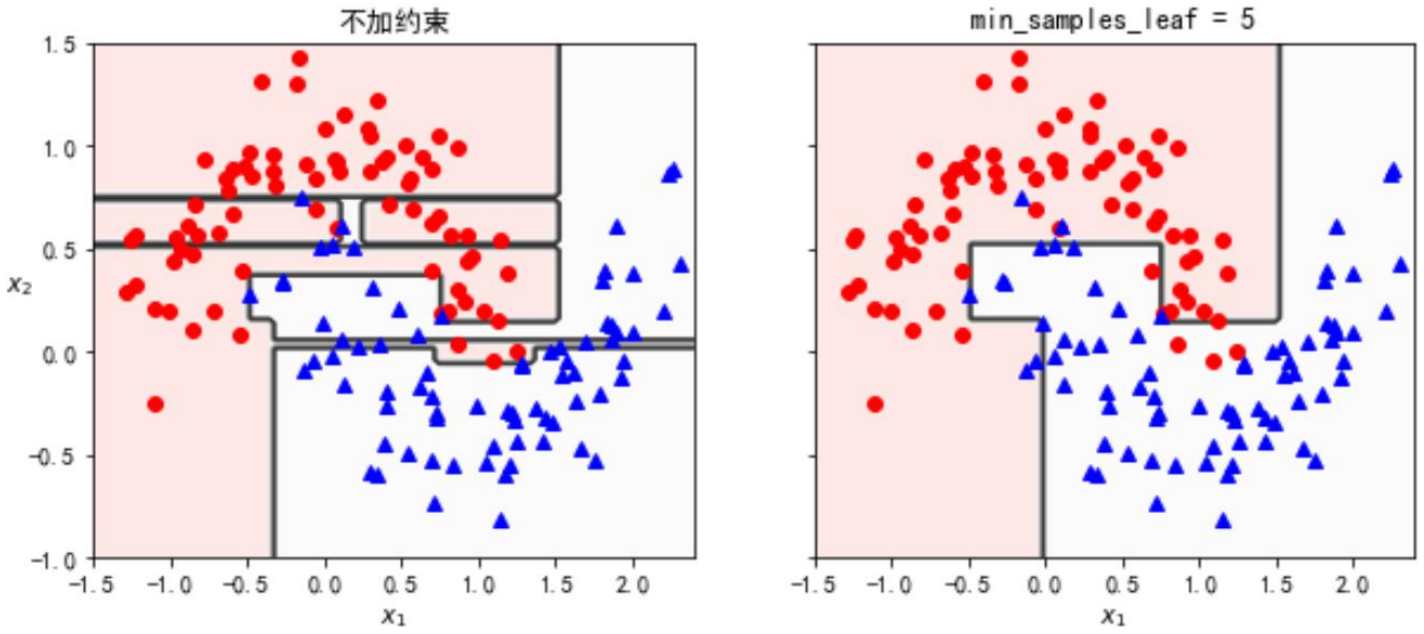
3.5.2 CART训练算法 | 终止条件

- CART分类算法采用**贪心策略**，递归地选出特征k和阈值t_k 将当前节点的样本集划分成两个子集，直到达到预设的最大深度，或者找不到一个划分能使不纯度降低。
- **终止条件**：
  - 最大深度（max_depth）：达到该深度就停止划分；若未设定最大深度，则扩展节点，直到所有叶子都是纯的或者所有叶子含有的样本数目少于 min_samples_split。
  - 找不到一个划分能使不纯度降低
- 还有一些附加终止条件：如scikit-learn的DecisionTreeClassifier类中超参数
  - min_samples_split：设定拆分内部节点所需的最小样本数目。
  - min_samples_leaf：叶节点所需的最小样本数目。只有在左右分支中的每个分支中至少留下min_samples_leaf个训练样本时，才会考虑分割。
  - …

3.5.2 CART训练算法 | 过拟合问题

决策树极少对训练数据做假设，树结构将随训练集变化，**易过拟合**。这种模型常被称为**非参数模型**，是指在训练之前没有确定参数的数量，导致模型结构自由而紧密地贴合数据。

如，为two-moons数据训练两棵CART分类器：一棵不限制树的生长，另一棵要求min_samples_leaf=5。两个树分类器的决策面如下图所示。



左图决策面比有图赋复杂得多，因此易过拟合！

3.5.2 CART训练算法 | 过拟合问题

- **决策树避免过拟合的常用策略：**预剪枝（正则化）、后剪枝（常简称剪枝）
  - 预剪枝：设置正则化超参数，在训练过程中降低模型的自由度。
  - 后剪枝：先不加约束地训练模型，然后再对不必要的节点进行剪枝(删除)。

    先从训练集生成一棵完整的决策树，然后自底向上地对非叶节点进行考察，若将该节点对应的子树替换为叶节点能带来决策树泛化性能提升则将该子树替换为叶节点。如，scikit-learn采用最小代价复杂度修剪。
- 预剪枝 训练快，但用贪心策略可能导致欠拟合。
- 后剪枝 训练慢，但不易欠拟合。

3.5.2 CART训练算法 | sklearn中的DecisionTreeClassifier类

Sklearn.tree.DecisionTreeClassifier类：可用于两分类、多分类

DecisionTreeClassifier(*criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, max_features=None, max_leaf_nodes=None,...*)

