《机器学习》课程系列

决策树*

武汉纺织大学数学与计算机学院

杜小勤

2020/06/13

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: June 20, 2020。

1 概述

决策树 (Decision Tree) 是一种基本的分类与回归方法，属于树形结构模型。例如，在分类问题中，可以把决策树的决策过程看作是，一系列针对输入 $\boldsymbol{x}$ 的主要和重要的特征分量 (或属性) 进行划分、分类或判断的过程。本质上，分类决策树等价于 If-Then 规则的集合，或者等价于特征空间与类别空间上的条件概率分布。

决策树的优点是，决策模型具有可读性与可解释性[1]，且决策速度快。它是一种将特征空间的若干重要特征或属性映射到对象类别或特定值的方法。对于分类决策树，其基本度量准则是信息论中的熵 (Entropy)——利用熵来反映分类前后数据 (子) 集的混乱程度，进而能够反映特征或属性的重要程度，从而为分类决策提供了重要的量化依据。

如图1-1所示，展示了一棵分类决策树。在该决策树中，不规则矩形表示决策节点，它对应输入 $\boldsymbol{x}$ 中的某个特征。一般而言，决策节点离根节点越近，表明其对应的属性或特征的重要性越高，属于分类决策的主要与重要特征。除了决策重要性之外，特征离根节点的远近还与决策顺序有关。例如，特征“Homework done?”为根节点，表明其最重要，属于全局重要特征，它决定了后续决策流程；随后，特征“Outlook”的重要性次之，将进一步决定决策对象的后续活动；最后，特征“Umbrella”属于决策分支“Rainy”的后续“重要”节点，属于局部重要特征，它决定了局部决策流程。蓝色圆角矩形为示例决策树的叶子节点，表示决策树的决

[1]例如，根据所生成的决策树，可以很容易地建立适用于人类决策的逻辑表达式。

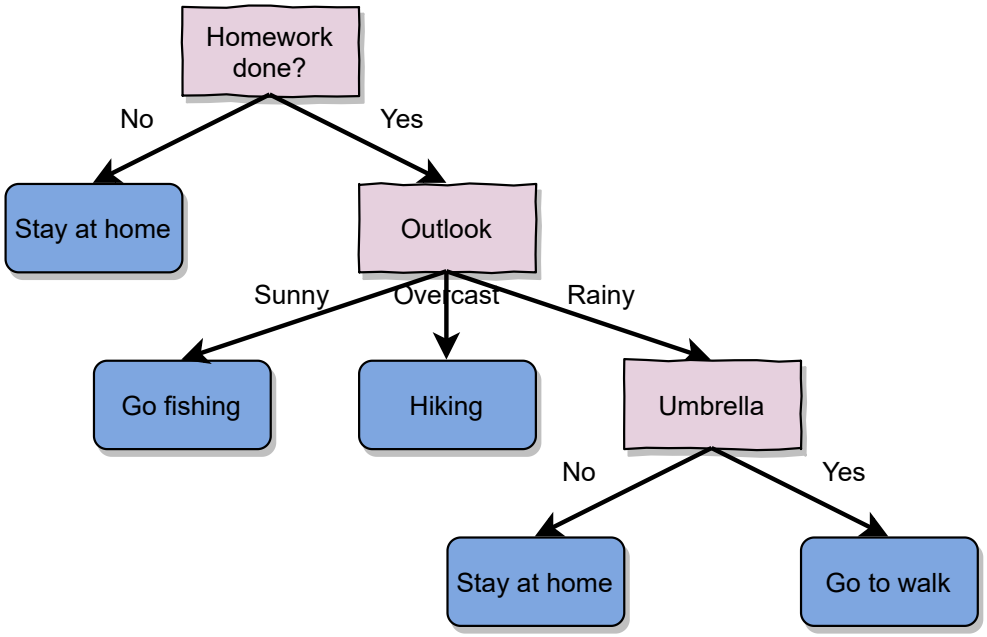


图 1-1: 一棵分类决策树

策结果。

实际上，从上述示例可以看出，分类决策树包含着分类与决策 2 个功能部分。其分类功能不仅体现在终端叶子节点上，而且也体现在非叶子节点上。正是因为这个原因，我们可以很容易地将分类决策树转换成人类所擅长的 If-Then 决策规则集合。

2 分类决策树

2.1 模型的定义

首先，给出分类决策树的定义。

定义 **2.1**（分类决策树）分类决策树是一种对输入 $\boldsymbol{x}$ 进行分类的树形结构。它由节点和有向边组成；节点分为内部节点与叶子节点。内部节点表示输入 $\boldsymbol{x}$ 的某个特征或属性，也被称为决策节点；而叶子节点表示输入 $\boldsymbol{x}$ 所属的类别或分类，也被称为类别节点。决策时，从根节点开始，依据输入 $\boldsymbol{x}$ 的相应特征及其具体取值，做出相应的选择，即沿着一条有向边，或者到达新的决策节点，或者到达类别节点：对于前者，按照上述方法，继续进行决策并执行流程，该过程一直持续下去，

直至到达类别节点；对于后者，得到输入 $\boldsymbol{x}$ 的类别，分类决策过程结束。

上述定义，给出了分类决策树模型的定性描述及其决策流程。一个非常自然的问题是，如何从给定的训练数据 $T=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)\}$(其中，$N$ 表示实例样本点的个数) 中生成一棵适用于实际应用问题的分类决策树呢？

本质上，分类决策树需要满足以下 2 个功能：

1. 从给定的训练数据集 T 中，归纳出一组能够对 T 中实例样本点 $\boldsymbol{x}_i$ 进行分类的规则，且分类结果需要达到一定的训练精度要求。

   很难想象，对于可见的训练数据集分类效果不好的分类决策树模型，能够在未见的实际数据上取得较好的分类效果。因此，本目标是首要的。

2. 然而，我们的真正目标是，第 1 步归纳出的分类决策树模型能够满足实际应用的需求，希望其分类效果在未见的实际数据上也能够达到一定的精度要求，即所获得的分类决策树模型也应该具有很好的泛化能力。

理想情况下，我们希望所获得的分类决策树模型，既具有很好的训练精度，也具有很好的泛化能力。但是，在绝大多数情况下，训练精度与泛化能力之间是有冲突的，算法需要在两者之间取得某种平衡或折中。

因此，粗略地讲，我们需要生成一棵与训练数据集“基本”相符的分类决策树，也需要生成一棵对实际数据有很好预测能力的分类决策树。

对于前者，分类决策树的学习算法可以从整个训练数据集开始，首先选择 1 个最优特征[2]，并根据所选特征对训练数据集进行划分，这样就可以得到具有“最好”分类效果的若干训练数据子集；然后，针对每个数据子集，再次选择最优特征对数据子集进行划分。上述过程，循环往复，一直持续下去，直至划分结束，或满足某些条件为止。最后，每个实例样本点都将被划分到某个叶子节点上，即都有了明确的类别。实际上，上述的递归过程，包含特征选择与分类决策树的生成这 2 个功能，能够生成一棵与训练数据集“基本”相符的分类决策树。

然而，需要注意的是，上述过程生成的分类决策树，可能存在“过拟合”现象，即算法对训练数据集有很好的分类精度，但对未知的测试数据却未必有很好的分类精度。因此，一般需要对生成的分类决策树进行剪枝，使之变得简单些，以增强其泛化能力。

[2]即优先选择那些最能刻画当前训练数据集特性的属性进行处理。一般而言，这种属性就是重要属性。

综上所述，分类决策树的学习算法包括特征选择、决策树生成与剪枝等功能。由于分类决策树可以表示为一个条件概率分布，所以不同复杂度的分类决策树对应着不同复杂度的概率模型。决策树的生成对应着模型的局部“最优化”过程，而决策树的剪枝对应着模型的全局“最优化”过程。

2.2 信息论的基本概念

2.2.1 自信息

自信息 (Self Information) 由信息论创始人 Claude Elwood Shannon 提出，它是与概率空间中的单一事件或离散随机变量的值相关的信息量的量度。它的单位为比特 (Bit)、纳特 (Nat) 与哈特 (Hart)，分别对应于以 2、自然常数 e 以及 10 为底的对数。事件 $X = x$ 的自信息定义如下：

$$I(X = x) = -\log P(X = x) = \log \frac{1}{P(X = x)} \tag{1}$$

其中，$P(X = x)$ 表示事件 $X = x$ 发生的概率。从上式可以看出，事件 $X = x$ 的概率越小，其发生后的自信息量越大，即不确定性减少的程度越大[3]。图2-2展示了自信息曲线 (对数底数为 2)。

当某个实体 (发送方) 将自身的信息传递给另一个实体 (接收方) 时，仅当接收方知道关于消息的先验知识少于 100% 时，信息才得到真正地传递；否则，如果接收方事先知道了消息的全部内容，那么这条消息所传递的信息量就是 0。

因此，自信息量具有如下特点：

- 随机事件 $X = x$ 发生后，所带来的自信息量，只与该事件发生的概率有关，与 X 的具体取值无关；
- 随机事件 $X = x$ 发生的概率越低，事件发生后，接收到的自信息量就越大；反之，自信息量越小；
- 如果 $P(X = x) = 1$，则 $I(X = x) = 0$；如果 $P(X = x) < 1$，那么 $I(X = x) > 0$，即自信息量是非负的；

[3]需要注意的是，信息量不同于信息作用量，它们是不同的概念。信息量的大小只表明不确定性的减少程度不同。而对于不同的信息接收者而言，所获得的信息可能非常重要，也可能无足轻重，这表示信息作用量的大小不同。

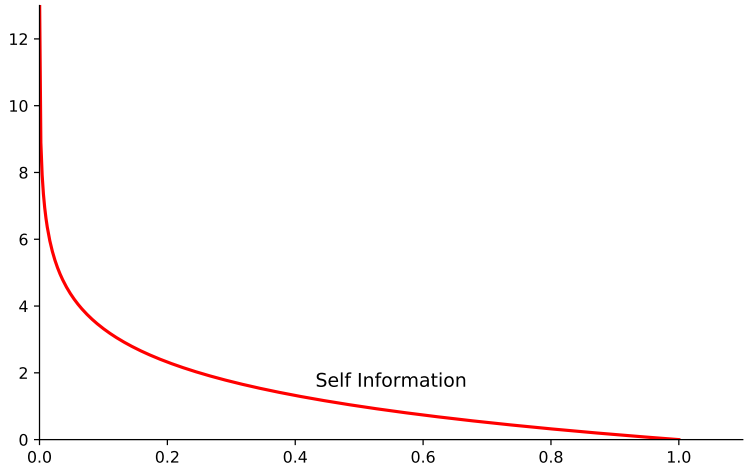


图 2-2: 自信息曲线图

- 如果事件 $Z = z$ 表示两个独立事件 $X = x$ 和 $Y = y$ 同时发生，那么事件 $Z = z$ 发生后产生的自信息量就等于事件 $X = x$ 与事件 $Y = y$ 分别发生后所产生的自信息量的总和，即：

$$I(Z = z) = I(X = x, Y = y) = I(X = x) + I(Y = y) \tag{2}$$

即自信息量具有可加性[4]。例如，抛 m 次硬币所获得的自信息量，应该是抛 1 次硬币所获自信息量的 m 倍。

2.2.2 信息熵

熵 (Entropy) 的概念最早起源于物理学，用于度量一个热力学系统的无序或混乱程度。Shannon 于 1948 年将熵从热力学领域引入到信息论中，因而也被称为信息熵、香农熵或平均自信息量，它是自信息的期望值或接收的每条消息[5]中包含的平均信息量[6]，也反映了随机变量概率分布的平均或整体不确定性程度。与自信

[4]实际上，对于上述的独立事件 $X = x$ 与 $Y = y$ 以及事件 $Z = z$ 而言，它们之间的概率存在关系：$P(Z = z) = P(X = x, Y = y) = P(X = x) \cdot P(Y = y)$，两端应用 $-\log$ 函数，将得到 $-\log P(Z = z) = -\log P(X = x) - \log P(Y = y)$，即 $I(Z = z) = I(X = x) + I(Y = y)$。这也从另一方面，解释了自信息量取 $-\log$ 的原因。

[5]与前述的自信息概念一样，“消息”指的是概率分布或数据流中的事件、样本或特征等。

[6]从信息传输的角度看，熵越高，传输的信息越多；反之，传输的信息越少。例如，英文数据流的熵比较低，原因在于英文数据流较容易被预测：字母、字母间的组合等出现的频率有较强的偏

