第 8 章 决策树与集成学习

目录

第 8 章 决策树与集成学习

8.1 决策树的基本思想

经过前面几个章节的介绍，我们已经学习过了 3 个分类算法模型，包括逻辑回归、K 近邻和朴素贝叶斯。今天笔者将开始介绍下一个分类算法模型决策树（Decision Tree）。

一说到决策树其实很读者或多或少都已经使用过，只是自己还不知道罢了。例如最简单的决策树就是通过输入年龄，判读其是否为成年人，即 if age >= 18 return True，想想自己是不是经常用到这样的语句?

8.1.1 冠军球队

关于什么是决策树，我们先来看这么一个例子。假如笔者错过了某次世界杯比赛，赛后笔者问一个知道比赛结果的人“哪支球队是冠军”？但是对方并不愿意直接说出结果，而是让笔者自己猜，且每猜一次对方都要收一元钱才肯告诉笔者是否猜对了。那现在的问题是要掏多少钱才能知道谁是冠军球队呢[①]?

现在我可以把球队从 1 到 16 编上号，然后提问：“冠军球队在 1-8 号中吗？”。假如对方告诉我猜对了，笔者就会接着问：“冠军在 1-4 号中吗？”。假如对方告诉我猜错了，那么笔者也就自然知道冠军在 5-8 号中。这样只需要 4 次，笔者就能知道哪支球队是冠军。上述过程背后所隐藏着的其实就是决策树的基本思想，并且还可以用更为直观的图来展示上述的过程，如图 8-1 所示。

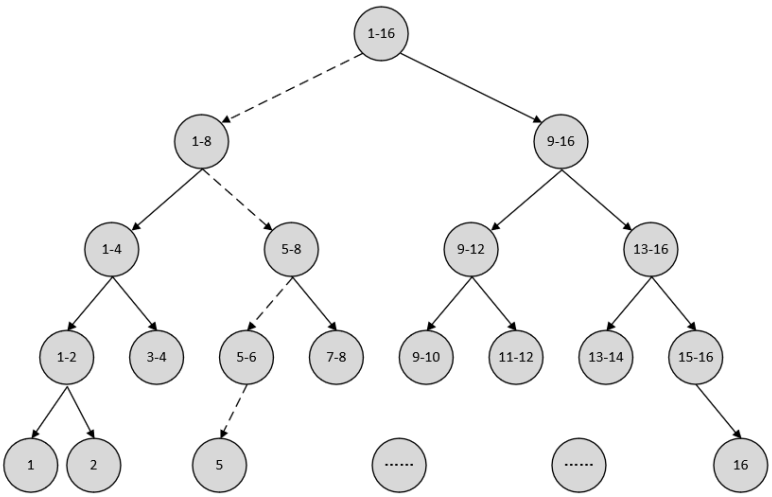


图 8-1 决策树思想图

由此可以得出，决策树每一步的决策过程就是降低信息不确定性的过程，甚至还可以将这些决策看成是一个 if-then 的规则集合。如图 8-1 所示，冠军球队一开始有 16 种可能性，经过一次决策后变成了 8 种。这意味着每次决策都能得到更多确定的信息，而减少更多的不确定性。

不过现在的问题是，为什么要像图 8-1 这样来划分球队？对于熟悉足球的读者来说，这样的决策树似乎略显多余。因为事实上只有少数几支球队才有夺冠的希望，而大多数是没有

[①] 吴军，数学之美，人民邮电出版社

的。因此，一种改进做法就是在一开始的时候就将几个热门的可能夺冠的球队分在一边，将剩余的放在另一边，这样就可以大大提高整个决策过程的效率。

例如最有可能夺冠的是 1,2,3,4 这 4 个球队，而其余球队夺冠的可能性远远小于这 4 个。那么一开始就可以将球队分成 1-4 和 5-16。如果冠军是在 1-4 中，那么后面很快就能知道谁是冠军。退一万步，假如冠军真的是在 5-16，那么接下来你同样可以按照类似的思路将剩余的球队划分成最有可能夺冠和最不可能夺冠这两个部分。这样也能快速的找出谁是冠军球队。

于是这时候可以发现，如何划分球队变成了建立这棵决策树的关键。如果存在一种划分，能够使得数据的“不确定性”减少得越多（谁不可能夺冠），也就意味着该划分能获取更多的信息，而我们也就更倾向于采取这样的划分。因此采用不同的划分就会得到不同的决策树。所以，现在的问题就变成了如何构建一棵“好”的决策树呢？要想回答这个问题，先来解决如何描述“信息”这个问题。

8.1.2 信息的度量

关于如何定量的来描述信息，几千年来都没有人给出很好的解答。直到 1948 年，香农在他著名的论文“通信的数学原理”中提出了信息熵（Information Entropy）的概念，这才解决了信息的度量问题，并且还量化出了信息的作用。

1) 信息熵

一条信息的信息量与其不确定性有着直接的关系。比如说，要搞清楚一件非常非常不确定的事，就需要了解大量的信息。相反，如果已经对某件事了解较多，则不需要太多的信息就能把它搞清楚。所以从这个角度来看可以认为，信息量就等于不确定性的多少。我们经常说，一句话包含有多少信息，其实就是指它不确定性的多与少。

于是，8.1.1 节中第一种划分方式的不确定性（信息量）就等于“4 块钱”，因为笔者花 4 块钱就可以解决这个不确定性。当然，香农用的不是钱，而是用“比特”（Bit）这个概念来度量信息量，一个字节就是 8 比特。在上面的第一种情况中，“谁是冠军”这条消息的信息量就是 4 比特。那 4 比特是怎么计算来的呢？第二种情况的信息量又是多少呢？

香农指出，它的准确信息量应该是

$$H = -(p_1 \cdot \log p_1 + p_2 \cdot \log p_2 + \cdots + p_{16} \cdot \log p_{16}) \tag{8-1}$$

其中 $\log$ 表示以 2 为底的对数，$p_1, p_2, ..., p_{16}$ 分别是这 16 支球队夺冠的概率。香农把 s 式(8-1)的结果称为“信息熵（Entropy），一般用符号 H 表示，单位是比特。由于在第一种情况中，默认条件是 16 支球队夺冠概率相同，因此对应的信息熵就是 4 比特。

对于任意一个随机变量 X （比如得冠军的球队），它的熵定义如下[①]：

$$H(X) = -\sum_{x \in X} P(x) \log P(x), \tag{8-2}$$

其中 log 表示以 2 为底的对数 。

例如在二分类问题中：设 $P(y=0) = p, P(y=1) = 1-p, 0 \le p \le 1$，那么此时的信息熵 $H(y)$ 即为：

$$H(y) = -(p \log p + (1-p) \log(1-p)) \tag{8-3}$$

根据式(8-3)还能画出其对应的函数图像，如图 8-2 所示。

[①] 李航，统计机器学习，清华大学出版社

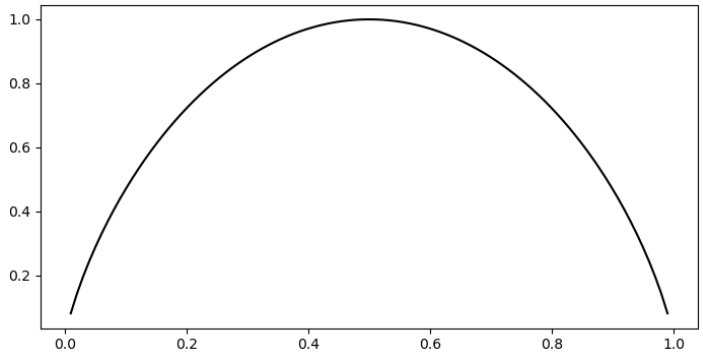


图 8-2 二分类中的信息熵

从图 8-2 可以发现，当两种情况发生的概率均等（$p=1-p=0.5$）时信息熵最大，也就是说此时的不确定性最大，要把这件事搞清楚所需要的信息量也就越大。并且这也很符合我们的常识，例如“明天下雨和不下雨的概率都是 50%”，那么这一描述所存在的不确定性是最大的。

2) 条件熵

在谈条件熵（Condition Entropy）之前，我们先来看看信息的作用。一个事物（那只球队会得冠），其内部会存有随机性也就是不确定性，假定其为U。从外部消除这个不确定性的唯一办法就是引入信息I（我来猜，另一个人给我反馈）。并且需要引入的信息量取决于这个不确定性的大小，即$I>U$才能完全消除这一不确定性。当$I<U$时，外部信息可以消除一部分不确定性，也就是说新的不确定性$U'=U-I$。反之，如果没有信息的引入，那么任何人都无法消除原有的不确定性[①]。而所谓条件熵，说得简单点就是在给定某种条件（有用信息）I下，事物U的熵，即$H(U|I)$。

到此，对于条件熵相信读者朋友们在感性上已经有了一定的认识。至于具体的数学定义将在决策树生成部分进行介绍。

3) 信息增益

在 8.1.1 节中笔者介绍到，若一种划分能使数据的“不确定性”减少得越多，也就意味着该种划分能获取更多的信息，而我们也就更倾向于采取这样的划分。也是就说，存在某个事物I，当引入事物I的信息后U的熵变小了。而我们要选的就是那个最能使得U的信息熵变小的I，即需要得到最大的信息增益（Information Gain）

$$g(U|D)=H(U)-H(U|I) \tag{8-4}$$

综上所述，采用不同的划分就会得到不同的决策树，而我们所希望得到的就是在每一次划分的时候都采取最优的方式，即局部最优解。这样每一步划分都能够使得当前的信息增益达到最大。因此可以得出，构建决策树的核心思想就是每次划分时，要选择使得信息增益达到最大时的划分方式，以此来递归构建决策树。

8.1.3 小结

在本节中，笔者首先介绍了决策树的核心思想，即决策树的本质就是降低信息不确定性的过程；然后总结出构建一棵决策树的关键在于找到一种合适的划分，使得信息的“不确定性”能够降低得最多；最后笔者介绍了如何以量化的方式来对信息进行度量。

[①] 吴军，数学之美，人民邮电出版社

8.2 决策树的生成之 ID3 与 C4.5

在正式介绍决策树的生成算法前，笔者先将 8.1.1 节中介绍的几个概念重新梳理一下；并且同时再通过一个例子来熟悉一下计算过程，以便于后续更好的理解决策树的生成算法。

8.2.1 基本概念与定义

1) 信息熵

设 X 是一个取值有限的离散型随机变量（例如上一小节中可能夺冠的 16 只球队），其概率分布为 $P(X=x_i)=p_i, i=1,2,...,n$（每个球队可能夺冠的概率），则随机变量 X 的信息熵定义为

$$H(X)=-\sum_{i=1}^{n} p_i \log p_i \tag{8-5}$$

其中，若 $p_i=0$，则定义 $0\log 0=0$；且通常 $\log$ 取 2 为底或 e 为底时，其熵的单位分别称为比特（Bit）或纳特（Nat）。如无特殊说明，默认以 2 为底。

3) 条件熵

设有随机变量 (X,Y)，其联合概率分布分 $P(X=x_i,Y=y_i)=p_{ij}$，其中 $i=1,2,...,n$，$j=1,2,...,m$；条件熵 $H(Y\mid X)$ 表示在已知随机变量 X 的条件下，随机变量 Y 的不确定性，其定义为

$$H(Y\mid X)=\sum_{i=1}^{n} p_i H(Y\mid X=x_i) \tag{8-6}$$

其中，$p_i=P(X=x_i), i=1,2,...,n$。

同时，当信息熵和条件熵中的概率由样本数据估计（特别是极大似然估计）得到时，所对应的信息熵与条件熵分别称之为经验熵（Empirical Entropy）和经验条件熵（Empirical Conditional Entropy）。这里暂时看不懂没关系，请结合后续计算示例。

3) 信息增益

从 8.1.1 节的内容可知，所谓信息增益指的就是事物 U 的信息熵 $H(U)$，在引入外部信息 I 后的变化量 $H(U)-H(U\mid I)$。因此，可以将特征 A 对训练数据集 D 的信息增益 $d(D,A)$ 定义为集合 D 信息熵 $H(D)$ 与特征 A 给定条件下 D 的条件熵 $H(D\mid A)$ 之差，即

$$g(D,A)=H(D)-H(D\mid A) \tag{8-7}$$

定义：设训练集为 D，$|D|$ 表示所有训练样本总数；同时 D 有 K 个类别 $C_k, k=1,2,...,K$；$|C_k|$ 为属于类 C_k 的样本总数，即 $\sum_{k=1}^{K}|C_k|=|D|$；设特征 A 有 n 个不同的取值 $a_1,a_2,...,a_n$，根据特征 A 的取值将 D 划分为 n 个子集 $D_1,D_2,...,D_n$，$|D_i|$ 为子集 D_i 中的样本个数，即 $\sum_{i=1}^{n}|D_i|=|D|$；同时记子集 D_i 中，属于类 C_k 的样本集合为 D_{ik}，即 $D_{ik}=D_i\bigcap C_k$，$|D_{ik}|$ 为 D_{ik} 的样本个数。此时如下定义[①]