参数	• *criterion* : string,默认'gini'。度量划分质量的函数,可选'entropy'；'gini'。 • *max_depth*: int,默认'None'。指定决策树的最大深度。若None,则不限制树的生长，直到叶子都纯。 • *min_samples_split*: int 或 float, 默认2。拆分内部节点所需的最小样本数。 • *min_samples_leaf*: int ,float,默认为1。叶节点中样本数目的最小值。 • *max_leaf_nodes*: int，默认None(叶节点不受限)。最大叶节点数目。 • *max_features*: int、float或{'sqrt','log2'}，默认None (即特征数目)。寻找最佳划分时要考虑的特征数量。若'sqrt', 则为sqrt(特征数目)；若'log2'，则为log2(特征数目)。
方法	• fit(X,y) 、 predict(X) 、score(X,y) 、predict_proba(X)：预测输入样本 X 的类别概率 • get_depth()：返回树深度；get_n_leaves()：返回叶节点数目。
属性	• feature_importances_ : 数组,形状 *[n_features]*。返回特征对决策的重要性。 • classes_:类标签 (单输出问题)，或类标签数组列表(多输出问题)。 • tree_：树对象。存储了整个二叉树结构，表示为多个数组。如，feature[i]是用于分割节点i的特征。threshold[i]是节点i的阈值。

正则化超参数，控制决策树的复杂性和大小

增大min_* 或 减少max_* 将使模型正则化

决策树可视化

常用方法有两种：

- 调用sklearn.tree下的plot_tree()函数，绘制决策树
- 使用sklearn.tree的导出器export_graphviz以Graphviz格式导出树模型，然后利用第三方包graphviz绘制树，并可输出为pdf格式文件。

需要安装graphviz包，pip install graphviz

决策树可视化

调用sklearn.tree 模块下的 plot_tree()函数

```
plot_tree(decision_tree,class_names=None,feature_names=None,
node_ids=False, filled=False, rounded=False, fontsize=None,…)
```

参数:

- *decision_tree*: 决策树分类器或回归器。
- *class_names*： 类别名称，字符串列表，默认值None。
- *feature_names*： 特征名称。字符串列表，默认值None。
- *node_ids*: 是否在每个节点上显示 ID 号，默认值False。
- *filled*: 是否填充，若True, 则用颜色填充节点，以表示分类时该节点的多少类。
- *rounded*: 是否绘制圆角节点框，默认值False。

引例 决策树预测鸢尾花类别

从鸢尾花数据集中取出petal length和petal width两个特征，构成由150个样本、两个特征的数据集。

① 为整个数据集训练一棵决策树。max_depth=2，其他超参数取默认值。

② 绘制这个决策树。

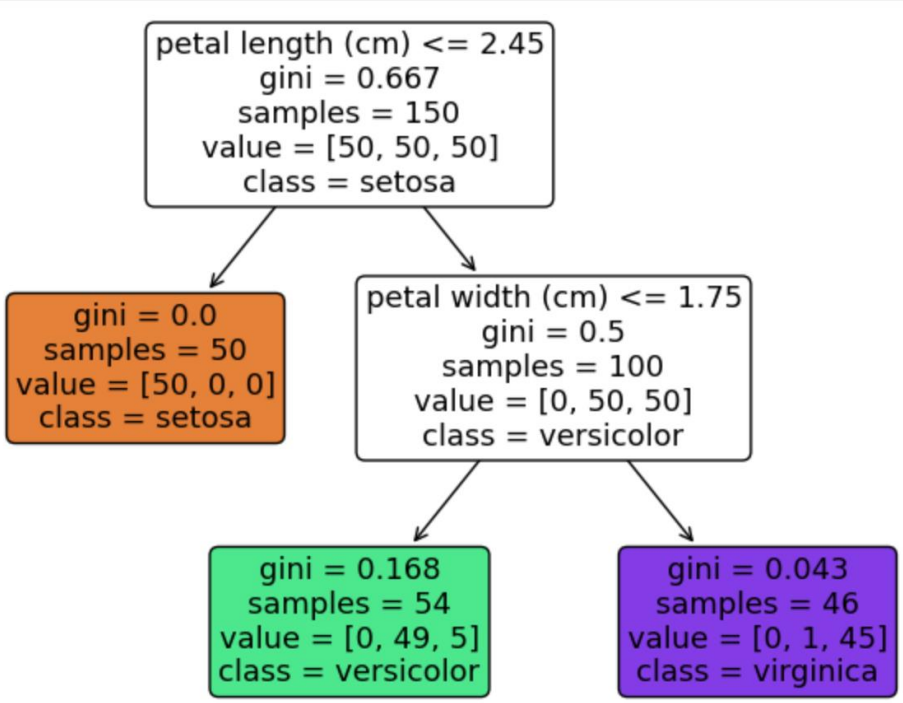
引例 决策树预测鸢尾花类别 | 代码

1.训练决策树

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

iris = load_iris(as_frame=True)
X_iris = iris.data[["petal length (cm)","petal width (cm)"]].values
y_iris = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)
```

2.可视化决策树　　调用plot_tree()函数绘制决策树

```
from sklearn import tree
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
tree.plot_tree(tree_clf,filled=True,rounded=True,
               feature_names=["petal length (cm)", "petal width (cm)"],
              class_names=iris.target_names)
plt.show()
```



例3.17 基于决策树的乳腺癌预测

要求：

1. 利用scikit-learn自带的乳腺癌数据，讨论预剪枝的作用。
2. 首先采用默认设置构建一个决策树分类器，即默认将树完全展开（树不断分支，直到所有叶节点都是纯的）。输出决策树分类器的训练得分和测试得分。
3. 设置**正则化超参数**max_depth=4，在到达指定深度后停止树的展开。观察分类器性能的变化。
4. 设置正则化超参数min_samples_leaf=4。观察决策树形态的变化。(请同学们自己完成)
5. 查看树的特征重要性。

关键代码(1)

1. 导入数据

```
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.model_selection import train_test_split
cancer=load_breast_cancer()
X_train,X_test,y_train,y_test=train_test_split(cancer.data, cancer.target,
                                               stratify=cancer.target,random_state=42)
```

2.创建决策树分类器 (默认设置)