息的单位一样，熵的单位有比特 (Bit)、纳特 (Nat) 与哈特 (Hart)，选用哪一个，取决于对数的底。

在实际应用中，熵的常用实际意义有如下几个：

- 度量事件或信源的平均 (期望) 信息量；
- 编码某个符号体系中所有符号所需的最少位数；
- 衡量某个系统的无序或混乱程度；

下面给出熵的定义。设 X 为取有限个值的离散随机变量，其概率分布为：

$$P(X = x_i) = p_i \quad i = 1, 2, \ldots, n \tag{3}$$

则随机变量 X 的熵被定义为[7]：

$$\begin{aligned} H(X) &= \mathbb{E}\left[I(X)\right] = \mathbb{E}\left[-\log P(X)\right] \\ &= \sum_{i=1}^{n} P(X = x_i) I(X = x_i) \\ &= -\sum_{i=1}^{n} P(X = x_i) \log P(X = x_i) \\ &= -\sum_{i=1}^{n} p_i \log p_i \end{aligned} \tag{4}$$

其中，如果 $p_i = 0$，则定义 $0 \log 0 = 0$。这与极限情形一致：

$$\lim_{p \to 0+} p \log p = 0 \tag{5}$$

由定义可知，熵只依赖于 X 的分布，与 X 的具体取值无关，因而可将 X 的熵记作 $H(\boldsymbol{p})$：

$$H(\boldsymbol{p}) = -\sum_{i=1}^{n} p_i \log p_i \tag{6}$$

向性。如果未经压缩，英文文本的每个字母需要 8 个比特来编码。但是，理论上，英文文本的信息熵大概只有 $-\sum_{i=1}^{26} \frac{1}{26} \log_2(\frac{1}{26}) \approx 4.7$ 比特。如果考虑到各字母的实际频率，则英文的信息熵为 4.03 比特。其它语言，例如法语为 3.98、西班牙语为 4.01 比特、德语为 4.10 比特、俄文为 4.8 比特，而中文高达 9.65 比特。上述语言的信息熵，都低于相应字符集的 (理论) 最大信息熵。

[7]熵的定义公式中，符号形式较为灵活，可以依据上下文进行判断。此外，如果随机变量为连续型随机变量，那么也可以给出相应的熵定义——使用积分替换求和。此处，对此不进行讨论。

定理 2.1 对于上述取有限个值的离散随机变量 X，熵 $H(\boldsymbol{p})$ 满足下式：

$$0 \leqslant H(\boldsymbol{p}) \leqslant \log n \tag{7}$$

证明：(方法一)

首先，定义拉格朗日函数：

$$L(\boldsymbol{p}, \lambda) = -\sum_{i=1}^{n} p_i \log p_i + \lambda \left(\sum_{i=1}^{n} p_i - 1 \right) \tag{8}$$

求解偏导数 $\frac{\partial L(\boldsymbol{p},\lambda)}{\partial p_i}$，并令其为 0：

$$\frac{\partial L(\boldsymbol{p}, \lambda)}{\partial p_i} = -\log p_i^* - 1 + \lambda = 0 \quad \Rightarrow \quad \lambda = \log p_i^* + 1 \tag{9}$$

上式两端同时乘以 p_i^* 并按 i 求和，利用公式 $\sum\limits_{i=1}^{n} p_i^* = 1$，得到：

$$\begin{aligned} &\lambda = \log p_i^* + 1 \quad \Rightarrow \quad \lambda p_i^* = p_i^* \log p_i^* + p_i^* \\ &\sum_{i=1}^{n} \lambda p_i^* = \sum_{i=1}^{n} (p_i^* \log p_i^* + p_i^*) \quad \Rightarrow \quad \lambda = \sum_{i=1}^{n} p_i^* \log p_i^* + 1 \end{aligned} \tag{10}$$

根据公式 (9) 和公式 (10)，得到：

$$\begin{aligned} &\begin{cases} \lambda = \log p_i^* + 1 \\ \lambda = \sum\limits_{i=1}^{n} p_i^* \log p_i^* + 1 \end{cases} \quad \Rightarrow \quad \log p_i^* = \sum_{i=1}^{n} p_i^* \log p_i^* \quad \Rightarrow \\ &\qquad \log p_1^* = \log p_2^* = \ldots = \log p_n^* \quad \Rightarrow \\ &\qquad p_1^* = p_2^* = \ldots = p_n^* = \frac{1}{n} \quad \Rightarrow \\ &\qquad H^*(\boldsymbol{p}^*) = -\sum_{i=1}^{n} p_i^* \log p_i^* = -\log \frac{1}{n} = \log n \end{aligned} \tag{11}$$

这表明下式成立：

$$H(\boldsymbol{p}) \leqslant H^*(\boldsymbol{p}^*) = \log n \tag{12}$$

而 $0 \leqslant H(\boldsymbol{p})$ 直接来源于概率性质 $(p_i \geqslant 0)$ 以及自信息的性质 $(-\log p_i \geqslant 0)$：

$$H(\boldsymbol{p}) = -\sum_{i=1}^{n} p_i \log p_i = \sum_{i=1}^{n} p_i \left(-\log p_i\right) \geqslant 0 \tag{13}$$

□

证明：(方法二)

利用 Jensen 不等式，得到：

$$H(\boldsymbol{p}) = -\sum_{i=1}^{n} p_i \log p_i = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \leqslant \log \sum_{i=1}^{n} \left(p_i \frac{1}{p_i} \right) = \log n \tag{14}$$

至于 $H(\boldsymbol{p}) \geqslant 0$ 的证明，与方法一相同。 □

证明：(方法三)

对于上述取有限个值的离散随机变量 X，假设其概率分布为 $\boldsymbol{p}$，并假设等概率分布为 $\boldsymbol{q}$，即 $q_i = \frac{1}{n}(i = 1, 2, \ldots, n)$。根据 KL 散度 (Kullback-Leibler Divergence)，亦称相对熵 (Relative Entropy) 或信息散度 (Information Divergence) 的定义，有：

$$\begin{aligned} & KL(\boldsymbol{p} \| \boldsymbol{q}) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i} \geqslant 0 \quad \Rightarrow \\ & \sum_{i=1}^{n} p_i \log p_i - \sum_{i=1}^{n} p_i \log q_i \geqslant 0 \quad \Rightarrow \\ & \sum_{i=1}^{n} p_i \log p_i - \sum_{i=1}^{n} p_i \log \frac{1}{n} \geqslant 0 \quad \Rightarrow \\ & \log n \geqslant -\sum_{i=1}^{n} p_i \log p_i = H(\boldsymbol{p}) \end{aligned} \tag{15}$$

至于 $H(\boldsymbol{p}) \geqslant 0$ 的证明，与方法一相同。 □

图2-3展示了随机变量 X 取 2 个值 (伯努利分布) 时熵的曲线图 (对数底数为 2)。可以看出，当 $p = \frac{1}{2}$ 时，熵 $H(p) = 1$，取得最大值。此时，随机变量 X 的不确定性最大。

下面，来看一下关于熵及自信息的一个有趣讨论。在英文文章中，假设 26 个字母出现的次数相同，那么每个字母的自信息量为 (对数以 2 为底)：

$$I(p_i) = -\log \frac{1}{26} \approx 4.70$$

但是，实际上，每个字母在文章中出现的次数并不相同。因此，那些出现次数少的字母，携带相对高的信息量。

对于中文文章而言，假设常用的汉字有 4808 个，并假设它们出现的次数相同，那么每个汉字的自信息量为：

$$I(p_i) = -\log \frac{1}{4808} \approx 12.23 \tag{16}$$

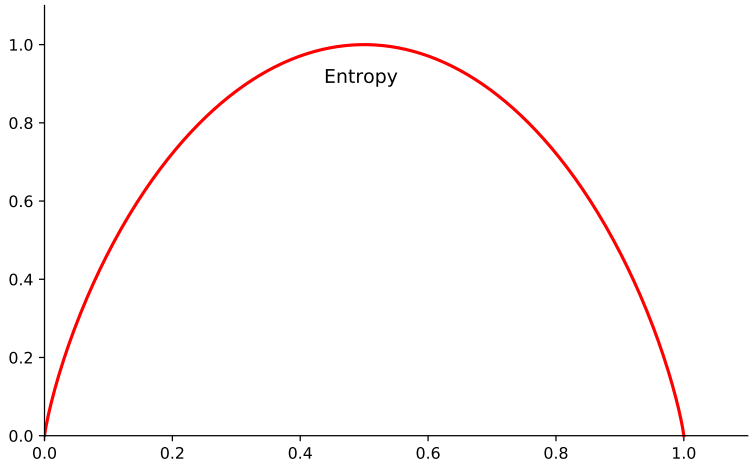


图 2-3: 二值随机变量 (伯努利分布) 的熵

同样，实际上，每个汉字在文章中出现的次数并不相同。因此，那些出现次数少的汉字，携带相对高的信息量。

从上面可以得出结论，一般而言，一个语言系统拥有的基本符号越多，其基本符号所携带的信息量就越大。

熵所衡量的是整个“系统”的平均消息量，即：

$$H(\boldsymbol{p}) = \sum_{i=1}^{n} p_i I(p_i) = -\sum_{i=1}^{n} p_i \log p_i \tag{17}$$

因此，一般而言，如果两篇文章拥有大致相同的消息量，例如使用英文与中文书写的同一篇文章，那么，由于每个汉字的信息量较大，中文文章使用的汉字就要比英文文章使用的字母要少，即中文文章比英文文章要短，即使一个汉字占用 2 个字符空间。

下面，给出熵的一些性质：

- 连续性

  熵具有连续性，即概率值的微小变化将引起熵的微小变化；

- 对称性或无序性

  $H(\boldsymbol{p})$ 的计算结果与 p_i 的求值顺序无关；

- 非负性与极值性，即满足不等式：

$$0 \leqslant H(\boldsymbol{p}) \leqslant \log n \tag{18}$$

- 扩展性

  增减概率为 0 的事件不改变熵：$H_{n+1}(p_1, p_2, \ldots, p_n, 0) = H_n(p_1, p_2, \ldots, p_n)$；或者小概率事件对熵的影响很小，可以忽略；

- 可加性

  如果事件 Z 表示两个独立事件 X 和 Y 同时发生，那么事件 Z 发生后产生的熵就等于事件 X 与事件 Y 分别发生后所产生的熵的总和，即：

$$H\left(P\left(Z\right)\right) = H\left(P\left(X,Y\right)\right) = H\left(P\left(X\right)\right) + H\left(P\left(Y\right)\right) \tag{19}$$

  或者，简记为：

$$H\left(Z\right) = H\left(X,Y\right) = H\left(X\right) + H\left(Y\right) \tag{20}$$

  利用事件 X 与 Y 之间的独立性 $(P(Z) = P(X,Y) = P(X) \cdot P(Y))$ 以及自信息的性质，有：

$$\begin{aligned}
&I(Z=z) = I(X=x, Y=y) = I(X=x) + I(Y=y) \quad \Rightarrow \\
&\mathbb{E}_{P(z)}\left[I(Z=z)\right] = \mathbb{E}_{P(x,y)}\left[I(X=x, Y=y)\right] = \mathbb{E}_{P(x,y)}\left[I(X=x) + I(Y=y)\right] \quad \Rightarrow \\
&\mathbb{E}_{P(z)}\left[I(Z=z)\right] = \mathbb{E}_{P(x,y)}\left[I(X=x, Y=y)\right] = \mathbb{E}_{P(x)}\left[I(X=x)\right] + \mathbb{E}_{P(y)}\left[I(Y=y)\right] \quad \Rightarrow \\
&H(P(Z)) = H\left(P\left(X,Y\right)\right) = H\left(P\left(X\right)\right) + H\left(P\left(Y\right)\right)
\end{aligned} \tag{21}$$

  如果 X 与 Y 之间不是独立的，那么利用贝叶斯公式以及自信息的性质，有：

$$\begin{aligned}
&P(X,Y) = P(Y|X)P(X) \quad \Rightarrow \\
&I(X=x, Y=y) = I(Y=y|X=x) + I(X=x) \quad \Rightarrow \\
&\mathbb{E}_{P(x,y)}\left[I(X=x, Y=y)\right] = \mathbb{E}_{P(x,y)}\left[I(Y=y|X=x) + I(X=x)\right] \quad \Rightarrow \\
&H\left(P(X,Y)\right) = H\left(P(Y|X)\right) + H\left(P(X)\right)
\end{aligned} \tag{22}$$

  或，简记为：

$$H(X,Y) = H(Y|X) + H(X) \tag{23}$$

实际上，Shannon 从数学上严格证明了，满足上述几个条件的不确定性度量函数具有唯一形式，即熵的计算公式 $H(X) = -\sum_{i=1}^{n} p_i \log p_i$。