- 数据集 D 的经验熵 $H(D)$ 为

[①] 李航，统计机器学习，清华大学出版社

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|} \tag{8-8}$$

从式(8-8)可以看出，它计算的是“任意样本属于其中一个类别”这句话所包含的信息量。

- 数据集 D 在特征值 A 下的经验条件熵 $H(D|A)$ 为

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|} \tag{8-9}$$

从式(8-9)可以看出，它计算的是特征 A 在各个取值条件下“任意样本属于其中一个类别”这句话所包含的信息量。

- 信息增益为

$$g(D,A)=H(D)-H(D|A) \tag{8-10}$$

8.2.2 计算示例

如果仅看上面的公式肯定会不那么容易理解，下面笔者再进行举例说明（将上面的公式同下面的计算过程对比看会更容易理解）。下表 8-1 同样是 6.1.3 节中用过的一个信用卡审批数据集，其一共包含 15 个样本和 3 个特征维度。其中特征 $X^{(1)}\in A_1=\{0,1\}$ 表示有无工作，特征 $X^{(2)}\in A_2=\{0,1\}$ 表示是否有房，特征 $X^{(3)}\in A_3=\{D,S,T\}$ 表示学历等级，$Y\in C=\{0,1\}$ 表示是否审批通过的类标记。

表 8-1 示例计算数据

样本	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$X^{(1)}$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$X^{(2)}$	1	1	1	0	1	0	0	0	0	0	1	1	1	0	0
$X^{(3)}$	T	S	S	T	T	T	D	T	T	D	D	T	T	S	S
Y	1	1	1	0	0	0	0	1	1	1	1	1	1	0	1

1) 计算信息熵

根据式(8-8)可得

$$H(D)=-\left(\frac{5}{15}\log_2\frac{5}{15}+\frac{10}{15}\log_2\frac{10}{15}\right)\approx 0.918 \tag{8-11}$$

2) 计算条件熵

由表 8-1 可知，数据集有 3 个特征（工作、房子、学历）A_1,A_2,A_3；接下来根据式(8-9)来计算 D 分别在 3 个特征取值条件下的条件熵 $H(D|A_i)$。

- 已知外部信息“工作”的情况下有

$$\begin{aligned}H(D|A_1)&=\left[\frac{7}{15}H(D_1)+\frac{8}{15}H(D_2)\right]\\&=-\frac{7}{15}\left(\frac{3}{7}\log_2\frac{3}{7}+\frac{4}{7}\log_2\frac{4}{7}\right)-\frac{8}{15}\left(\frac{7}{8}\log_2\frac{7}{8}+\frac{1}{8}\log_2\frac{1}{8}\right)\approx 0.75\end{aligned} \tag{8-12}$$

式(8-12)中，D_1,D_2 分别是 A_1 取值为“无工作”和“有工作”时，训练样本划分后对应的子集。

- 已知外部信息“房子”的情况下有

$$
\begin{aligned}
H(D|A_2) &= \left[\frac{8}{15}H(D_1)+\frac{7}{15}H(D_2)\right] \\
&= -\frac{8}{15}\left(\frac{4}{8}\log_2\frac{4}{8}+\frac{4}{8}\log_2\frac{4}{8}\right)-\frac{7}{15}\left(\frac{1}{7}\log_2\frac{1}{7}+\frac{6}{7}\log_2\frac{6}{7}\right)\approx 0.81
\end{aligned}
\tag{8-13}
$$

式(8-13)中，D_1, D_2 分别是 A_2 取值为“无房”和“有房”时，训练样本划分后对应的子集。

- 已知外部信息“学历”的情况下有

$$
\begin{aligned}
H(D|A_3) &= \left[\frac{3}{15}H(D_1)+\frac{4}{15}H(D_2)+\frac{8}{15}H(D_3)\right] \\
&= -\frac{3}{15}\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)-\frac{4}{15}\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{3}{4}\log_2\frac{3}{4}\right) \\
&-\frac{8}{15}\left(\frac{3}{8}\log_2\frac{3}{8}+\frac{5}{8}\log_2\frac{5}{8}\right)\approx 0.91
\end{aligned}
\tag{8-14}
$$

式(8-14)中，D_1, D_2, D_3 分别是 A_3 取值为“D”、“S”和“T”时，训练样本划分后对应的子集。

3) 计算信息增益

根据上面的计算结果便可以计算得到各特征划分下的信息增益为

$$
\begin{aligned}
g(D, A_1) &= 0.918-0.75=0.168 \\
g(D, A_2) &= 0.918-0.81=0.108 \\
g(D, A_3) &= 0.918-0.91=0.08
\end{aligned}
\tag{8-15}
$$

到目前为止，我们已经知道了在生成决策树的过程中所需要计算的关键步骤信息增益，接下来，笔者就开始正式介绍如何生成一棵决策树。

8.2.3 ID3 生成算法

在 8.1 节的末尾笔者总结到，构建决策树的核心思想就是：每次划分时，要选择使得信息增益最大的划分方式，以此来递归构建决策树。如果利用一个特征进行分类的结果与随机分类的结果没有很大差别，则称这个特征没有分类能力。因此，对于决策树生成的一个关键步骤就是选取对训练数据具有分类能力的特征，这样可以提高决策树学习的效率，而通常对于特征选择的准则就是 8.1 节谈到的信息增益。

ID3（Interactive Dichotomizer-3）算法的核心思想是在选择决策树的各个节点时，采用信息增益来作为特征选择的标准，从而递归地构建决策树。其过程可以概括为，从根节点开始计算所有可能划分情况下的信息增益；然后选择信息增益最大的特征作为划分特征，由该特征的不同取值建立子节点；最后对子节点递归地调用以上方法，构建决策树；直到所有特征的信息增益均很小或没有可以选择为止。例如根据 8.2.2 节最后的计算结果可知，首先应该将数据样本以特征 A_1（有无工作）作为划分方式对数据集进行第一次划分。下面开始介绍通过 ID3 来生成决策树的步骤及示例。

- 生成步骤

输入：训练数据集 D，特征集 A，阈值 ε；

输出：决策树[①]。

(1) 若 D 中所有样本属于同一类 C_k（即此时只有一个类别）则 T 为单节点树，将 C_k 作为该节点的类标记，返回 T;

(2) 若 $A=\varnothing$，则 T 为单节点树，并将 D 中样本数最多的类 C_k 作为该节点的类标记，并返回 T；

(3) 否则，计算 A 中各特征对 D 的信息增益，选择信息增益最大的特征 A_g；

(4) 如果 A_g 的信息增益小于阈值 ε，则置 T 为单节点树，并将 D 中样本数最多的类 C_k 作为该节点的类标记，返回 T；

(5) 否则，对 A_g 的每一个可能值 a_i，以 $A_g=a_i$ 将 D 分割为若干非空子集，并建立为子节点；

(6) 对于第 i 个子节点，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用(1)-(5)，得到子树 T_i，返回 T_i。

❑ 生成示例

下面就用 ID3 算法来对表 8-1 中的数据样本进行决策树生成示例。易知该数据集不满足步骤(1)(2)中的条件，所以开始执行步骤(3)。同时，根据 8.2.2 小节最后的计算结果可知，对于特征 A_1,A_2,A_3 来说，在 A_1 条件下信息增益最大，所以应该选择特征 A_1 作为决策树的根节点。

由于本例中未设置阈值，所以接着执行步骤(5)按照 A_1 的取值将训练集 D 划分为两个子集 D_1,D_2，如表 8-2 所示。

表 8-2 第一次划分表

样本	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$X^{(1)}$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$X^{(2)}$	1	1	1	0	1	0	0	0	0	0	1	1	1	0	0
$X^{(3)}$	T	S	S	T	T	T	D	T	T	D	D	T	T	S	S
Y	1	1	1	0	0	0	0	1	1	1	1	1	1	0	1
第一次划分	D_1							D_2							

接着开始执行步骤(6)，由于 D_1,D_2 均不满足步骤(1)(2)中的条件，所以两部分需要分别继续执行后续步骤，此时生成的决策树如图 8-3 所示。

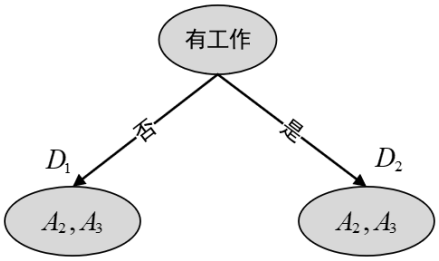


图 8-3 第一次划分

此时，对于子集 D_1 来说，需要从特征 $A-\{A_g\}$ 中即 A_2,A_3 中选择新的特征，并计算信息增益。

1) 计算信息熵 $H(D_1)$

[①] 李航，统计机器学习，清华大学出版社

$$H(D_1)=-\left(\frac{3}{7}\log_2\frac{3}{7}+\frac{4}{7}\log_2\frac{4}{7}\right)\approx 0.985 \tag{8-16}$$

2) 计算条件熵

$$H(D_1\mid A_2)=-\frac{3}{7}\left(\frac{3}{3}\log_2\frac{3}{3}+0\right)-\frac{4}{7}\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{3}{4}\log_2\frac{3}{4}\right)\approx 0.464 \tag{8-17}$$

$$H(D_1\mid A_3)=-\frac{1}{7}(0+0)-\frac{2}{7}(0+0)-\frac{4}{7}\left(\frac{3}{4}\log_2\frac{3}{4}+\frac{1}{4}\log_2\frac{1}{4}\right)\approx 0.464 \tag{8-18}$$

根据式(8-17)和(8-18)的结果可知，对于子集 D_1 来说，无论其采用 A_2, A_3 中的哪一个特征进行划分，最后计算得到的信息增益都是相等的，所以这里不妨就以特征 A_2 进行划分。

同理，对于子集 D_2 来说，也需要从特征 $A-\{A_g\}$ 中即 A_2, A_3 中选择新的特征，并计算信息增益。

1) 计算信息熵

$$H(D_2)=-\left(\frac{1}{8}\log_2\frac{1}{8}+\frac{7}{8}\log_2\frac{7}{8}\right)\approx 0.544 \tag{8-19}$$

2) 计算条件熵

$$H(D_2\mid A_2)=-\frac{5}{8}\left(\frac{1}{5}\log_2\frac{1}{5}+\frac{4}{5}\log_2\frac{4}{5}\right)-\frac{3}{8}(0+0)\approx 0.451 \tag{8-20}$$

$$H(D_2\mid A_3)=-\frac{2}{8}(0+0)-\frac{2}{8}\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)-\frac{4}{8}(0+0)=0.25 \tag{8-21}$$

3) 信息增益

$$\begin{aligned} g(D_2\mid A_2)&=0.544-0.451=0.093\\ g(D_2\mid A_3)&=0.544-0.25=0.294 \end{aligned} \tag{8-22}$$

根据式(8-22)的计算结果可知，对于子集 D_2 来说，采用特征 A_3 来对其进行划分时所产生的信息增益最大，因此应该选择特征 A_3 来作为子集 D_2 的根节点。

到此，根据上述计算过程，便可以得到第二次划分后的结果，如表 8-3 所示。

表 8-3 第二次划分表

样本	4	6	7	1	2	3	5	10	11	14	15	8	9	12	13
$X^{(1)}$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$X^{(2)}$	0	0	0	1	1	1	1	0	1	0	0	0	0	1	1
$X^{(3)}$	T	T	D	T	S	S	T	D	D	S	S	T	T	T	T
Y	0	0	0	1	1	1	0	1	1	0	1	1	1	1	1
第一次划分	D_1							D_2							
第二次划分	D_{11}			D_{12}				D_{21}		D_{22}		D_{23}			

从表 8-3 中的结果可知，D_{11}, D_{21}, D_{23} 这三个子集中样本均只有一个类别，既满足生成步骤中的第(1)步，故此时可以得到第二次划分后的决策树，如图 8-4 所示。

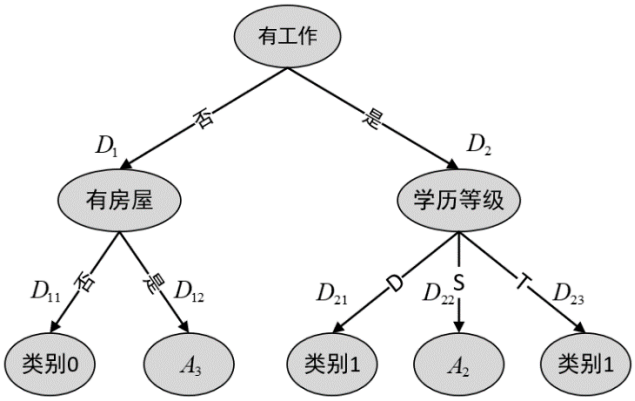


图 8-4 第二次划分

由于子集 D_{12}, D_{22} 均不满足终止条件(2)(4)，且此时两个子集中均只有一个特征可以选择，所以并不需要再进行比较直接划分即可得，如表 8-4 所示。

表 8-4 第二次划分表

样本	4	6	7	2	3	1	5	10	11	14	15	8	9	12	13
$X^{(1)}$	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$X^{(2)}$	0	0	0	1	1	1	1	0	1	0	0	0	0	1	1
$X^{(3)}$	T	T	D	S	S	T	T	D	D	S	S	T	T	T	T
Y	0	0	0	1	1	1	0	1	1	0	1	1	1	1	1
第一次划分	D_1							D_2							
第二次划分	D_{11}			D_{12}				D_{21}		D_{22}		D_{23}			
第三次划分	D_{11}			D_{121}		D_{122}		D_{21}		D_{221}		D_{23}			