```
clf=DecisionTreeClassifier(random_state=0)
clf.fit(X_train,y_train)
print("训练集上的准确度：{:.3f}".format(clf.score(X_train,y_train)))
print("测试集上的准确度：{:.3f}".format(clf.score(X_test,y_test)))
```

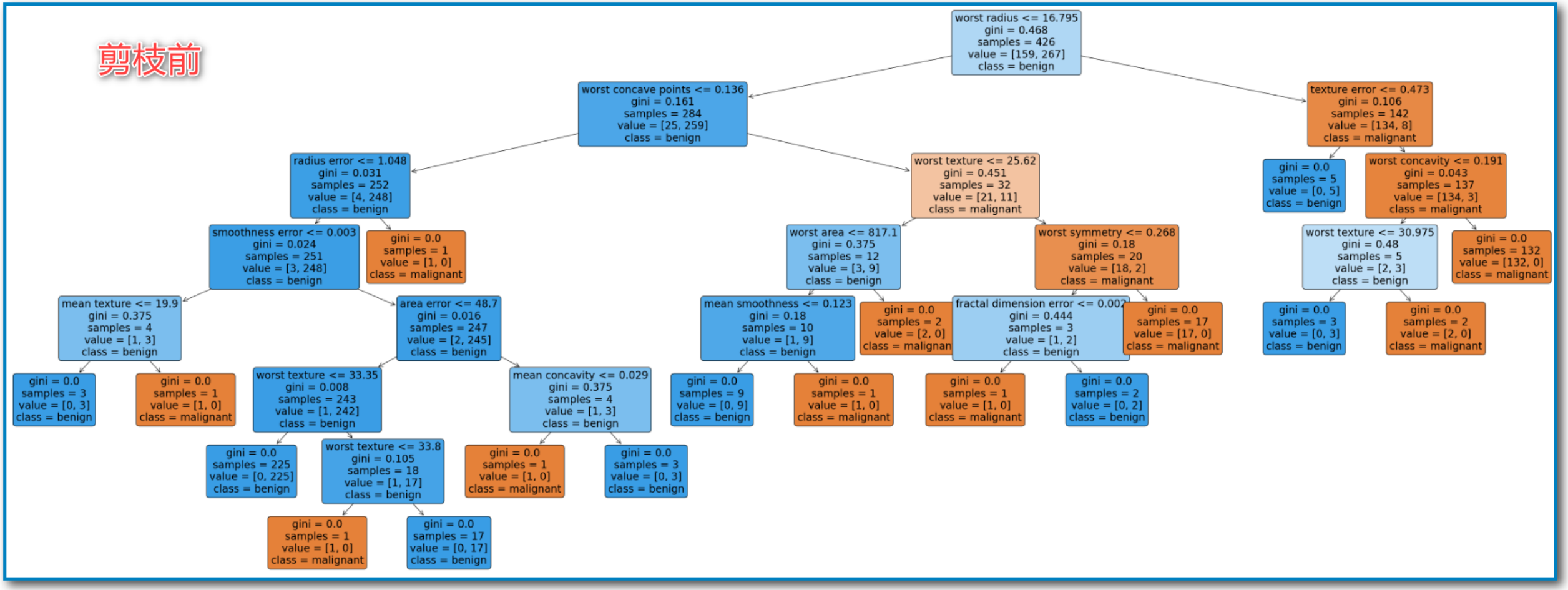
```
训练集上的准确度：1.000
测试集上的准确度：0.937
```

训练集上的准确度为100%，说明叶节点都是纯的，树的深度和复杂度都很高。

过拟合！

3.可视化未剪枝的决策树

