第 5 章

机 器 学 习

5.1 朴素贝叶斯算法

本节将详细研究朴素贝叶斯(Naive Bayes,NB)算法。本节内容主要包括：

- 朴素贝叶斯算法的基本原理及思想；
- 朴素贝叶斯算法的流程；
- 朴素贝叶斯算法的模型；
- 朴素贝叶斯算法的特性及应用场景。

5.1.1 朴素贝叶斯算法的基本概念

NB 算法是应用最为广泛的分类算法之一，也是为数不多的基于概率论的分类算法。它是在贝叶斯算法的基础上进行了相应的简化，即假定给定目标值时属性之间相互条件独立。虽然这个简化方式在一定程度上降低了贝叶斯分类算法的分类效果，但是在实际的应用场景中，极大地简化了贝叶斯算法的复杂性，且容易实现。在现实生活中，朴素贝叶斯算法广泛应用于垃圾邮件过滤、垃圾邮件的分类、信用评估及钓鱼网站检测等。

1. 基本原理

朴素贝叶斯分类(Naive Bayesian Classification,NBC)是以贝叶斯定理为基础并且假设特征条件之间相互独立的方法，先通过已给定的训练集，以特征词之间独立作为前提假设，学习从输入到输出的联合概率分布，再基于学习到的模型，输入 x 求出使得后验概率最大的输出 Y。

设有样本数据集 $D=\{d_1,d_2,\cdots,d_n\}$，对应样本数据的特征属性集为 $X=\{x_1,x_2,\cdots,x_d\}$，类变量为 $Y=\{y_1,y_2,\cdots,y_m\}$，即 D 可以分为 y_m 类别。如果 $x_1,x_2,\cdots,x_d$ 相互独立且随机，则 Y 的先验概率 $P_{\text{prior}}=P(Y)$，Y 的后验概率 $P_{\text{post}}=P(Y|X)$，由朴素贝叶斯算法可知，后验概率可以由先验概率 $P_{\text{prior}}=P(Y)$、证据 $P(X)$及类条件概率 $P(X|Y)$得出：

$$P(Y \mid X)=\frac{P(Y)P(X \mid Y)}{P(X)} \tag{5-1}$$

朴素贝叶斯基于各特征之间相互独立，在给定类别为 y 的情况下，式(5-1)可以进一步表示为

$$P(X \mid Y=y)=\prod_{i=1}^{d} P(x_i \mid Y=y) \tag{5-2}$$

整理式(5-1)和式(5-2)可得后验概率为

$$P_{\text{post}}=P(Y \mid X)=\frac{P(Y)\prod_{i=1}^{d} P(x_i \mid Y)}{P(X)} \tag{5-3}$$

由于 $P(X)$的大小是固定不变的，因此在比较后验概率时，只比较式(5-3)的分子部分即可，因此可以得到一个样本数据属于类别 y_i 的朴素贝叶斯：

$$P(y_i \mid x_1,x_2,\cdots,x_d)=\frac{P(y_i)\prod_{j=1}^{d} P(x_j \mid y_i)}{\prod_{j=1}^{d} P(x_j)} \tag{5-4}$$

2. 基本思想

上面介绍了朴素贝叶斯算法的基本原理，下面通过一个简单例子理解朴素贝叶斯算法的基本思想。

【例 5-1】 某医院早上接诊了 6 个门诊病人，现在又来的第 7 个病人是一个打喷嚏的建筑工人。请问他患感冒的概率有多大(假定“症状”与“职业”两个特征相互独立)？病人情况对照见表 5.1。

表 5.1　病人情况对照表

症　　状	职　　业	疾　　病
打喷嚏	护士	感冒
打喷嚏	农夫	过敏
头痛	建筑工人	脑震荡
头痛	建筑工人	感冒
打喷嚏	教师	感冒
头痛	教师	脑震荡

解：根据贝叶斯定理

$$P(Y \mid X)=\frac{P(Y)P(X \mid Y)}{P(X)}$$

可得：

$$P(\text{感冒} \mid \text{打喷嚏} \times \text{建筑工人})=\frac{P(\text{打喷嚏} \times \text{建筑工人} \mid \text{感冒}) \times P(\text{感冒})}{P(\text{打喷嚏} \times \text{建筑工人})}$$

假定“打喷嚏”和“建筑工人”这两个特征是独立的，所以有：

$$P(\text{感冒} \mid \text{打喷嚏} \times \text{建筑工人}) = \frac{P(\text{打喷嚏} \mid \text{感冒}) \times P(\text{建筑工人} \mid \text{感冒}) \times P(\text{感冒})}{P(\text{打喷嚏}) \times P(\text{建筑工人})}$$

则

$$P(\text{感冒} \mid \text{打喷嚏} \times \text{建筑工人}) = \frac{\frac{2}{3} \times \frac{1}{3} \times \frac{1}{2}}{\frac{1}{3} \times \frac{1}{2}} = \frac{2}{3}$$

因此，这个打喷嚏的建筑工人，有 66％的概率是感冒了。同理，可以计算这个病人过敏或脑震荡的概率。比较这几个概率，就可以知道病人最可能得了什么病。

这就是朴素贝叶斯算法的基本方法，即在概率基础上，依据某些特征，计算各个类别的概率，从而实现分类。

5.1.2 朴素贝叶斯算法的流程与模型

1. 具体流程

朴素贝叶斯算法分为 3 个阶段，具体流程如图 5.1 所示。

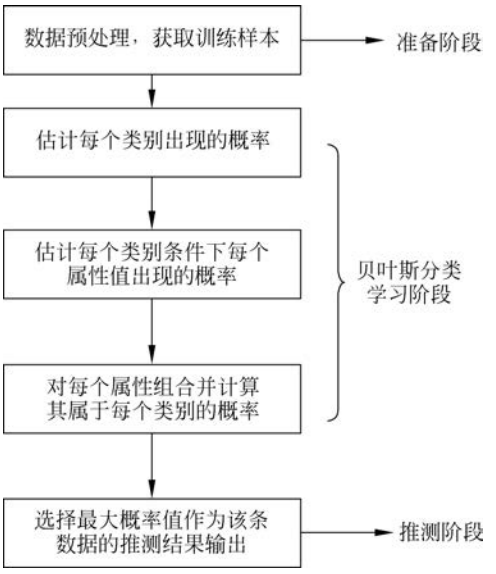


图 5.1 朴素贝叶斯算法流程

第一阶段——准备阶段，根据具体情况确定特征属性，对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其优劣对整个过程将有重要影响，分类器的优劣很大程度上由特征属性、特征属

性划分及训练样本质量决定。

第二阶段——贝叶斯分类学习阶段。这个阶段的任务就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并记录结果。其输入是特征属性和训练样本，输出是分类器。这一阶段是机械性阶段，可以根据前面讨论的公式由程序自动计算完成。

第三阶段——推测阶段。这个阶段的任务是使用分类器对待分类项进行分类，其输入是待分类项，输出是待分类项与类别的映射关系。这一阶段也是机械性阶段，由程序完成。

2. 常用模型

朴素贝叶斯算法有 3 个常用的模型。

(1) 高斯模型：处理特征是连续型变量的情况。当特征是连续型变量时，运用多项式模型就会导致很多 $P(x_i|y_k)=0$(不做平滑的情况下)，此时即使做平滑，所得到的条件概率也难以描述真实情况，所以处理连续的特征变量，应该采用高斯模型。

(2) 多项式模型：最常见且要求特征是离散数据。当特征是离散数据时，使用多项式模型。多项式模型在计算先验概率 $P(y_k)$和条件概率 $P(x_i|y_k)$时，会做一些平滑处理，如果不做平滑处理，当某一维特征的值 x_i 没在训练样本中出现过时($P(x_i|y_k)=0$)，会导致后验概率为 0，加上平滑就可以克服这个问题。

(3) 伯努利模型：要求特征是离散的，且为布尔类型，即 true 和 false，或者 1 和 0。与多项式模型一样，伯努利模型适用于离散特征的情况，但伯努利模型中每个特征的取值只能是 1 和 0(以文本分类为例，某个单词在文档中出现过，则其特征值为 1，否则为 0)。伯努利模型中，条件概率 $P(x_i|y_k)$的计算方式为

$$\begin{cases} P(x_i \mid y_k)=P(x_i=1 \mid y_k), & x_i=1 \\ P(x_i \mid y_k)=1-P(x_i=1 \mid y_k), & x_i=0 \end{cases}$$