根据表 8-4 中的结果可知，子集 D_{121} 满足生成步骤中的第(1)步，故此时可以得到第三次划分后的决策树，如图 8-5 所示。

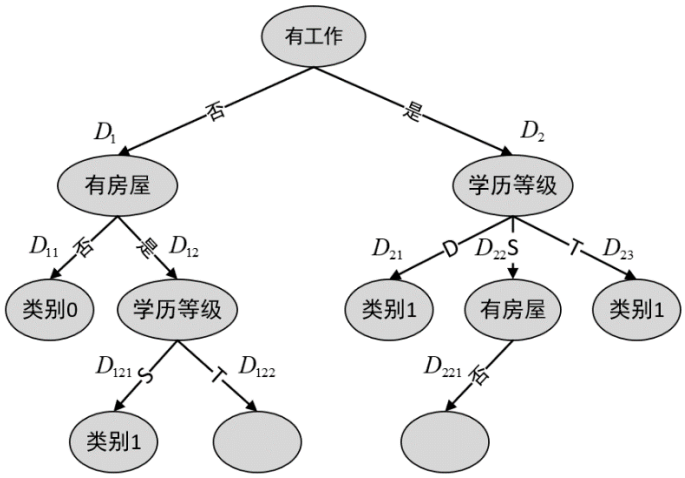


图 8-5 第三次划分

此时，由于子集 D_{122}, D_{221} 满足生成步骤中第 (2)步的终止条件，即再无特征可以进行划分，需要选择样本数最多的类别作为该节点的类别进行返回。但巧合的是子集 D_{122}, D_{221} 中不同类别均只有一个样本，因此随机选择一个类别即可。不过在实际过程中很少会出现这样的情况，因为一般当节点的样本数小于某个阈值时也会停止继续划分。这样便能得到最终生成的决策树，如图 8-6 所示。

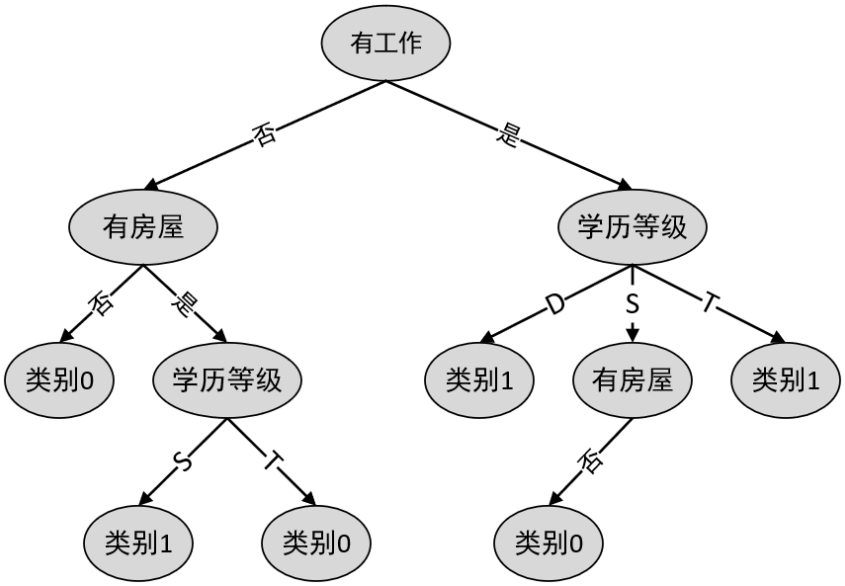


图 8-6 完整决策树

如上就是通过 ID3 算法生成整个决策树的详细过程。根据生成步骤可以发现，如果单纯以 $g(D,A)$ 作为标准的话，会存在模型倾向于选择取值较多的特征进行划分（比如上面的 A_2 ）。虽然在上面这个例子中不存在，但是我们仍可以从直观上理解为什么 ID3 会倾向于选取特征值取值较多的特征。由于 $g(D,A)$ 的直观意义是 D 被 A 划分后不确定性的减少量，因此可想而知当 A 的取值情况越多，那么 D 会被划分得到的子集就越多，于是其不确定性自然会减少得越多，从而 ID3 算法会倾向于选择取值较多的特征进行划分。可以想象，在这样的情况下最终得到的决策树将会是一颗很胖很矮的决策树，进而导致最后生成的决策树容易出现过拟合现象。

8.2.4 C4.5 生成算法

为了解决 ID3 算法的弊端，进而产生了 C4.5 算法。C4.5 算法与 ID3 算法相似，不同之处仅在于 C4.5 算法在选择特征的时候采用了信息增益比作为标准，即选择信息增益比最大的特征作为当前样本集合的根节点。具体为，特征 A 对训练集 D 的信息增益比 $g_R(D,A)$ 定义为其信息增益 $g(D,A)$ 与其训练集 D 关于特征 A 的信息熵 $H_A(D)$ 之比，即

$$g_R(D,A)=\frac{g(D,A)}{H_A(D)} \tag{8-23}$$

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。

因此，8.2.3 节的例子中，对于选取根节点时其增益比计算如下：

1) 计算得到信息增益

$$\begin{aligned} g(D,A_1)&=0.918-0.75=0.168\\ g(D,A_2)&=0.918-0.81=0.108\\ g(D,A_3)&=0.918-0.91=0.08 \end{aligned} \tag{8-24}$$

2) 计算各特征的信息熵

$$
\begin{aligned}
H_{A_1}(D) &= -\sum_{i=1}^{2}\frac{|D_i|}{|D|} = -(\frac{7}{15}\log_2\frac{7}{15}+\frac{8}{15}\log_2\frac{8}{15}) \approx 0.997 \\
H_{A_2}(D) &= -(\frac{8}{15}\log_2\frac{8}{15}+\frac{7}{15}\log_2\frac{7}{15}) \approx 0.997 \\
H_{A_3}(D) &= -(\frac{3}{15}\log_2\frac{3}{15}+\frac{4}{15}\log_2\frac{4}{15}+\frac{8}{15}\log_2\frac{8}{15}) \approx 1.456
\end{aligned}
\tag{8-25}
$$

3) 计算信息增益比

$$
\begin{aligned}
g_R(D,A_1) &= \frac{0.168}{0.997} = 0.169 \\
g_R(D,A_2) &= \frac{0.108}{0.997} = 0.108 \\
g_R(D,A_3) &= \frac{0.08}{1.456} = 0.055
\end{aligned}
\tag{8-26}
$$

根据式(8-26)的计算结果，此时应该以特征 A_1 的各个取值对样本集合 D 进行划分。

由此，可以将利用 C4.5 算法来生成决策树的过程总结如下：

输入：训练数据集 D，特征集 A，阈值 ε；

输出：决策树。

(1) 若 D 中所有样本属于同一类 C_k，则 T 为单节点树，并将 C_k 作为该节点的类标记，返回 T;

(2) 若 $A=\varnothing$，则 T 为单节点树，并将 D 中样本数最多的类 C_k 作为该 节点的类标记，返回 T；

(3) 否则，计算 A 中各特征对 D 的信息增益比，选择信息增益比最大的特征 A_g；

(4) 如果 A_g 的信息增益比小于阈值 ε，则置 T 为单节点树，并将 D 中样本数最多的类 C_k 作为该节点的类标记，返回 T；

(5) 否则，对 A_g 的每一个可能值 a_i，以 $A_g = a_i$ 将 D 分割为若干非空子集，并建立为子节点；

(6) 对于第 i 个子节点，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用(1)-(5)，得到子树 T_i，并返回 T_i

从以上生成步骤可以看出，C4.5 算法与 ID3 算法的唯一为别就是选择的标准不同，而其它的步骤均一样。到此为止，我们就学习完了决策树中的 ID3 与 C4.5 生成算法。在 8.3 节中，将来通过 sklearn 来完成决策树的建模工作。

8.2.5 特征划分

在经过上述两个小节的介绍之后，相信读者朋友们对于如何通过 ID3 和 C4.5 算法来构造一棵简单的决策树已经有了基本的了解。不过细心的读者可能会有这样一个疑问，那就是如何来处理连续型的特征变量。

从上面的示例数据集可以看出，工作、房子、学历这三个都属于离散型的特征变量（Discrete Variable），即每个特征的取值都属于某一个类别；而通常在实际建模过程中，更多的会是连续型的特征变量（Continuous Variable），例如年龄、身高等。在 sklearn 所实现

的决策树算法中，对于这种连续型的特征变量，其具体做法便是先对其进行排序处理，然后取所有连续两个值的均值来离散化整个连续型特征变量[①]。

假设现在某数据集其中一个特征维度为

$$[0.5, 0.2, 0.8, 0.9, 1.2, 2.1, 3.2, 4.5] \tag{8-27}$$

则首先需要对其进行排序处理，排序后的结果为

$$[0.2, 0.5, 0.8, 0.9, 1.2, 2.1, 3.2, 4.5] \tag{8-28}$$

接着再计算所有连续两个值之间的平均值

$$[0.35, 0.65, 0.85, 1.05, 1.65, 2.65, 3.85] \tag{8-29}$$

这样，便得到了该特征离散化后的结果。最后在构造决策树时，只需要使用式(8-29)中离散化后的特征进行划分指标的计算即可。同时，值得一说的地方是目前 sklearn 在实际处理时，把所有的特征均看作连续型变量在进行处理。

8.2.6 小结

在本节中，笔者首先回顾了决策树中几个重要的基本概念，并同时进行了相关示例计算；接着介绍了如何通过信息增益这一划分标准（即 ID3 算法）来构造生成决策树，并以一个真实的例子进行了计算示例；然后，介绍了通过引入信息增益比（即 C4.5 算法）这一划分标准来解决 ID3 算法在生成决策树时所存在的弊端；最后，介绍了在决策树生成时，如何处理连续型特征变量的一种常用方法。

8.3 决策树生成与可视化

在清楚决策树的相关生成算法后，再利用 sklearn 建模就变得十分容易了。下面使用的依旧是前面介绍的 iris 数据集，完整代码见/Chapter08/01_decision_tree_ID3.py 文件。

8.3.1 ID3 算法示例代码

在正式建模之前，笔者先来对 sklearn 中类 DecisionTreeClassifier 里的几个常用参数进行简单的介绍。

```
def __init__(self, *,
    criterion="gini",
    splitter="best",
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features=None,
    min_impurity_split=None):
```

在上述代码中，criterion 用来选择划分时的度量标准，当 criterion="entropy"时表示使用信息增益作为划分指标；splitter 用来选择节点划分时的特征选择策略，当 splitter="best"时则每次节点进行划分时均在所有特征中通过度量标准来选择最优划分方式，而当 splitter="random"时则每次节点进行划分时只会随机选择 max_features 个特征，并在其中选择最优划分方式；max_depth 表示决策树的最大深度，默认为 None 表示直到所有叶子节点的样本均为同一类别或者是样本数小于 min_samples_split 时停止划分；min_samples_leaf 指定构成一个叶子节点所需要的最少样本数，即如果划分后叶子节点中的样本数小于该阈值，则不会进行划分；min_impurity_split 用来提前停止节点划分的阈值，默认为 None 即无阈值。

[①] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

1) 载入数据集

在介绍完类 DecisionTreeClassifier 的基本用法后，便可以通过其来完成决策树的生成。首先需要载入训练模型时所用到的数据集，同时为了后续更好的观察可视化后的决策树，所以这里也要返回各个特征的名称，代码如下：

```
1 def load_data():
2     data = load_iris()
3     X, y = data.data, data.target
4     feature_names = data.feature_names
5     X_train, X_test, y_train, y_test = /
6         train_test_split(X, y, test_size=0.3, random_state=42)
7     return X_train, X_test, y_train, y_test, feature_names
```

在上述代码中，第 4 行代码便是得到特征维度的名称，其结果为：

```
1 ['sepal length(cm)','sepal width(cm)','petal length (cm)','petal width (cm)']
```

2) 训练模型

在完成数据载入后，便可通过类 DecisionTreeClassifier 来完成决策树的生成。这里除了指定划分标准为'entropy'之外（即使用 ID3 算法），其它参数保持默认即可：

```
1 def train(X_train, X_test, y_train, y_test, feature_names):
2     model = tree.DecisionTreeClassifier(criterion='entropy')
3     model.fit(X_train, y_train)
4     print("在测试集上的准确率为：",model.score(X_test, y_test))
```

训练完成后，可以得到模型在测试集上的准确率为：

```
1 在测试集上的准确率为： 1.0
```

8.3.2 决策树可视化

当拟合完成决策树后，还可以借助第三方工具 graphviz[①]来对生成的决策树进行可视化。具体的，需要下载页面中 Windows 环境下的 ZIP 压缩包 graphviz-2.46.1-win32.zip。在下载完成并解压成功后，可以得到一个名为 Graphviz 的文件夹。接着将文件夹 Graphviz 里的 bin 目录添加到环境变量中。步骤为右击“此电脑”，点击“属性”，再点击“高级系统设置”，继续点击“环境变量”，最后双击系统变量里的 Path 变量并新建一个变量输入内容为 Graphviz 中 bin 的路径即可，例如笔者添加时的路径为 C: /graphviz-2.46.1-win32/Graphviz/bin。

添加完成环境变量后，再安装 graphviz 包即可完成可视化的前期准备工作，命令为

```
1 pip install graphviz
```

要实现决策树的可视化，只需要在 8.3.1 节中 train()函数后添加如下代码即可

```
1     dot_data = tree.export_graphviz(model, out_file=None,
2                                     feature_names=feature_names,
3                                     filled=True,rounded=True,
4                                     special_characters=True)
5     graph = graphviz.Source(dot_data)
6     graph.render('iris')
```