```
import matplotlib.pyplot as plt
plt.figure(figsize=(55,20))
plot_tree(clf,filled=True,rounded=True,feature_names=cancer.feature_names,
          class_names=cancer.target_names, fontsize=20)
plt.show()
```


剪枝前
worst radius <= 16.795
gini = 0.468
samples = 426
value = [159, 267]
class = benign
worst concave points <= 0.136
gini = 0.161
samples = 284
value = [25, 259]
class = benign
texture error <= 0.473
gini = 0.106
samples = 142
value = [134, 8]
class = malignant
radius error <= 1.048
gini = 0.031
samples = 252
value = [4, 248]
class = benign
worst texture <= 25.62
gini = 0.451
samples = 32
value = [21, 11]
class = malignant
gini = 0.0
samples = 5
value = [0, 5]
class = benign
worst concavity <= 0.191
gini = 0.043
samples = 137
value = [134, 3]
class = malignant
smoothness error <= 0.003
gini = 0.024
samples = 251
value = [3, 248]
class = benign
gini = 0.0
samples = 1
value = [1, 0]
class = malignant
worst area <= 817.1
gini = 0.375
samples = 12
value = [3, 9]
class = benign
worst symmetry <= 0.268
gini = 0.18
samples = 20
value = [18, 2]
class = malignant
worst texture <= 30.975
gini = 0.48
samples = 5
value = [2, 3]
class = benign
gini = 0.0
samples = 132
value = [132, 0]
class = malignant
mean texture <= 19.9
gini = 0.375
samples = 4
value = [1, 3]
class = benign
area error <= 48.7
gini = 0.016
samples = 247
value = [2, 245]
class = benign
mean smoothness <= 0.123
gini = 0.18
samples = 10
value = [1, 9]
class = benign
gini = 0.0
samples = 2
value = [2, 0]
class = malignant
fractal dimension error <= 0.002
gini = 0.444
samples = 3
value = [1, 2]
class = benign
gini = 0.0
samples = 17
value = [17, 0]
class = malignant
gini = 0.0
samples = 3
value = [0, 3]
class = benign
gini = 0.0
samples = 2
value = [2, 0]
class = malignant
gini = 0.0
samples = 3
value = [0, 3]
class = benign
gini = 0.0
samples = 1
value = [1, 0]
class = malignant
worst texture <= 33.35
gini = 0.008
samples = 243
value = [1, 242]
class = benign
mean concavity <= 0.029
gini = 0.375
samples = 4
value = [1, 3]
class = benign
gini = 0.0
samples = 9
value = [0, 9]
class = benign
gini = 0.0
samples = 1
value = [1, 0]
class = malignant
gini = 0.0
samples = 1
value = [1, 0]
class = malignant
gini = 0.0
samples = 2
value = [0, 2]
class = benign
gini = 0.0
samples = 225
value = [0, 225]
class = benign
worst texture <= 33.8
gini = 0.105
samples = 18
value = [1, 17]
class = benign
gini = 0.0
samples = 1
value = [1, 0]
class = malignant
gini = 0.0
samples = 3
value = [0, 3]
class = benign
gini = 0.0
samples = 1
value = [1, 0]
class = malignant
gini = 0.0
samples = 17
value = [0, 17]
class = benign

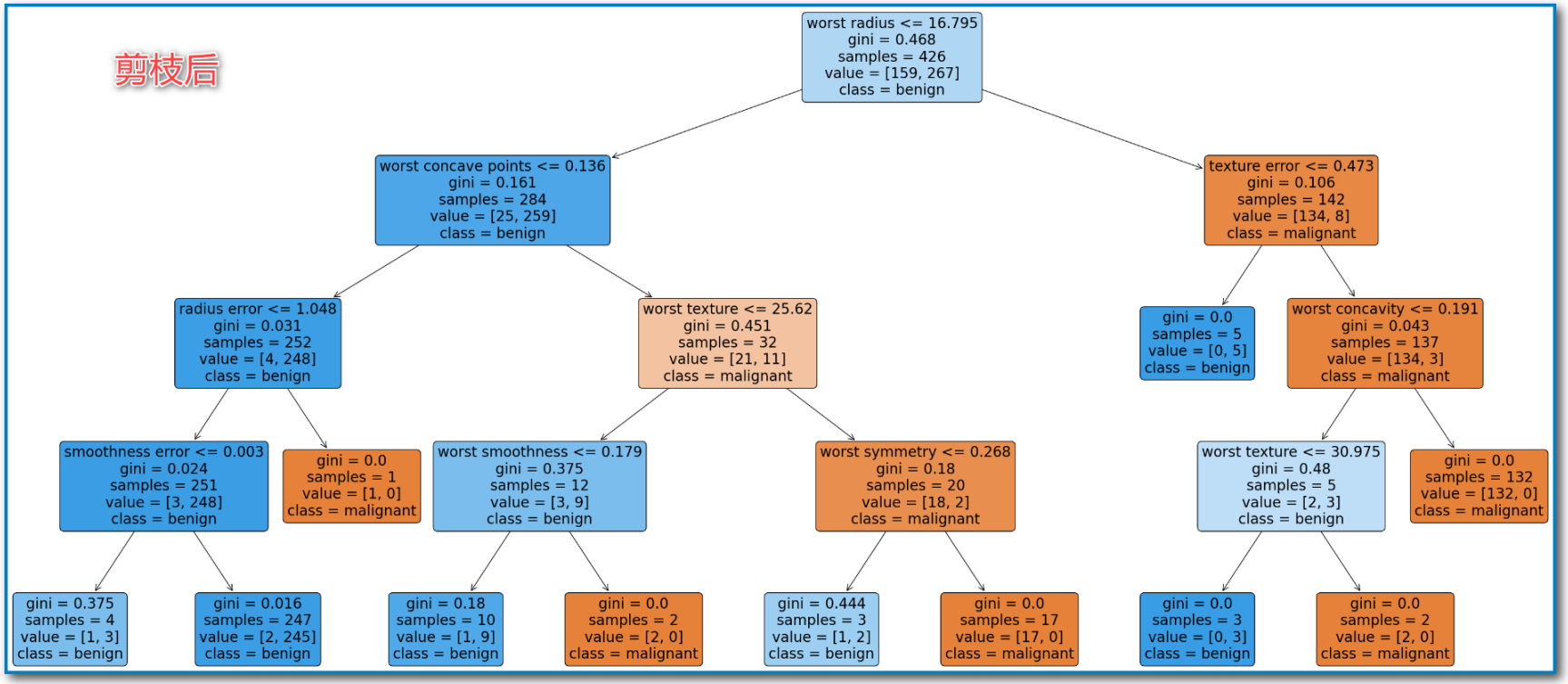
4.创建预剪枝的决策树（max_depth=4）

```
tree=DecisionTreeClassifier(max_depth=4,random_state=0) #设置max_depth=4
tree.fit(X_train,y_train)
print("剪枝后训练集上的准确度：{:.3f}".format(tree.score(X_train,y_train)))
print("剪枝后测试集上的准确度：{:.3f}".format(tree.score(X_test,y_test)))
```