5.1.3 朴素贝叶斯算法的特性与应用场景

1. 算法特性

朴素贝叶斯算法假设数据集属性之间是相互独立的，因此算法的逻辑性十分简单，并且算法较为稳定。当数据呈现不同的特点时，朴素贝叶斯的分类性能不会有太大的差异。换句话说，就是朴素贝叶斯算法的健壮性比较好，对于不同类型的数据集不会呈现出太大的差异性。当数据集属性之间的关系相对比较独立时，朴素贝叶斯分类算法会有较好的效果。

属性独立性的条件同时也是朴素贝叶斯分类器的不足之处。数据集属性的独立性在很多情况下是很难满足的，因为数据集的属性之间往往都存在相互关联，如果在分类过程中出现这种问题，会导致分类的效果大大降低。

2. 实际应用场景

朴素贝叶斯算法应用广泛，主要用于以下几个方面。

(1) 文本分类。

(2) 垃圾邮件过滤。

(3) 病人分类。

(4) 拼写检查。

(5) 信用评估。

(6) 钓鱼网站检测。

5.1.4 朴素贝叶斯算法的相关应用与 MATLAB 算例

1. 应用实例 1——判断人是否有对象

假设给定一个数据集,以这个数据集为基础,对于一个新的决策向量,基于式(5-1)～式(5-4)能够得到一个最大概率的值,将大概率事件作为目标值输出,就是对此决策向量的分类。

【例 5-2】 以表 5.2 中的数据为例,其中 1 表示是,0 表示否,那么现在得知一个人声音好听、颜值低、情商低、智商高($X_1=1, X_2=0, X_3=0, X_4=1$),是否可以判断他有(将有)对象?

表 5.2 对照表

X_1:声音好听	X_2:颜值高	X_3:情商高	X_4:智商高	Y:能否找到另一半
1	0	0	0	0
0	1	1	0	1
0	0	1	1	1
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0
0	0	0	1	0

解:

$$P(X_i \mid Y=1)=\left[\frac{1}{4},\frac{2}{4},0,\frac{1}{4}\right]$$

$$P(X_i \mid Y=0)=\left[\frac{2}{3},\frac{2}{3},\frac{2}{3},\frac{2}{3}\right]$$

$$P(Y=1)=\frac{4}{7}$$

$$P(Y=0)=\frac{3}{7}$$

有对象的概率为 $P=0$,没对象的概率为 $P=48/567=0.0847$,说明这个人很可能不会有对象。其 MATLAB 代码如下:

```
clc;clear all;
input = load("BayesData.txt")
[l,w] = size(input);
count = zeros(2,w);
for i = 1:1:l
for j = 1:1:w
if input(i,j) == 1 && input(i,end) == 1
count(1,j) = count(1,j) + 1;
elseif input(i,j) == 1 && input(i,end) == 0
count(2,j) = count(2,j) + 1;
end
end
end
count(2,end) = l - count(1,end);
test_data = [1 0 0 1];
answer = [0,0];
% case 1:
temp = 1;
for i = 1:1:w - 1
if test_data(i) == 1
temp = temp * count(1,i)/count(1,end);
else
temp = temp * (1 - count(1,i)/count(1,end));
end
end
answer(1) = count(1,end)/l * temp;
% case 0:
temp = 1;
for i = 1:1:w - 1
if test_data(i) == 1
temp = temp * count(2,i)/count(2,end);
else
temp = temp * (1 - count(2,i)/count(2,end));
end
end
answer(2) = count(2,end)/l * temp;
answer
if answer(1)> answer(2)
disp("可能会有对象")
else
disp("可能没有对象")
end
```

实现结果如图 5.2 所示。

从例 5-2 的结果可以看出，一个人声音好听且智商高，但是长得不好看还情商低，还是比较难找对象的。

2. 应用实例 2——判断西瓜好坏

【例 5-3】 现在有一个西瓜，它的属性值如下：

图 5.2 MATLAB 代码实现结果

色泽：青绿；根蒂：蜷缩；敲声：浊响；纹理：清晰；

脐部：凹陷；触感：硬滑；密度：0.697；糖率：0.460。

西瓜属性对照表如表 5.3 所示，判断该西瓜是好瓜还是坏瓜。

表 5.3 西瓜属性对照表

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	糖率	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.460	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.364	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软黏	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软黏	0.481	0.149	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软黏	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软黏	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软黏	0.360	0.370	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103	否

解：首先求每个类的先验概率，就是好瓜和坏瓜的比例。

$$P(\text{好瓜}) = \frac{8}{17} = 0.471$$

$$P(\text{坏瓜}) = \frac{9}{17} = 0.529$$

然后为每个属性值估计概率：

$$P(\text{色泽} = \text{青绿} \mid \text{好瓜} = \text{是}) = \frac{3}{8} = 0.375$$

$$P(\text{色泽} = \text{青绿} \mid \text{好瓜} = \text{否}) = \frac{3}{9} = 0.333$$

$$P(\text{根蒂} = \text{蜷缩} \mid \text{好瓜} = \text{是}) = \frac{5}{8} = 0.625$$

$$P(\text{根蒂} = \text{蜷缩} \mid \text{好瓜} = \text{否}) = \frac{3}{9} = 0.333$$

$$P(\text{敲声} = \text{浊响} \mid \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750$$

$$P(\text{敲声} = \text{浊响} \mid \text{好瓜} = \text{否}) = \frac{4}{9} = 0.444$$

$$P(\text{纹理} = \text{清晰} \mid \text{好瓜} = \text{是}) = \frac{7}{8} = 0.875$$

$$P(\text{纹理} = \text{清晰} \mid \text{好瓜} = \text{否}) = \frac{2}{9} = 0.222$$

$$P(\text{脐部} = \text{凹陷} \mid \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750$$

$$P(\text{脐部} = \text{凹陷} \mid \text{好瓜} = \text{否}) = \frac{2}{9} = 0.222$$

$$P(\text{触感} = \text{硬滑} \mid \text{好瓜} = \text{是}) = \frac{6}{8} = 0.750$$

$$P(\text{触感} = \text{硬滑} \mid \text{好瓜} = \text{否}) = \frac{6}{9} = 0.667$$

$$P(\text{密度} = 0.697 \mid \text{好瓜} = \text{是}) = \frac{1}{\sqrt{2\pi} \cdot 0.129} \mathrm{e}^{-\left(\frac{(0.697-0.574)^2}{2 \cdot 0.129^2}\right)} = 1.959$$

$$P(\text{密度} = 0.697 \mid \text{好瓜} = \text{否}) = \frac{1}{\sqrt{2\pi} \cdot 0.195} \mathrm{e}^{-\left(\frac{(0.697-0.496)^2}{2 \cdot 0.195^2}\right)} = 1.203$$

$$P(\text{含糖率} = 0.460 \mid \text{好瓜} = \text{是}) = \frac{1}{\sqrt{2\pi} \cdot 0.101} \mathrm{e}^{-\left(\frac{(0.460-0.279)^2}{2 \cdot 0.101^2}\right)} = 0.788$$

$$P(\text{含糖率}=0.460 \mid \text{好瓜}=\text{否})=\frac{1}{\sqrt{2\pi}\cdot 0.108}\mathrm{e}^{-\left(\frac{(0.460-0.154)^2}{2\cdot 0.108^2}\right)}=0.066$$