① http://www.graphviz.org/download/

在整个代码运行结束后，便会在当前目录中生成一个名为 iris.pdf 的文件，这就是可视化后的结果，如图 8-7 所示。

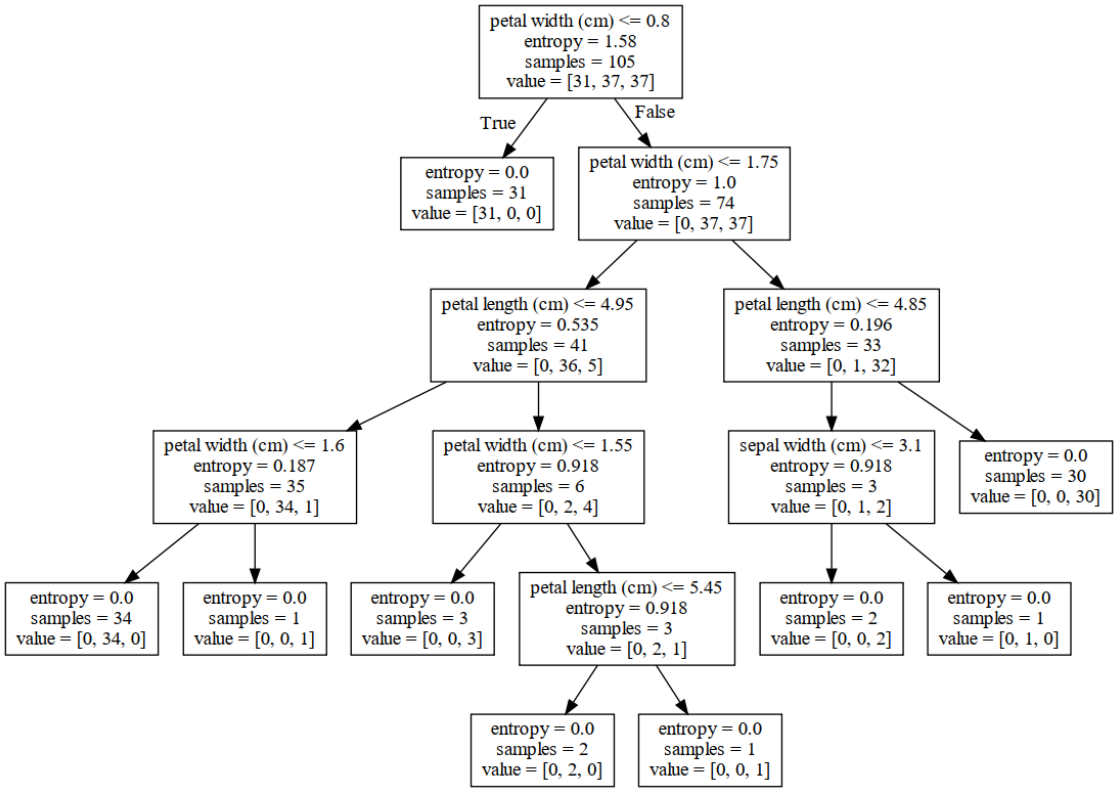


图 8-7 决策树可视化结果

在图 8-7 中，samples 表示当前节点的样本数，value 为一个列表表示每个类别对应的样本数。从图中可以看出，随着决策树不断向下分裂，每个节点对应的信息熵总体上也在逐步减小，直到最终变成 0 结束。

8.3.3 小结

在本节中，笔者首先介绍了类 DecisionTreeClassifier 的使用方法，包括其中一些常见的重要参数及其含义；接着介绍了如何根据现有的数据集来训练一个决策树模型；最后介绍了如何利用开源的 graphviz 工具来实现决策树的可视化。

8.4 决策树剪枝

8.4.1 剪枝思想

在 8.2 节内容中笔者介绍到，使用 ID3 算法进行构建决策树时容易产生过拟合现象，因此我们需要使用一种方法来缓解这一现象。通常，决策树过拟合的表现形式为这棵树有很多的叶子节点。想象一下如果这棵树为每个样本点都生成一个叶节点，那么也就代表着这棵树能够拟合所有的样本点，因为决策树的每个叶节点都表示一个分类类别。同时，出现过拟合的原因在于模型在学习时过多地考虑如何提高对训练数据的正确分类，从而构建出了过于复杂的决策树。因此，解决这一问题的办法就是考虑减少决策树的复杂度，对已经生成的决策树进行简化，也就是剪枝（Pruning）。

8.4.2 剪枝步骤

决策树的剪枝往是通过最小化决策树整体的损失函数或者代价函数来实现。设树T的叶节点个数为$|T|$，t是树T的一个叶节点，该叶节点有N_t个样本点，其中类别k的样本点有N_{tk}个，$k=1,2,...,K$，$H_t(T)$为叶节点t上的经验熵，$\alpha \geq 0$为参数，则决策树的损失函数可以定义为

$$C_\alpha(T)=\sum_{t=1}^{|T|}N_tH_t(T)+\alpha|T| \tag{8-30}$$

其中经验熵为

$$H_t(T)=-\sum_k \frac{N_{tk}}{N_t}\log\frac{N_{tk}}{N_t} \tag{8-31}$$

进一步有

$$C(T)=\sum_{t=1}^{|T|}N_tH_t(T)=-\sum_{t=1}^{|T|}\sum_{k=1}^{K}N_{tk}\log\frac{N_{tk}}{N_t} \tag{8-32}$$

此时损失函数可以写为

$$C_\alpha(T)=C(T)+\alpha|T| \tag{8-33}$$

其中$C(T)$表示模型对训练数据的分类误差，即模型与训练集的拟合程度，$|T|$表示模型复杂度，参数$\alpha \geq 0$控制两者之间的平衡。较大的α促使选择较简单的模型（树），较小的α促使选择较复杂的模型（树）。$\alpha=0$意味着只考虑模型与训练集的拟合程度，不考虑模型的复杂度。可以发现，这里α的作用就类似于正则化中惩罚系数的作用。

具体的，决策树的剪枝步骤如下

输入：生成算法产生的整个树T，参数α；

输出：修剪后的子树T_α

(1) 计算每个叶节点的经验（信息）熵；

(2) 递归地从树的叶节点往上回溯；设一组叶节点回溯到其父节点之前与之后的整体树分别为T_B,T_A，其对应的损失函数值分别是$C_\alpha(T_B),C_\alpha(T_A)$，如果$C_\alpha(T_A)\leq C_\alpha(T_B)$则进行剪枝，即将父节点变为新的叶节点。

(3) 返回(2)，直到不能继续为止，得到损失函数最小的子树T_α。

当然，如果仅看这些步骤依旧会很模糊，下面笔者再来通过一个实际计算示例进行说明。

8.4.3 剪枝示例

如图 8-8 所示，在考虑是否要减掉“学历等级”这个节点时，首先需要计算的就是剪枝前的损失函数数值$C_\alpha(T_B)$。由于剪枝时，每次只考虑一个节点，所以在计算剪枝前和剪枝后的损失函数值时，仅考虑该节点即可。因为其它叶节点的经验熵对于剪枝前和剪枝后都没有变化。

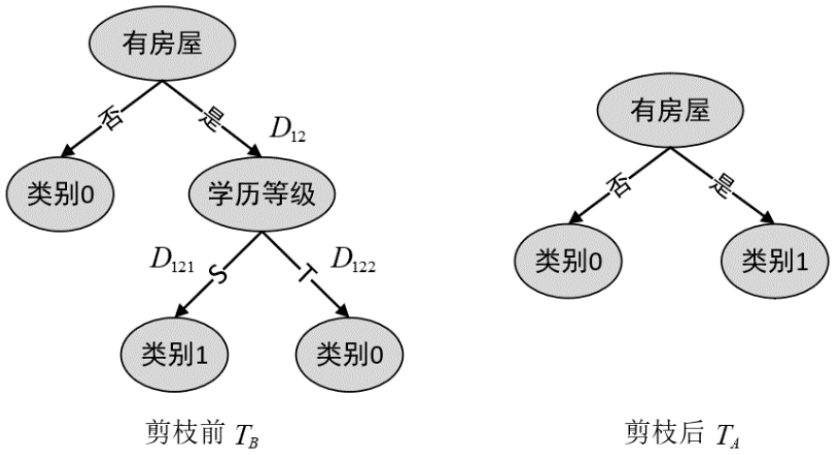


图 8-8 决策树剪枝图

根据表 8-3 可知，“学历等级”这个节点对应的训练数据如表 8-5 所示。

表 8-5 学历等级样本分布表

样本	4	6	7	2	3	1	5
$X^{(1)}$	0	0	0	0	0	0	0
$X^{(2)}$	0	0	0	1	1	1	1
$X^{(3)}$	T	T	D	S	S	T	T
Y	0	0	0	1	1	1	0
第一次划分	D_1						
第二次划分	D_{11}			D_{12}			
第三次划分	D_{11}			D_{121}		D_{122}	

根据式(8-32)有

$$C(T_B)=-\sum_{t=1}^{2}\sum_{k=1}^{2}N_{tk}\log\frac{N_{tk}}{N_t}=-\left((2\log_2\frac{2}{2}+0)+(1\log_2\frac{1}{2}+1\log_2\frac{1}{2})\right)=2 \quad (8\text{-}34)$$

进一步，根据式(8-33)有

$$C_\alpha(T_B)=C(T_B)+\alpha|T_B|=2+2\alpha \quad (8\text{-}35)$$

同理可得，剪枝完成后树T_A损失为

$$C_\alpha(T_A)=-\left(3\log_2\frac{3}{4}+1\log_2\frac{1}{4}\right)+\alpha\approx 3.25+\alpha \quad (8\text{-}36)$$

由(8-35)(8-36)的结果可知，当设定$\alpha\geq 1.25$时决策树便会执行剪枝操作，因为此时$C_\alpha(T_A)=3.25+\alpha\leq C_\alpha(T_B)=2+2\alpha$满足剪枝条件。

从上述过程可以发现，通过剪枝来缓解决策树的过拟合现象算是一种事后补救的措施，即先生成决策树，然后进行简化处理。但实际上，还可以在决策树生成时就施加相应的条件来避免产生过拟合的现象。例如限制树的深度、限制每个叶节点的最少样本数等，当然这些都可以通过网格搜索来进行参数寻找。在图 8-7 所示的决策树中，如果将叶节点的最少数量设置为 min_samples_leaf=10，那么便可以得到一个更加简单的决策树，如图 8-9 所示。

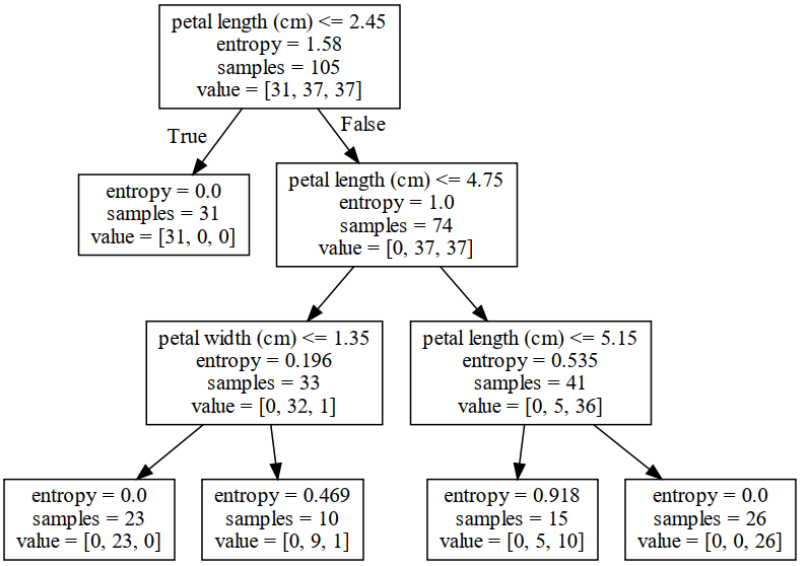


图 8-9 决策树简化后结果

8.4.4 小结

在本节中，笔者首先介绍了决策树中的过拟合现象，阐述了为什么决策树会出现过拟合的现象；然后介绍了可以通过对决策树剪枝来实现缓解决策树过拟合的现象；最后进一步介绍了决策树剪枝的原理以及详细的剪枝计算过程，并且还提到可以通过在构建决策树时施加相应限制条件的方法来避免决策树产生过拟合现象。

8.5 CART 生成与剪枝算法

在 8.4 节中，笔者分别介绍了用 ID3 和 C4.5 这两种算法来生成决策树。其中 ID3 算法每次用信息增益最大的特征来划分数据样本，而 C4.5 算法每次用信息增益比最大的特征来划分数据样本。接下来，再来看另外一种采用基尼不纯度（Gini Impurity）为标准的划分方法，CART 算法。

8.5.1 CART 算法

分类与回归树（Classification and Regression Tree, CART），是一种既可以用于分类也可以用于回归的决策树，同时它也是应用最为广泛的决策树学习方法之一。CART 假设决策树是二叉树，内部节点特征的取值均为“是”和“否”，左分支取值为“是”，右分支取值为“否”。这样，决策树在构建过程中就等价于递归地二分每个特征，将整个特征空间划分为有限个单元[①]。

在本书中，笔者暂时只对其中的分类树进行介绍。总体来说，利用 CART 算法来构造一棵分类需要完成两步：①基于训练数据集生成决策树，并且生成的决策树要尽可能的大；②用验证集来对已生成的树进行剪枝并选择最优子树。

8.5.2 分类树生成算法

在介绍分类树的生成算法前，让我们先来看一看新引入划分标准基尼不纯度。在分类问题中，假设某数据集包含有 K 个类别，样本点属于第 k 类的概率为 p_k，则概率分布的基尼不纯度定义为

[①] 李航，统计机器学习，清华大学出版社

$$Gini(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2 \tag{8-37}$$

因此，对于给定的样本集合D，其基尼不纯度为

$$Gini(D)=1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2 \tag{8-38}$$

其中，C_k是D中属于第k类的样本子集，$|C_k|$表示类别k中的样本数，K是类别的个数。

从基尼不纯度的定义可以看出，若集合D中存在样本数的类别越多，那么其对应的“不纯度”也就会越大，直观的说也就是该集合“不纯”，这也很类似于信息熵的性质。相反，若是该集合中只存在一个类别，那么其对应的基尼不纯度就会是0。因此，在通过CART算法构造决策树时，会选择使基尼不纯度达到最小值的特征取值进行样本划分。