```
剪枝后训练集上的准确度：0.988
剪枝后测试集上的准确度：0.951
```

5.可视化预剪枝的决策树

代码略



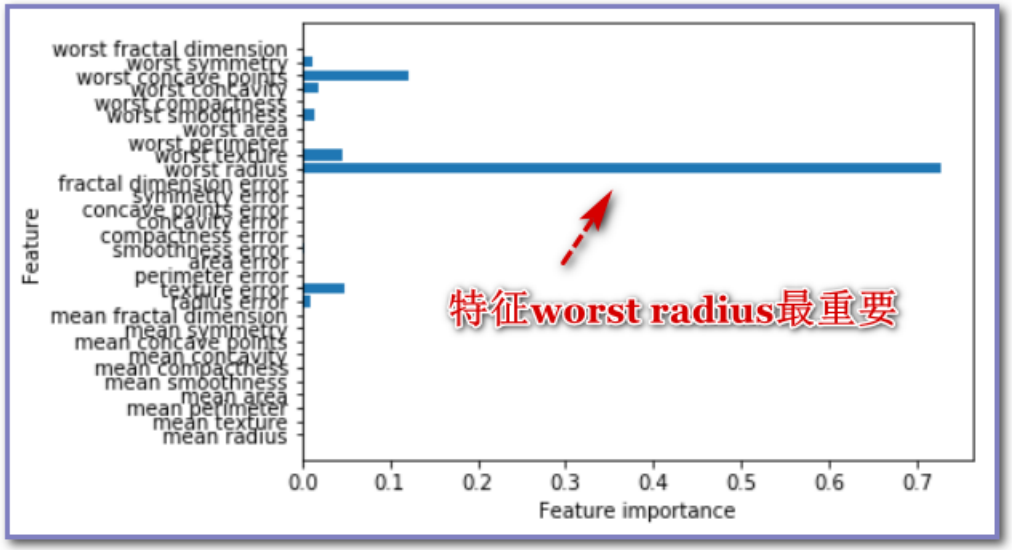
剪枝后，训练集上的准确度为98.8%，虽然略有下降，但在测试集上的准确性有明显提高，说明泛化能力得到提升。

关键代码(2)

6. 查看树的特征重要性

```
print("Feature importances:\n{}".format(tree.feature_importances_))
```

```
Feature importances:
[ 0.          0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.01019737  0.04839825  0.          0.
  0.0024156   0.          0.          0.          0.          0.
  0.72682851  0.0458159   0.          0.          0.0141577   0.          0.018188
  0.1221132   0.01188548  0.        ]
```



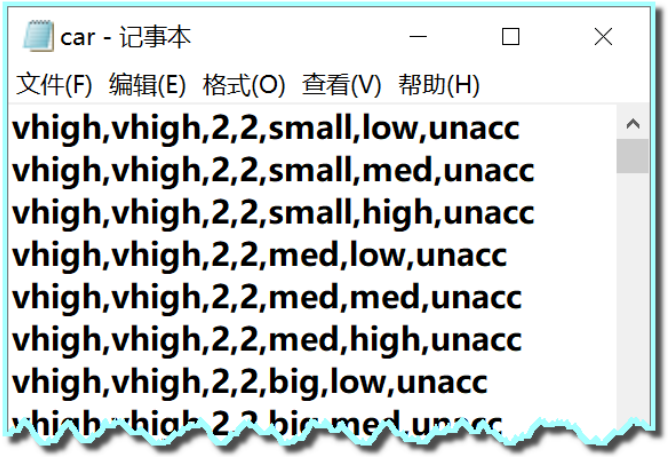

```
#将特征重要性可视化
import matplotlib.pyplot as plt
import numpy as np

n_features=cancer.data.shape[1] #特征数目
plt.barh(range(n_features),tree.feature_importances_,align='center')
plt.yticks(np.arange(n_features),cancer.feature_names)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
```

例3.18 汽车等级评估

使用决策树根据用户购买汽车时关注的指标（如价格、舒适性、安全性），为一款汽车做等级评估。

列号	列名	涵义	可取值
1	buying	购买价格	vhigh,high,med,low
2	maint	维护费用	vhigh,high,med,low
3	doors	车门数量	2,3,4,5more
4	persons	定员人数	2,4,more
5	lugs_boot	行李箱大小	small,med,big
6	safety	安全等级	low,med,high
7	class	分类标签	unacc,acc,good,vgood

- 数据集文件名car.data
- 数据集共有1728条数据，其中每一行包含6个特征、1个评估等级。



下载网址：https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

回顾：有序分类变量的转换

若数据是有序关系的分类变量，编码转换应保持序关系！ **Sklearn提供了两种**

1. 用sklearn中LabelEncoder编码器类，对分类变量进行标签编码。

`preprocessing.LabelEncoder()`

一般用于目标变量的编码

作用： 若一个数组（1维）包含n个不同的值，只要值不变且可比较，可用LabelEncoder 将原数组的值转换为0~n-1之间的数字。

方法：

- fit(*a*)：训练数组a 的编码器。
- transform(*a*)：用编码器对数组a进行编码转换。
- fit_transform(*a*)：用数组a训练编码器，并返回编码。
- inverse_transform(*a*)：将数据转换回原始表示形式。

属性： classes_：查看每个编码对应的原始值。<编码器名>.classes_

例　标签编码

y取值["Paris", "BeiJing", "Paris","BeiJing","Berlin"]，进行标签编码。

```
In [1]: from sklearn.preprocessing import LabelEncoder

In [2]: y = ["Paris", "BeiJing", "Paris","BeiJing","Berlin"] # 一个分类变量
        le = LabelEncoder()  # 创建一个编码器对象
        le.fit_transform(y)

Out[2]: array([2, 0, 2, 0, 1], dtype=int64)

In [3]: le.classes_  # 通过classes_属性可以查看转换值对应的原值

Out[3]: array(['BeiJing', 'Berlin', 'Paris'], dtype='<U7')
```

原值	编码
BeiJing	0
Berlin	1
Paris	2

若数据中有多个特征是分类变量，需要对这些特征分别进行标签编码，非常麻烦！

标签编码器LabelEncoder

```
# 数据标签编码，方法一：
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
data1 = data
for col in data.columns:
    data1[col] = labelencoder.fit_transform(data[col])
```