计算西瓜是好瓜和坏瓜的概率，哪个概率大就认为它是哪种瓜。

$$\begin{aligned}&P(\text{好瓜}=\text{是})\times P(\text{色泽}=\text{青绿}\mid\text{好瓜}=\text{是})\times P(\text{根蒂}=\text{蜷缩}\mid\text{好瓜}=\text{是})\times\\&P(\text{敲声}=\text{浊响}\mid\text{好瓜}=\text{是})\times P(\text{纹理}=\text{清晰}\mid\text{好瓜}=\text{是})\times\\&P(\text{脐部}=\text{凹陷}\mid\text{好瓜}=\text{是})\times P(\text{触感}=\text{硬滑}\mid\text{好瓜}=\text{是})\times\\&P(\text{密度}=0.697\mid\text{好瓜}=\text{是})\times P(\text{含糖率}=0.460\mid\text{好瓜}=\text{是})\\=&0.038\end{aligned}$$

$$\begin{aligned}&P(\text{好瓜}=\text{否})\times P(\text{色泽}=\text{青绿}\mid\text{好瓜}=\text{否})\times P(\text{根蒂}=\text{蜷缩}\mid\text{好瓜}=\text{否})\times\\&P(\text{敲声}=\text{浊响}\mid\text{好瓜}=\text{否})\times P(\text{纹理}=\text{清晰}\mid\text{好瓜}=\text{否})\times\\&P(\text{脐部}=\text{凹陷}\mid\text{好瓜}=\text{否})\times P(\text{触感}=\text{硬滑}\mid\text{好瓜}=\text{否})\times\\&P(\text{密度}=0.697\mid\text{好瓜}=\text{否})\times P(\text{含糖率}=0.460\mid\text{好瓜}=\text{否})\\=&6.80\times10^{-5}\end{aligned}$$

由此可知，该西瓜是好瓜。

在实现过程中，将数据分成训练集和测试集，计算测试集中每个类的先验概率（就是每个类在训练集中占的比例），然后为样本的每个属性估计条件概率（就是属性值相同的样本在每一类中占的比例）。其 MATLAB 代码如下：

```
[b] = xlsread('mix.xlsx',1,'A1:C1628');
x = b(:,1);
y = b(:,2);
c = b(:,3);
data = [x,y];
NUM = 500; % 样本数量
Test = sortrows([x(1:NUM,1),y(1:NUM,1),c(1:NUM,1)],3); % 按类对样本排序
temp = zeros(23,5); % 存储样本中各属性的均值、方差和每个类的概率
% 计算样本中各属性的均值、方差和每个类的概率
for i = 1:23
    X = [];
    Y = [];
    count = 0;
    for j = 1:NUM
        if Test(j,3) == i
            X = [X;Test(j,1)];
            Y = [Y;Test(j,2)];
            count = count + 1;
        end
    end
    temp(i,1) = mean(X);
    temp(i,2) = std(X);
    temp(i,3) = mean(Y);
    temp(i,4) = std(Y);
    temp(i,5) = count/NUM;
end
```

```
%计算预测结果
result = [];
for m = 1:1628
    pre = [];
    for n = 1:23
        PX = 1/temp(n,2) * exp(((data(m,1) - temp(n,1))^2)/ - 2/(temp(n,2)^2));
        PY = 1/temp(n,4) * exp(((data(m,2) - temp(n,3))^2)/ - 2/(temp(n,4)^2));
        pre = [pre;PX * PY * temp(n,5) * 10^8];
    end
    [da,index] = max(pre);
    result = [result;index];
end
xlswrite('mix.xlsx',result,'E1:E1628');
%画图
for i = 1:1628
    rand('seed',result(i,1));
    color = rand(1,3);
    plot(x(i,1),y(i,1),'*','color',color);
    hold on;
end
%查看正确率
num = 0;
for i = 1:1628
    if result(i) == c(i)
        num = num + 1;%正确的个数
    end
end
```

实现结果如图 5.3 所示。

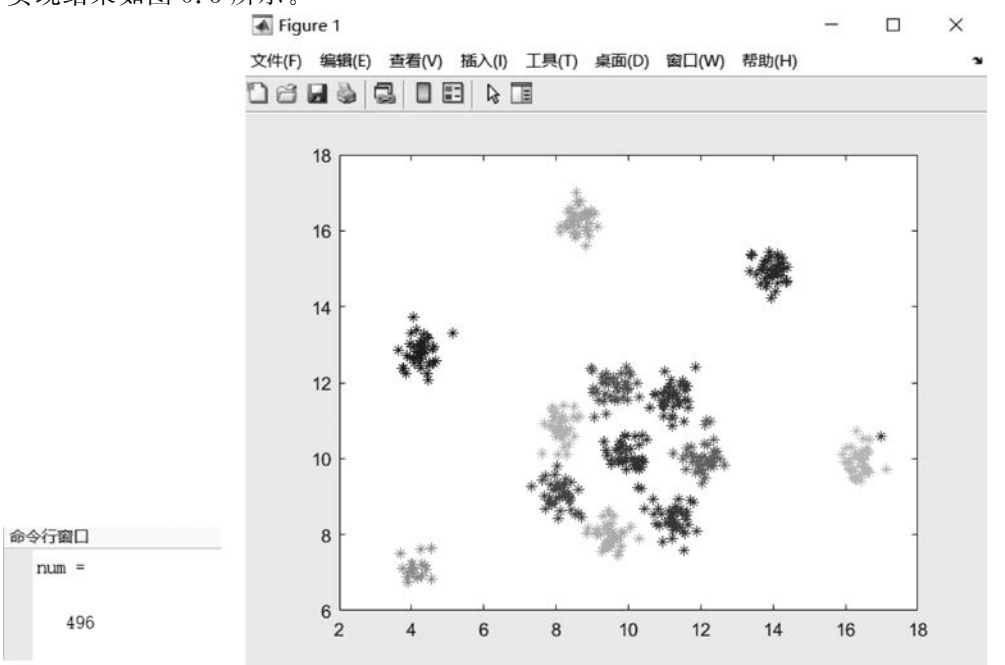


图 5.3　MATLAB 代码实现结果

3. 应用实例 3——判断花的类别

【例 5-4】 假设有三类花，且它们在自然界的数量都相同，即在这三类中任意取一枝花，$P(A)=P(B)=P(C)=1/3$。现有一枝花，判断其属于哪一类。在没有任何提示的情况下，可以得知，属于三类花的可能性一样。若此时用 4 维向量 x：花萼的长度、花萼的宽度、花瓣的长度、花瓣的宽度表示各自的特征，并且这些特征已知，这时判断它属于哪一类花。

解：已知某样本的特征，判断它是哪一类，就是模式识别的任务，而已知某样本的特征，得出它属于这些类的概率，最大者为所属的类，就是贝叶斯分类的方法。以 A 为例，利用贝叶斯公式：

$$P(A\mid \boldsymbol{x})=\frac{P(A,\boldsymbol{x})}{P(\boldsymbol{x})}=\frac{P(\boldsymbol{x}\mid A)P(A)}{P(\boldsymbol{x})}=\frac{P(\boldsymbol{x}\mid A)P(A)}{\sum_{i=1}^{3}P(\boldsymbol{x}\mid \omega_i)P(\omega_i)},\quad \omega_i=A,B,C$$

其中，$P(\boldsymbol{x})$表示这三类花中，花萼的长度、花萼的宽度、花瓣的长度和花瓣的宽度的总体密度分布，对于三类花来说，都是一致的。$P(A)=1/3$ 称为先验概率，在实践中有已知的统计。$P(\boldsymbol{x}|A)$为类条件密度，即 A 类花的花萼的长度、花萼的宽度、花瓣的长度、花瓣的宽度服从的分布，在朴素贝叶斯分类中，假设该分布密度为 4 元高斯分布；$P(A|\boldsymbol{x})$称为后验概率。所以，在求解 $P(A|\boldsymbol{x})$时，只需求解其展开公式的分子即可。

