02-决策树算法（1天）

https://blog.csdn.net/m0_66890670/article/details/140649113

1、什么是决策树

决策树（decision tree）：决策树是一种树形结构的监督学习算法，广泛应用于分类任务和回归任务中。它通过递归地将数据集分割成更小的子集，最终形成一个树形模型，用于预测新数据的输出。

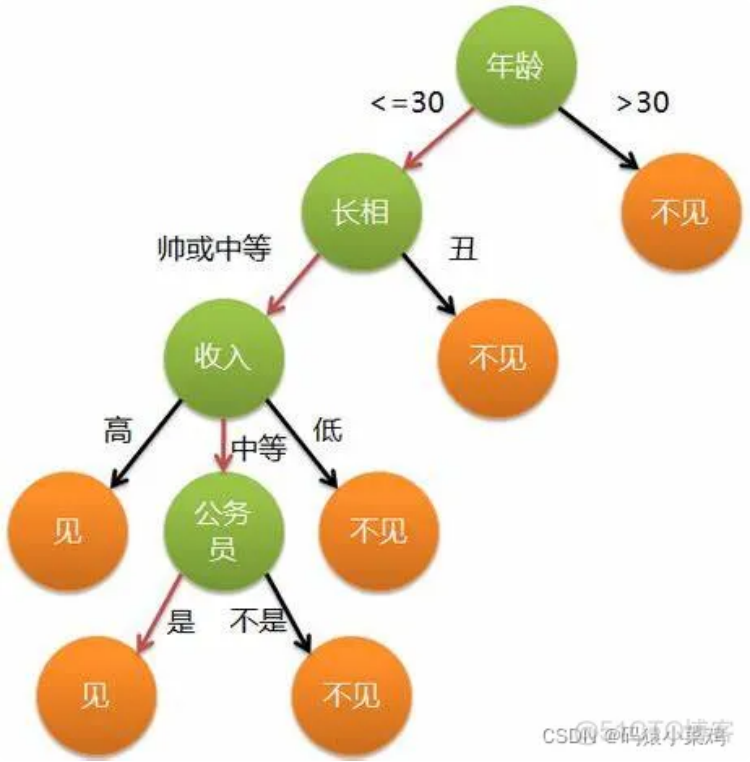
决策树即通过一步步决策得到最终结果的树

1.1、案例引入

比如，现在有如下的一些数据样本，每个样本的特征有 4 个：年龄，长相，收入，公务员，每个样本对应一个 2 个标签数据：见面，不见面；现在如果有一个人他的基本特征如下：