```
# 数据标签编码，方法二：dataframe对象名.apply(LabelEncoder().fit_transform)
from sklearn.preprocessing import LabelEncoder
data2=data.apply(LabelEncoder().fit_transform)
data2.head(3)
```

有序分类变量的转换（新方法）

2. 用sklearn中OrdinalEncoder编码器类，对分类数据进行有序编码。

`preprocessing.OrdinalEncoder()`

一般用于分类特征矩阵的编码

作用：将分类特征编码为整数数组

该编码器输入应为整数或字符串数组，表示分类（离散）特征所取的值。这些特征将转换为序数，即每个特征将产生一列整数（0 到 n_categories - 1）。

方法：

- fit(*X*)：编码器拟合X。 X是形状为 (n_samples, n_features) 的数组
- transform(*X*) : 用编码器对X进行编码转换。返回转换结果，同形状数组。
- fit_transform(*X*)：用X训练编码器，并返回转换结果。
- inverse_transform(*X*)：逆变换，将数据转换回原始表示形式。

属性： categories_：在fit过程中确定的每个特征的类别。

feature_names_in_：在拟合过程中看到的特征名称。仅在X的所有特征名称都是字符串时定义。

例　序数编码

X = [["Paris",78],["BeiJing",98],["Berlin",89],["BeiJing",90]]，进行编码。

In [1]:
```
from sklearn.preprocessing import OrdinalEncoder
coder = OrdinalEncoder()
X = [["Paris",78],["BeiJing",98],["Berlin",89],["BeiJing",90]]
X_encoded=coder.fit_transform(X) # 返回二维数组
```

In [2]:
```
X_encoded
```

Out[2]:
```
array([[2., 0.],
       [0., 3.],
       [1., 1.],
       [0., 2.]])
```

如何查看编码对应的原数据?

In [3]:
```
coder.categories_
```

Out[3]:
```
[array(['BeiJing', 'Berlin', 'Paris'], dtype=object),
 array([78, 89, 90, 98], dtype=object)]
```

In [4]:
```
coder.inverse_transform([[1., 1.]]) # 逆变换
```

Out[4]:
```
array([['Berlin', 89]], dtype=object)
```

例3.18 汽车等级评估 | 代码

课堂演示代码，程序名：案例 汽车等级评估.ipynb

分12步：

1.准备数据

2. 数据预处理（序数编码）

3. 将编码器保存在pickle文件中，以备后用

4. 模型创建、训练、评估

5. 画出决策树

6. 预剪枝处理（max_depth正则化）

7. 寻找最优决策树

8. 查看树的特征重要性

9. 将最优决策树模型保存在pickle文件中，以备后用

10. 预测新数据

①读取一条新数据;

②创建一个数据框来存储用户输入的数据;

③加载编码器和决策树模型，对新数据进行编码，然后进行预测。

例3.18 汽车等级评估 | 代码

1. 准备数据

```
import pandas as pd
data = pd.read_csv('car.data',header=None,
     names=['buying','maint','doors','persons',
            'lug_boot','safety','class’])
print(data.head(3))
```

给每列特征添加特征名称

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc

```
# 拆分数据
X = data.drop('class',axis=1) # 特征矩阵
y = data['class'] #目标数组
```

例3.18 汽车等级评估_OrdinalEncoder版.ipynb

2. 数据预处理（序数编码）

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split

car_encoder = OrdinalEncoder()
X_ = car_encoder.fit_transform(X)
print(X_[:3])    # 输出编码后的前3个特征向量

Xtrain,Xtest,ytrain,ytest = train_test_split(X_,y, test_size = 0.3, random_state=10)
```

```
[[3. 3. 0. 0. 2. 1.]
 [3. 3. 0. 0. 2. 2.]
 [3. 3. 0. 0. 2. 0.]]
```

3.保存标签编码器在pickle文件中，以备后用

```
import pickle
with open('car_encoder.pkl','wb') as f:
    pickle.dump(car_encoder,f,pickle.HIGHEST_PROTOCOL)
# 使用最高协议。使用的协议越高，读取生成的pickle所需的Python版本就越新。
```

4. 模型创建、训练、评估

```
from sklearn.tree import DecisionTreeClassifier


clf = DecisionTreeClassifier(random_state=42)
clf.fit(Xtrain,ytrain)
print("训练集上的准确度: {:.3f}".format(clf.score(Xtrain,ytrain)))
print("测试集上的准确度: {:.3f}".format(clf.score(Xtest,ytest)))
```

```
训练集上的准确度: 1.000
测试集上的准确度: 0.988
```

5. 画出决策树

```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

plt.figure(figsize=(55,20))
feature_names=['buying','maint','doors','persons','lug_boot','safety']
plot_tree(clf,class_names=['acc', 'good', 'unacc', 'vgood'],
                feature_names=feature_names,
                filled=True,rounded=True)
plt.show()
```

```
print(clf.tree_.max_depth)
```

```
14
```

6. 预剪枝处理

```
#设置max_depth=10
tree=DecisionTreeClassifier(max_depth=10,random_state=42)
tree.fit(Xtrain,ytrain)
print("剪枝后训练集上的准确度：{:.3f}".format(tree.score(Xtrain,ytrain)))
print("剪枝后测试集上的准确度：{:.3f}".format(tree.score(Xtest,ytest)))
```

```
剪枝后训练集上的准确度：0.984
剪枝后测试集上的准确度：0.981
```

7. 寻找最优决策树

```
from sklearn.model_selection import GridSearchCV
import numpy as np
grid_para={'max_depth':np.arange(4,16)}
grid = GridSearchCV(clf,grid_para,cv=7)
grid.fit(Xtrain,ytrain)
```

```
print(grid.best_params_)
```