其 MATLAB 代码如下：

```
clear; clc;
```

A=[5.1,3.5,1.4,0.2	B=[6.4,3.2,4.5,1.5	C=[6.3,3.3,6.0,2.5
4.9,3.0,1.4,0.2	6.9,3.1,4.9,1.5	5.8,2.7,5.1,1.9
4.7,3.2,1.3,0.2	5.5,2.3,4.0,1.3	7.1,3.0,5.9,2.1
4.6,3.1,1.5,0.2	6.5,2.8,4.6,1.5	6.3,2.9,5.6,1.8
5.0,3.6,1.4,0.2	5.7,2.8,4.5,1.3	6.5,3.0,5.8,2.2
5.4,3.9,1.7,0.4	6.3,3.3,4.7,1.6	7.6,3.0,6.6,2.1
4.6,3.4,1.4,0.3	4.9,2.4,3.3,1.0	4.9,2.5,4.5,1.7
5.0,3.4,1.5,0.2	6.6,2.9,4.6,1.3	7.3,2.9,6.3,1.8
4.4,2.9,1.4,0.2	5.2,2.7,3.9,1.4	6.7,2.5,5.8,1.8
4.9,3.1,1.5,0.1	5.0,2.0,3.5,1.0	7.2,3.6,6.1,2.5
5.4,3.7,1.5,0.2	5.9,3.0,4.2,1.5	6.5,3.2,5.1,2.0
4.8,3.4,1.6,0.2	6.0,2.2,4.0,1.0	6.4,2.7,5.3,1.9
4.8,3.0,1.4,0.1	6.1,2.9,4.7,1.4	6.8,3.0,5.5,2.1
4.3,3.0,1.1,0.1	5.6,2.9,3.6,1.3	5.7,2.5,5.0,2.0
5.8,4.0,1.2,0.2	6.7,3.1,4.4,1.4	5.8,2.8,5.1,2.4
5.7,4.4,1.5,0.4	5.6,3.0,4.5,1.5	6.4,3.2,5.3,2.3
5.4,3.9,1.3,0.4	5.8,2.7,4.1,1.0	6.5,3.0,5.5,1.8
5.1,3.5,1.4,0.3	6.2,2.2,4.5,1.5	7.7,3.8,6.7,2.2
5.7,3.8,1.7,0.3	5.6,2.5,3.9,1.1	7.7,2.6,6.9,2.3
5.1,3.8,1.5,0.3	5.9,3.2,4.8,1.8	6.0,2.2,5.0,1.5
5.4,3.4,1.7,0.2	6.1,2.8,4.0,1.3	6.9,3.2,5.7,2.3
5.2,4.1,1.5,0.1	6.3,2.5,4.9,1.5	5.6,2.8,4.9,2.0

5.5,4.2,1.4,0.2 4.9,3.1,1.5,0.1 5.0,3.2,1.2,0.2 5.5,3.5,1.3,0.2 4.9,3.1,1.5,0.1 4.4,3.0,1.3,0.2 5.1,3.4,1.5,0.2 5.0,3.5,1.3,0.3 4.5,2.3,1.3,0.3 4.4,3.2,1.3,0.2 5.0,3.5,1.6,0.6 5.1,3.8,1.9,0.4 4.8,3.0,1.4,0.3 5.1,3.8,1.6,0.2 4.6,3.2,1.4,0.2 5.3,3.7,1.5,0.2 5.0,3.3,1.4,0.2 7.0,3.2,4.7,1.4];	6.1,2.8,4.7,1.2 6.4,2.9,4.3,1.3 6.6,3.0,4.4,1.4 6.8,2.8,4.8,1.4 6.7,3.0,5.0,1.7 6.0,2.9,4.5,1.5 5.7,2.6,3.5,1.0 5.5,2.4,3.8,1.1 5.5,2.4,3.7,1.0 5.8,2.7,3.9,1.2 6.0,2.7,5.1,1.6 5.4,3.0,4.5,1.5 6.0,3.4,4.5,1.6 6.7,3.1,4.7,1.5 6.3,2.3,4.4,1.3 5.6,3.0,4.1,1.3 5.7,2.8,4.1,1.3];	7.7,2.8,6.7,2.0 6.3,3.4,5.6,2.4 6.4,3.1,5.5,1.8 6.0,3.0,4.8,1.8 6.9,3.1,5.4,2.1 6.7,3.1,5.6,2.4 6.9,3.1,5.1,2.3 5.8,2.7,5.1,1.9 6.8,3.2,5.9,2.3 6.7,3.3,5.7,2.5 6.7,3.0,5.2,2.3 6.3,2.5,5.0,1.9 6.5,3.0,5.2,2.0 6.2,3.4,5.4,2.3 5.9,3.0,5.1,1.8];

```
NA = size(A,1);NB = size(B,1);NC = size(C,1);
A_train = A(1:floor(NA/2),:); % 训练数据取 1/2(或者 1/3,3/4,1/4)
B_train = B(1:floor(NB/2),:);
C_train = C(1:floor(NC/2),:);
u1 = mean(A_train)';u2 = mean(B_train)';u3 = mean(C_train)';
S1 = cov(A_train);S2 = cov(B_train);S3 = cov(C_train);
S11 = inv(S1);S22 = inv(S2);S33 = inv(S3);
S1_d = det(S1);S2_d = det(S2);S3_d = det(S3);
PA = 1/3;PB = 1/3;PC = 1/3; % 假设各类花的先验概率相等,即都为 1/3
A_test = A((floor(NA/2) + 1):end,:);
B_test = B((floor(NB/2) + 1):end,:);
C_test = C((floor(NC/2) + 1):end,:);
 % test of Sample_A
right1 = 0;
error1 = 0;
for i = 1:size(A_test,1)
P1 = ( - 1/2) * (A_test(i,:)' - u1)' * S11 * (A_test(i,:)' - u1) - (1/2) * log(S1_d) + log(PA);
P2 = ( - 1/2) * (A_test(i,:)' - u2)' * S22 * (A_test(i,:)' - u2) - (1/2) * log(S2_d) + log(PB);
P3 = ( - 1/2) * (A_test(i,:)' - u3)' * S33 * (A_test(i,:)' - u3) - (1/2) * log(S3_d) + log(PC);
P = [P1 P2 P3];
[Pm,ind] = max(P);
if ind == 1
right1 = right1 + 1;
else
error1 = error1 + 1;
end
end
right_rate = right1/size(A_test,1) % 计算 A 中测试数据的准确率,同理可以计算 B 和 C
```

5.2 决策树

本节将详细研究决策树(Decision Tree,DT)。本节内容主要包括:

- 决策树的定义及特性;
- 决策树分裂属性的选择;
- 决策树停止分裂的条件;
- 决策树的剪枝;
- 决策树的三种算法;
- 决策树生成算法的步骤。

5.2.1 决策树的基本概念

决策树是一种较为通用的分类方法,决策树模型简单并易于理解,且决策树转换为SQL语句很容易,能有效地访问数据库。特别地,很多情况下,决策树分类器的准确度与其他分类方法相似,甚至更好。目前已形成了多种决策树算法,如ID3、CART、C4.5、SLIQ、SPRINT等。本节主要介绍决策树定义及其特性。

1. 决策树的定义

决策树是在已知各种情况发生概率的基础上,通过构建决策树求取净现值的期望值大于或等于零的概率,评价项目风险,判断其可行性的决策分析方法,是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干,故称决策树。

决策树是一种树形结构,其中每个内部节点表示一个属性上的测试,每个分支代表一个测试输出,每个叶节点代表一种类别。决策树是一种十分常用的分类方法。

【例5-5】 某高尔夫俱乐部的经理为了调控俱乐部雇员数量,减少资金浪费,通过下周天气预报判断什么时候人们会打高尔夫球,以适时调整雇员数量。

解:通过收集天气状况(晴、多云和雨)、相对湿度(百分比)、有无风以及顾客是不是在这些日子光顾俱乐部,最终得到14列5行的数据表格。根据上面的自变量,建立了图5.4所示的决策树。

类似人类的思维过程,决策树就像一棵从根长出来的树(这里是倒着长的,也有横着长的)。最上面的节点叫作根节点,而最下面放置判断结果的节点叫作叶节点或者终节点。上面的描述性决策树是多分叉的,即每个非叶节点有两个分叉或三个分叉。决策树的节点上的变量可能是各种形式的(连续、离散、有序、分类变量等),一个变量也可以重复出现在不同的节点。一个节点前面的节点称为父节点(母节点或父母节点),而该节点为前面节点的子节点(女节点或子女节点),并列的节点称为兄弟节点(姊妹节点)。