同时，在决策树的生成过程中，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,D_2两个部分，即

$$D_1=\{(x,y)\in D\,|\,A(x)=a\},D_2=D-D_1 \tag{8-39}$$

则在特征A的条件下，集合D的基尼不纯度定义为：

$$Gini(D,A)=\frac{|D_1|}{|D|}Gini(D_1)+\frac{|D_2|}{|D|}Gini(D_2) \tag{8-40}$$

其中$Gini(D,A)$表示经$A=a$分割后集合D的不确定性。可以看出这类似于条件熵，$Gini(D,A)$越小则表示特征A越能降低集合D的不确定性。

在介绍完成基尼不纯度后，就能够列出CART分类树的生成步骤：

输入：训练数据集D，停止计算条件；

输出：CART分类决策树。

根据训练集，从根节点开始，递归地对每个节点进行如下操作，并构建二叉决策树。

(1) 设训练集为D，根据式(8-38)计算现有特征对该数据集的基尼不纯度。接着，对于每一个特征A，对其可能的每一个值a，根据样本点对$A=a$是否成立将D划分成D_1,D_2两个部分，然后再利用式(8-40)计算$A=a$时的基尼不纯度；

(2) 在所有可能的特征A以及它们所有可能的切分点a中，选择基尼不纯度最小的特征取值作为划分标准将原有数据集划分为两个部分，并分配到两个子节点中去；

(3) 对两个子节点递归的调用步骤(1)和(2)，直到满足停止条件；

(4) 生成CART决策树。

其中，算法停止计算的条通常是节点中的样本点个数小于设定阈值，或样本集合的基尼不纯度小于设定阈值，亦或是没有更多特征，这一点同8.2节中ID3和C4.5算法的停止条件类似。

8.5.3 分类树生成示例

在介绍完上述理论性的内容后，这里我们同样还是拿之前的数据集来对具体的生成过程进行详细的示例。由表8-1中的数据可知，此时的其基尼不纯度为

$$Gini(D)=1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2=1-\left(\frac{5}{15}\right)^2-\left(\frac{10}{15}\right)^2\approx 0.444 \tag{8-41}$$

进一步，对于特征 A_1 来说，根据其取值是否为 1，可以将原始样本划分为 D_1 和 D_2 两个部分。由式(8-40)得

$$Gini(D,A_1=1)=\frac{7}{15}Gini(D_1)+\frac{8}{15}Gini(D_2)=\frac{7}{15}\times\frac{24}{49}+\frac{8}{15}\times\frac{14}{64}\approx 0.345 \quad (8\text{-}42)$$

同理，对于特征 A_2 来说，根据其取值是否为 1，也可以将原始样本划分为 D_1 和 D_2 两个部分。此时有

$$Gini(D,A_2=1)=\frac{8}{15}Gini(D_1)+\frac{7}{15}Gini(D_2)=\frac{8}{15}\times\frac{1}{2}+\frac{7}{15}\times\frac{12}{49}\approx 0.381 \quad (8\text{-}43)$$

进一步，对于特征 A_3 来说，根据其分别取值是否为 D,S,T ,每一次也可将原始样本划分为 D_1 和 D_2 两个部分。此时有

$$Gini(D,A_3=D)=\frac{12}{15}Gini(D_1)+\frac{3}{15}Gini(D_2)=\frac{12}{15}\times\frac{4}{9}+\frac{3}{15}\times\frac{4}{9}\approx 0.444 \quad (8\text{-}44)$$

$$Gini(D,A_3=S)=\frac{11}{15}Gini(D_1)+\frac{4}{15}Gini(D_2)=\frac{11}{15}\times\frac{56}{121}+\frac{4}{15}\times\frac{3}{8}\approx 0.439 \quad (8\text{-}45)$$

$$Gini(D,A_3=T)=\frac{7}{15}Gini(D_1)+\frac{8}{15}Gini(D_2)=\frac{7}{15}\times\frac{20}{49}+\frac{8}{15}\times\frac{30}{64}\approx 0.440 \quad (8\text{-}46)$$

注意：每次划分时都是将样本集合划分为两个部分，即 $A_i=a$ 和 $A_i\neq a$

由以上计算结果可知，使用 $A_1=1$ 对样本集合进行划分所得到的基尼不纯度最小。故，根节点应该以 $A_1=1$ 是否成立来进行分割，如图 8-10 所示。

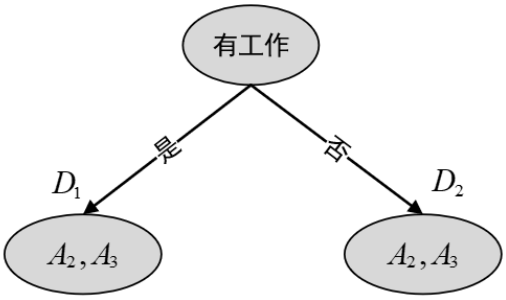


图 8-10 CART 第一次划分

经过这次划分后，原始的样本集合就被特征“有工作”分割成了左右两个部分。接下来，再对左右两个集合递归的进行上述步骤，最终便可以得到通过 CART 算法生成的分类决策树，如图 8-11 所示。

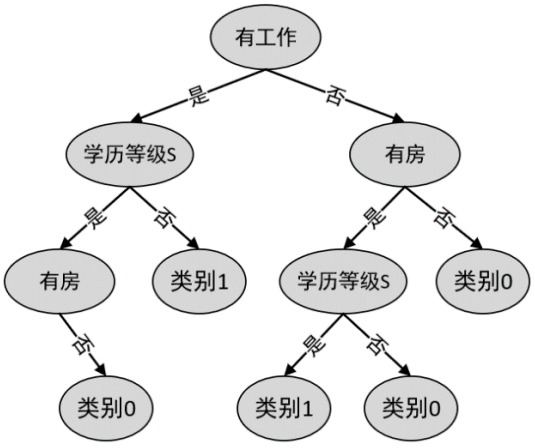


图 8-11 CART 分类决策树

8.5.4 分类树剪枝步骤

在 8.5.3 节中，笔者介绍了 CART 中决策树的生成算法，接下来再来看看在 CART 中如何对生成后的决策树进行剪枝。根据第 4 章内容的介绍可知，总体上来说模型（决策树）越复杂，越容易产生过拟合现象，此时对应的代价函数值也相对较小。在决策树中，遇到这种情况时也就需要进行剪枝处理。CART 剪枝算法由两部组成：

(1) 首先是从之前生成的决策树T_0底端开始不断剪枝，直到T_0的根节点，形成一个子序列$\{T_0,T_1,...,T_n\}$；

(2) 然后通过交叉验证对这一子序列进行测试，从中选择最优的子树。

从上面的两个步骤可以看出，第二步并没有什么难点，关键就在于如何通过剪枝来生成这样一个决策树子序列。

- 剪枝，形成一个子序列

  在剪枝过程中，计算子树的损失函数

$$C_\alpha(T)=C(T)+\alpha|T| \tag{8-47}$$

其中，T为任意子树，$C(T)$为对训练集的预测误差，$|T|$为子树的叶节点个数，$\alpha\geq 0$为参数。需要指出的是不同与之前 ID3 和 C4.5 中剪枝算法的α，前者是人为设定，而此处则是通过计算得到。

具体地，从整体树T_0开始剪枝，如图 8-12 所示。

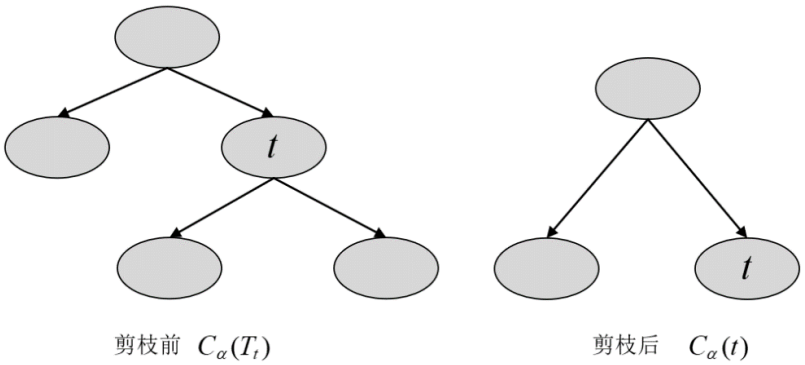


图 8-12 CART 剪枝图

对于T_0的任意内部节点t，以t为根节点的子树T_t（可以看作是剪枝前）的损失函数为

$$C_\alpha(T_t)=C(T_t)+\alpha|T_t| \tag{8-48}$$

同时，以t为单节点的子树（可以看作是剪枝后）的损失函数为

$$C_\alpha(t)=C(t)+\alpha\cdot 1 \tag{8-49}$$

① 当$\alpha=0$或者极小的时候，有不等式

$$C_\alpha(T_t)<C_\alpha(t) \tag{8-50}$$

不等式成立的原因是因为，当$\alpha=0$或者极小的时候，起决定作用的就是预测误差$C(t)$和$C(T_t)$，而模型越复杂其训练误差总是越小的，因此不等式成立。

② 当α增大时，在某一α有

$$C_\alpha(T_t)=C_\alpha(t) \tag{8-51}$$

等式成立的原因是因为，当α慢慢增大时，就不能忽略模型复杂度所带来的影响（也就是式(8-47)第二项）。所以总会存在一个取值α使得等式(8-51)成立。

③ 当α再增大时，不等式(8-50)反向。

因此，当$C_\alpha(T_t)=C_\alpha(t)$时，有$\alpha=\dfrac{C(t)-C(T_t)}{|T_t|-1}$，此时的子树$T_t$和单节点树$t$有相同的损失函数值，但$t$的节点少模型更简单，因此$t$比$T_t$更可取，即对$T_t$进行剪枝。

为此，对决策树T_0中每一个内部节点t来说，都可以计算

$$g(t)=\frac{C(t)-C(T_t)}{|T_t|-1} \tag{8-52}$$

它表示剪枝后整体损失函数减少的程度。因为每个$g(t)$背后都对应着一个决策树模型，而不同的$g(t)$则表示损失函数变化的不同程度。接着，在树T_0中减去$g(t)$最小的子树T_t，将得到的子树作为T_1。如此剪枝下去，直到得到根节点。

注意，此时得到的一系列$g(t)$即α，都能使得在每种情况下剪枝前和剪枝后的损失值相等，因此按照上面第②种情况中的规则要进行剪枝。但为什么是减去其中$g(t)$最小的呢？

对于树T来说，其内部可能的节点t有t_0,t_1,t_2,t_3；t_i表示其中任意一个，如图 8-13 所示。

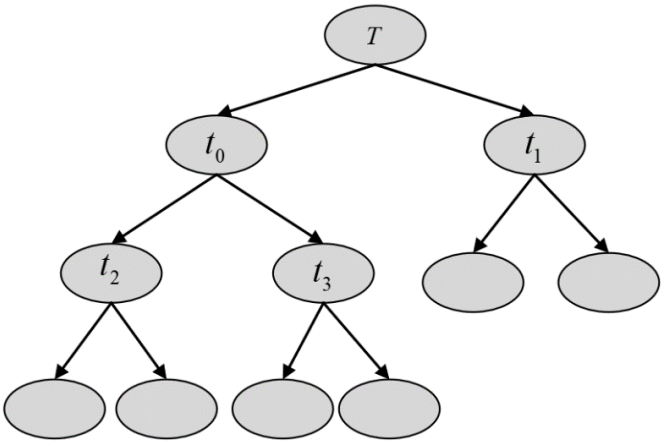


图 8-13 CART 剪枝过程图

因此便可以计算得到$g(t_0),g(t_1),g(t_2),g(t_3)$，也即对应的$\alpha_0,\alpha_1,\alpha_2,\alpha_3$。从上面的第②种情况可以知道，$g(t)$是根据式(8-52)所计算得到，因此这 4 种情况下t_i比T_{t_i}更可取，都满足剪枝。但是由于以t_i为根节点的子树对应的复杂度各不相同，也就意味着$\alpha_i\neq\alpha_j,(i,j=0,1,2,3;i\neq j)$,即$\alpha_i,\alpha_j$存在着大小关系。又因为，当$\alpha$大的时候，最优子树$T_\alpha$偏小；当$\alpha$小的时候，最优子树$T_\alpha$偏大；并且由于在这 4 中情况下剪枝前和剪枝后损失值都相等（即都满足剪枝条件），因此选择减去其中$g(t)$最小的子树。

接着，在得到子树T_1后，再通过上述步骤对T_1进行剪枝得到T_2。如此剪枝下去直到得到根节点，此时便得到了子树序列$T_0,T_1,T_2,....T_n$。

❑ 交叉验证选择最优子树T_α

通过上一步步我们便可以得到一系列的子树序列$T_0,T_1,...,T_n$，最后再通过交叉验证来选取最优决的策树T_α。

进一步，CART 分类树的剪枝步骤可以总结为[①]：

输入：CART 算法生成的分类树 T_0；

输出：最优决策树 T_α.