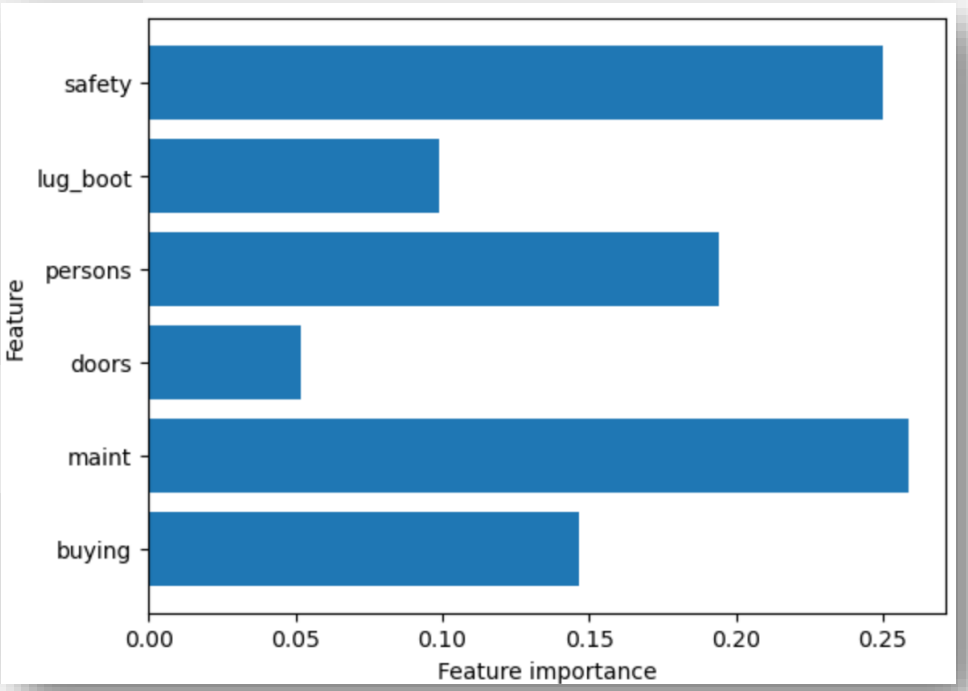
```
{'max_depth': 11}
```

```
best_tree = grid.best_estimator_
print("最优树在训练集上的准确度: {:.3f}".format(best_tree.score(Xtrain,ytrain)))
print("最优树在测试集上的准确度: {:.3f}".format(best_tree.score(Xtest,ytest)))
```

```
最优树在训练集上的准确度: 0.993
最优树在测试集上的准确度: 0.981
```

8.查看树的特征重要性

```
n_features=X.shape[1] # 特征数目
plt.barh(range(n_features),best_tree.feature_importances_,
        align='center')
plt.yticks(np.arange(n_features),feature_names)
plt.xlabel("Feature importance")
plt.ylabel("Feature")
```



9. 保存模型,存入pickle文件

```
import joblib
joblib.dump(best_tree,'car_tree.pkl')
```

10. 预测新数据

```
newdata=['vhigh','vhigh','2','2','small','high'] # 新数据
newdf = pd.DataFrame([newdata]) # 用一个数据框来存储新数据
newdf.columns=['buying','maint','doors','persons','lug_boot','safety']
```

```
# 加载标签编码器
with open('car_encoder.pkl','rb') as f:
    coder = pickle.load(f)  # 加载编码器
# 用于新数据的序数编码
newX=coder.transform(newdf)
```

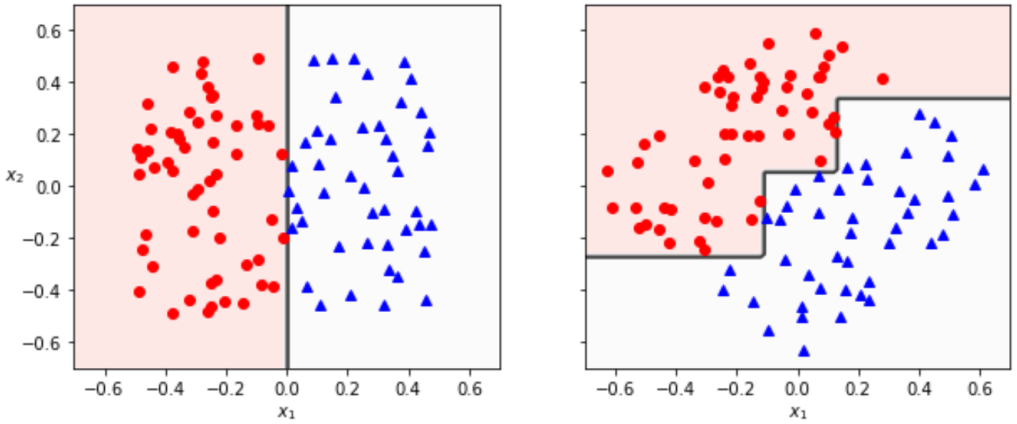
```
# 加载模型，对新数据进行预测
model=joblib.load('car_tree.pkl')
Ypred=model.predict(newX)
print('新数据预测结果：',Ypred)
```

```
新数据预测结果：  ['unacc']
```