2.2.3 条件熵

首先给出条件熵 (Conditional Entropy) 的定义。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：

$$\begin{aligned} H(Y|X) &= \sum_x P(x) H(Y|X=x) \\ &= -\sum_x P(x) \sum_y P(y|x) \log P(y|x) \\ &= -\sum_x \sum_y P(x,y) \log P(y|x) \\ &= -\sum_{x,y} P(x,y) \log P(y|x) \end{aligned} \tag{24}$$

从上式可以看出，条件熵 $H(Y|X)$ 表示为给定 X 条件下 Y 的条件熵 $H(Y|X=x)$ 对 X 的数学期望。

另一种等价性定义，可表示为条件自信息 $I(y|x)$ 关于 X 与 Y 联合分布的数学期望：

$$\begin{aligned} H(Y|X) &= \mathbb{E}_{P(x,y)} \left[I(y|x) \right] \\ &= -\sum_x \sum_y P(x,y) \log P(y|x) \\ &= \sum_x P(x) \left[-\sum_y P(y|x) \log P(y|x) \right] \\ &= \sum_x P(x) H(Y|x) \end{aligned} \tag{25}$$

其中，$H(Y|x) = H(Y|X=x)$ 表示 X 取某个特定值 x 时 Y 的条件熵。

实际上，条件熵还具有其它等价形式。例如，对公式 (24) 进行变形：

$$\begin{aligned} H(Y|X) &= -\sum_x \sum_y P(x,y) \log P(y|x) \\ &= -\sum_x \sum_y P(x,y) \log \frac{P(x,y)}{P(x)} \\ &= -\sum_x \sum_y P(x,y) \log P(x,y) + \sum_x \sum_y P(x,y) \log P(x) \\ &= H(X,Y) - H(X) \end{aligned} \tag{26}$$

或者，直接利用贝叶斯公式 $P(Y|X)=\frac{P(X,Y)}{P(X)}$ 以及 (条件) 自信息的性质，得到[8]：

$$\begin{aligned}
&I(Y=y|X=x)=I(X=x,Y=y)-I(X=x) \quad \Rightarrow \\
&\mathbb{E}_{P(x,y)}\left[I(Y=y|X=x)\right]=\mathbb{E}_{P(x,y)}\left[I(X=x,Y=y)-I(X=x)\right] \quad \Rightarrow \\
&\mathbb{E}_{P(x,y)}\left[I(Y=y|X=x)\right]=\mathbb{E}_{P(x,y)}\left[I(X=x,Y=y)\right]-\mathbb{E}_{P(x,y)}\left[I(X=x)\right] \quad \Rightarrow \\
&H\left(P\left(Y|X\right)\right)=H\left(P\left(X,Y\right)\right)-H\left(P\left(X\right)\right)
\end{aligned} \tag{27}$$

或者，简记为：

$$H\left(Y|X\right)=H\left(X,Y\right)-H\left(X\right) \tag{28}$$

2.2.4 联合熵

实际上，在推导条件熵 (公式 (26)、(27)) 的过程中，已经使用了联合熵 (Joint Entropy)。联合熵被定义为：

$$\begin{aligned}
H(X,Y)&=-\sum_x\sum_y P(x,y)\log P(x,y)\\
&=-\sum_x\sum_y P(x,y)\log\left(P(y|x)P(x)\right)\\
&=-\sum_x\sum_y P(x,y)\log P(y|x)-\sum_x\sum_y P(x,y)\log P(x)\\
&=H(Y|X)+H(X)
\end{aligned} \tag{29}$$

也可以直接利用 (条件) 自信息的性质，并使用类似于条件熵的推导方式 (公式 (27))，得到上式；或者，利用上述关于条件熵的结论，直接得到上式。同理可得：

$$H(X,Y)=H(X|Y)+H(Y) \tag{30}$$

于是，得到：

$$H(X,Y)=H(Y|X)+H(X)=H(X|Y)+H(Y) \tag{31}$$

进一步变形，得到[9]：

$$H(X)-H(X|Y)=H(Y)-H(Y|X) \tag{32}$$

[8]根据自信息的定义，对公式 $P(Y=y|X=x)=\frac{P(X=x,Y=y)}{P(X=x)}$ 两端应用 $-\log$，得到 $I(Y=y|X=x)=I(X=x,Y=y)-I(X=x)$。

[9]实际上，我们得到了互信息，参见下文。在决策树中，训练数据集 D 与特征 A 的互信息 $H(D)-H(D|A)$，也被称为信息增益 $\text{gain}(D,A)$。它是决策树生成过程中用于特征选择的主要依据之一。

如果 X 与 Y 相互独立，那么根据熵的性质，可以得到：

$$\begin{cases} H(X,Y) = H(X) + H(Y) \\ H(X,Y) = H(Y|X) + H(X) = H(X|Y) + H(Y) \end{cases} \Rightarrow \begin{cases} H(Y|X) = H(Y) \\ H(X|Y) = H(X) \end{cases} \tag{33}$$

上式表明，如果 X 与 Y 相互独立，那么在已知 X(或 Y) 的情况下，不影响我们对 Y(或 X) 的认知。

如果 X 与 Y 不是相互独立的，那么[10]：

$$\begin{aligned} H(Y|X) < H(Y) \\ H(X|Y) < H(X) \end{aligned} \tag{34}$$

它们表达的含义是，在已知 X(或 Y) 的情况下，Y(或 X) 的不确定性将会降低。因此，得到：

$$H(X,Y) \leqslant H(X) + H(Y) \tag{35}$$

当且仅当 X 与 Y 相互独立时，等式成立。上式的含义是，两个事件同时发生后的熵，不大于每个事件单独发生后的熵之和。

现在，假设 $Y = f(X)$ 且 f 为确定性函数 (X 为随机变量)，则 $H\left(f\left(X\right)|X\right) = 0$[11]。于是：

$$\begin{aligned} &H(X) + H\left(f\left(X\right)|X\right) = H\left(f(X)\right) + H\left(X|f(X)\right) \quad \Rightarrow \\ &H(X) = H\left(f(X)\right) + H\left(X|f(X)\right) \quad \Rightarrow \\ &H(X) \geqslant H\left(f(X)\right) \end{aligned} \tag{36}$$

上式表明，当随机变量 X 通过确定性函数 $y = f(X)$ 传递后，熵将会降低。

2.2.5 互信息

互信息 (Mutual Information) 或转移信息 (Transinformation) 度量随机变量之间的相互依赖性。它与相关系数不同，互信息并不局限于实值随机变量——它更加具有一般性，用于度量联合分布 $P(X,Y)$ 与边缘分布乘积 $P(X)P(Y)$ 之间的相

[10] 利用下文介绍的互信息的非负性，可以正式地证明这 2 个不等式。

[11] 它表达的含义是，在随机变量 X 已知的情况下，$Y = f(X)$ 是确定的，即 $Y = f(X)$ 的熵或不确定性为 0。

似程度[12]：

$$\begin{aligned} I(X;Y) &= D_{KL}\left(P(X,Y)\|P(X)P(Y)\right) = \sum_x \sum_y P(x,y)\log\frac{P(x,y)}{P(x)P(y)} \\ &= \sum_x \sum_y P(x,y)\log\frac{P(y|x)}{P(y)} \\ &= \sum_x \sum_y P(x,y)\log P(y|x) - \sum_x \sum_y P(x,y)\log P(y) \\ &= H(Y) - H(Y|X) \end{aligned} \tag{37}$$

同理可得：

$$I(X;Y) = H(X) - H(X|Y) \tag{38}$$

于是，得到：

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{39}$$

上式表达的含义是，在已知 Y(或 X) 的情况下，X(或 Y) 的不确定性下降了多少，或者，变量 X(或 Y) 的不确定性，有多少与 Y(或 X) 有关。直观地理解，互信息越小，两个随机变量的相互依赖性就越弱；否则，互信息越高，它们之间的依赖性越强[13]。

互信息具有如下性质：

1. 对称性

   即 $I(X;Y) = I(Y;X)$，前面已经证明了这一点。

2. 非负性

   即 $I(X;Y) \geqslant 0$。

下面证明 $I(X;Y) \geqslant 0$。

[12]因此，从 KL 散度 (参见下文) 的角度看，$I(X;Y) = D_{KL}\left(P(X,Y)\|P(X)P(Y)\right)$，即联合分布 $P(X,Y)$ 与边缘分布乘积 $P(X)P(Y)$ 之间的 KL 散度。

[13]2 个极端的情况是：X 与 Y 相互独立时，$I(X;Y) = 0$；$I(X;X) = H(X)$，参见下文。

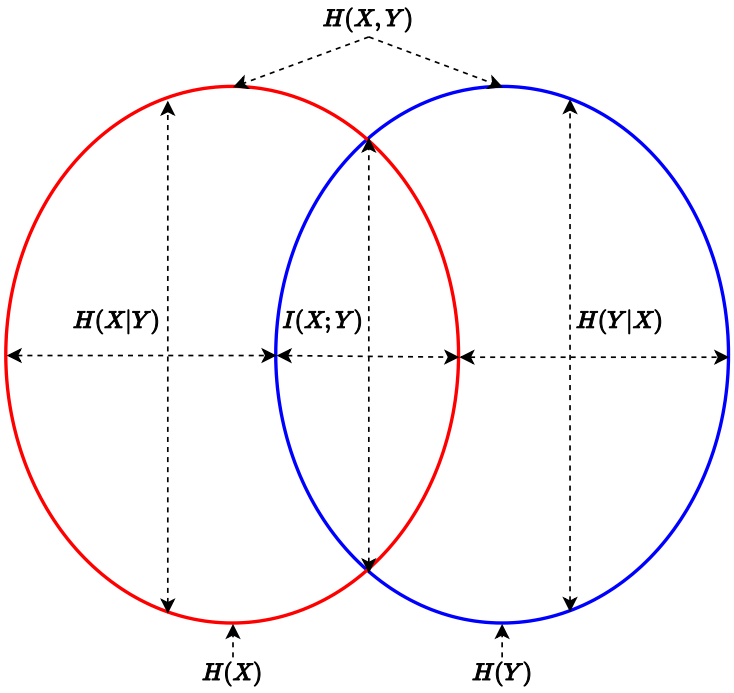


图 2-4: 几种熵之间的 Venn Diagram

证明：根据互信息的定义，可得：

$$\begin{aligned} I(X;Y) &= \sum_x \sum_y P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \\ &= -\sum_x \sum_y P(x,y) \log \frac{P(x)P(y)}{P(x,y)} \\ &\geqslant -\log \sum_x \sum_y \left(P(x,y) \frac{P(x)P(y)}{P(x,y)} \right) = 0 \end{aligned} \tag{40}$$

对于上式，当且仅当 X 与 Y 相互独立时，等式成立。 □

利用联合熵、条件熵与 (边缘) 熵之间的关系，即：

$$H(X,Y) = H(Y) + H(X|Y) \tag{41}$$

可以得到：

$$\begin{aligned} I(X;Y) &= H(X) + H(Y) - H(X,Y) \\ &= H(X,Y) - H(X|Y) - H(Y|X) \end{aligned} \tag{42}$$

图2-4，使用 Venn Diagram 展示了几种熵：$H(X,Y)$、$H(X)$、$H(Y)$、$H(X|Y)$、$H(Y|X)$ 与 $I(X;Y)$ 之间的关系。

如果 X 与 Y 相互独立，那么：

$$I(X;Y) = H(X) - H(X|Y) = H(X) - H(X) = 0 \tag{43}$$

上式表明，Y 对 X 没有提供任何“有价值”的信息。

对于 $I(X;X)$，有：

$$I(X;X) = H(X) - H(X|X) = H(X) \tag{44}$$

其中，$H(X|X)$ 表示在 X 已知的情况下，X 的不确定性，它当然为 0。因此，(信息) 熵是一种特殊的互信息，被称为自信息 (Self-Information)[14]。于是，再利用熵与互信息的非负性，可得：

$$I(X;Y) = H(X) - H(X|Y) = I(X;X) - H(X|Y) \geqslant 0 \quad \Rightarrow \quad I(X;X) \geqslant I(X;Y) \tag{45}$$