年龄：28，长相：帅气，收入：中等，公务员：是

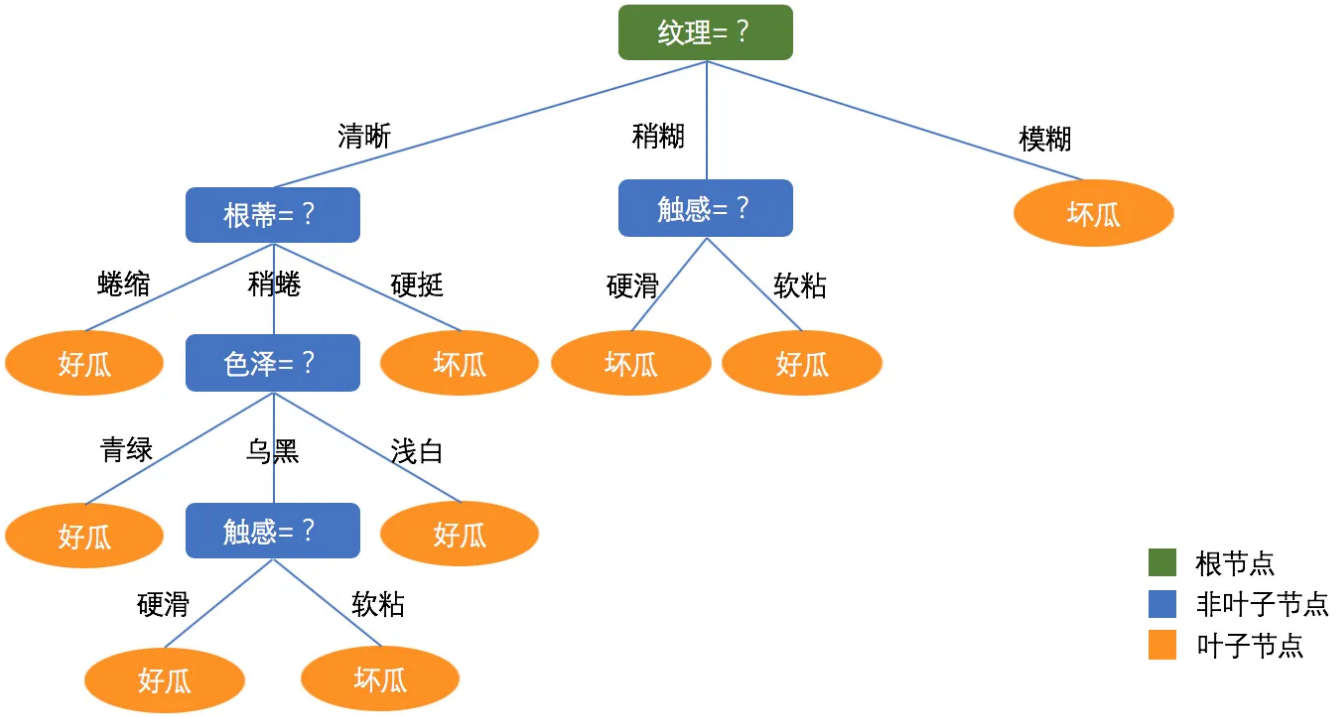
现在需要根据这个男孩的特征，通过分类算法，决定是否与他见面。



决策树基于树的结构进行决策，从根节点开始，沿着划分属性进行分支，直到叶节点

1.2、基本概念

- **节点（Node）**：表示一个特征或属性。
  - 根节点（Root Node）：树的最顶层节点，表示整个数据集的初始分割。
- **内部节点（Internal Node）/子节点**：除根节点和叶节点外的节点，用于进一步分割数据。
- **叶节点（Leaf Node）**：树的末端节点，表示分类或回归结果。
- **边（Edge）**：节点之间的连接，表示特征或属性的可能取值。
- **路径（Path）**：从根节点到叶节点的一条路径，表示一系列决策。



如上决策树分类图所示：从根节点（绿框）开始，对实例的某一**特征**（纹理、根蒂、色泽）进行测试，根据测试结果将实例分配到其内部节点（蓝框），此时每个子节点对应着该特征的一个取值，如此递归的对实例进行测试并分配，直到到达叶节点（橙框），最后将实例分到叶节点的类中。

2、决策树的构造

决策树学习的算法通常是一个递归地选择最优特征，并根据该特征对数据集进行分割，使得各个子数据集有一个最好的分类的过程。这一过程对应着对特征空间的划分，也对应着决策树的构造。

- 选择最佳特征：构建根节点，将所有训练数据都放在根节点，根据某种指标（如信息增益，信息增益比，基尼系数等）选择最能区分数据的特征作为当前节点的分割标准
- 数据分割：根据选择的特征将数据集分割成子集，每个子集对应特征的一个取值。如果这些子集已经能够被基本正确分类，那么构建叶节点，并将这些子集分到对应的叶节点中去；如果还有子集不能被基本正确分类，那么就对这些子集选择新的最优特征，继续对其进行分割，构建相应的节点。
- 递归构建子树：对子集重复上述过程，构建子树，直到满足停止条件：
    - 达到最大树深度，
    - 叶节点数据量小于最小样本数（数据量不足）
    - 节点纯度达到阈值
- 生成叶节点：当无法进一步分割时（基本正确分类或没有合适特征），生成叶节点并赋予分类或回归结果。最后每个子集被分到叶节点上，即都有了明确的类，这样就生成了一颗决策树。

3、特征选择标准

- 划分数据集的大原则是：将无序数据变得更加有序；
- 特征选择在于选取对训练数据具有分类能力的特征，这是决策树学习的关键。
- 常见的特征选择的准则是信息增益和信息增益率。
- 信息增益表示得知特征 X 的信息而使得类Y的信息不确定性减少的程度，即在划分数据集前后信息发生的变化，获得信息增益最高的特征就是最好的选择。

3.1、熵/信息熵

- 用来衡量一个系统的**不确定性**或**混乱度，熵越大，随机变量的不确定性就越大**
- 在决策树中，熵表示数据集的杂乱程度，熵越高，表示数据集中的样本类别越混乱，越不容易进行分类。
- 一个例子：A集合 [1,1,2,2], B集合[1,2,3,4]

显然A集合的熵值要低，因为A里面只有两种类别，相对稳定一些，而B中类别太多了，熵值就会大很多。

- **A 集合 [1,1,2,2]**:

  包含 2 个类别（1 和 2），每个类别各有 2 个样本，概率均为 42=0.5。

  分布相对集中，不确定性低，因此熵值小。

- **B 集合 [1,2,3,4]**:

  包含 4 个类别（1、2、3、4），每个类别仅 1 个样本，概率均为 41=0.25。

  分布高度分散，不确定性高，因此熵值大。

3.2、熵的计算

对于数据集 *D*，假设包含 *c* 个类别，第 *i* 类样本的概率为 *pi*（即 $p_i = \frac{第i类样本数}{总样本数}$），则熵的计算公式为：

$$H(D) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

- 对数底数通常取 2（单位为“比特”），也可使用自然对数（单位为“奈特”），核心是衡量概率分布的不确定性。
- 公式中添加负号是为了保证熵值非负（因 $pi \in [0,1]$，$\log(pi) \leqslant 0$，负号后 $H(D) \geqslant 0$）。