3.5.3 决策树的不稳定性

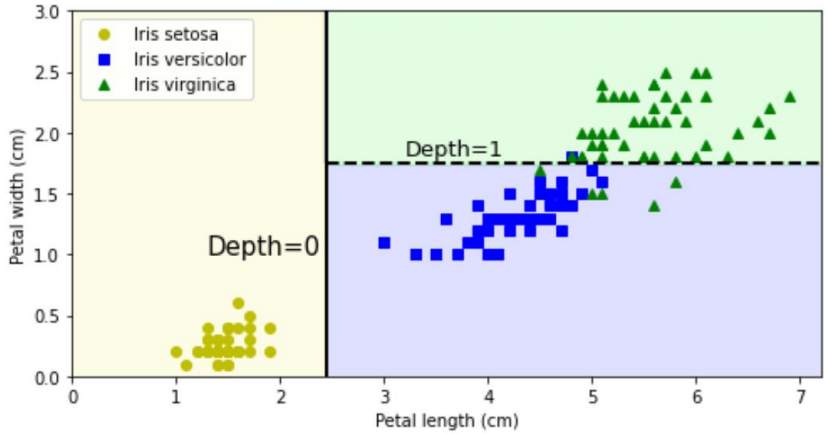
- **对数据集的微小变化敏感**

  如旋转数据集，将得到不同的决策树

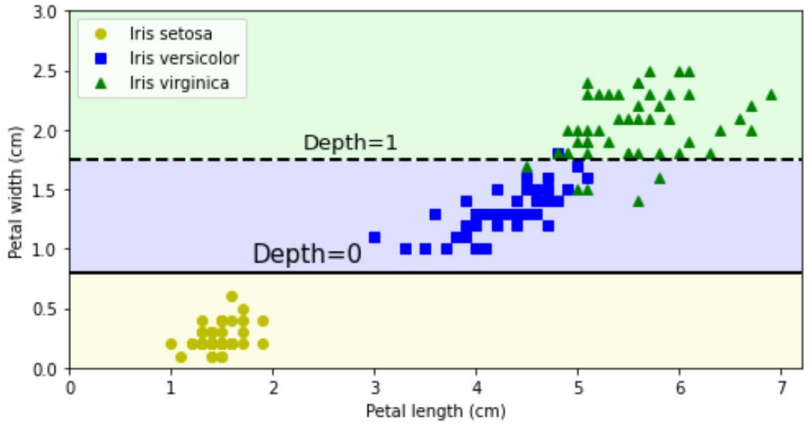


- **决策树具有高方差**

即使在相同的数据上训练相同的模型，每次也可能产生一个非常不同的模型，因为Scikit Learn中的CART训练算法是随机的，它随机地为每个节点选择待评估的特征集合。



DecisionTreeClassifier(max_depth=2, **random_state=42**)



DecisionTreeClassifier(max_depth=2, **random_state=40**)

决策树的优缺点

- **优点**:
  - 直观，易解释，属于白盒模型（white box model）；
  - 计算复杂度不高，预测复杂性大约$O(\log_2(m))$, m是样本数目，独立于特征数目。因此预测快；训练复杂性$O(n \times m \log_2(m))$，对于小数据集（少于几千个样本）可通过预排序（preorder=True）来加快训练速度，而大数据集这种方法会使训练变慢。
  - 算法不受数据尺度影响， 也不用中心化，不需特征规范化预处理。
- **缺点**:
  - 即使做了预剪枝，也易过拟合，泛化性能差。因此，大多数情况下，使用集成方法（随机森林、梯度提升决策树）代替单棵树。

小结

决策树是一种用于分类和回归的非参数监督学习技术。

- **优点**：直观，可理解；计算复杂度不高；算法不需特征规范化预处理。
- **缺点**：易过拟合，泛化性能差。因此，大多数情况下用集成方法代替单棵树。
- **难点**：划分特征的选择
- 防止过拟合方法：剪枝处理（预剪枝、后剪枝）
- 决策树分类，Scikit-learn中的DecisionTreeClassifier。
- 决策树回归，Scikit-learn中的DecisionTreeRegressor。不能外推，即不能在训练数据范围之外进行预测。

作业 6

6.1 不纯度度量

一个集合中有三类样本组成，各类样本数目分别为1,45,4，请分别用gini值和熵测量集合的不纯度。

作业6　　6.2 推销员决策树

某汽车推销员收集的一些顾客信息存入buyCar.csv (如表，可从课程文件夹中下载) 。希望建立一个能预测客户会否购车的模型。

1. 对表中给出的特征对进行必要的编码转换。
2. 训练一棵决策树。
3. 画出决策树
4. 预测新数据“28岁，男性，高收入”是否买车。

CustomerID	Age	Sex	Income	Bought
1	28	Male	Middle	No
2	35	Female	Middle	No
3	33	Female	Middle	No
4	38	Female	Low	No
5	22	Male	High	No
6	43	Female	Low	No
7	23	Female	Low	No
8	22	Female	High	Yes
9	44	Male	Middle	Yes
10	20	Male	High	No
11	37	Female	Middle	No
12	23	Male	Low	No
13	45	Female	Middle	No
14	43	Male	Low	Yes
15	40	Male	Middle	Yes
16	36	Female	Low	No

作业6　6.3 网页横幅广告判别

ad数据集下载：http://archive.ics.uci.edu/ml/datasets/Internet+Advertisements

ad数据集（文件名ad.data，可从课程文件夹下载）包括3279张图片数据，其中459张图片是广告，另外2820张图片是文章内容。

数据集共1559列，最后一列是类别标签(取值“ad.”是广告，“nonad.”不是)。前1558列是特征，包括图片的维度、网页URL中的文字、图片URL中的文字、图片的anchor属性文字、以及围绕图片标签的文字窗等。前3个特征是实数，分别是图片的宽度、高度和长宽比的编码数值。余下的特征对文字变量出现的频率进行二元编码。有1/4的样本缺少至少一个维度的值，这些缺失值使用一个空格和一个问号来标注。

要求：

(1)先用-1来替换这些缺失值。

(2)创建一个能判定是否为广告的决策树分类器。利用网格搜索，找出最优参数组合。输出最优模型的分类性能报告。

```
params = {'max_depth':[1,5,10,15,20,30,35,40],  'min_samples_split':[2,3], 'min_samples_leaf':[1,2,3]}
```

提示：1) 读取数据后，将最后一列赋给y，前面1558列赋给X
2) 替换缺失值：X.replace(to_replace=r'\?',value=-1,regex=True,inplace=True)