上式可以解释为，一个随机变量自身提供的信息至少包含其它任何变量可以提供的跟它有关的信息。

现在，假设 $Y = f(X)$ 是 X 的确定性函数，且 $X = f^{-1}(Y)$ 也是 Y 的确定性函数，那么类似于公式 (36) 情形，有 $H\left(f(X)|X\right) = 0$ 和 $H\left(f^{-1}(Y)|Y\right) = 0$，则

$$\begin{aligned}
&H(X) + H\left(f\left(X\right)|X\right) = H\left(f(X)\right) + H\left(X|f(X)\right) \quad \Rightarrow \\
&I(X;Y) = H(X) - H\left(X|f(X)\right) = H\left(f(X)\right) - H\left(f(X)|X\right) \quad \Rightarrow \\
&I(X;Y) = H(X) - H\left(f^{-1}(Y)|Y\right) = H\left(f(X)\right) - H\left(f(X)|X\right) \quad \Rightarrow \\
&I(X;Y) = H(X) = H\left(Y\right)
\end{aligned} \tag{46}$$

上式表明，在此情形下，互信息与 Y 或 X 单独包含的不确定性相同，即互信息等于 X 的熵或 Y 的熵。显然，前述的 $I(X;X) = H(X)$ 是其特殊情形。

互信息的一个应用例子是，在决策树的生成过程中，利用 $I(D;A) = H(D) - H(D|A)$ 进行重要特征的选择，其中 D 表示训练数据集，A 表示输入 $\boldsymbol{x}$ 的特征，此时的 $I(D;A)$ 被称为信息增益，记作 $\text{gain}(D,A)$ 或 $g(D,A)$。一般而言，信息增益越大，特征越重要。特征选择的任务就是，计算每个特征的信息增益，然后选择信息增益最大的特征作为当前节点的划分特征，并将当前训练数据集划分成子训练数据集。

[14]这样看来，将熵定义为自信息的期望，也许要表达这样的含义。

在前面给出互信息定义的时候，已经指出，它实际上是一种 KL 散度。下面，继续对其进行变形：

$$\begin{aligned} I(X;Y) &= D_{KL}\left(P(X,Y)\|P(X)P(Y)\right) \\ &= \sum_x \sum_y P(x,y)\log\frac{P(x,y)}{P(x)P(y)} \\ &= \sum_x \sum_y P(y)P(x|y)\log\frac{P(x|y)}{P(x)} \\ &= \sum_y P(y) D_{KL}\left(P(x|y)\|P(x)\right) \\ &= \mathbb{E}_{P(y)}\left[D_{KL}\left(P(x|y)\|P(x)\right)\right] \end{aligned} \tag{47}$$

上式表明，最内层的 KL 散度仅对 X 求和，其结果将以 Y 为变量。分布 $P(x|y)$ 与 $P(x)$ 之间的平均差异越大，互信息 (或信息增益) 将越大。

2.2.6 相对熵

相对熵 (Relative Entropy)，又称 KL 散度 (Kullback-Leibler Divergence) 或信息散度 (Information Divergence)，用于度量 2 个概率分布之间的差异：

$$\begin{aligned} D_{KL}(P\|Q) &= \sum_x P(x)\log\frac{P(x)}{Q(x)} \\ &= -\sum_x P(x)\log\frac{Q(x)}{P(x)} \\ &\geqslant -\log\left(\sum_x P(x)\frac{Q(x)}{P(x)}\right) = 0 \end{aligned} \tag{48}$$

最后一行的推导，利用了 Jensen 不等式。上式表明，相对熵具有非负性 $D_{KL}(P\|Q) \geqslant 0$；当且仅当 $P(x) = Q(x)$ 时，等式成立。继续对相对熵变形：

$$\begin{aligned} D_{KL}(P\|Q) &= \sum_x P(x)\log\frac{P(x)}{Q(x)} \\ &= \sum_x P(x)\log P(x) - \sum_x P(x)\log Q(x) \\ &= H(P,Q) - H(P) \end{aligned} \tag{49}$$

上式表明，相对熵等于概率分布 P 与 Q 的交叉熵与 P 熵之差[15]。在机器学习中，训练数据的分布 P 是固定的，那么最小化相对熵 $D_{KL}(P\|Q)$，等价于最小化交叉

[15]注意，$H(P,Q)$ 表示交叉熵，其中 P 与 Q 是概率分布。不要同 $H(X,Y)$ 联合熵相混淆，其中 X 与 Y 是随机变量。

熵 $H(P,Q)$，也等价于给定训练数据集时的最大似然估计。因此，在分类算法中，根据需要，可以使用交叉熵作为损失函数，以优化参数。

相对熵的性质如下：

- 非负性；
- 非对称性

  即 $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$，因而 KL 散度不是一个度量 (Metric)[16]；

2.2.7 交叉熵

设 $P(x)$ 和 $Q(x)$ 表示随机变量 X 的两个概率分布，其中 $P(x)$ 为训练数据集的真实分布，$Q(x)$ 为预测得到的非真实分布，那么交叉熵定义为：

$$H(P,Q) = -\sum_{x} P(x)\log Q(x) \tag{50}$$

在逻辑回归中，使用的损失函数就是交叉熵，也被称为负对数似然函数：

$$L(\boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log\sigma(\boldsymbol{x}_i) + (1-y_i)\log(1-\sigma(\boldsymbol{x}_i))\right) \tag{51}$$

其中，y_i 表示第 i 个样本的真实输出 (取值为 1 或 0)，$\sigma(\boldsymbol{x}_i)$ 表示第 i 个样本的实际输出，$L(\boldsymbol{w})$ 是凸函数，可以得到全局最优解。

作为代价函数或损失函数，对于给定的实际训练数据集 (提供了 y 标签数据，即“真实”分布)，交叉熵给出了执行非真实分布 Q 所对应的策略时，为消除系统的不确定性，所付出的努力。

2.3 特征选择

假设在给定的数据集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$ 中，每个输入样本点 $\boldsymbol{x}_i$ 有 n 个特征分量，现在需要选择若干特征对该数据集进行分类。

首先，需要明确的是，在 n 个特征中，有一些特征对当前的分类任务而言是主要和重要的特征，而另一些特征对当前的分类任务而言可能是无用的或无关紧要的。

[16]度量应满足四个基本性质：非负性、同一性、对称性与直递性，参见文献 [2]。

一个典型的例子是，银行的信用卡防欺诈系统，需要准确地甄别出每个申请信用卡客户的信用状况，并给出分类建议：申请通过与不通过。对此问题，可以构建一个训练数据集，其中的每个数据样本包括 $\boldsymbol{x}$ 和相应的分类标签 y(例如，使用 1 表示申请通过，-1 表示申请不通过)。在 $\boldsymbol{x}$ 中，可以列出每个客户的一些相关信息，例如姓名、身份证号、年龄、职业、信贷情况、住房情况、收入、家庭住址等。

而在上述若干信息 (特征) 中，尽管客户的姓名与身份证号对识别客户的唯一性而言是关键信息，但是对于当前的分类决策而言，并不是主要和重要的特征。如果选择这样的特征对客户进行分类，那么可以预料的是，得到的分类结果与随机分类的结果没有多大的差别，我们称这种特征没有分类能力。相反，类似于职业、信贷情况、住房情况、收入等这样的特征，对当前的分类决策任务而言就是主要和重要的特征。我们应该选择这种具有很强分类能力的特征，用于构建分类决策系统。当然，特征的重要性程度与分类任务的性质及目标有关，不具有绝对性。

那么，如何选择一个分类能力强的特征进行分类呢？假设对于给定的数据集 D，初始时，它的正负样本比例为 $1:1$。这意味着，此时的数据集具有最高的类别混乱程度——正负样本实例全部混合在一起，且数量一样多。现在，假设找到了一个特征分量 x_j，使得正负样本实例刚好完全分开[17]。换句话说，在选择了特征 x_j 之后，数据集的混乱程度由最高值一下降低到了最低值 0。于是，我们认为特征分量 x_j 具有“超强”的分类能力。它应该就是我们在分类时需要选择的重要特征。

在上述讨论中，我们已经意识到了数据集的混乱程度以及混乱程度变化的重要性，它们就是我们进行特征选择的依据。而信息论中的熵与信息增益 (或互信息) 就能够很好地对上述依据进行量化。

虽然熵与互信息的概念在前面已经进行了讨论，但是针对特定的问题，仍然有必要给出专门的定义。

定义 **2.2**（信息增益）特征 A 关于训练数据集 D 的信息增益(*Information Gain*)或互信息指的是，对于给定的训练数据集 D 与特征 A，D 的经验熵 $H(D)$ 与给定特征 A 时 D 的经验条件熵 $H(D|A)$ 的差，即：

$$I(D;A) = g(D,A) = H(D) - H(D|A) \tag{52}$$

从上述定义可以看出，决策树中的信息增益等价于训练数据集与特征的互信息。经验熵 $H(D)$ 表示对数据集 D 进行分类的不确定性，而 $H(D|A)$ 表示依据给

[17]这当然是一种最理想的分类情形了。

定的特征 A 对数据集 D 进行分类的不确定性。显然，根据互信息的非负性，可知 $H(D) \geqslant H(D|A)$，或者，从定性的角度看，在已知特征 A 时，$H(D|A)$ 的不确定性肯定不比特征 A 未知时的 $H(D)$ 高。因此，如果 $H(D) - H(D|A)$ 值大，即信息增益大，则表明特征 A 让数据集 D 的分类不确定性大大降低，因而表明特征 A 对数据集 D 的分类能力强；反之，则弱。

综上所述，根据信息增益进行特征选择的方法为，对数据集 D 及其子集，计算每个特征的信息增益，并比较它们的大小，算法将选择信息增益最大的特征作为划分特征；该过程一直持续下去——针对划分之后的新子集——迭代往复，直至满足划分要求为止。

下面给出计算信息增益的算法。

算法 **2.1**（信息增益的计算）

Input:

Train Dataset: $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Feature: A

Numbers of Class: K

Output:

Information Gain: $g(D, A)$

Algorithm:

1. $H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log \frac{|C_k|}{|D|}$, $|C_k|$: Numbers of Class C_k in D, $|D|$: Numbers of D's Samples
2. $H(D|A) = \sum_{v=1}^{V} \frac{|D_v|}{|D|} H(D_v) = -\sum_{v=1}^{V} \frac{|D_v|}{|D|} \sum_{k=1}^{K} \frac{|C_k|}{|D_v|} \log \frac{|C_k|}{|D_v|}$, $|C_k|$: Numbers of Class C_k in D_v, V: Numbers of feature A's value, D_v: Samples with same feature A's value
3. $g(D, A) = H(D) - H(D|A)$

上述基于信息增益的特征选择方法，具有相对的绝对性——偏向于选取那些取值个数较多的特征。使用比值，即信息增益比 (Information Gain Ratio) 可以对此问题进行一定程度的校正——消除取值个数的影响。于是，得到另一个选取特征的准则。

定义 **2.3**（信息增益比）对于给定的训练数据集 D 与特征 A，特征 A 关于 D 的信息增益比 $g_r(D,A)$ 被定义为：

$$g_r(D,A) = \frac{g(D,A)}{H_A(D)} \tag{53}$$

其中：

$$H_A(D) = -\sum_{v=1}^{V} \frac{|D_v|}{|D|} \log \frac{|D_v|}{|D|} \tag{54}$$

其中，V 表示特征 A 的取值个数；$|D_v|$ 表示数据集 D 中该特征分量具有某个相同值的子数据集的样本点个数。

2.4 决策树的生成：ID3 与 C4.5 算法

下面，介绍一种决策树的生成算法：ID3 算法。该算法的核心操作是，使用信息增益作为特征选取的准则。首先，从整个数据集 D 开始，依据最大信息增益准则，选取最佳划分特征作为当前节点的特征；然后，依据该特征的取值个数，将数据集划分成不同的新子数据集，并递归地应用上述操作，从而构建出一棵分类决策树。上述过程，一直持续下去，直至剩余特征的信息增益低于阈值或没有特征可以选择为止。

下面，直接给出 ID3 算法。

算法 **2.2**（*ID3* 算法）

Input:

Train Dataset: $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Feature Set: A

Threshold of Information Gain: ϵ

Output:

Decision Tree for Classification: T

Algorithm: $ID3(D, A)$

1. Create a *Root* node
2. If all samples are belong to C_k, return *Root* with label=C_k
3. if $A = \emptyset$, return *Root* with label=$\arg\max_{C_k} |C_k|$
4. Otherwise:
5. $A_g = \arg\max_{A_i} H(D) - H(D|A_i)$, $i = 1, 2, \ldots, n$, n: Numbers of Feature in A
6. if $g(D, A_g) < \epsilon$, return *Root* with label=$\arg\max_{C_k} |C_k|$
7. for v=1 to V , index of A_g's value:
8. $Subtree = ID3(D_v, A - \{A_g\})$, recursively call $ID3(\ldots)$
9. Add *Subtree* to *Root*
10. return *Root*