1. 计算 A 集合的熵

- 类别：2 类（1 和 2），概率 $p1=0.5$，$p2=0.5$
- 代入公式：$H(A)=-[0.5\log2(0.5)+0.5\log2(0.5)]$因 $\log2(0.5)=-1$,
- 故：$H(A)=-[0.5\times(-1)+0.5\times(-1)]=-[-0.5-0.5]=1$ 比特

2. 计算 B 集合的熵

- 类别：4 类（1、2、3、4），概率均为 $pi=0.25$
- 代入公式：$H(B)=-[4\times(0.25\log2(0.25))]$因 $\log2(0.25)=-2$,
- 故：$H(B)=-[4\times(0.25\times-2)]=-[4\times(-0.5)]=-[-2]=2$ 比特

3. 结果分析

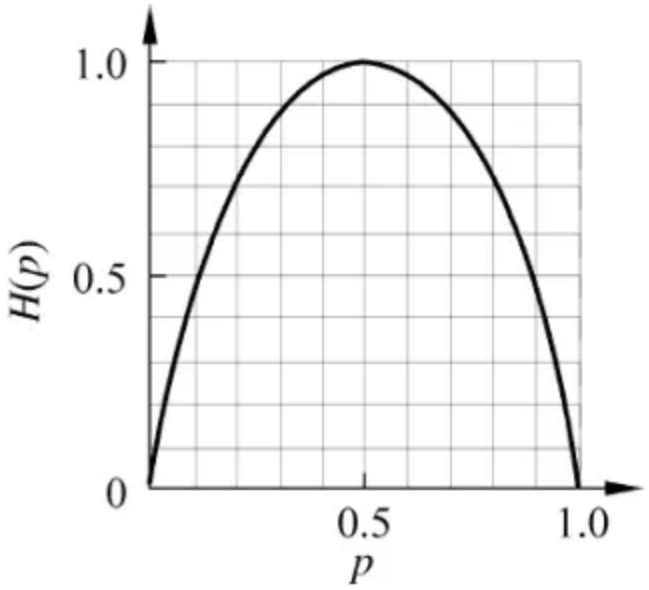
- *H*(*A*)=1，*H*(*B*)=2，符合直观判断：A 的熵值更低，不确定性更小。
- 极端情况：
  - 若数据集所有样本属于同一类别（如 [1,1,1,1]），则 *p*1=1，熵 *H*=−1×log2(1)=0（完全确定，熵最小）。
  - 若类别均匀分布（如 8 个类别各 1 个样本），熵会更大（*H*=3 比特），不确定性更高。

4. 如果数据集只有2个类别的时候，例如1,0时，那么X的分布为

$$P(X=1)=p, P(X=0)=1-p, 0 \le p \le 1$$

熵为：$H(D)=-plog_2p-(1-p)log_2(1-p)$

这时，熵$H(D)$随着概率p变化的曲线如下图所示：



当p=0或p=1时$H(D)=0$，随机变量完全没有不确定性。当p=0.5时，$H(D)=1$，熵取值最大，随机变量的不确定性最大。

3.3、条件熵

条件熵（Conditional Entropy） 是信息论中的一个重要概念，用来度量在已知某些条件下，数据集的剩余不确定性。简单来说，条件熵表示给定一个条件下，目标变量的不确定性。

条件熵H(Y|X)表示在已知随机变量X的条件下随机变量Y的不确定性，随机变量X给定的条件下随机变量Y的条件熵 (conditional entropy) H(Y|X)，定义X给定条件下Y的条件概率分布的熵对X的数学期望：$H(Y|X) = \sum_{i=1}^{n} p_i H(Y|X = x_i)$

其中，$p_i = P(X = x_i)$

例：在给定属性 的条件下，数据集的条件熵表示在属性 上分裂后，数据集的不确定性。假设属性有个可能的取值，将数据集分成 个子集 ，其中 是取第个值时的数据子集，则属性 的条件熵定义为：$H(D|A) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i)$

当熵和条件熵中的概率由数据估计(特别是最大似然估计)得到时，所对应的熵和条件熵分别称为经验熵(empirical entropy)和经验条件熵（empirical conditional entropy）。

3.4、信息增益

概念：信息增益是通过某个特征对数据集的划分来衡量“纯度”提升的程度。它表示通过特征划分数据集后，熵的减少量。信息增益越大，表示该特征的划分效果越好。

公式：例如，特征A对训练数据集D的信息增益g(D,A)，定义为集合D的经验熵H(D)与特征A给定条件下D的经验条件熵H(D|A)之差，即：$g(D, A) = H(D) - H(D|A)$

信息增益值的大小相对于训练数据集而言的，并没有绝对意义，在分类问题困难时，也就是说在训练数据集经验熵大的时候，信息增益值会偏大，反之信息增益值会偏小，使用信息增益比可以对这个问题进行校正，这是特征选择的另一个标准。

3.5、信息增益比（Information Gain Ratio）

信息增益比（信息增益率）：特征A对训练数据集D的信息增益比gR(D,A)定义为其信息增益g(D,A)与训练数据集D的关于特征A的值的熵$H_A(D)$之比,即：$g_R(D, A) = \frac{g(D,A)}{H_A(D)}$