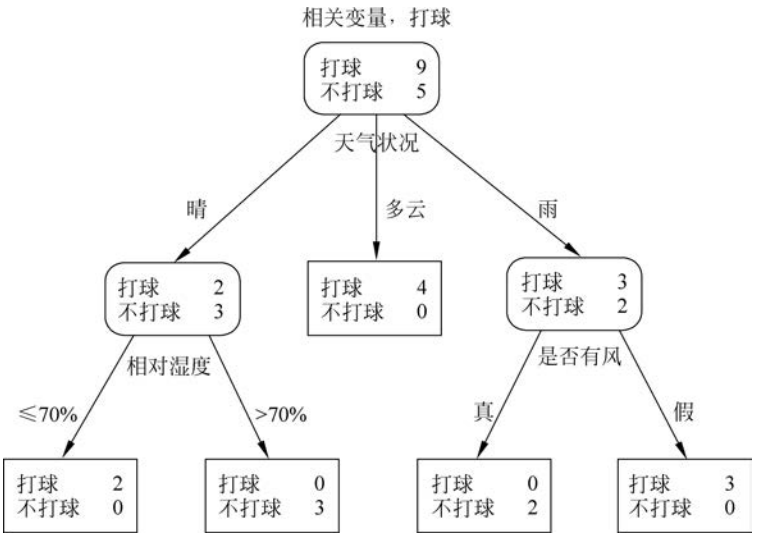


图 5.4　决策树实例

2. 决策树的特性

决策树的优点如下。

(1) 决策树易于理解和实现，能够直接体现数据的特点，人们在学习过程中不需要使用者了解很多的背景知识，只要通过必要的解释即可理解决策树所表达的意义。

(2) 对于决策树，数据的准备往往是简单的或者是不必要的，而且能够同时处理数据型和常规型属性，在相对短的时间内能够对大型数据源得出可行且效果良好的结果。

(3) 易于通过静态测试来对模型进行评测，可以测定模型可信度。如果给定一个观察的模型，那么根据所产生的决策树很容易推出相应的逻辑表达式。

决策树的缺点如下。

(1) 对连续性的字段比较难预测。

(2) 对有时间顺序的数据，需要很多预处理的工作。

(3) 当类别太多时，错误可能就会增加得比较快。

(4) 算法分类时，一般只是根据一个字段来分类。

5.2.2　决策树的构建

决策树的构建是数据逐步分裂的过程，构建的步骤如下。

步骤 1：将所有的数据看作一个节点，进入步骤 2。

步骤 2：从所有的数据特征中挑选一个数据特征对节点进行分割，进入步骤 3。

步骤 3：生成若干子节点，对每一个子节点进行判断，如果满足停止分裂的条件，进入步骤 4；否则，进入步骤 2。

步骤 4：设置该节点是子节点，其输出的结果为该节点数量占比最大的类别。

从上述步骤可以看出，决策生成过程中有两个重要的问题：如何选择分裂的特征、什么时候停止分裂。

1. 分裂属性的选择

决策树采用贪婪思想进行分裂，即选择可以得到最优分裂结果的属性进行分裂。那么怎样才算是最优的分裂结果？最理想的情况当然是能找到一个属性刚好能够将不同类别分开，但是大多数情况下分裂很难一步到位，希望每一次分裂之后子节点的数据尽量“纯”，以图 5.5 和 5.6 为例。

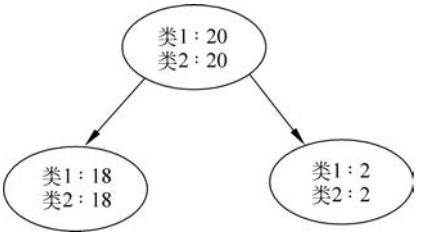


图 5.5　分裂属性 1

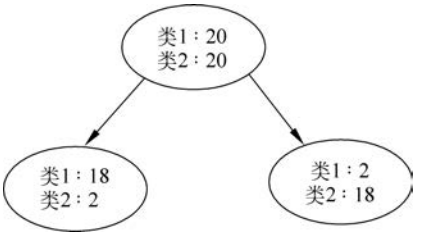


图 5.6　分裂属性 2

从图 5.5 和图 5.6 可以明显看出，属性 2 分裂后的子节点比属性 1 分裂后的子节点更纯：属性 1 分裂后每个节点的两类的数量还是相同，与根节点的分类结果相比完全没有提高；按照属性 2 分裂后每个节点各类的数量相差比较大，可以大概率认为第 1 个子节点的输出结果为类 1，第 2 个子节点的输出结果为 2。

选择分裂属性是要找出能够使所有子节点数据最纯的属性，决策树使用信息增益或者信息增益率作为选择属性的依据。

1）信息增益

用信息增益表示分裂前后的数据复杂度和分裂节点数据复杂度的变化值，计算公式为

$$\text{Info\_Gain} = \text{Gain} - \sum_{i=1}^{n} \text{Gain}_i$$

其中，Gain 表示节点的复杂度，Gain 越高，说明复杂度越高。简单来讲，信息增益就是分裂前的数据复杂度减去子节点的数据复杂度的和，信息增益越大，分裂后的复杂度减小得越多，分类的效果越明显。

节点的复杂度有以下两种不同的计算方式。

（1）熵描述了数据的混乱程度，熵越大，混乱程度越高，也就是纯度越低；反之，熵越小，混乱程度越低，纯度越高。熵的计算公式为

$$\text{Entropy} = -\sum_{i=1}^{n} P_i \log P_i$$

其中，P_i 表示类 i 的数量占比。以二分类问题为例，如果两类的数量相同，此时分类节点的

纯度最低，熵等于 1；如果节点的数据属于同一类时，此时节点的纯度最高，熵等于 0。

（2）基尼值的计算如下：

$$\text{Gini}=1-\sum_{i=1}^{n}p_i^2$$

其中，P_i 表示类 i 的数量占比。仍以上述熵的二分类例子为例，当两类数量相等时，基尼值等于 0.5；当节点数据属于同一类时，基尼值等于 0。基尼值越大，数据越不纯。

【例 5-6】 以熵作为节点复杂度的统计量，分别求出下面例子的信息增益。图 5.7 表示节点选择属性 1 进行分裂的结果，图 5.8 表示节点选择属性 2 进行分裂的结果，通过计算两个属性分裂后的信息增益，选择最优的分裂属性。

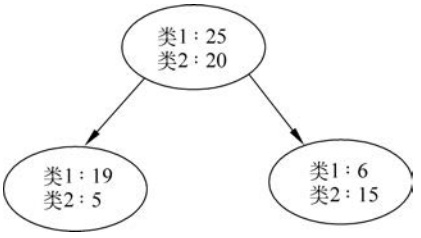


图 5.7　分裂属性 1

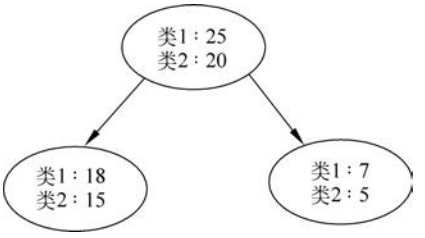


图 5.8　分裂属性 2

解：图 5.7 所示的分裂属性 1 求解过程如下。

$$\text{Info1}=\text{entropy}-\sum_{i=1}^{n}\text{entropy}_i=\left(\frac{25}{25+20}\right)\log\left(\frac{25}{25+20}\right)+\left(\frac{20}{25+20}\right)\log\left(\frac{20}{25+20}\right)$$

$$\Rightarrow \text{entropy}-\left[\left(\frac{19}{19+5}\right)\log\left(\frac{19}{19+5}\right)+\left(\frac{5}{19+5}\right)\log\left(\frac{5}{19+5}\right)\right]$$

$$\Rightarrow \text{entropy}_1-\left[\left(\frac{6}{15+6}\right)\log\left(\frac{6}{15+6}\right)+\left(\frac{15}{15+6}\right)\log\left(\frac{15}{15+6}\right)\right]$$

$$\Rightarrow \text{entropy}_2=0.423$$

图 5.8 所示的分裂属性 2 求解过程如下。

$$\text{Info2}=\text{entropy}-\sum_{i=1}^{n}\text{entropy}_i=\left(\frac{25}{25+20}\right)\log\left(\frac{25}{25+20}\right)+\left(\frac{20}{25+20}\right)\log\left(\frac{20}{25+20}\right)$$

$$\Rightarrow \text{entropy}-\left[\left(\frac{15}{15+18}\right)\log\left(\frac{15}{15+18}\right)+\left(\frac{18}{18+15}\right)\log\left(\frac{18}{18+15}\right)\right]$$