(1) 设 $k=0, T=T_0$；

(2) 设 $\alpha=+\infty$；

(3) 对于树 T 来说，自下而上地对其每个内部节点 t 计算 $C(T_t), |T_t|$ 以及

$$g(t)=\frac{C(t)-C(T_t)}{|T_t|-1}, \alpha=min(\alpha, g(t)) \tag{8-53}$$

其中，T_t 表示以 t 为根节点的子树，$C(T_t)$ 是子树 T_t 在训练集上的误差，$|T_t|$ 是子树 T_t 叶节点的个数。

(4) 对 $g(t)=\alpha$ 对应的内部节点 t 进行剪枝，并对叶节点以多数表决法决定其类别得到树 T；

(5) 令 $k=k+1$，$\alpha_k=\alpha$，$T_k=T$；

(6) 如果 T_k 不是由根节点及两个叶节点构成的树，则继续执行步骤(3)；否则令 $T_k=T_n$；

(7) 采用交叉验证在子树序列 $T_0, T_1, ..., T_n$ 中选择最优子树 T_α。

当然，如果仅看这些步骤可能依旧会很模糊，下面笔者再通过一个简单的图示进行说明。

8.5.5 分类树剪枝示例

现在假设通过 CART 算法生成了一棵如图 8-14 所示的决策树。

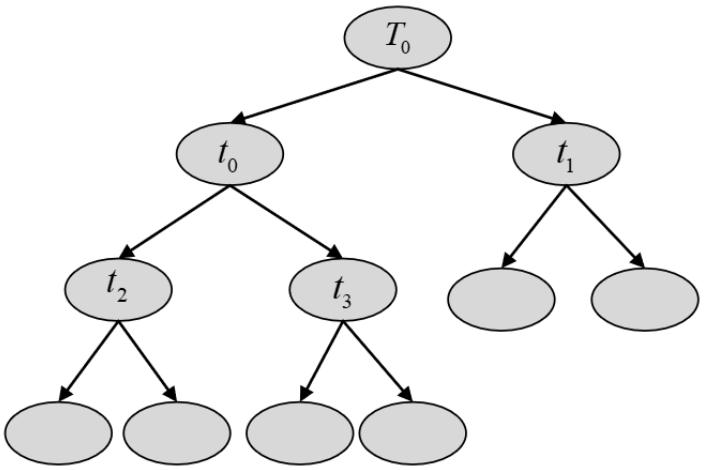


图 8-14 CART 分类子树 T_0

从图 8-14 可以看出，可对树 T_0 进行剪枝的内部节点有 t_0, t_1, t_2, t_3，因此根据剪枝步骤(3)可以分别算出 $g(t_0), g(t_1), g(t_2), g(t_3)$。假设此时算出 $g(t_3)$ 最小，那么根据剪枝步骤(4)便可以得到如图 8-15 所示的子树 T_1，且 $\alpha_1=g(t_3)$。

[①] 李航，统计机器学习，清华大学出版社

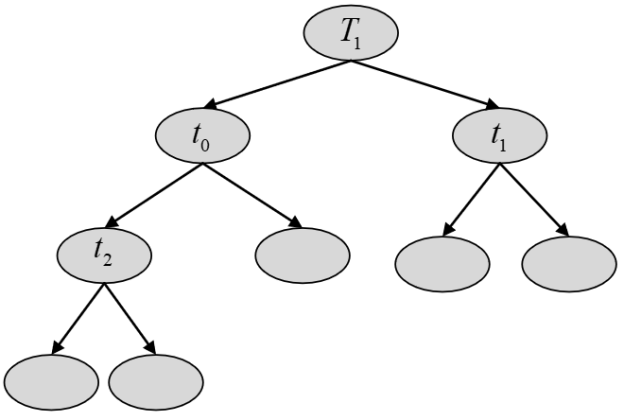


图 8-15 CART 分类子树 T_1

从图 8-15 可以看出，根据剪枝步骤(6)可知需要再次对 T_1 进行剪枝，且此时可对树 T_1 进行剪枝的内部节点有 t_0, t_1, t_2。进一步，根据剪枝步骤(3)可以分别算出 $g(t_0), g(t_1), g(t_2)$。假设此时算出 $g(t_1)$ 最小，那么根据剪枝步骤(4)便可以得到如图 8-16 所示的子树 T_2，且 $\alpha_2 = g(t_1)$。

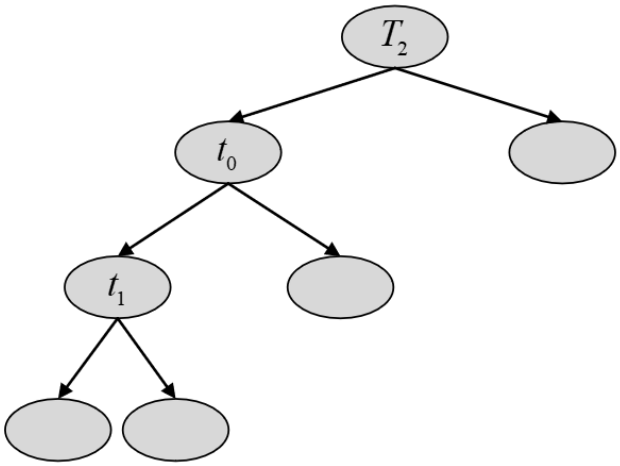


图 8-16 CART 分类子树 T_2

从图 8-16 可以看出，根据剪枝步骤(6)可知需要再次对 T_2 进行剪枝，且此时可对树 T_2 进行剪枝的内部节点有 t_0, t_1。进一步，根据剪枝步骤(3)可以分别算出 $g(t_0), g(t_1)$。假设此时算出 $g(t_0)$ 最小，那么根据剪枝步骤(4)便可以得到如图 8-17 所示的子树 T_3，且 $\alpha_3 = g(t_0)$。

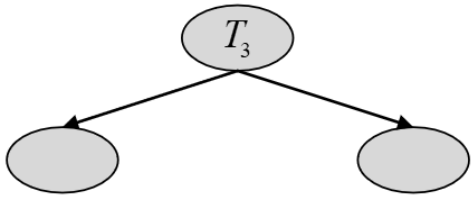


图 8-17 CART 分类子树 T_3

从图 8-17 可以看出，根据剪枝步骤(6)可知满足停止条件不需要继续进行剪枝。到此，便得到了整个子树序列 T_0, T_1, T_2, T_3，接着采用交叉验证在子树序列中选择最优子树 T_α 即可。

到此为止，对于 CART 决策树的整个生成与剪枝过程就介绍完了。最后，通过 sklearn 来完成对于 CART 分类树的使用也很容易，只需要将类 DecisionTreeClassifier 中的划分标准设置为 criterion="gini"即可，其它地方依旧不变。

8.5.6 小结

在本节中，笔者首先介绍了什么是 CART 算法；然后进一步介绍了 CART 分类树中的划分标准基尼不纯度；接着详细介绍了 CART 分类树的生成过程，并通过一个实际例子展示整个决策树生成的计算过程；最后介绍了 CART 分类树剪枝过程的基本原理，并且也通过一个简单的图示展示了整个剪枝过程。

8.6 集成学习

8.6.1 集成学习思想

通过前面几章的学习，我们已经了解了机器学习中的多种分类和回归模型。现在有一个问题就是，这么多模型哪一个最好呢？以分类任务为例，当拿到一个实际的数据集时，如果是你你会选择哪种模型进行建模呢？一个狡猾的办法就是挨个都试一下，那这样做有没有道理呢？还别说，在实际的情况中真的可能会都去试一下。假如现在选择 A、B、C 这三个模型进行建模，最后得到结果是：A 的分类准确率为 0.93，B 的分类准确率为 0.95，C 的准确率为 0.88。那最终应该选择哪一个模型呢？是模型 B 吗？

假设现在一共有 100 个样本，其标签为二分类（正、负两类），三个模型的部分分类结果如表 8-6 所示。

表 8-6 不同模型分类结果对比表

模型	样本 1	样本 2	样本 3	样本 4	样本 5
模型 A	负	正	正	负	正
模型 B	正	负	负	正	负
模型 C	负	正	正	负	正

在表 8-6 中的 5 个样本，模型 A 和 C 均能分类正确，而模型 B 不能分类正确。但如果此时将这三个模型一起用于分类任务的预测，并且对于每一个样本的最终输出结果采用基于投票的规则在三个模型的输出结果中进行选择。例如表 8-6 中的第 1 个样本，模型 A 和 C 均判定为“负类”只有 B 判定为“正类“，则最后的输出便为”负类“。那么此时，我们就有可能得到一个分类准确率为 1 的“混合”模型。

注意：在其余的 95 个样本中，假设根据投票规则均能分类正确。

8.6.2 集成学习种类

在机器学习中，基于这种组合思想来提高模型精度的方法被称为集成学习（Ensemble Learning）。俗话说“三个臭皮匠，赛过诸葛亮”，这句话就完美阐述了集成学习的潜在思想——通过将多个模型结合在一起来提高整体的泛化能力[①]。

常见的集成模型主要包括以下三种：

1) Bagging 集成学习

Bagging 的核心思想为并行地训练一系列各自独立的同类模型，然后再将各个模型的输出结果按照某种策略进行组合，并输出最终结果。例如分类中可采用投票策略，回归中可采用平均策略。通常来说，模型越容易过拟合，则越适用于 Bagging 集成学习方法。

[①] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

2) Boosting 集成学习

Boosting 的核心思想为先串行地训练一系列前后依赖的同类模型，即后一个模型用来对前一个模型的输出结果进行修正，最后再通过某种策略将所有的模型组合起来，并输出最终的结果。通常来说，模型越容易欠拟合，则越适用于 Boosting 集成学习方法。

3) Stacking 集成学习

Stacking 的核心思想为并行地训练一系列各自独立的不同类模型，然后再将各个模型的输出结果作为输入来训练一个新模型（例如：逻辑回归），并通过这个新模型来输出最终预测的结果[①]。通常来说，Stacking 集成学习也适用于欠拟合的机器学习模型。

下面的内容，笔者就来大致介绍一下各类集成模型中常见的算法和使用示例。

8.6.3 Bagging 集成学习

Bagging 的全称为 Bootstrap Aggregation，而这两个单词也分别代表了 Bagging 在执行过程中的两个步骤，①Bootstrap Samples；②Aggregate Outputs。总结起来就是 Bagging 首先从原始数据中随机抽取多组包含若干数量样本的子训练集，以及对于各子训练集来说再随机抽取若干特征维度作为模型输入；然后分别以不同的子训练集来训练得到不同的基模型，同时将各个模型的预测结果进行聚合，即

$$y=\frac{1}{M}\sum_{m=1}^{M}f_m(x) \tag{8-54}$$

其中 M 表示基模型的数量，$f_m(x)$ 表示不同的基模型。

同时，由于 Bagging 的策略是取所有基模型的“平均”值作为最终模型的输出结果，所以 Bagging 集成方法能够很好的降低模型高方差（过拟合）的情况。因此通常来说，在使用 Bagging 集成方法的时候，可以尽量使得每个基模型都出现过拟合的现象。下面，笔者就来介绍在 sklearn 中如何使用 Bagging 集成学习方法。

1) Bagging on KNN

在 sklearn 中，可以通过 from sklearn.ensemble import BaggingClassifier 来导入 Bagging 集成学习方法中的分类模型。下面先来介绍一下 BaggingClassifier 类中常见的重要参数及其含义。

```
def __init__(self,
    base_estimator=None,
    n_estimators=10,
    max_samples=1.0,
    max_features=1.0,
    bootstrap=True,
    bootstrap_features=False,
    n_jobs=None):
```

上述代码是类 BaggingClassifier 初始化方法中的部分参数，其中 base_estimator 表示所使用的基模型；n_estimators 表示需要同时训练多少个基模型；max_samples 表示每个子训练集中最大的样本数量，其可以是整数也可以是 0 到 1 之间的浮点数（此时表示在总样本数中

① https://en.wikipedia.org/wiki/Ensemble_learning

的占比）；max_features 表示子训练集中特征维度的数量（由于是随机抽样，所以不同的子训练集特征维度可能不一样）；bootstrap=True 表示在同一子训练集中同一样本可以重复抽样出现；bootstrap_features=False 表示在同一子训练集中同一特征维度不能重复出现（如果设置为 True，极端情况下所有的特征维度可能都一样）；n_jobs 表示同时要使用多个 CPU 核并行进行计算。

下面以 KNN 作为基模型通过 sklearn 中的 BaggingClassifier 类来进行 Bagging 集成学习建模，完整代码见 Chapter08/02_ensemble_bagging_knn.py 文件。

```
1 bagging = BaggingClassifier(KNeighborsClassifier(n_neighbors=3),
2                                 n_estimators=5,
3                                 max_samples=0.8,
4                                 max_features=3,
5                                 bootstrap_features=False,
6                                 bootstrap=True)
7 bagging.fit(x_train, y_train)
8 print(bagging.estimators_features_)
9 print(bagging.score(x_test, y_test))
```

在上述代码中，第 1 行表示使用了 KNN 作为基模型，且 K 值设置为了 3；第 2-3 行分别表示一共采用了 5 个 KNN 分类器，每个分类器在进行训练时使用原始训练集 80%的样本进行训练；第 4-5 行表示每个子训练集的仅使用其中 3 个特征维度（一共有 4 个），且不能重复；第 6 行表示每个子训练集在划分样本时可以重复。

训练完成后可以得到如下结果：

```
1 [array([1, 3, 0]), array([0, 1, 2]), array([3, 1, 2]), array([0, 1, 2]), array([1, 3, 0])]
2 0.9777
```