其中，$H_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} log_2 \frac{|D_i|}{|D|}$，n是特征A取值的个数。

3.5、案例推到

假设有一个数据集 D，包括以下实例：

天气（climatic）	玩游戏
阳光(sunny)	是(yes)
阴天(cloudy)	是
雨天(rainy)	否(no)
阳光	否
阴天	是
雨天	否

我们要计算在属性“天气”上进行分裂的信息增益。

- **计算数据集 D 的熵**：

  总共6个实例，其中3个玩游戏（是），3个不玩游戏（否）。

  概率 $p_{\text{yes}} = \frac{3}{6} = 0.5, \quad p_{\text{no}} = \frac{3}{6} = 0.5$

  熵 $H(D) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$

- **计算在“天气”属性上分裂后的条件熵 $H(D|\text{climatic})$**：

  天气”属性有三个取值：阳光、阴天、雨天。

  计算每个子集的熵：

  阳光：2个实例（1是,1否），熵 $H(D_{\text{sunny}}) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$

  阴天：2个实例（2是），熵 $H(D_{\text{cloudy}}) = -(1 \log_2 1) = 0$。

  雨天：2个实例（2否），熵 $H(D_{\text{rainy}}) = -(1 \log_2 1) = 0$ 。

  条件熵 $H(D|\text{climatic}) = \left(\frac{2}{6}\right) \cdot 1 + \left(\frac{2}{6}\right) \cdot 0 + \left(\frac{2}{6}\right) \cdot 0 = \frac{1}{3}$

- **计算信息增益**$g(D, \text{climatic})$：

$$g(D, \text{climatic}) = H(D) - H(D|\text{climatic}) = 1 - \frac{1}{3} = \frac{2}{3}$$

- **计算信息增益比**$g_R(D, climatic)$

$$H_{climatic}(D) = -\left(\frac{2}{6}\log_2\frac{2}{6} + \frac{2}{6}\log_2\frac{2}{6} + \frac{2}{6}\log_2\frac{2}{6}\right) = \log_2 3 \approx 1.58496$$

$$g_R(D, climatic) = \frac{g(D,\text{climatic})}{H_{climatic}(D)} = \frac{\frac{2}{3}}{1.58496} \approx 0.421$$

3.6、基尼指数（Gini Index）

基尼指数（Gini Index）通过度量数据集的不纯度（或不一致性），来决定如何分裂节点。具体来说，基尼系数用于衡量一个节点的纯度程度，数值范围在0到0.5之间，其中0表示节点纯度最高（即节点内的样本全属于同一类），0.5表示节点纯度最低（即节点内的样本均匀分布在各类中）。

- **基尼指数**：分类问题中，假设有 K 个类，样本点属于第 k 类的概率为 p_k，则概率分布的基尼指数定义为

$$Gini(p) = \sum_{k=1}^{K} p_k(1 - p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

- 对于二类分类问题，若样本点属于第 1 个类的概率为 p，则概率分布的基尼指数为

$$\text{Gini}(p) = 2p(1 - p)$$

- 对于给定的样本集合 D，其基尼指数为

$$Gini(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|}\right)^2$$

  这里，C_k是 D 中属于第 k 类的样本子集，K 是类的个数。

- 如果样本集合 D 根据特征 A 是否取某一可能值 a 被分割成 D_1 和 D_2 两部分，即

$$D_1 = \{(x, y) \in D | A(x) = a\}, \quad D_2 = D - D_1$$

- 则在特征 A 的条件下，集合 D 的基尼指数定义为

$$Gini(D, A) = \frac{|D_1|}{|D|}\text{Gini}(D_1) + \frac{|D_2|}{|D|}\text{Gini}(D_2)$$

基尼指数 Gini(D) 表示集合 D 的不确定性，基尼指数 Gini(D, A) 表示经 A = a 分割后集合 D 的不确定性。基尼指数值越大，样本集合的不确定性也就越大，这一点与熵相似。

3.7、特征选择

在特征选择过程中，基尼系数用于评估每一个特征的分裂效果。具体步骤如下：

- 计算初始节点的基尼系数： 计算当前节点（即父节点）的基尼系数$Gini(D)$。
- 计算特征的基尼系数：对于每一个候选特征，基于该特征的不同取值将数据集 D 划分为若干子集$D_1, D_2, \ldots, D_m$， ，并计算这些子集的加权基尼系数。假设第j个特征有m个取值，计算该特征的基尼系数：$Gini(D, j) = \sum_{i=1}^{m} \frac{|D_i|}{|D|} \cdot \text{Gini}(D_i)$

  其中：

  D_i表示根据第j个特征划分得到的第i个子集。

  $|D_i|$和 $|D|$分别表示子集D_i 和原始数据集 D的样本数量。

  $Gini(D_i)$表示子集D_i的基尼系数。
- **选择最优特征**：选择基尼系数最小的特征作为当前节点的分裂特征，即选取能够最大程度减少不纯度的特征。

3.8、案例推到

假设数据集 D包含三类样本A、B和 C，且样本数量分别为 20、30 和 50。计算初始节点的基尼系数：

$$p_A = \frac{20}{100} = 0.2, \quad p_B = \frac{30}{100} = 0.3, \quad p_C = \frac{50}{100} = 0.5$$

$$Gini(D) = 1 - (0.2^2 + 0.3^2 + 0.5^2) = 1 - (0.04 + 0.09 + 0.25) = 1 - 0.38 = 0.62$$