$$\Rightarrow \text{entropy}_1-\left[\left(\frac{5}{5+7}\right)\log\left(\frac{5}{5+7}\right)+\left(\frac{7}{5+7}\right)\log\left(\frac{7}{5+7}\right)\right]$$

$$\Rightarrow \text{entropy}_2=0.6812$$

由于 Info2＞Info1 ，所以与属性 1 相比，属性 2 是更优的分裂属性，故选择属性 1 作为分裂的属性。

2）信息增益率

使用信息增益作为选择分裂的条件有一个不可避免的缺点：倾向选择分支比较多的属性进行分裂。为了解决这个问题，引入了信息增益率这个概念。信息增益率是在信息增益的基础上除以分裂节点数据量的信息增益，其计算公式如下：

$$\text{Info\_Ratio}=\frac{\text{Info\_Gain}}{\text{InstrinsicInfo}}$$

其中，Info_Gain 表示信息增益，InstrinsicInfo 表示分裂子节点数据量的信息增益。InstrinsicInfo 的计算公式如下：

$$\text{InstrinsicInfo}=-\sum_{i=1}^{n}\frac{n_i}{N}\cdot\log\left(\frac{n_i}{N}\right)$$

其中，n 表示子节点的数量，n_i 表示第 i 个子节点的数据量，N 表示父节点数据量。也就是说，InstrinsicInfo 是分裂节点的熵，如果节点的数据链越接近，InstrinsicInfo 越大，如果子节点越大，InstrinsicInfo 越大，而 Info_Ratio 就会越小，能够降低节点分裂时选择子节点多的分裂属性的倾向性。信息增益率越高，说明分裂的效果越好。

【例 5-7】 求解图 5.7 和图 5.8 的属性分裂后的信息增益率。

解：属性 1 的信息增益率计算如下。

$$\text{Info\_Gain}_1=0.423$$

$$\text{InstrinsicInfo}_1=-\left[\left(\frac{24}{24+21}\right)\log\left(\frac{24}{24+21}\right)+\left(\frac{21}{24+21}\right)\log\left(\frac{21}{24+21}\right)\right]=0.6909$$

$$\text{Info\_Ratio}_1=\frac{\text{Info\_Gain}_1}{\text{InstrinsicInfo}_1}=0.6122$$

属性 2 的信息增益率计算如下：

$$\text{Info\_Gain}_2=0.6812$$

$$\text{InstrinsicInfo}_2=-\left[\left(\frac{33}{33+12}\right)\log\left(\frac{33}{33+12}\right)+\left(\frac{12}{33+12}\right)\log\left(\frac{12}{33+12}\right)\right]=0.5799$$

$$\text{Info\_Ratio}_2=\frac{\text{Info\_Gain}_2}{\text{InstrinsicInfo}_2}=1.1747$$

2. 停止分裂的条件

决策树不可能不限制地生长，总有停止分裂的时候，最极端的情况是当节点分裂到只剩下一个数据点时自动结束分裂，但这种情况下树过于复杂，而且预测的精度不高。一般情况下，为了降低决策树的复杂度和提高预测的精度，会适当提前终止节点的分裂。

以下是决策树节点停止分裂的一般性条件。

（1）最小节点数。当节点的数据量小于一个指定的数量时，不继续分裂，主要有两个原因：一是数据量较少时，再做分裂容易强化噪声数据的作用；二是降低树生长的复杂性。

提前结束分裂一定程度上有利于降低过拟合的影响。

(2) 熵或者基尼值小于阈值。熵和基尼值的大小表示数据的复杂程度,当熵或者基尼值过小时,表示数据的纯度比较大。如果熵或者基尼值小于一定程度,节点停止分裂。

(3) 决策树的深度达到指定的条件。节点的深度可以理解为节点与决策树根节点的距离,如根节点的子节点的深度为1,因为这些节点与根节点的距离为1,子节点的深度要比父节点的深度大1。决策树的深度是所有叶节点的最大深度,当深度到达指定的上限大小时,停止分裂。

(4) 所有特征已经使用完毕,不能继续进行分裂。被动式停止分裂的条件,当已经没有可分的属性时,直接将当前节点设置为叶节点。

5.2.3 决策树的剪枝

决策树剪枝的基本策略有“预剪枝”和“后剪枝”。

预剪枝是指在决策树生成过程中,对每个节点在划分前先进行估计,若当前节点的划分不能带来决策树泛化性能提升,则停止划分并将当前节点标记为叶节点。

后剪枝则是先从训练集上生成一棵完整的决策树,然后自底向上对非叶节点进行考察,若将该节点对应的子树替换为叶节点能带来决策树泛化性能提升,则将该子树替换为叶节点。

1. 预剪枝

预剪枝使得决策树的很多分支没有“展开”,这不仅降低了过拟合的风险,还显著减少了决策树的训练时间开销和测试时间开销。

但是,采用预剪枝决策,有些分支的当前划分虽不能提升泛化性能、甚至可能导致泛化性能暂时下降,但在其基础上进行的后续划分却又可能导致性能显著提高;预剪枝基于贪心算法,本质上禁止了这些分支展开,给预剪枝决策树带来了欠拟合的风险。

在树生成的过程中设定一定的准则来决定是否继续生长树,例如设定决策树的最大高度(层数)来限制树的生长,或设定每个节点必须包含的最少样本数,当节点中样本的个数小于某个数值时就停止分割;也可在构造树时,用信息增益等度量评估分割的优良性,如果在一个节点划分样本将导致低于预定阈值的分支,则停止进一步划分给定的子集。然而适当的阈值的选取是困难的,较高的阈值可能导致过分简化树,较低的阈值可能使得树的化简太少,需要根据专门应用领域的知识或经过多次测试评估确定阈值。

2. 后剪枝

后剪枝决策树通常比预剪枝决策树保留了更多的分支。一般情形下,后剪枝决策树的欠拟合风险很小,泛化性能往往优于预剪枝决策树。

后剪枝过程是在生成完全决策树之后进行的,并且要自底向上对树中的所有非叶节点进行逐一考察,因此其训练时间开销比未剪枝决策树和预剪枝决策树都要大得多。

先允许树尽量生长，然后通过删除节点的分支剪除树的节点，把树修剪到较小的尺寸。当然在修剪的同时要求尽量保持决策树的准确度不要下降太多。后剪枝方法所需的计算比预剪枝方法多，但通常产生更可靠的树。

5.2.4 决策树的算法实现

1. ID3 算法

ID3 算法只能处理定性变量，而且一个变量用过之后就不再使用了。

首先定义一个节点在选择了一个定性变量 X 之后，根据其取值产生若干子节点之后的信息增益(information gain)。用 T 表示母节点 t 处的数据样本，记该母节点的样本量为 T，熵为 I，而其各个子节点的样本及样本量分别为 X 及 $T_1, T_2, \cdots, T_n$。如果母节点的熵为 $I(T)$，而根据变量 X 所得到的各子节点的熵为 $I(X, T_1), I(X, T_2), \cdots, I(X, T_n)$，那么定义

$$I(X, Y) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} I(X, T)$$

在该节点对变量 X 的信息增益定义为

$$\text{Gain}(X, T) = I(T) - I(X, I)$$

显然，源于变量 X 的信息增益代表了识别 T 中元素所需信息和在得到 X 之后识别 T 中元素所需信息的差。根据信息增益在每个节点对变量进行排序，并选择信息增益最大的变量在该节点继续构造树。这样做的意图在于产生最小可能的树或者要使树增长最快。

图 5.9 是 ID3 算法的一个形式代码。

```
function ID3 (R：尚未用过的变量集，T：在该节点训练数据集)
    If T为空集，返回失败信息；
    If T包含所有同样的分类变量的值，返回一个具有该值的单独节点；
    If R为空，那么返回一个具有最大频率的当前变量的值；
    Let D∈R具有最大Gain(D, T)的变量；
    Let {dj | j=1,2,…,m}为D的值；
    Let {Sj | j=1,2,…,m}相应于D的值的T的子集；
Return 以D为标签的节点及标为d1,d2,…,dm的树枝；
end ID3;
```