C4.5 算法与 ID3 算法类似，只是在使用信息增益的地方，使用信息增益比替换即可，此处不再赘述。

2.5 剪枝算法

在决策树的递归生成过程中，由于每次选取信息增益或信息增益比最大的特征作为划分特征，而信息增益或信息增益比具有某种贪心性质[18]，且该递归过程一直进行到不能继续下去为止——这样造成的结果是，算法可能会一味地追求对训练数据集的拟合精度，使得生成的分类决策树过于复杂，这将会导致“过拟合”现象。在过拟合发生后，分类决策树对训练数据集的分类精度高，但是对于未见数据的分类精度较低，泛化能力较差，分类器缺乏实用性。

为了解决上述问题，可以采取两种策略：

- 预剪枝 (Prepruning)

  预剪枝指的是，在决策树的生成过程中，对每个节点在划分前，先进行估计：如果当前节点的划分，不能带来泛化性能的提升，则停止划分，并将当前节点标记为叶节点。

- 后剪枝 (Postpruning)

  后剪枝指的是，先从训练数据集生成一棵完整的决策树，然后，以自底向上的方式，考察每一个非叶子节点，如果对它进行剪枝，能够带来泛化性能的提升，则执行剪枝。

无论哪种策略，都是为了在训练精度与泛化性能之间取得某种平衡，或者，在决策树的复杂度与训练数据集的拟合精度之间取得某种平衡。具体而言，需要借助于带正则化的损失函数或具有全局优化性质的损失函数，来实现子树或叶子节点的裁剪功能。

一般而言，预剪枝使得决策树的一些分支不被扩展，因而降低了过拟合的风险，还显著地减少了决策树的训练时间开销。但是，这种策略也是有代价的——有些分支的当前划分虽然不能提升泛化性能，但是后续划分却有可能使得泛化性能得到显著的提升，即预剪枝没有考虑剪枝的整体优化性能与延迟优化性能，仅追求当前的划分优化性能。因此，预剪枝的这种“短见”显示了其“贪心”的一面，具有欠拟合风险。

而后剪枝决策树通常会比预剪枝决策树保留更多的分支。一般情形下，后剪枝所造成的欠拟合风险很小，泛化性能往往优于预剪枝。但是，后剪枝在决策树

[18]换句话说，其对应的损失函数不具有全局最优化性质，或者缺乏类似于正则化那样的平衡机制，导致优化结果只具有局部最优性。

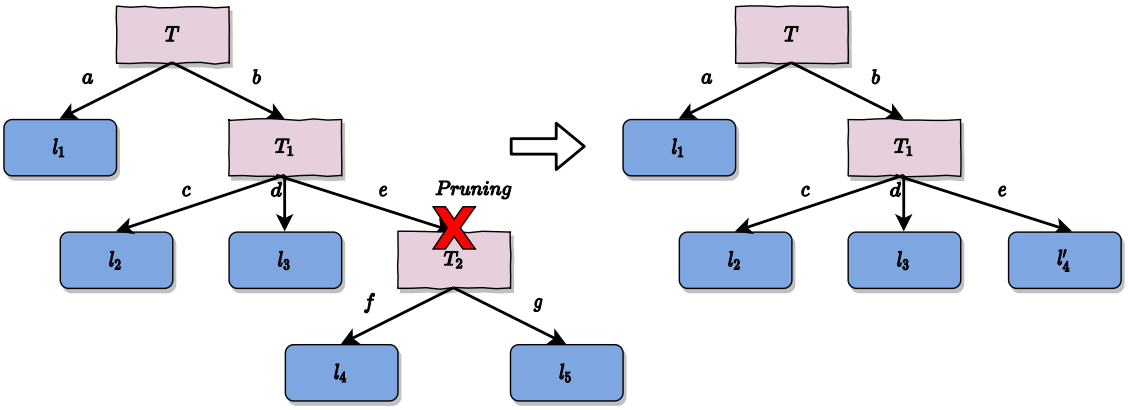


图 2-5: 决策树的后剪枝

生成之后进行，需要以自底向上的方式逐一地考察所有非叶子节点，因而其训练时间的开销要比预剪枝过程大得多。

下面，讨论一下后剪枝的基本思路。后剪枝的基本想法是，针对某个非叶子节点，计算剪枝前后的某种性能度量 (例如损失函数)，如果剪枝后的损失函数值要小于剪枝前的损失函数值，那么该节点的剪枝是可行的。

如图2-5所示，展示了决策树的后剪枝操作示意图。其中，不规则矩形表示非叶子节点，圆角矩形表示叶子节点。由于新决策树的整体损失函数值比原决策树的整体损失函数值要小，因而原决策树中的子树 T_2 被剪枝，成为新决策树中的叶子节点 l'_4。原子树 T_2 中的所有样本点集中于新叶子节点 l'_4 中。

总体上来说，后剪枝通过极小化决策树的整体损失函数来实现。假设树 T 的叶子节点个数为 L，l_i 表示树 T 的叶子节点，该叶子节点有 N_i 个样本点，其中属于 k 类别的样本点有 N_{ik} 个 (其中 $k = 1, 2, \ldots, K$)，$H(l_i)$ 为叶子节点 l_i 的经验熵，$\alpha \geqslant 0$ 为正则项权重，则决策树的整体损失函数可以表示为：

$$L(T) = \sum_{i=1}^{L} N_i H(l_i) + \alpha L \tag{55}$$

其中：

$$H(l_i) = -\sum_{k=1}^{K} \frac{N_{ik}}{N_i} \log \frac{N_{ik}}{N_i} \tag{56}$$

上式表明，决策树的整体损失有 2 项：所有叶子节点的总熵、叶子节点的加权总个数。前者，表示决策树对训练数据集的拟合精度，值越小，叶子节点中样本点的

类别越“纯粹”，拟合精度越高；同时，这也意味着需要在决策树中添加更多的叶子节点，才能达到这种效果。后者，表示决策树模型的复杂度，权重系数 α 一定时，L 值越大，叶子节点越多，模型越复杂。权重系数 α 用于控制拟合精度与模型复杂度的重要性程度。当 α 较大时，对决策树模型的复杂度有较大的惩罚，从而促使算法选择较简单的决策树；当 α 较小时，对决策树模型的复杂度有较小的惩罚，从而促使算法选择较复杂的决策树。一个极端情况是，当 $\alpha = 0$ 时，算法只考虑训练数据的拟合精度，生成的决策树模型具有最高的复杂度。

由此可以看出，决策树的后剪枝算法，不仅考虑了训练数据集的拟合精度，也考虑了决策树的模型复杂度，属于全局优化算法。而决策树生成算法，只考虑了通过信息增益或信息增益比来实现对训练数据的更好拟合精度，属于局部优化算法。

另外，针对剪枝的局部性特点，计算时，使用局部损失函数替代整体损失函数，这不会影响到剪枝的效果。例如，在图2-5中，除去子树 T_2 及叶子节点 l'_4，决策树的其余部分均相同。因此，在算法实现时，每次不需要计算整体损失函数值，而只需要计算局部损失函数值，从而大大减少了计算开销。

3 分类回归树：CART 算法

CART(Classification and Regression Tree) 算法属于分类回归树，既可以用于分类，也可以用于回归，由 Breiman 等人于 1984 年提出。

对于回归树，CART 使用平方误差最小化准则拟合训练数据集；对于分类树，CART 使用基尼指数 (Gini Index) 最小化准则进行特征选择。两者均生成一棵二叉树。

3.1 回归树的生成

设训练数据集为 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$，其中 $\boldsymbol{x}_i \in \mathbb{R}^d$，$y_i$ 为连续型变量。现在考虑，如何使用二叉回归树拟合训练数据集。

由于只使用二叉树进行拟合，因而在每个节点处进行判断并选择分支时，只有 2 种可能。我们规定，左分支表示 $x^{(j)} \leqslant s$，右分支表示 $x^{(j)} > s$，其中 $x^{(j)}$ 表示输入 $\boldsymbol{x}$ 的第 j 个分量，s 表示其切分点 (Splitting Point) 或切分变量 (Splitting Variable)，表示一个具体的划分值。因此，在该节点处，当前的样本数据集被一分

为二，且形成了 2 个区域：

$$\begin{aligned} R_1(j,s) &= \{\boldsymbol{x}|x^{(j)} \leqslant s\} \\ R_2(j,s) &= \{\boldsymbol{x}|x^{(j)} > s\} \end{aligned} \tag{57}$$

对于一棵完整的二叉回归树而言，给定某个样本实例点 $\boldsymbol{x}_i$，从根节点开始，依据节点的特征分量 $x^{(j)}$ 的取值，进行分支选择，最终将形成一条路径并到达某个叶子节点。在这条路径上，有若干的切分节点，所有这些切分节点的 (左或右) 划分区域相交而形成最终的一个划分单元，该划分单元上的值就作为该样本点的输出值。

假设一棵二叉回归树已经生成，它将输入空间划分为 M 个单元 R_1、R_2、...、R_M，且每个单元 R_m 上的输出值为 c_m，那么二叉回归树模型可表示为函数：

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m I(\boldsymbol{x} \in R_m) \tag{58}$$

实际上，上述二叉回归树模型给出了一种二叉回归树的生成算法。如果将二叉回归树模型 (公式 (58)) 视作一个可以动态生长的二叉树，那么对于给定的训练数据集 D，可以使用平方误差损失函数表示当前的二叉回归树对于训练数据集 D 的预测误差。此时，我们的任务就是，计算当前二叉回归树的预测误差，如果其满足要求，则停止对输入空间[19]进行进一步的划分；否则，采取启发式方法，选择最优的特征分量 $x^{(j)}$ 及其切分点 s 对当前的数据集进行进一步的划分。

如图3-6所示，展示了一棵二叉回归树。图中，假设除 T_4 分支外的所有节点与分支均已经稳定且无数据样本点可以处理。但是，此时，假设在 T_4 节点处，聚集着一些训练样本点且整体二叉回归树的训练误差还没有达到要求。那么，就需要对 T_4 处的实例样本点所在的输入空间进行进一步的划分，其实质是对这些实例样本点进行划分。于是，依据某种最优准则，选取最优的特征分量-切分点对 $(x^{(j)}, s)$ ——图示选取了 $(x^{(2)}, s_2)$，对 T_4 所在的数据集进行划分，从而生成了 2 个新节点：T_5 和 l_7。后续节点的生成，可以做类似的分析。

在上述二叉回归树的生成算法中，有 2 个关键问题还没有解决：如何确定每个划分单元的最优输出值 c_m^*；如何确定最优特征分量-切分点对 $(x^{(j)*}, s^*)$。

对于第 1 个问题，设区域为 R_m，需要确定该区域上的最佳输出值 c_m^*。对于

[19]输入空间由当前所有的划分单元共同组成，每个叶子节点形成一个划分单元。

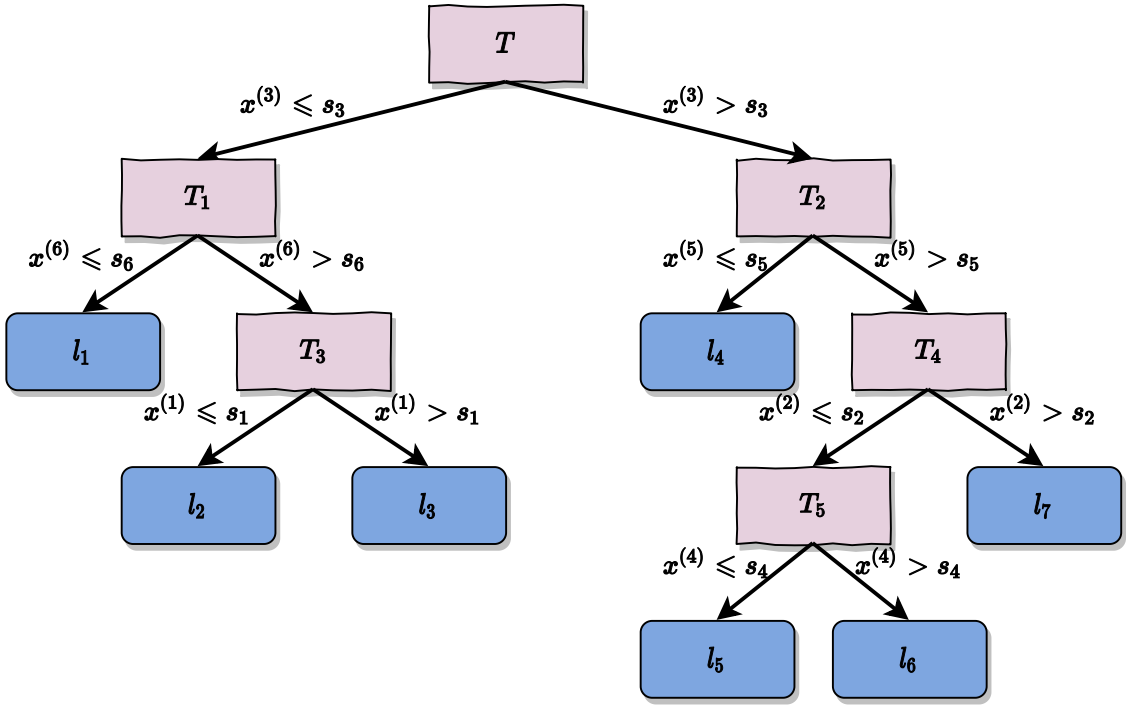


图 3-6: 一棵二叉回归树

拟合问题，可以采用平方误差损失最小化的原则得到 c_m^*：

$$c_m^* = \arg \min_{c_m, \boldsymbol{x}_i \in R_m} \frac{1}{2} \sum_{i=1}^{N'} (y_i - c_m)^2 \tag{59}$$