假设某特征 X将数据集D分为D_1和 D_2，其中 D_1有 40 个样本（包含 10 个A、20 个 B和 10 个 C），D_2有 60 个样本（包含 10 个A、10 个 B和 40 个 C）。计算特征 X的基尼系数：

$$p_{A1} = \frac{10}{40} = 0.25, \quad p_{B1} = \frac{20}{40} = 0.5, \quad p_{C1} = \frac{10}{40} = 0.25$$

$$\text{Gini}(D_1) = 1 - (0.25^2 + 0.5^2 + 0.25^2) = 1 - (0.0625 + 0.25 + 0.0625) = 1 - 0.375 = 0.625$$

$$p_{A2} = \frac{10}{60} = 0.167, \quad p_{B2} = \frac{10}{60} = 0.167, \quad p_{C2} = \frac{40}{60} = 0.667$$

$$Gini(D_2) = 1 - (0.167^2 + 0.167^2 + 0.667^2) = 1 - (0.0278 + 0.0278 + 0.4449) = 1 - 0.5 = 0.5$$

$$Gini(D, X) = \frac{40}{100} \cdot 0.625 + \frac{60}{100} \cdot 0.5 = 0.25 + 0.3 = 0.55$$

由于$Gini(D, X) = 0.55$小于初始基尼系数$Gini(D) = 0.62$，所以特征X是一个较好的分裂特征。

基尼系数在决策树中的作用是通过衡量节点纯度，帮助选择最优的分裂特征，从而构建更有效的分类模型。使用基尼系数能够在较大程度上减少数据的不纯度，提高分类的准确性。

4、常用决策树算法

决策树算法中，ID3、C4.5和CART是三种最基本且广泛应用的算法。每种算法都有其独特的特征选择标准和树构建机制。

4.1、ID3（Iterative Dichotomiser 3）

特点：

- ID3 是基于信息增益（Information Gain）作为特征选择的标准来构建决策树的。
- 主要用于分类问题。
- 只能处理离散特征，不能直接处理数值型数据。
- 不支持剪枝，容易过拟合。
- 构建的树通常是二叉树或多叉树。

计算过程：

- 1）计算数据集的总熵。
- 2）对每个特征，计算信息增益。
- 3）选择信息增益最大的特征作为节点。

- 4）根据选定的特征分割数据集，重复以上步骤，直至所有特征被使用，或数据不再可分。

4.2、C4.5

特点：

- C4.5 是 ID3 算法的改进版本，使用信息增益比（Gain Ratio）来选择特征。
- 可以处理离散和连续特征。
- 引入了树的剪枝技术，减少过拟合。
- 结果通常是多叉树，而非二叉树。

计算过程：

- 1）使用信息增益比而非信息增益作为特征选择的标准。
- 2）对于连续特征，找到一个“最优”切分点将数据分为两部分，以最大化信息增益比。
- 3）递归构建决策树。
- 4）构建完整树后，应用剪枝技术，剪去那些对最终预测准确性贡献不大的分支。

4.3、CART（Classification and Regression Tree）

特点：

- CART 是一种既可以用于分类也可以用于回归的决策树算法。
- 使用基尼指数（Gini Index）作为分类任务的特征选择标准；使用最小二乘偏差（Least Squares Deviation）作为回归任务的标准。
- 始终构建二叉树。
- 内置有剪枝机制，使用成本复杂度剪枝（Cost Complexity Pruning）来防止过拟合。

计算过程：

- 1）对于分类任务，选择基尼指数最小的特征进行分割。
- 2）对于回归任务，选择使得结果变量的平方误差最小化的特征进行分割。
- 3）递归分割直至满足停止条件（如节点大小、树深度等）。

- 4）应用剪枝策略，通过设定不同的参数优化模型的泛化能力。

这三种算法各有优缺点。ID3简单直观但容易过拟合且只能处理离散数据；C4.5在ID3基础上进行了多项改进，更加灵活且有效地减少了过拟合；CART算法则提供了一个统一的框架适用于分类与回归任务，且总是构建二叉树，使得模型更加简洁易理解。在选择决策树算法时，应根据具体的数据特征和需求决定使用哪一种。

5、决策树的剪枝

5.1、为什么需要剪枝

决策树容易在训练数据上过拟合，尤其是当树变得特别深或复杂时。过拟合的树可能在训练数据上表现得很好，但在新的、未见过的数据上表现不佳。剪枝有助于减少这种过拟合，从而提高模型的泛化能力。剪枝通过去除决策树中的某些分支来实现，这些分支可能代表过度拟合训练数据的噪声或异常值。