注：这时ID3的函数和参数为ID3($R-\{D\}$, T_1), ID3($R-\{D\}$, T_2), …, ID3($R-\{D\}$, T_m)。

图 5.9 ID3 算法形式代码

2. C4.5 算法

C4.5 算法与 ID3 算法决策树的生成过程相似，C4.5 算法对 ID3 算法进行了改进，用信息增益率(比)选择特征。改进主要是针对样本特征。

(1) 基本决策树要求特征 A 取值为离散值,如果 A 是连续值,假如 A 有 v 个取值,则对特征 A 的测试可以看成是对 $v-1$ 个可能条件的测试。其实可以把这个过程看成是离散化的过程,只不过这种离散的值间隙会相对小一点。当然也可以采用其他方法,如将连续值按段进行划分,然后设置亚变量。

(2) 特征 A 的每个取值都会产生一个分支,有时会导致划分出的子集样本量过小,统计特征不充分而停止继续分支,这样在强制标记类别的时候也会带来局部的错误。针对这种情况可以采用 A 的一组取值作为分支条件;或者采用二元决策树,每一个分支代表一个特征取值的情况(只有是否两种取值)。

(3) 某些样本在特征 A 上值缺失,针对这种空值的情况,可以采用很多方法,比如用其他样本中特征 A 出现最多的值来填补空缺。在某些领域的数据中可以采用样本内部的平滑来补值,当样本量很大的时候也可以丢弃这些有缺失值的样本。

(4) 随着数据集的不断减小,子集的样本量会越来越小,所构造出的决策树就可能出现碎片、重复、复制等,这时可以利用样本的原有特征构造新的特征进行建模。

(5) 信息增益法会倾向于选择取值比较多的特征(这是信息熵的定义决定了的),针对这一问题,人们提出了增益比率法(gain ratio),将每个特征取值的概率考虑在内,及基尼索引法、G 统计法等。

3. CART 算法

CART 算法既可以做分类,也可以做回归,只能形成二叉树。CART 算法的分支条件是二分类问题。

对于连续特征的情况,CART 算法的分支方法是比较阈值,高于某个阈值就属于某一类,低于某个阈值属于另一类。对于离散特征,分支方法是抽取子特征,如颜值这个特征,有帅、丑、中等三个水平,可以先分为帅和不帅的,不帅的里面再分成丑和中等的。

CART 算法的得分函数对于分类树取的是分类最多的那个结果(也即众数),对于回归树取的是均值。

CART 算法的损失函数就是分类的准则,也就是求最优化的准则。对于分类树(目标变量为离散变量),损失函数是同一层所有分支假设函数的基尼系数的平均。对于回归树(目标变量为连续变量),损失函数是同一层所有分支假设函数的平方差损失。

对于分类树(目标变量为离散变量),使用基尼系数作为分裂规则。比较分裂前的 Gini 和分裂后的 Gini 减少多少,减少得越多,则选取该分裂规则。对于回归树(目标变量为连续变量),使用最小方差作为分裂规则,只能生成二叉树。

1) CART 分类树算法对于连续特征和离散特征处理的改进

对于 CART 分类树连续值的处理问题,其思想和 C4.5 算法相同,都是将连续的特征离散化。唯一的区别在于在选择划分点时的度量方式不同,C4.5 使用的是信息增益比,则 CART 分类树使用的是基尼系数。

具体的思路如下，比如 m 个样本的连续特征 A 有 m 个，从小到大排列为 $a_1, a_2, \cdots, a_m$，则 CART 算法取相邻两样本值的平均数，一共取得 $m-1$ 个划分点，其中第 i 个划分点表示为 $T_i = \frac{a_i + a_{i+1}}{2}$。对于这 $m-1$ 个点，分别计算以该点作为二元分类点时的基尼系数。选择基尼系数最小的点作为该连续特征的二元离散分类点。需要注意的是，与 ID3 或者 C4.5 处理离散属性不同的是，如果当前节点为连续属性，则该属性后面还可以参与子节点的产生选择过程。

对于 CART 分类树离散值的处理问题，采用的思路是不停地进行二分离散特征。

对于 ID3 或者 C4.5，如果某个特征 A 被选取建立决策树节点，如果它有 $A1$、$A2$、$A3$ 三种类别，我们会在决策树上建立一个三叉的节点，这样导致决策树是多叉树。但是 CART 分类树使用的方法不同，它采用的是不停地二分。对于 m 个样本的特征，CART 分类树会考虑把 A 分成 $\{A_1\}$ 和 $\{A_2, A_3\}$ $\{A_1\}$ 和 $\{A_2, A_3\}$，$\{A_2\}$ 和 $\{A_1, A_3\}$ $\{A_2\}$ 和 $\{A_1, A_3\}$，$\{A_3\}$ 和 $\{A_1, A_2\}$ $\{A_3\}$ 和 $\{A_1, A_2\}$ 三种情况，找到基尼系数最小的组合，比如 $\{A_2\}$ 和 $\{A_1, A_3\}$ $\{A_2\}$ 和 $\{A_1, A_3\}$。然后建立二叉树节点，一个节点是 A_2 对应的样本，另一个节点是 $\{A_1, A_3\}$ 对应的节点。同时，由于这次没有把特征 A 的取值完全分开，后面还有机会在子节点继续选择到特征 A 划分 A_1 和 A_3。这和 ID3 或者 C4.5 不同，在 ID3 或者 C4.5 的一棵子树中，离散特征只会参与一次节点的建立。

2) CART 分类树建立算法的具体流程

上面介绍了 CART 算法和 C4.5 的一些不同之处，下面介绍 CART 分类树建立算法的具体流程，之所以加上了建立，是因为 CART 树算法还有独立的剪枝算法。

算法的输入包括训练集 D、基尼系数的阈值和样本个数阈值。算法输出是决策树 T。

算法从根节点开始，用训练集递归地建立 CART 树。

(1) 对于当前节点的数据集为 D，如果样本个数小于阈值或者没有特征，则返回决策子树，当前节点停止递归。

(2) 计算样本集 D 的基尼系数，如果基尼系数小于阈值，则返回决策树子树，当前节点停止递归。

(3) 计算当前节点现有的各个特征值对数据集 D 的基尼系数，对于离散值和连续值的处理方法和基尼系数的计算见 5.2.2 节。缺失值的处理方法和 C4.5 算法中描述的相同。

(4) 在计算出来的各个特征值对数据集 D 的基尼系数中，选择基尼系数最小的特征 A 和对应的特征值 a。根据这个最优特征和最优特征值，把数据集划分成 D_1 和 D_2 两部分，同时建立当前节点的左右节点，左节点的数据集为 D_1，右节点的数据集为 D_2（由于是二叉树，故这里的 D_1 和 D_2 有集合关系，$D_2 = D - D_1$）。

(5) 对左右子节点的递归调用(1)～(4)步骤，生成决策树。

3）CART 回归树建立算法

CART 回归树和 CART 分类树的建立算法大部分是类似的，所以这里只讨论 CART 回归树和 CART 分类树的建立算法不同的地方。

首先，回归树和分类树的区别在于样本输出，如果样本输出是离散值，那么这是一棵分类树；如果果样本输出是连续值，那么这是一棵回归树。

除了概念的不同，CART 回归树和 CART 分类树的建立和预测的区别主要有两点。

（1）连续值的处理方法不同。

（2）决策树建立后做预测的方式不同。

对于连续值的处理，CART 分类树采用基尼系数的大小度量特征的各个划分点的优劣情况。这比较适合分类模型，但是对于回归模型，可以使用常见的和方差的度量方式，CART 回归树的度量目标是，对于任意划分特征 A，对应的任意划分点 s 两边划分的数据集 D_1 和 D_2，求出使 D_1 和 D_2 各自集合的均方差最小，同时 D_1 和 D_2 的均方差之和最小所对应的特征和特征值划分点。表达式为

$$\min_{A,s}\left[\min_{c_1}\sum_{x_i\in D_1(A,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in D_2(A,s)}(y_i-c_2)^2\right]$$

其中，c_1 为 D_1 数据集的样本输出均值；c_2 为 D_2 数据集的样本输出均值。

对于决策树建立后做预测的方式，上面讲到了 CART 分类树采用叶节点中概率最大的类别作为当前节点的预测类别。而回归树输出不是类别，它采用的是用最终叶节点的均值或者中位数预测输出结果。