其中[1, 3, 0]表示该模型在训练时使用的是第 1、3 和 0 个特征维度，其它同理。

2) Bagging on Decision Tree

正如上面介绍到，在通过 Bagging 方法进行集成学习时其基模型可以是其它任意模型，所以自然而然也可以是决策树。同时，由于对决策树使用 Bagging 集成方法是一个较为热门的研究方向，因此它还有另外一个响亮的名字随机森林（Random Forests）。根据 Bagging 的思想来看，随机森林这个名字也很贴切，一系列的树模型就变成了森林。在 sklearn 中，如果是通过类 BaggingClassifier 来实现 Bagging 集成学习，当参数 base_estimator=None 时，默认就会采用决策树作为基模型。由于这部分的内容较多且应用也比较广泛，所以详细内容将会单独放在 8.7 节中进行介绍。

8.6.4 Boosting 集成学习

Boosting 同 Bagging 一样，都是用于提高模型的泛化能力。不同的是 Boosting 方法是通过串行地训练一系列模型来达到这一目的。在 Boosting 集成学习中，每个基模型都会对前一个基模型的输出结果进行改善。如果前一个基模型对某些样本进行了错误的分类，那么后一个基模型就会针对这些错误的结果进行修正。这样在经过一系列串行基模型的拟合后，最终就会得到一个更加准确的结果。因此， Boosting 集成学习方法经常被用于改善模型高偏差的情况（欠拟合现象）。

在 Boosting 集成学习中最常见的算法就是 AdaBoost，关于该算法的具体原理笔者在这里就暂不阐述，各位读者朋友要是兴趣可以自行去查找相关资料。下面直接来看 AdaBoost 算法在 sklearn 中的用法。

在 sklearn 中，可以通过 from sklearn.ensemble import AdaBoostClassifier 来导入 AdaBoosting 集成学习方法中的分类模型。下面先来介绍一下 AdaBoostClassifier 类中常见的重要参数及其含义。

```
def __init__(self,
    base_estimator=None,
    n_estimators=50,
    learning_rate=1.):
```

上述代码是类 BaggingClassifier 初始化方法中的部分参数，其中 base_estimator 表示所使用的基模型，如果设置为 None 则模型使用决策树；n_estimators 表示基模型的数量，默认为 50 个；learning_rate 用来控制每个基模型的贡献度，默认为 1 即等权重。

下面以决策树作为基模型通过 sklearn 中的 AdaBoostClassifier 类来进行 Boosting 集成学习建模，完整代码见 Chapter08/ 03_AdaBoosting.py 文件。

```
x_train, x_test, y_train, y_test = load_data()
dt = DecisionTreeClassifier(criterion='gini', max_features=4, max_depth=1)
model = AdaBoostClassifier(base_estimator=dt, n_estimators=100)
model.fit(x_train, y_train)
print("模型在测试集上的准确率为：", model.score(x_test, y_test)) # 1.0
```

在上述代码中，第 2 行表示定义决策树基模型；第 3 行表示定义 AdaBoost 分类器，并将决策树基模型作为参数传入到类 AdaBoostClassifier 中。

到此，对于 AdaBoost 的示例用法就介绍完了。不过细心的读者可能会问，此时基分类器决策树和 AdaBoost 均有自己的超参数，如果要在上述训练过程中使用网格搜索 GridSearchCV 该怎么操作呢？关于这部分内容笔者在此就不做介绍，可以直接参见 Chapter08/04_AdaBoosting_gridsearch.py 文件。

8.6.5 Stacking 集成学习

不同于 Bagging 和 Boosting 这两种集成学习方法，Stacking 集成学习方法首先通过训练得到多个基于不同算法的基模型，然后再将通过训练一个新模型来对其它模型的输出结果进行组合。例如选择以逻辑回归和 KNN 作为基模型，以决策树作为组合模型。那么 Stacking 集成方法的做法为：首先将训练得到前两个基模型；然后再以基模型的输出作为决策树的输入训练组合模型；最后以决策树的输出作为真正的预测结果。

在 sklearn 中，可以通过 from sklearn.ensemble import StackingClassifier 来导入 StackingClassifier 集成学习方法中的分类模型。下面先来介绍一下 StackingClassifier 类中常见的重要参数及其含义。

```
def __init__(self,
    estimators,
    final_estimator=None,
    passthrough=False)
```

上述代码是类 StackingClassifier 初始化方法中的部分参数，其中 estimators 表示所使用的基模型；final_estimator 表示最后使用的组合模型；当 passthrough=False 时，表示在训练

最后的组合模型时只将各个基模型的输出作为输入，当 passthrough=True 时表示同时也将原始样本也作为输入。

下面以逻辑回归、K 近邻作为基模型，决策树作为组合模型，并通过 sklearn 中的 StackingClassifier 类来进行 Stacking 集成学习建模，完整代码见 Chapter08/05_ensemble_stacking.py 文件。

```
1 estimators = [('logist', LogisticRegression(max_iter=500)),
2               ('knn', KNeighborsClassifier(n_neighbors=3))]
3 stacking = StackingClassifier(estimators=estimators,
4                               final_estimator=DecisionTreeClassifier())
5 stacking.fit(x_train, y_train)
6 acc = stacking.score(x_test, y_test)
7 print("模型在测试集上的准确率为：", acc)#0.956
```

在上述代码中，第 1-2 行分别用来定义两个基模型，并进行相应的初始化；第 3-4 行用来定义 Stacking 分类器，并指定组合模型为决策树。

到此，对于 sklearn 中 Stacking 集成学习的示例用法就介绍完了。同时，上述训练过程的网格搜索示例用法可以参见 Chapter08/06_ensemble_stacking_gridsearch.py 文件。

8.6.6 小结

在本节中，笔者首先介绍了机器学习中集成学习的基本思想；接着介绍了 3 种常见集成学习方法 Bagging、Boosting 和 Stacking 的基本原理；最后，分别就这 3 种集成学习方法各自在 sklearn 中的示例用法进行了详细介绍。

8.7 随机森林

8.7.1 随机森林原理

正如笔者在第 8.6.3 节中介绍的那样，随机森林本质上也就是基于决策树的 Bagging 集成学习模型。因此，随机森林的建模过程总体上可以分为三步[①]：

第一步，对原始数据集进行随机采样，得到多个训练子集；

第二步，在各个训练子集上训练得到不同的决策树模型；

第三步，将训练得到的多个决策树模型进行组合，然后得到最后的输出结果。

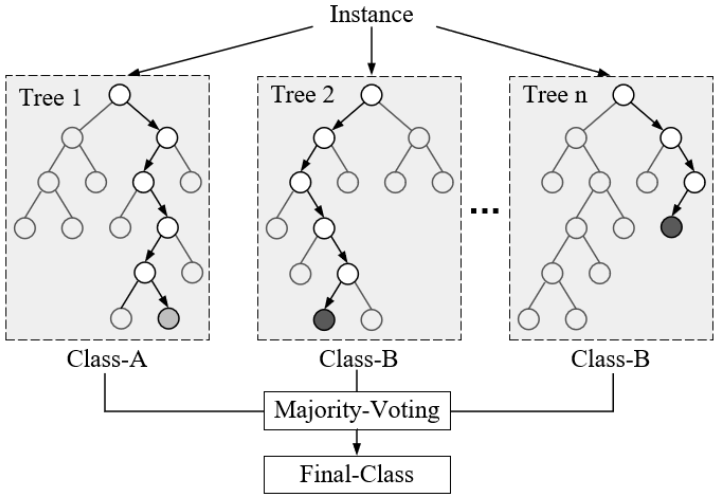


图 8-18 随机森林原理图

[①] https://en.wikipedia.org/wiki/Random_forest

如图 8-18 所示为随机对样本点和特征采样后训练得到的若干决策树模型组成的随机森林。从图中可以看出，即使同一个样本在不同树中所归属的叶子节点也不尽相同，甚至连类别也可能不同。但是这也充分体现了 Bagging 集成模型的优点，通过“平均”来提高模型的泛化能力。

在图 8-18 中，多个不同结构的决策树模型构成了随机森林，并且在模型输出时将会以投票的方式决策出最终的输出类别。同时，随机森林与普通 Bagging 集成学习方法存在的一点差别就是，随机森林中每一个决策树在每次划分节点的过程中，还会有一个随机的过程[①]，即只会从已有的特征中再随机选择部分特征参与节点划分，这一过程也被称为“Feature Bagging”。之所以要这么做是为了减小各个树模型之间的关联性。例如训练数据中存在着某些差异性较大的特征，那么所有的决策树在节点划分时就会选择同样的特征，使得最终得到的决策树之间具有较强的关联性，即每棵树都类似。

8.7.2 随机森林示例代码

介绍完随机森林的基本原理后，我们再来看一看如何通过 sklearn 完成随机森林的建模任务。在 sklearn 中，可以通过 from sklearn.ensemble import RandomForestClassifier 来导入模块随机森林。下面先来介绍一下 RandomForestClassifier 类中常见的重要参数及其含义。

```
def __init__(self,
                 n_estimators=100,
                 criterion="gini",
                 max_depth=None,
                 min_samples_split=2,
                 min_samples_leaf=1,
                 max_features="auto",
                 bootstrap=True,
                 max_samples=None):
```

上述代码是类 RandomForestClassifier 初始化方法中的部分参数，其中 n_estimators 表示在随机森林中决策树的数量；criterion 表示指定构建决策树的算法；max_depth 表示允许决策树的最大深度；min_samples_split 表示节点允许继续划分的最少样本数，即如果划分后的节点中样本数少于该值，将不会进行划分；min_samples_leaf 叶子节点所需要的最少样本数；max_features 表示每次对节点进行划分时候选特征的最大数量，即节点每次在进行划分时会先在原始特征中随机的选取 max_features 个候选特征，然后在候选特征中选择最佳特征；bootstrap 表示是否对原始数据集进行采样，如果为 False 则所有决策树在构造时均使用相同的样本；max_samples 表示每个训练子集中样本数量的最大值（当 bootstrap=True 时），其默认值为 None，即等于原始样本的数量。

注意：max_samples=None 仅仅只是表示采样的样本数等于原始训练集的样本数，不代表抽样后的子训练等同于原始训练集，因为采样时样本可以重复。

一般来说，在 sklearn 的各个模型中，对于大多数参数来说保持默认即可，对于少部分关键参数可采样交叉验证进行选择。

下面以 iris 数据集为例来进行 RandomForestClassifier 的集成学习建模任务，完整代码见 Chapter08/07_ensemble_random_forest.py 文件。

```
if __name__ == '__main__':
    x_train, x_test, y_train, y_test = load_data()
    model = RandomForestClassifier(n_estimators=2, max_features=3,
```

```
4                                   random_state=2)
5     model.fit(x_train, y_train)
6     print(model.score(x_test, y_test)) # 0.95
```

可以看到，尽管随机森林这么复杂的一个模型，在 sklearn 中同样可以通过几行代码来完成建模。同时，在完成随机森林的训练后，可以通过 model.estimators_属性来得到所有的决策树对象，然后分别对其进行可视化就可以得到整个随机森林可视化结果。当然，最重要的是可以通过 model.feature_importances_属性来得到每个特征的重要性程度以进行特征筛选去掉无关特征。

8.7.3 特征重要性评估

从决策树的构造原理便可以看出，越是靠近决策树顶端的特征维度越能够对不同类别的样本进行区分，也就意味着越是接近于根节点的特征维度越重要。因此，在 sklearn 中的类 DecisionTreeClassifier 里面，同样也有 feature_importances_属性来输出每个特征的重要性值。只是通过随机森林来进行特征重要性评估更加准确，因此笔者才将这部分内容放到了这里。不过想要弄清楚随机森林中的特征重要性评估过程，还得从决策树说起。

1) 决策树中的特征评估

在 sklearn 中，决策树是通过基于基尼纯度的减少量来对特征进行重要性评估，当然基尼纯度也可以换成信息增益或者是信息增益比。具体的，对于决策树中划分每个节点的特征来说，其特征重要性计算公式为[①]

$$importance = \frac{N_t}{N} * (impurity - \frac{N_{tL}}{N_t} * left\_impurity - \frac{N_{tR}}{N_t} * right\_impurity) \quad (8\text{-}55)$$

其中 N 表 的样本数；N_t 表示当前节点的样本数；$impurity$ 表示当前节点的纯度；N_{tL} 表示当前节点左孩子中的样本数；$left\_impurity$ 表示当前节点左孩子的纯度；N_{tR} 表示当前节点右孩子中的样本数；$right\_impurity$ 表示当前节点右孩子的纯度。

以 8.7.2 小节随机森林里的其中一棵决策树为例，其在每次进行节点划分时的各项信息如图 8-19 所示。

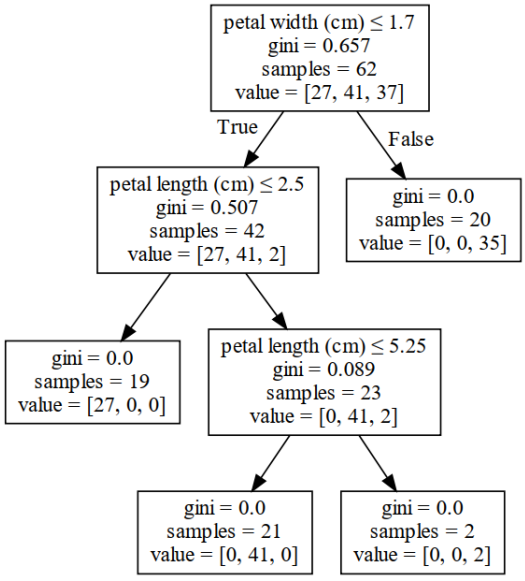


图 8-19 决策树特征重要性评估

[①] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

这里有一个小细节需要注意的地方便是，在图 8-19 中每个节点里 samples 的数量指的是不重复的样本数（因为采样会有重复），而列表 value 中的值则包含有重复样本。例如在根节点中，samples=62 表示一共有 62 个不同的样本点，但实际上该节点中有 105 个样本点，即有 43 个重复出现。