5.2、剪枝的类型

- 决策树剪枝主要有两种类型：
    - **预剪枝（Pre-pruning）**：在决策树完全形成之前停止树的进一步生长。预剪枝的方法包括限制树的最大深度、限制节点中的最小样本数、或者当信息增益低于某个阈值时停止分裂。
    - **后剪枝（Post-pruning）**：首先构建决策树，让它成长到最大深度，然后开始移除那些对最终决策不产生重要影响的叶节点。常见的后剪枝技术包括成本复杂度剪枝（Cost Complexity Pruning，简称CCP）。
    - **成本复杂度剪枝（CCP）**：成本复杂度剪枝是一种常用的后剪枝方法，它通过引入“复杂度参数”（通常表示为α）来平衡树的大小和模型的拟合度。具体步骤如下：
        - 计算每个节点的错误率：在树的每个节点计算错误分类的代价。
        - 计算每个节点的“净增益”：这是通过将节点的错误率与树的复杂度（如树的深度或节点数）结合起来计算的。

- 选择合适的α：通过交叉验证或其他方法选择一个最优的α值。
- 剪枝：对于每个内部节点，如果移除该节点（及其所有子节点）并将其替换为叶节点能减少复杂度调整后的总错误率，则执行这一剪枝操作。

5.3、案例推到

设树T的叶节点个数为|T|，t是树T的叶节点，该叶节点有N_t个样本点，其中 k 类的样本点有 N_{tk} 个，； $H_t(T)$ 为叶结点t上的经验熵， $\alpha \geq 0$为参数，则决策树学习的损失函数可以定义为

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t H_t(T) + \alpha|T| \quad (1)$$

其中经验熵为

$$H_t(T) = \sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t} \quad (2)$$

在损失函数中，将式（1）右端的第1项记作

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t} \quad (3)$$

于是

$$C_\alpha(T) = C(T) + \alpha|T| \quad (4)$$

式（4）中，表示模型对训练数据的拟合损失， 表示模型复杂度，参数控制两者之间的影响。较大的促使选择较简单的模型（树），较小的促使选择较复杂的模型（树）。意味着只考虑对训练数据的拟合程度，不考虑模型的复杂度。

剪枝就是当确定时，选择损失较小的模型，即损失较小的子树， 越大，树越小。在训练数据集和测试数据集不同时，最优模型的复杂度就越低，但是在训练数据集和测试数据集一致时，损失较小的模型可以不剪枝。

可以看出，剪枝的生效仅仅只是通过提高信息度量（或信息增益比）对训练数据进行更好的拟合。通过正则化对拟合损失和模型复杂度进行权衡，从而实现剪枝。在生产系统中剪枝模型可以有效的选择出准确率更高的模型。

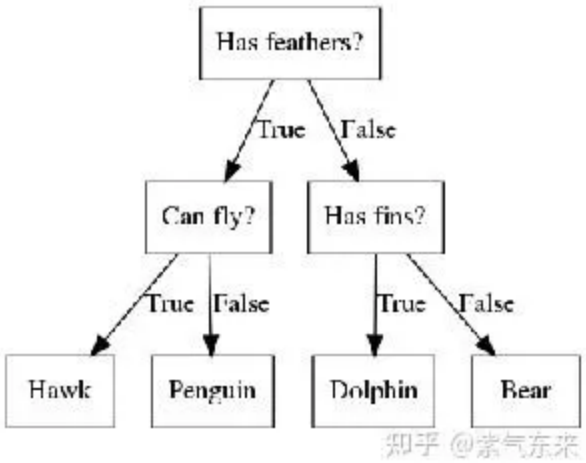
式（1）或（4）定义的损失函数的极小化等价于正则化的极大似然估计。因此，利用损失函数的原则进行剪枝就是利用正则化的极大似然估计进行模型选择。

6、使用sklearn库实现动物分类

可以调用mglearn库，展现动物分类的过程，如下：

```
CSS
```

```
import mglearn
mglearn.plots.plot_animal_tree()
```