其中 N' 表示区域 R_m 上实例样本点的个数。令目标函数关于 c_m 的导数为 0，得到：

$$-\sum_{i=1}^{N'} (y_i - c_m^*) = 0 \quad \Rightarrow \quad c_m^* = \frac{1}{N'} \sum_{i=1}^{N'} y_i \tag{60}$$

上式意义非常明确，划分 (叶子) 单元 R_m 的最优输出值 c_m^*，就是该区域内所有样本实例点输出值 y 的均值。

对于第 2 个问题，可以遍历所有的特征分量，且针对每一个特征分量，遍历所有的粗 (或离散) 切分点，并执行如下的 2 次最优化过程，即：

$$(x^{(j)*}, s^*) = \arg \min_{x^{(j)}, s} \left[\min_{c_1} \sum_{x^{(j)} \leqslant s, \boldsymbol{x}_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x^{(j)} > s, \boldsymbol{x}_i \in R_2} (y_i - c_2)^2 \right] \tag{61}$$

其中，R_1 表示 $x^{(j)} \leqslant s$ 形成的区域，R_2 表示 $x^{(j)} > s$ 形成的区域。c_1 和 c_2 的最优值，已经在公式 (60) 中给出。

使用上述方法生成的二叉回归树，被称为最小二乘回归树 (Least Squares Regression Tree)。下面，给出最小二乘回归树生成算法。

算法 **3.1**（最小二乘回归树生成算法）

Input:

Train Dataset: $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Feature Set: A

Output:

Binary Regression Tree: T and $f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m^* I(\boldsymbol{x} \in R_m^*)$, $R_m^* \in T$

Algorithm: $LSRT(D, A)$

1. Create a *Root* node
2. $(x^{(j)*}, s^*) = \arg\min_{x^{(j)}, s} \left[\min_{c_1} \sum_{x^{(j)} \leqslant s, \boldsymbol{x}_i \in R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{x^{(j)} > s, \boldsymbol{x}_i \in R_2} (y_i - c_2)^2 \right]$
3. $j = 1, 2, \ldots, d$, d: Feature Numbers of $\boldsymbol{x}$
4. $c_1^* = \frac{1}{|R_1^*|} \sum_{\boldsymbol{x}_i \in R_1^*} y_i$
5. $c_2^* = \frac{1}{|R_2^*|} \sum_{\boldsymbol{x}_i \in R_2^*} y_i$
6. if R_1^* is not empty:
7. $Left = LSRT(\{\boldsymbol{x} | \boldsymbol{x} \in R_1^*\}, A - \{x^{(j)*}\})$
8. Add $Left$ to *Root*
9. if R_2^* is not empty:
10. $Right = LSRT(\{\boldsymbol{x} | \boldsymbol{x} \in R_2^*\}, A - \{x^{(j)*}\})$
11. Add $Right$ to *Root*
12. return *Root*

3.2 分类树的生成

3.2.1 基尼指数与特征选择

在 CART 分类树中，使用基尼指数 (Gini Index) 作为特征选取的准则。另外，由于 CART 为二叉树，因而还需要选取最优二值切分点。

下面给出基尼指数的定义。设 X 为取有限个值的离散随机变量，其概率分布为：

$$P(X=x_i)=p_i \quad i=1,2,\ldots,n \tag{62}$$

则随机变量 X 的基尼指数被定义为：

$$\begin{aligned} Gini(X)=Gini(\boldsymbol{p}) &= \mathbb{E}_{P(x)}\left[1-P(x)\right] \\ &= \sum_{i=1}^{n} p_i(1-p_i) \\ &= 1-\sum_{i=1}^{n} p_i^2 \end{aligned} \tag{63}$$

例如，在分类问题中，假设有 K 个类别，样本点 $\boldsymbol{x}$ 属于第 k 个类别的概率为 p_k，那么样本点 $\boldsymbol{x}$ 的基尼指数为：

$$Gini(\boldsymbol{x})=\sum_{k=1}^{K} p_k(1-p_k)=1-\sum_{k=1}^{K} p_k^2 \tag{64}$$

其特例是，如果 $K=2$，样本点 $\boldsymbol{x}$ 属于第 1 个类别的概率为 p，则其基尼指数为：

$$Gini(\boldsymbol{x})=2p(1-p) \tag{65}$$

对于给定的数据集 D，其基尼指数为：

$$Gini(D)=\sum_{k=1}^{K}\frac{|C_k|}{|D|}(1-\frac{|C_k|}{|D|})=1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2 \tag{66}$$

其中，C_k 表示类别 k 的样本子集，$|C_k|$ 表示该样本子集中样本的个数；$|D|$ 表示数据集 D 的样本个数。

从公式形式上看，基尼指数将熵中的期望目标由 $-\log p_i$ 替换为 $1-p_i$。从曲线形状看，基尼指数与相同条件下的 $\frac{1}{2}$ 熵，非常相似，如图3-7所示[20]。

[20]注意，该图中，熵中对数以 2 为底，熵与基尼指数均为二分类情形。

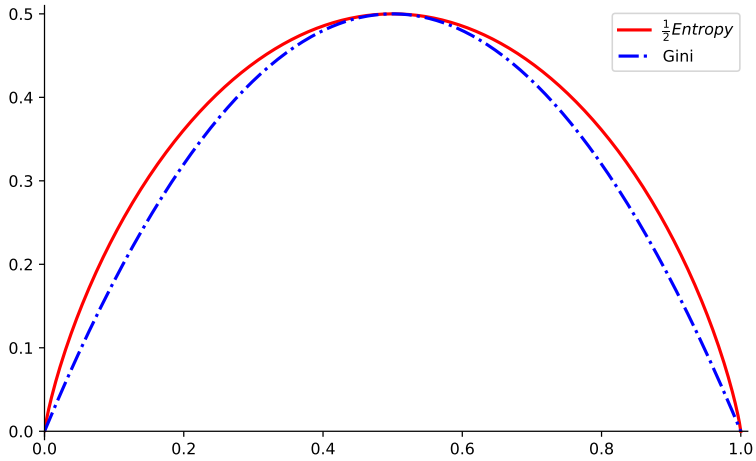


图 3-7: $\frac{1}{2}$ 熵与基尼指数的关系

正是由于这个原因，基尼指数也具有熵的一些性质，也能够度量数据集的不确定性。例如，在给定特征取值 $A=a$ 的条件下，数据集 D 的基尼指数为：

$$Gini(D|A=a)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2) \tag{67}$$

其中，D_1 和 D_2 为特征 A 取值 a 时将数据集 D 分割之后得到的 2 个数据子集：$D_1=\{(\boldsymbol{x},y)\in D|A(\boldsymbol{x})=a\}$，$D_2=D-D_1$。$Gini(D_1)$ 和 $Gini(D_2)$ 的计算，可以利用公式 (66)。

于是，利用公式 (66) 可以计算原数据集 D 的基尼指数，那么其“信息增益”为：

$$g(D,A=a)=Gini(D)-Gini(D|A=a) \tag{68}$$

类似于基于熵的信息增益算法，下面给出基于基尼指数的信息增益算法。

算法 **3.2**（基于基尼指数的信息增益算法）

Input:

Train Dataset: $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Feature: $A = a$

Numbers of Class: K

Output:

Information Gain: $g(D, A = a)$

Algorithm:

1. $Gini(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|}\right)^2$: Numbers of Class C_k in D, $|D|$: Numbers of D's Samples
2. $Gini(D|A = a) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2)$, $Gini(D_1) = 1 - \sum_{k=1}^{K} \left(\frac{|C_{k1}|}{|D_1|}\right)^2$, $Gini(D_2) = 1 - \sum_{k=1}^{K} \left(\frac{|C_{k2}|}{|D_2|}\right)^2$, $D_1 = \{(\boldsymbol{x}, y) \in D | A(\boldsymbol{x}) = a\}$, $D_2 = D - D_1$
3. $g(D, A = a) = Gini(D) - Gini(D|A = a)$

3.2.2 分类树的生成算法

下面，直接给出 CART 分类树的生成算法。

算法 **3.3**（*CART* 分类树的生成算法）

Input:

Train Dataset: $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$

Feature Set: A

Threshold of Information Gain: ϵ

Output:

CART for Classification: T

Algorithm: $CART_C(D, A)$

1. Create a *Root* node
2. If all samples are belong to C_k, return *Root* with label=C_k
3. if $A = \emptyset$, return *Root* with label=$\arg\max_{C_k} |C_k|$
4. Otherwise:
5. $A_g, a^* = \arg\max_{A_i, a} Gini(D) - Gini(D|A_i = a)$, $i = 1, 2, \ldots, n$,
   n: Numbers of Feature in A, $A_i = a$: all possible values for A_i
6. if $g(D, A_g) < \epsilon$, return *Root* with label=$\arg\max_{C_k} |C_k|$
7. $Left = CART_C(D_1, A - \{A_g\})$, $D_1 = \{(\boldsymbol{x}, y) \in D | A_g(\boldsymbol{x}) = a^*\}$
8. Add $Left$ to *Root*
9. $Right = CART_C(D_2, A - \{A_g\})$, $D_2 = D - D_1$
10. Add *Right* to *Root*
11. return *Root*

3.3 剪枝算法

CART 的剪枝算法要比前述的剪枝算法复杂些。其基本思想是，由于改变正则项的权重系数 α，就可以改变决策树的复杂程度——如果将 α 从 0 增加到 $+\infty$，且每次只减少一个合适的非叶子节点，这样就可以得到当前条件下的一棵最优子

树；那么，在此过程结束后，将得到复杂度逐级降低的最优子树序列[21]。最后，再利用独立的验证数据集，计算序列中每棵最优子树的平方误差 (回归树) 或基尼指数 (分类树)，从中选取平方误差或基尼指数最小的决策树作为最终的最优决策树。

首先，讨论一下如何生成最优子树序列。在剪枝情况下，需要定义一个带正则项的损失函数或考虑整体最优的损失函数。一个可选的损失函数是公式 (55)。需要注意的是，在引入基尼指数后，公式中的 $H(l_i)$ 可以使用熵或基尼指数。为表达这种变化，将其修改为 $L(l_i)$。于是，得到：

$$L(T) = \sum_{i=1}^{L} N_i L(l_i) + \alpha L \tag{69}$$

其中，L 表示树 T 中叶子节点的个数，l_i 表示树 T 的叶子节点，N_i 表示叶子节点 l_i 中样本点的个数。

对于某个固定的 $\alpha(0 \leqslant \alpha < +\infty)$，一定存在使得损失函数 $L(T)$ 最小的唯一子树 T，记为 T_α^*[22]：

$$\begin{aligned} T_\alpha^* &= \arg\min_T L(T) \\ &= \arg\min_T \sum_{i=1}^{L} N_i L(l_i) + \alpha L \end{aligned} \tag{70}$$

当 α 小时，T_α^* 偏大；极端情况时，即 $\alpha = 0$ 时，T_0^* 就是由决策树生成算法得到的原始决策树。当 α 大时，T_α^* 偏小；极端情况时，即 $\alpha \to +\infty$，T_α^* 仅有一个根节点，它将所有的训练数据样本点集中于根节点中。

Breiman 等人已经证明，利用上述方法生成的最优子树序列 $\{T_{\alpha_0}, T_{\alpha_1}, \ldots, T_{\alpha_n}\}$ 以及相应的 $\alpha_i(i = 0, 1, \ldots, n)$ 具有如下特点：