此时，对于特征 petal width 来说，根据式(8-55)其特征重要性值为

$$\frac{105}{105}\times\left(0.657-\frac{70}{105}\times 0.507-\frac{35}{105}\times 0\right)\approx 0.319 \tag{8-56}$$

对于特征 petal length 来说，由于其在两次节点划分中均有参与，所以它的特征重要性为

$$\frac{70}{105}\times\left(0.507-\frac{27}{70}\times 0-\frac{43}{70}\times 0.089\right)+\frac{43}{105}\times(0.089-0)\approx 0.338 \tag{8-57}$$

对于另外两个特征 sepal length 和 sepal width 来说，由于两者并没有参与决策树节点的划分，所以其重要性均为 0。

2) 随机森林中的特征评估

在介绍完决策树中的特征重要性评估后，再来看随机森林中的特征重要性评估过程就相对容易了。在 sklearn 中，随机森林的特征重要性评估主要也是基于多棵决策树的特征重要性结果计算而来，称为平均纯度减少量（Mean Decrease in Impurity,MDI）。MDI 的主要计算过程就是将多棵决策树的特征重要性值取了一次平均。

对于 8.7.2 节中的随机森林来说，其另外一棵决策树在每次进行节点划分时的各项信息如图 8-20 所示。

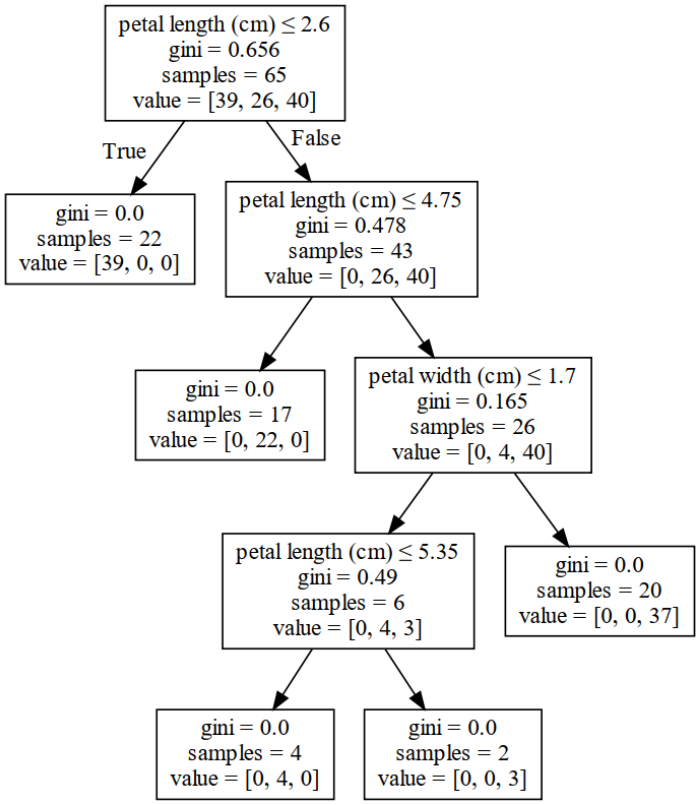


图 8-20 随机森林特征重要性评估

从图 8-20 可以看出，一共有 2 个特征参与到了节点的划分过程中。根据式(8-55)可知，特征 petal length 的重要性为

$$\frac{105}{105}\times\left(0.656-\frac{66}{105}\times 0.478\right)+\frac{66}{105}\times\left(0.478-\frac{44}{66}\times 0.165\right)+\frac{7}{105}\times 0.49\approx 0.619 \quad (8\text{-}58)$$

特征 petal width 的重要性为

$$\frac{44}{105}\times\left(0.165-\frac{7}{44}\times 0.49\right)\approx 0.036 \quad (8\text{-}59)$$

到此为止，对于 8.7.2 节中的随机森林，其两棵决策树对应的计算得到特征重要性如表 8-7 所示。

表 8-7 决策树特征重要性表

特征 / 模型	sepal length	sepal width	petal length	petal width
Tree 1	0	0	0.338	0.319
Tree 2	0	0	0.619	0.036

在 sklearn 中，对于决策树计算得到的特征重要性值默认情况下还会进行标准化，即每个维度均会除以所有维度的和。进一步，对于随机森林来说，其各个特征的重要性值则为所有决策树对应特征重要性的平均值。因此，对于表 8-6 中的结果来说最终每个特征重要性值为 0，0，0.729 和 0.271。关于上述详细的计算过程可以参见 Chapter08/08_ensemble_random_forest_features.py 文件。

从最后的结果可以看出，在数据集 iris 中对分类起决定性作用的为最后两个特征维度。因此，各位读者也可以进行一个对比，只用最后两个维度来进行分类并观察其准确率。完整示例代码见 Chapter08/09_feature_importance_comp.py 文件。

8.7.4 小结

在本节中，笔者首先介绍了随机森林的基本原理；然后介绍了 sklearn 中随机森林模块 RandomForestClassifier 的基本用法以及其中常见参数的作用；最后详细介绍了如何通过随机森林来对特征进行重要性评估，包括具体的计算以及示例代码等。

8.8 泰坦尼克号生还预测

在本章的前面几节内容中，笔者陆续介绍了几种决策树的生成算法以及常见的集成学习方法。在接下来的这节内容中，笔者会将以泰坦尼克号生还预测（分类）[①]为例来进行实战演示；并且还会介绍到相关的数据预处理方法，例如缺失值填充和类型特征转换等。

本次用到的数据集为泰坦尼克号生还预测数据集，原始数据集一共包含 891 个样本，11 个特征维度。但是需要注意的是，在实际处理时这 11 个特征维度不一定都要用到，只选择你认为有用的即可。同时，由于部分样本存在某些特征维度出现缺失的状况，因此还需要对其进行填充。完整代码参见 Chapter08/10_titanic.py 文件。

8.8.1 读取数据集

本次用到的数据集一共包含两个文件，其中一个为训练集，另一个为测试集。下载完成后放在本代码所在目录的 data 目录中即可。接着可以通过如下代码来读入数据：

[①] https://www.kaggle.com/c/titanic/data

```
1 import pandas as pd
2 def load_data():
3     train = pd.read_csv('data/train.csv',sep=',')
4     test = pd.read_csv('./data/test.csv')
```

在上述代码中，第 1 行用来导入库 pandas 用于读取本地文件；第 3 行代码表示通过 pandas 中的 read_csv()方法来读取.csv 文件，它返回的是一个 DataFrame 格式的数据类型，可以方便进行各类数据预处理操作。这里值得一提的是，read_csv 不仅仅可以用来读取.csv 格式的数据，只要读取的数据满足条件：①它是一个结构化的文本数据，即 m 行 n 列；②列与列之间有相同的分隔符，例如默认情况下 sep=','。那么这样的数据都可以通过该方法来进行读取，不管文件的后缀是.csv 还是.txt，亦或是没有后缀。当然，pandas 还提供很多常见数据的读取方法，例如 excel、json 等。

在读取完成后，可以通过如下方式来查看数据集的相关信息：

```
1 print(train.info())
2 Data columns (total 12 columns):
3  #   Column       Non-Null Count  Dtype
4 ---  ------       --------------  -----
5  0   PassengerId  891 non-null    int64
6  1   Survived     891 non-null    int64
7  2   Pclass       891 non-null    int64
8  3   Name         891 non-null    object
9  4   Sex          891 non-null    object
10 5   Age          714 non-null    float64
11 6   SibSp        891 non-null    int64
12 7   Parch        891 non-null    int64
13 8   Ticket       891 non-null    object
14 9   Fare         891 non-null    float64
15 10  Cabin        204 non-null    object
16 11  Embarked     889 non-null    object
```

从上述输出信息可以知道，该数据集一共有 11 个特征维度（第 1 列 Survived 为标签），891 个样本。同时还可以具体的看到每个特征维度的数据类型、是否为非空值等信息。

8.8.2 特征选择

在完成原始数据的载入后，就需要对特征进行选择。对于特征的选择这一步显然是仁者见仁智者见智，可以都用上也可以只选择你认为对最后预测结果有影响的特征。在本示例中，笔者选择的是'Pclass'、'Sex'、 'Age'、 'SibSp'、 'Parch'、'Fare'和'Embarked'这 7 个特征维度，其分别表示船舱等级、性别、年龄、乘客在船上兄弟姐妹/配偶的数量、乘客在船上父母/孩子的数量、船票费用和登船港口。当然， 'Survived'这一列特征是作为最终进行预测的类标。接着通过如下代码便可完成特征的选择工作：

```
1 features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']
2 x_train = train[features]
3 x_test = test[features]
4 y_train = train['Survived']
```

在上述代码中，第 1 行用来定义需要选择的特征；第 2-3 行用来在训练集和测试集中取对应的特征维度；第 4 行用来获取得到训练集中的标签。由于这是比赛中的数据集，所以真实的测试集中并不含有标签。

8.8.3 缺失值填充

在选择完成特征后下一步就是对其中的确实值进行填充。从上面的输出结果可以看出，在训练集和测试集中特征‘Age’、‘Embarked’和‘Fare’存在缺失值的情况。且特征‘Age’和‘Fare’均为浮点型，对于浮点型的缺失值一般可采用该特征维度所有值的平均作为填充；而特征‘Embarked’为类型值，对于类型值的缺失一般可以采用该特征维度出现次数最多的类型值进行填充。因此下面开始分别用这两种方法来进行缺失值的补充。

```
1 x_train['Age'].fillna(x_train['Age'].mean(), inplace=True)
2 print(x_train['Embarked'].value_counts())# S 644 C 168  Q 77
3 x_train['Embarked'].fillna('S', inplace=True)
4 x_test['Age'].fillna(x_train['Age'].mean(), inplace=True)
5 x_test['Fare'].fillna(x_train['Fare'].mean(), inplace=True)
```

在上述代码中，第 1、4 行用来对特征‘Age’以均值进行填充，这里需要注意的是测试集中的缺失值也应该用训练集中的均值来进行填充；第 2 行用来统计输出特征‘Embarked’中各个取值出现的次数，可以发现‘S’出现次数最多（644 次）；第 3 行则用来对特征‘Embarked’的缺失值以‘S’进行填充。

8.8.4 特征值转换

在进行完上述几个步骤后，最后一步需要完成的就是对特征进行转换。所谓特征转换就是将其中的非数值型特征，用数值进行代替，例如特征‘Embarked’和‘Sex’。

```
1 x_train.loc[x_train['Sex'] == 'male', 'Sex'] = 0
2 x_train.loc[x_train['Sex'] == 'female', 'Sex'] = 1
3 x_train.loc[x_train['Embarked'] == 'S', 'Embarked'] = 0
4 x_train.loc[x_train['Embarked'] == 'C', 'Embarked'] = 1
5 x_train.loc[x_train['Embarked'] == 'Q', 'Embarked'] = 2
```

在上述代码中，.loc 方法用来获取对应行列索引中的值，而类似 x_train['Sex'] == 'male' 则是用来得到满足条件的行索引。经过上述步骤后，数据集中的字符特征就被替换成了对应的数值特征。

8.8.5 乘客生还预测

在完成数据预处理的所有工作后，便可以建立相应的分类模型来对测试集中的乘客生还情况进行预测。下面以随机森林模型进行示例。

```
1 def random_forest():
2     x_train, y_train, x_test = load_data()
3     model = RandomForestClassifier()
4     paras = {'n_estimators': np.arange(10, 100, 10),
5              'criterion': ['gini', 'entropy'],
6              'max_depth': np.arange(5, 50, 5)}
7     gs = GridSearchCV(model, paras, cv=5, verbose=2, n_jobs=2)
8     gs.fit(x_train, y_train)
```

```
 9      y_pre = gs.predict(x_test)
10      print('best score:', gs.best_score_) # 0.827
11      print('best parameters:', gs.best_params_)
```

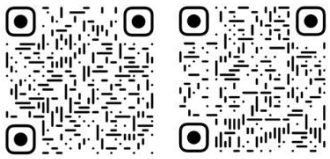
由于上述示例代码在 8.7.2 节中均有介绍，所以在此就不再赘述。

8.8.6 小结

在本节中，笔者以泰坦尼克号生还预测数据集为例，首先介绍了如何通过 pandas 来读取结构化的文本数据；然后详细的展示了从数据预处理到模型预测的每一个步骤，包括读取数据集、特征选择、缺失值补充、特征转换和模型选择等；最后以随机森林为例，完成了随机森林模型的训练以及在测试集上的预测任务。

总结一下，在本章中笔者首先介绍了决策树的基本思想，以及 3 种常见的决策树生成算法，包括 ID3 算法、C4.5 和 CART 算法；然后介绍了集成学习算法的基本思想，并就 3 种常见的集成学习算法 Bagging、Boosting 和 Stacking 进行了简单的介绍和示例；最后，笔者通过一个真实的比赛数据集，详细介绍了从数据预处理到模型训练与预测的全过程。经过以上内容的学习，会使得我们对于如何从零构建一个机器学习模型有了更深的理解。

本次内容就到此结束，感谢您的阅读！如果你觉得上述内容对你有所帮助，欢迎分享至一位你的朋友！若有任何疑问与建议，请添加笔者微信'nulls8'或加群进行交流。**青山不改，绿水长流，我们月来客栈见！**



扫码关注@月来客栈可获得更多优质内容！

代码仓库： https://github.com/moon-hotel/MachineLearningWithMe

目录

2021年