使用sklearn库中DecisionTreeClassifier对癌症数据集进行处理，结果如下：

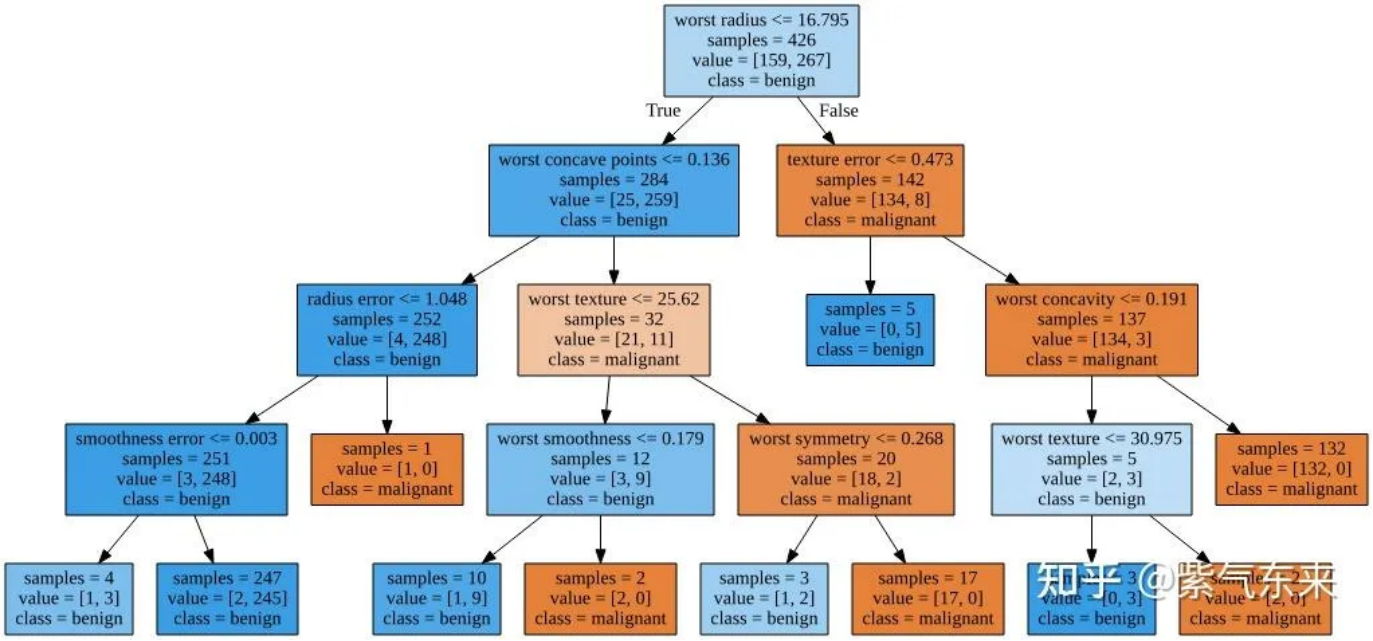
```
CSS
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import export_graphviz
import graphviz

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)

tree = DecisionTreeClassifier(max_depth=4,random_state=0)
tree.fit(X_train,y_train)

print('accuracy on training set:{:.3f}'.format(tree.score(X_train,y_train)))
print('accuracy on test set:{:.3f}'.format(tree.score(X_test,y_test)))
```

accuracy on training set:0.988 accuracy on test set:0.951

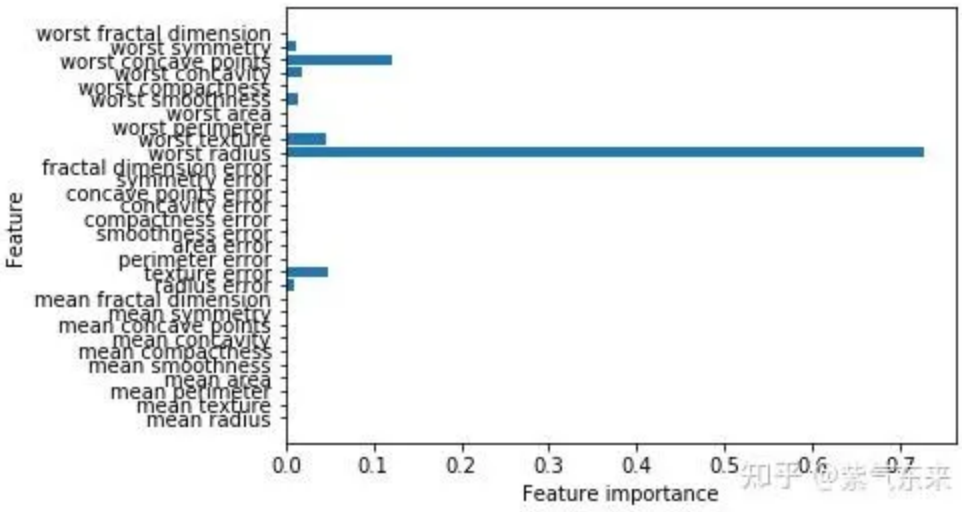
绘制其分类过程如下：

```
export_graphviz(tree,out_file='tree.dot',class_names=['malignant','benign'],feature_names=cancer.feature_names,impurity=False,filled=True)
with open('tree.dot') as f:
    dot_graph = f.read()

graphviz.Source(dot_graph)
```



同样，可找出其特征重要性，如下：

```
n_features = cancer.data.shape[1]
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')
```



CSS

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 加载数据
data = pd.read_csv('data.csv')
X = data.drop('target', axis=1)
y = data['target']

# 划分训练集和测试集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
andom_state=42)

# 构建决策树模型
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# 预测
y_pred = clf.predict(X_test)

# 评估模型
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# 输出决策树结构
tree_rules = export_text(clf, feature_names=list(X.columns))
print(tree_rules)
```

ID3决策树的实现（不调包）