- 最优子树序列 $\{T_{\alpha_0}, T_{\alpha_1}, \ldots, T_{\alpha_n}\}$ 按从左到右的顺序，复杂度依次降低[23]，且每棵子树被其左边所有的子树所包含，形成一个所谓的嵌套结构；
- α_i 与 α_{i+1} 不是连续的，而是形成一个区间。该区间内所有的 α 值都对应同一棵最优子树；

[21] 当 $\alpha = 0$ 时，最初的决策树是最优的。当 $\alpha \to +\infty$ 时，根节点组成的单节点树是最优的；此时，根节点包括所有的数据集样本。

[22] 关于最优子树的存在性与唯一性，请阅读文献 [3]。

[23] 从左到右，每棵树依次减少一个非叶子节点，而不是减少一个节点，因为每个非叶子节点可能包含若干节点。

因此，将生成的最优子树序列改写为 $\{T_0, T_1, \ldots, T_n\}(n+1$ 表示最优子树的个数)，其对应的 α 序列为 $0 = \alpha_0 < \alpha_1 < \alpha_2 < \ldots < \alpha_n < \alpha_{n+1} = +\infty$，其中 T_i 对应的 α 区间为 $[\alpha_i, \alpha_{i+1})$。

下面，给出推导过程。设 T_0 为前述决策树生成算法生成的原始决策树——在基于熵或基尼指数的信息增益意义下，它是一棵最优决策树。首先，从 T_0 开始剪枝：以自底向上的方式，考察 T_0 的所有非叶子节点，命名为 $T_{0,i}$。计算剪枝前后 $T_{0,i}$ 的损失函数，即：

- 剪枝前

  将 $T_{0,i}$ 为根节点的子树命名为 $T_{0,i}^{old}$，则 $T_{0,i}^{old}$ 的损失函数为：

$$\begin{aligned} L(T_{0,i}^{old}) &= \sum_{i=1}^{L} N_i L(l_i) + \alpha L \\ &= C(T_{0,i}^{old}) + \alpha |T_{0,i}^{old}| \end{aligned} \tag{71}$$

  其中，L 为子树 $T_{0,i}^{old}$ 中叶子节点的个数，l_i 为叶子节点，N_i 为叶子节点 l_i 中样本点的个数，$L(l_i)$ 为叶子节点 l_i 的熵或基尼指数。为表达方便，令 $C(T_{0,i}^{old}) = \sum_{i=1}^{L} N_i L(l_i)$，$L = |T_{0,i}^{old}|$。

- 剪枝后

  此时，$T_{0,i}$ 为单节点树，将它命名为 $T_{0,i}^{new}$，则 $T_{0,i}^{new}$ 的损失函数为：

$$L(T_{0,i}^{new}) = C(T_{0,i}^{new}) + \alpha \tag{72}$$

  其中，$C(T_{0,i}^{new})$ 表示单节点树 $T_{0,i}^{new}$ 的熵或基尼指数。

从剪枝的角度看，$T_{0,i}^{new}$ 是 $T_{0,i}^{old}$ 的剪枝结果。但是，从决策树生成或特征划分的角度看，$T_{0,i}^{old}$ 是 $T_{0,i}^{new}$ 的生成或划分结果。因此，当 $\alpha = 0$ 时，$L(T_{0,i}^{old}) < L(T_{0,i}^{new})$[24]。

进一步地，如果逐渐增大 α，那么两个损失函数中的 $\alpha|T_{0,i}^{old}|$ 和 α 开始发挥作用：$L(T_{0,i}^{old})$ 和 $L(T_{0,i}^{new})$ 将逐渐增大。然而，$L(T_{0,i}^{old})$ 比 $L(T_{0,i}^{new})$ 增长得更快，这使

[24]从决策树生成或特征划分的角度看，当 $\alpha = 0$ 时，表示单纯地使用信息增益的准则，对节点 $T_{0,i}^{new}$ 进行划分，因而降低了节点 $T_{0,i}^{new}$ 的不确定性 (熵或基尼指数)，或者该划分获得了信息增益。因此，$C(T_{0,i}^{old}) < C(T_{0,i}^{new}) \Rightarrow L(T_{0,i}^{old}) < L(T_{0,i}^{new})$。直接对照信息增益的公式，$C(T_{0,i}^{new})$ 等价于 $H(D)$ 或 $Gini(D)$，而 $C(T_{0,i}^{old})$ 等价于 $H(D|A)$ 或 $Gini(D|A = a^*)$，其中 D 是节点 $T_{0,i}^{new}$ 所对应的数据集。

得 $L(T_{0,i}^{old})$ 逐渐地接近 $L(T_{0,i}^{new})$。在某个 α 处，它们相等，则：

$$\begin{aligned} & L(T_{0,i}^{old}) = L(T_{0,i}^{new}) \quad \Rightarrow \quad C(T_{0,i}^{old}) + \alpha |T_{0,i}^{old}| = C(T_{0,i}^{new}) + \alpha \quad \Rightarrow \\ & \alpha = \frac{C(T_{0,i}^{new}) - C(T_{0,i}^{old})}{|T_{0,i}^{old}| - 1} \end{aligned} \tag{73}$$

上式表达的含义是，只要 α 取该等式的值，则剪枝得到的 (单节点) 子树 $T_{0,i}^{new}$，其损失函数至少与剪枝前子树 $T_{0,i}^{old}$ 的损失函数一样好[25]，因而剪枝操作可行，执行剪枝。为表达方便，令 $\alpha_{1,i}$ 等于上式中的 α。

假设原始最优决策树 T_0 总共有 I 个非叶子节点，那么将得到 I 个 $\alpha_{1,i}(i = 1, 2, \ldots, I)$ 值。于是，令：

$$\alpha_1 = \min(\{\alpha_{1,1}, \alpha_{1,2}, \ldots, \alpha_{1,I}\}) \tag{74}$$

在得到 α_1 之后，在原始最优决策树 T_0 中，对 α_1 所对应的子树 (例如 $T_{0,k}^{old}$) 执行剪枝，从而将 T_0 转变为另一棵复杂度稍低的新树 (将其命名为 T_1)。

显然，区间 $[\alpha_0, \alpha_1)$(其中 $\alpha_0 = 0$) 对应原始最优决策树 T_0，而区间 $[\alpha_1, \alpha_2)$ 对应决策树 T_1[26]。

上述剪枝过程，一直持续下去，直至只剩下根节点为止。在此过程中，α_i 的值将会不断地增加——对应的 α 序列为 $0 = \alpha_0 < \alpha_1 < \alpha_2 < \ldots < \alpha_n < \alpha_{n+1} = +\infty$，且每个区间 $[\alpha_i, \alpha_{i+1})$ 对应一棵被剪枝的最优子树 T_i。由此，得到一个最优子树序列 $\{T_0, T_1, \ldots, T_n\}$。

最后，使用独立的验证数据集，并利用交叉验证方法，从上述序列中，选取平方误差或基尼指数最小的最优子树，作为剪枝后的最优决策树。

下面，给出 CART 剪枝算法。

算法 **3.4**（*CART* 剪枝算法）

[25]注意，如果继续增加 α，则 $L(T_{0,i}^{old}) > L(T_{0,i}^{new})$。

[26]α_2 在下一阶段按同样的方法确定。

Input:

Original Optimal CART Tree: T_0

Output:

Pruned Optimal CART Tree: T^*_{pruned}, $0 = \alpha^*_0 < \alpha^*_1 < \ldots < \alpha^*_n < \alpha^*_{n+1} = +\infty$

Algorithm:

1. $\tau = 1$ and $\alpha^*_0 = 0$
2. while $|T_{\tau-1}'s\ nodes| > 1$:
3. $\{\alpha_i\} = \frac{C(T^{new}_{\tau-1,i}) - C(T^{old}_{\tau-1,i})}{|T^{old}_{\tau-1,i}| - 1}$, $i = 1, 2, \ldots, I$ for all $T_{\tau-1}$'s nonleaf nodes from bottom to top in $T_{\tau-1}$
4. $\alpha^*_\tau = \min(\{\alpha_i\})$
5. T_τ=Pruned $T_{\tau-1}$ at nonleaf node identified by α^*_τ with label=$\arg\max_{C_k} |C_k|$, $k = 1, 2, \ldots, K$, K: class numbers
6. $\tau = \tau + 1$
7. Get T^*_{pruned} from $\{T_0, T_1, \ldots, T_n\}$ by cross validation

4 决策树实验

ID3 算法的实现

In [1]:
```
import numpy as np
from collections import Counter
from math import log
```

In [2]:
```
class DTNode:
    def __init__(self, i_feature = None, label = None):
        self.i_feature = i_feature # 特征索引
        self.label = label
        self.tree = {} # 存放子树

    def isLeaf(self):
        return len(self.tree) == 0

    def AddChild(self, feature_value, child_node):
        self.tree[feature_value] = child_node

    def _predict(self, x):
        if self.isLeaf():
            return self.label
        return self.tree[x[self.i_feature]]._predict(x)

    def predict(self, X):
        results = []
        for x in X:
            results.append(self._predict(x))
        return results

    def __repr__(self):
        return 'label={}, i_feature={}, subtree={}'.format(self.label,
         self.i_feature, self.tree)

class ID3Tree:
    def __init__(self, threshold = 0.1):
        # 信息增益阈值，低于阈值，不会进一步划分
        self.threshold = threshold
        self.root = None # 决策树的根节点

    # 计算数据集的经验熵
    def CalcDatasetEntropy(self, Y):
        N = len(Y)
        labels_counter = Counter(Y) # 提取出类标签及相应的样本个数
        return -sum([Ni/N*log(Ni/N, 2)
         for Ni in labels_counter.values()])

    # 计算数据集 / 特征的经验条件熵
    def CalcConditionEntropy(self, X, Y, i_feature):
```

```
        N = len(X)
        subsets = {} # 存放第 i 个特征（每个）离散值所对应的数据子集
        for i in range(N):
            feature_value = X[i][i_feature]
            if feature_value not in subsets:
                subsets[feature_value] = []
            subsets[feature_value].append(Y[i])
        return sum([len(subset)/N*self.CalcDatasetEntropy(subset)
         for subset in subsets.values()])

    # 选取最佳特征
    def SelectBestFeature(self, X, Y, used):
        features_info = []
        # 遍历每一个特征，但是需要排除已经使用过的特征
        for i in range(len(X[0])):
            if i in used:
                continue
            features_info.append((i, self.CalcConditionEntropy(X, Y, i)))
        return min(features_info, key = lambda x:x[-1])

    # 使用数据集递归生成 ID3 决策树
    def train(self, X, Y, used):
        if (len(X[0]) - len(used) == 0): # 没有可以使用的特征
            return DTNode(label = max(Counter(Y),
             key = lambda x:x[-1])[0])

        if len(Counter(Y)) == 1: # 所有实例属于一个类别
            return DTNode(label = Y[0])

        best_i_feature, best_cond_entropy = self.SelectBestFeature(X, Y,
         used)
        # 信息增益小于阈值
        if self.CalcDatasetEntropy(Y) - best_cond_entropy < self.threshold:
            return DTNode(label = max(Counter(Y), key = lambda x:x[-1])[0])

        # 非叶子节点
        current_dtnode = DTNode(i_feature = best_i_feature)
        used.append(best_i_feature)

        # 重新划分 X 与 Y
        subsets = {} # 存放特征 best_i_feature（每个）离散值所对应的数据子集
        for i in range(len(X)):
            feature_value = X[i][best_i_feature]
            if feature_value not in subsets:
                subsets[feature_value] = []
            subsets[feature_value].append((X[i], Y[i]))
        # 处理每个特征离散值对应的新数据集
        for feature_value, newdataset in subsets.items():
```

```
                NEWX = np.array([xy[0] for xy in newdataset])
                NEWY = np.array([xy[1] for xy in newdataset])
                child = self.train(NEWX, NEWY, used)
                current_dtnode.AddChild(feature_value, child)
            return current_dtnode

        # 使用数据集生成 ID3 决策树
        def fit(self, X, Y):
            # 最后一个列表用来保存已经使用过的特征索引
            self.root = self.train(X, Y,[])
            print(self.root)
            print('ID3 decision tree training completed!')

        # 预测
        def predict(self, X):
            return self.root.predict(X)
```