```
import numpy as np

class DecisionTreeNode:
    def __init__(self, name):
        self.name = name
        self.branches = {}

def entropy(y):
    _, counts = np.unique(y, return_counts=True)
    probabilities = counts / counts.sum()
    return -np.sum(probabilities * np.log2(probabilities))

def information_gain(X, y, feature_index):
    total_entropy = entropy(y)
    values, counts = np.unique(X[:, feature_index], return_counts=True)
    weighted_entropy = 0

    for value, count in zip(values, counts):
        subset_y = y[X[:, feature_index] == value]
        weighted_entropy += (count / len(X)) * entropy(subset_y)

    return total_entropy - weighted_entropy

def id3(X, y, feature_names):
    if len(np.unique(y)) == 1:
        return DecisionTreeNode(str(y[0]))

    if len(feature_names) == 0:
        return DecisionTreeNode(str(np.bincount(y).argmax()))

    gains = [information_gain(X, y, index) for index in range(X.shape[1])]
    best_feature_index = np.argmax(gains)
    best_feature_name = feature_names[best_feature_index]

    node = DecisionTreeNode(best_feature_name)
    values = np.unique(X[:, best_feature_index])

    for value in values:
        sub_index = X[:, best_feature_index] == value
        sub_X = X[sub_index]
        sub_y = y[sub_index]
        sub_feature_names = feature_names[:best_feature_index] + feature_n
ames[best_feature_index + 1:]

        child = id3(sub_X, sub_y, sub_feature_names)
```

```
            node.branches[value] = child

    return node

def print_tree(node, level=0):
    if node.branches:
        for value, child in node.branches.items():
            print(' ' * level * 4, node.name, '=', value)
            print_tree(child, level + 1)
    else:
        print(' ' * level * 4, 'RESULT:', node.name)

# 示例数据
feature_names = ['Outlook', 'Temperature', 'Humidity', 'Windy']
X = np.array([
    ['Sunny', 'Hot', 'High', 'False'],
    ['Sunny', 'Hot', 'High', 'True'],
    ['Overcast', 'Hot', 'High', 'False'],
    ['Rain', 'Mild', 'High', 'False'],
    ['Rain', 'Cool', 'Normal', 'False'],
    ['Rain', 'Cool', 'Normal', 'True'],
    ['Overcast', 'Cool', 'Normal', 'True'],
    ['Sunny', 'Mild', 'High', 'False'],
    ['Sunny', 'Cool', 'Normal', 'False'],
    ['Rain', 'Mild', 'Normal', 'False'],
    ['Sunny', 'Mild', 'Normal', 'True'],
    ['Overcast', 'Mild', 'High', 'True'],
    ['Overcast', 'Hot', 'Normal', 'False'],
    ['Rain', 'Mild', 'High', 'True']
])
y = np.array(['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes',
'Yes', 'Yes', 'Yes', 'Yes', 'No'])

tree = id3(X, y, feature_names)
print_tree(tree)
```

7、总结

6.1、优点：

- 简单易理解：
    - 决策树的结构类似于人类的决策过程，具有很好的可解释性。

- 可以通过树状图直观地展示决策过程和规则，易于理解和解释。

- 非线性关系：
  - 决策树能够处理线性和非线性关系，不要求数据线性可分。
  - 通过分裂节点，决策树可以捕捉复杂的模式和关系。
- **特征选择**：
  - 决策树可以自动进行特征选择，重要特征会被优先分裂，提高了模型的性能和效率。
- **处理缺失值**：
  - 决策树在处理缺失值时具有一定的鲁棒性，能够有效地处理部分缺失的数据。
- **数据预处理要求低**：
  - 决策树对数据的预处理要求较低，不需要进行特征缩放或归一化处理。
  - 可以处理连续型和离散型变量。
- **可处理多分类问题**：
  - 决策树不仅可以处理二分类问题，还可以处理多分类问题，应用广泛。

6.3、缺点

- **容易过拟合**：
  - 决策树在训练过程中容易对训练数据过拟合，特别是当树的深度过大时，模型的泛化能力较差。
  - 需要通过剪枝或设置最大深度等方法进行正则化处理。
- **对噪声敏感**：
  - 决策树对数据中的噪声较为敏感，噪声数据可能导致树结构发生较大变化，影响模型稳定性。
- **不稳定性**：
  - 决策树对数据的微小变化较为敏感，训练数据的轻微变化可能导致生成完全不同的树。
  - 需要通过集成学习方法（如随机森林、梯度提升树等）来提高模型的稳定性和鲁棒性。

- **偏差–方差权衡**：
    - 决策树通常存在较高的方差和较低的偏差，容易产生高方差问题。
    - 需要结合其他模型或通过集成方法来平衡偏差和方差。
- **计算复杂度**：
    - 对于高维数据或特征数量较多的数据集，决策树的计算复杂度较高，训练时间较长。
    - 尤其是在节点分裂过程中，需要遍历所有特征及其取值，计算代价较大

8、结论

决策树作为一种直观、易理解的模型，适用于各种分类和回归任务，但其在处理复杂、高维数据时可能面临过拟合、对噪声敏感等问题。因此，在实际应用中，常结合其他模型或集成方法来提高性能和稳定性。