创建测试数据集，《统计学习方法》第 2 版，例 5.1

```
In [3]: import pandas as pd
```

```
In [4]: datasets = [['青年', '否', '否', '一般', '否'],
                    ['青年', '否', '否', '好', '否'],
                    ['青年', '是', '否', '好', '是'],
                    ['青年', '是', '是', '一般', '是'],
                    ['青年', '否', '否', '一般', '否'],
                    ['中年', '否', '否', '一般', '否'],
                    ['中年', '否', '否', '好', '否'],
                    ['中年', '是', '是', '好', '是'],
                    ['中年', '否', '是', '非常好', '是'],
                    ['中年', '否', '是', '非常好', '是'],
                    ['老年', '否', '是', '非常好', '是'],
                    ['老年', '否', '是', '好', '是'],
                    ['老年', '是', '否', '好', '是'],
                    ['老年', '是', '否', '非常好', '是'],
                    ['老年', '否', '否', '一般', '否'],
                   ]
        labels = [u'年龄', u'有工作', u'有自己的房子', u'信贷情况', u'类别']
        train_data = pd.DataFrame(datasets, columns = labels)
        X = np.array(train_data.iloc[:, :-1])
        Y = np.array(train_data.iloc[:, -1])
        train_data
```

Out[4]:

	年龄	有工作	有自己的房子	信贷情况	类别
0	青年	否	否	一般	否
1	青年	否	否	好	否
2	青年	是	否	好	是
3	青年	是	是	一般	是
4	青年	否	否	一般	否

```
5   中年   否      否   一般   否
6   中年   否      否    好   否
7   中年   是      是    好   是
8   中年   否      是  非常好   是
9   中年   否      是  非常好   是
10  老年   否      是  非常好   是
11  老年   否      是    好   是
12  老年   是      否    好   是
13  老年   是      否  非常好   是
14  老年   否      否   一般   否
```

训练 ID3 决策树

```
In [5]: id3_dt = ID3Tree()
        id3_dt.fit(X, Y)
```

```
label=None, i_feature=2, subtree={' 是': label= 是, i_feature=None, subtree={},
 ' 否': label=None, i_feature=1, subtree={' 是': label= 是, i_feature=None,
 subtree={}, ' 否': label= 否, i_feature=None, subtree={}}}
ID3 decision tree training completed!
```

使用测试数据进行测试

```
In [6]: XTEST = [['老年', '否', '否', '一般'],
                 ['中年', '是', '否', '一般'],
                 ['中年', '否', '否', '一般'],
                 ['青年', '否', '否', '一般'],
                 ['青年', '否', '否', '好'],
                 ['青年', '否', '是', '一般'],
                ]
        id3_dt.predict(XTEST)
```

```
Out[6]: [' 否', ' 是', ' 否', ' 否', ' 否', ' 是']
```

使用 sklearn.tree.DecisionTreeClassifier

```
In [7]: from sklearn import preprocessing
        from sklearn.tree import DecisionTreeClassifier
```

对数据集进行预处理
生成字符串离散值的编码

```
In [8]: age_str = ['青年', '中年', '老年']
        age = preprocessing.LabelEncoder()
        age.fit(age_str)
        print(age.transform(age_str))

        yes_str = ['是', '否']
```

```
yes = preprocessing.LabelEncoder()
yes.fit(yes_str)
print(yes.transform(yes_str))

good_str = ['一般', '好', '非常好']
good = preprocessing.LabelEncoder()
good.fit(good_str)
print(good.transform(good_str))
```

```
[2 0 1]
[1 0]
[0 1 2]
```

对数据集进行编码转换

```
In [9]: sk_train_data = train_data.copy(deep = True)

sk_train_data[u'年龄'] = sk_train_data[u'年龄'].apply(lambda x:
 age.transform([x])[0])
sk_train_data[u'有工作'] = sk_train_data[u'有工作'].apply(lambda x:
 yes.transform([x])[0])
sk_train_data[u'有自己的房子'] = sk_train_data[u'有自己的房子'].
 apply(lambda x:yes.transform([x])[0])
sk_train_data[u'信贷情况'] = sk_train_data[u'信贷情况'].
 apply(lambda x:good.transform([x])[0])
sk_train_data[u'类别'] = sk_train_data[u'类别'].apply(lambda x:
 float(yes.transform([x])[0]))

sk_train_data
```

Out[9]:

	年龄	有工作	有自己的房子	信贷情况	类别
0	2	0	0	0	0.0
1	2	0	0	1	0.0
2	2	1	0	1	1.0
3	2	1	1	0	1.0
4	2	0	0	0	0.0
5	0	0	0	0	0.0
6	0	0	0	1	0.0
7	0	1	1	1	1.0
8	0	0	1	2	1.0
9	0	0	1	2	1.0
10	1	0	1	2	1.0
11	1	0	1	1	1.0
12	1	1	0	1	1.0
13	1	1	0	2	1.0
14	1	0	0	0	0.0

生成适应 sklearn 的数据集

```
In [10]: X = np.array(sk_train_data.iloc[:, :-1])
         Y = np.array(sk_train_data.iloc[:, -1])
         clf = DecisionTreeClassifier(criterion = 'entropy')
         clf.fit(X, Y)

Out[10]: DecisionTreeClassifier(class_weight=None, criterion='entropy',
                     max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False,
                     random_state=None, splitter='best')
```

对测试数据集进行转换

```
In [11]: sk_test_data = pd.DataFrame(XTEST, columns = labels[:-1])
         sk_test_data
```

Out[11]:

	年龄	有工作	有自己的房子	信贷情况
0	老年	否	否	一般
1	中年	是	否	一般
2	中年	否	否	一般
3	青年	否	否	一般
4	青年	否	否	好
5	青年	否	是	一般

```
In [12]: sk_test_data[u'年龄'] = sk_test_data[u'年龄'].apply(lambda x:
          age.transform([x])[0])
         sk_test_data[u'有工作'] = sk_test_data[u'有工作'].apply(lambda x:
          yes.transform([x])[0])
         sk_test_data[u'有自己的房子'] = sk_test_data[u'有自己的房子'].
          apply(lambda x:yes.transform([x])[0])
         sk_test_data[u'信贷情况'] = sk_test_data[u'信贷情况'].
          apply(lambda x:good.transform([x])[0])
         sk_test_data
```

Out[12]:

	年龄	有工作	有自己的房子	信贷情况
0	1	0	0	0
1	0	1	0	0
2	0	0	0	0
3	2	0	0	0
4	2	0	0	1
5	2	0	1	0

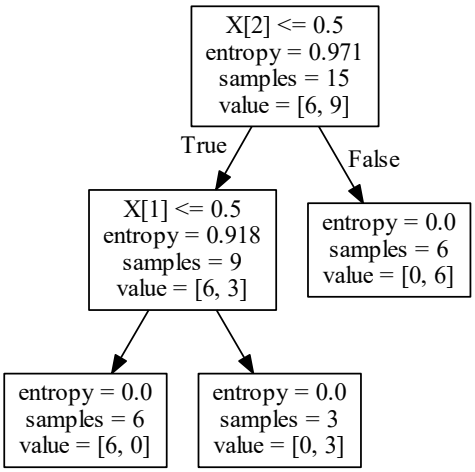
```
In [13]: XTEST = np.array(sk_test_data.iloc[:, :])
         print(yes.inverse_transform([int(r) for r in clf.predict(XTEST)]))

[' 否' ' 是' ' 否' ' 否' ' 否' ' 是']
```

绘制决策树

In [14]:
```
# 注意: Graphviz 功能与 jupyter notebook 的 latex 文件输出功能有冲突
# 在 Graphviz 功能存在的情况下, jupyter notebook 的 latex 输出功能失效

# 先安装 Graphviz 应用程序 (graphviz-2.38.msi),
# 然后安装包 pip install graphviz
from sklearn.tree import export_graphviz
import graphviz
```

In [15]:
```
# 需要安装 pydot 包: pip install pydot
import pydot
import os

# 添加路径“E:\Program Files (x86)\Graphviz2.38\bin”到 PATH 中
# 根据 Graphviz 的实际安装路径, 修改下面的目录
os.environ['PATH'] = os.environ['PATH'] + \
    (';E:\\Program Files (x86)\\Graphviz2.38\\bin\\')
```

In [16]:
```
export_graphviz(clf, out_file = "sklearn_dt_tree.dot")
with open('sklearn_dt_tree.dot') as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



生成 PDF 文件

In [17]:
```
(graph, ) = pydot.graph_from_dot_file('sklearn_dt_tree.dot')
# 需要根据实际安装目录, 修改下面的文件
```

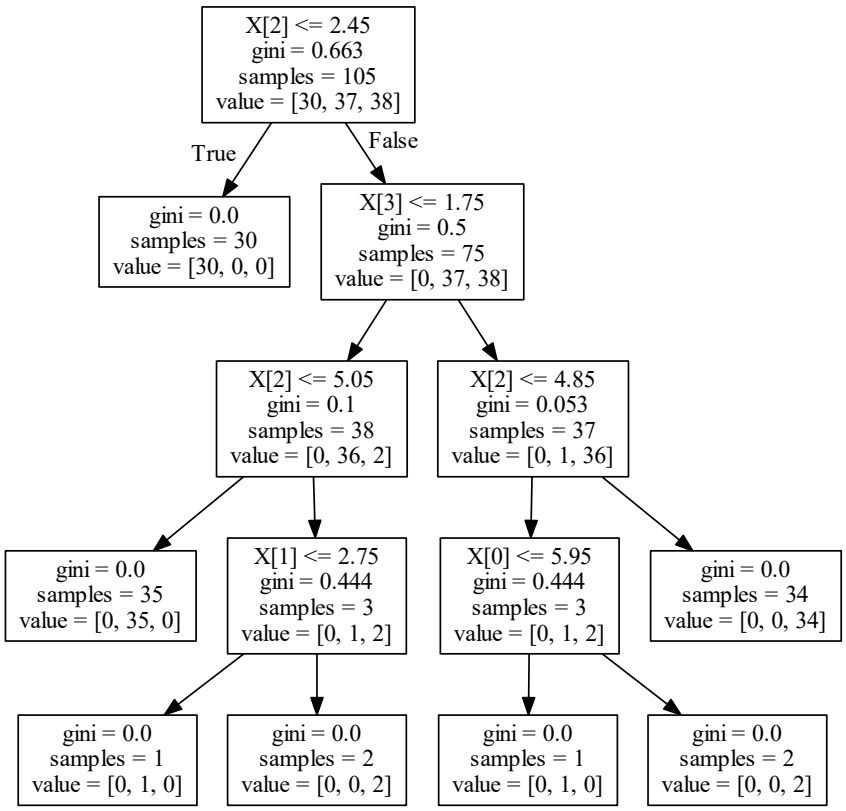
```
        # 需要修改文件 E:\Anaconda3\envs\DXQ\Lib\site-packages\pydot.py
        # 第 1712 行: self.prog='dot.exe'
        graph.write_pdf('sklearn_dt_tree.pdf')
```

使用鸢尾花数据集的所有数据进行训练和测试

```
In [18]: from sklearn.model_selection import train_test_split

         iris_npz = np.load('iris_full.npz')
         data = iris_npz['data']
         X = iris_npz['X']
         Y = iris_npz['Y']

         XTRAIN, XTEST, YTRAIN, YTEST = train_test_split(X, Y, test_size = 0.3)

In [19]: clf = DecisionTreeClassifier(criterion = 'gini')
         clf.fit(XTRAIN, YTRAIN)

Out[19]: DecisionTreeClassifier(class_weight=None, criterion='gini',
                     max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False,
                     random_state=None, splitter='best')

In [20]: clf.score(XTEST, YTEST)

Out[20]: 0.93333333333333335
```

绘制决策树

```
In [21]: export_graphviz(clf, out_file = "sklearn_iris_dt_tree.dot")
         with open('sklearn_iris_dt_tree.dot') as f:
             dot_graph = f.read()
         graphviz.Source(dot_graph)
```



生成 PDF 文件

```
In [22]: (graph, ) = pydot.graph_from_dot_file('sklearn_iris_dt_tree.dot')
         graph.write_pdf('sklearn_iris_dt_tree.pdf')
```

5 参考文献

1. 李航。《统计学习方法》，清华大学出版社，2019 年 5 月第 2 版。
2. 周志华。《机器学习》，清华大学出版社，2016 年 1 月第 1 版。
3. Leo Breiman. Classification and Regression Trees. CRC Press, 1998.
4. Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.
5. Kevin P. Murphy. Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.
6. https://zh.wikipedia.org/wiki/自信息.
7. https://zh.wikipedia.org/wiki/熵 _(信息论).
8. https://zh.wikipedia.org/wiki/互信息.
9. https://zh.wikipedia.org/wiki/条件熵.
10. https://en.wikipedia.org/wiki/ID3_algorithm.
11. https://www.zhihu.com/question/22539777.
12. https://github.com/wzyonggege/statistical-learning-method.