4）CART 树算法的剪枝

CART 回归树和 CART 分类树的剪枝策略除了在度量损失的时候一个使用均方差，一个使用基尼系数，算法基本完全一样。

由于决策时算法很容易对训练集过拟合，而导致泛化能力差，为了解决这个问题，需要对 CART 树进行剪枝，即类似于线性回归的正则化，来增加决策树的泛化能力。但是，有很多的剪枝方法，应该怎么选择呢？CART 采用的办法是后剪枝法，即先生成决策树，然后产生所有可能的剪枝后的 CART 树，然后使用交叉验证来检验各种剪枝的效果，选择泛化能力最好的剪枝策略。

也就是说，CART 树的剪枝算法可以概括为两步，第一步是从原始决策树生成各种剪枝效果的决策树，第二步是用交叉验证来检验剪枝后的预测能力，选择泛化预测能力最好的剪枝后的数作为最终的 CART 树。

首先看剪枝的损失函数度量，在剪枝的过程中，对于任意的一刻子树 T，其损失函数为

$$C_\alpha(T_t)=C(T_t)+\alpha\mid T_t\mid$$

其中，α 为正则化参数，这和线性回归的正则化一样。$C(T_t)$ 为训练数据的预测误差，分类树是用基尼系数度量，回归树是用均方差度量。$|T_t|$ 是子树 T 的叶子节点的数量。

当 $\alpha=0$ 时，即没有正则化，原始生成的 CART 树即为最优子树。当 $\alpha=\infty$ 时，即正则化强度达到最大，此时由原始生成的 CART 树的根节点组成的单节点树为最优子树。当然，这是两种极端情况。一般来说，α 越大，则剪枝越厉害，生成的最优子树相比原生决策树就越偏小。对于固定的 α，一定存在使损失函数 $C_\alpha(T)$ 最小的唯一子树。

了解了剪枝的损失函数度量后，再看剪枝的思路，对于位于节点 t 的任意一棵子树 T_t，如果没有剪枝，它的损失是：

$$C_\alpha(T_t)=C(T_t)+\alpha\mid T_t\mid$$

如果将其剪掉，仅仅保留根节点，则损失是：

$$C_\alpha(T)=C(T)+\alpha$$

当 $\alpha=0$ 或者 α 很小时：

$$C_\alpha(T_t)<C_\alpha(T)$$

当 α 增大到一定的程度时：

$$C_\alpha(T_t)=C_\alpha(T)$$

当 α 继续增大时，不等式反向，也就是说，如果满足：

$$\alpha=\frac{C(T)-C(T_t)}{\mid T_t\mid-1}$$

T_t 和 T 有相同的损失函数，但是 T 节点更少，因此可以对子树 T_t 进行剪枝，也就是将它的子节点全部剪掉，变为一个叶节点 T。

最后看 CART 树的交叉验证策略。通过上述步骤可以计算出每个子树是否剪枝的阈值 α，如果把所有的节点是否剪枝的值 α 都计算出来，然后分别针对不同的 α 所对应的剪枝后的最优子树做交叉验证。这样就可以选择一个最好的 α，有了这个 α，就可以用对应的最优子树作为最终结果。

5) CART 树的剪枝算法

输入是 CART 树建立算法得到的原始决策树 T，输出是最优决策子树 T_α。

算法过程如下。

(1) 初始化 $\alpha_{\min}=\infty$，最优子树集合 $\omega=\{T\}$。

(2) 从叶节点开始自下而上计算各内部节点 t 的训练误差损失函数 $C_\alpha(T_t)$（回归树为均方差，分类树为基尼系数），叶节点数 $|T_t|$，以及正则化阈值：

$$\alpha=\min\left\{\frac{C(T)-C(T_t)}{\mid T_t\mid-1},\alpha_{\min}\right\}$$

更新 $\alpha_{\min}=\alpha$。

(3) 得到所有节点的 α 值的集合 M。

(4) 从 M 中选择最大的值 α_k，自上而下的访问子树 t 的内部节点，如果

$$\frac{C(T)-C(T_t)}{|T_t|-1}\leqslant\alpha_k$$

时，进行剪枝，并决定叶节点 t 的值。如果是分类树，则是概率最高的类别；如果是回归树，则是所有样本输出的均值。这样得到 α_k 对应的最优子树 T_k。

(5) 最优子树集合 $\omega=\omega\cup T_k$，$M=M-\{\alpha_k\}$。

(6) 如果 M 不为空，则回到步骤(4)；否则就得到了所有的可选最优子树集合 ω。

(7) 采用交叉验证在 ω 选择最优子树 T_α。

4. 决策树生成算法的步骤

输入为训练数据集 S 和属性集合(包含类标号属性)；输出为决策树 T，具体步骤如下。

(1) 建立节点 R。

(2) 如果样本集 S 中所有的数据样本在同一个类 C 中，则 r 为叶节点，以类 C 作为该叶节点标号。

(3) 如果没有剩余属性，则 R 为叶节点，以样本中的多数所在类标号标记 R。

(4) 计算每个属性 A 的属性选择度量(信息增益或基尼系数等)，确定分割属性 At。

(5) 将属性 Aa 作为节点 R 的分割属性。

(6) 对于每次分割中的属性值 a_1，用不同的方法处理属性值为离散或连续的情况，这里给出离散时的方法，连续的情况后面将会提到。

(7) 由节点 R 分割一个条件为 $St=a_1$ 的分支。

(8) 设 S_1 是样本集 S 中 Salt$=a_1$ 的样本子集。

(9) 如果样本集 S 为空，则增加一个叶节点，将其标记为样本集 S 中最多的类；否则增加一个由语句 building_tree(S，去除分割属性 Salt 的属性集)返回的节点。

5.2.5　决策树的相关应用与 MATLAB 算例

1. 应用实例 1——基于信息的心理活动判断

决策树算法是基于信息增益来构建的，信息增益可以由训练集的信息熵算得。

【例 5-8】 data=[心情好　天气好　出门
心情好　天气不好　出门
心情不好　天气好　出门
心情不好　天气不好　不出门]

求其信息增益。

解：前面两列是分类属性，最后一列是分类，分类的信息熵可以计算得到：

出门＝3，　不出门＝1，　总行数＝4

分类信息熵＝－(3/4)log2(3/4)－(1/4)log2(1/4)

第一列属性有两类：心情好、心情不好。

心情好，出门＝2，　不出门＝0，　行数＝2

心情好信息熵＝－(2/2)log2(2/2)＋(0/2)log2(0/2)

同理，

心情不好信息熵＝－(1/2)log2(1/2)－(1/2)log2(1/2)

心情的信息增益＝分类信息熵－心情好的概率×心情好的信息熵－

心情不好的概率×心情不好的信息熵。

由此可以得到每个属性对应的信息熵，信息熵最大的即为最优划分属性。

【例 5-9】 基于例 5-4，加入最优划分属性为心情，如图 5.10 所示，求其决策树。

解：区分在心情属性的每个具体情况下的分类是否全部为同一种，若为同一种则该节点标记为此类别，在心情好的情况下，不管什么天气结果都是出门，如图 5.11 所示。

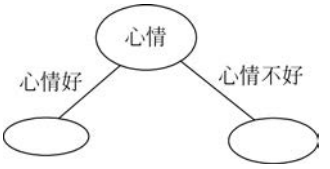


图 5.10　最优属性划分

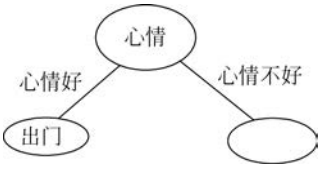


图 5.11　最优属性划分结果

心情不好的情况下有不同的分类结果，继续计算在心情不好的情况下其他属性的信息增益，把信息增益最大的属性作为分支节点。这里只有天气这个属性，那么这个节点就是天气，天气属性有两种情况，如图 5.12 所示。

在心情不好并且天气好的情况下，若分类全为同一种，则改节点标记为此类别。心情不好并且天气好为出门，心情不好并且天气不好为不出门，结果如图 5.13 所示。

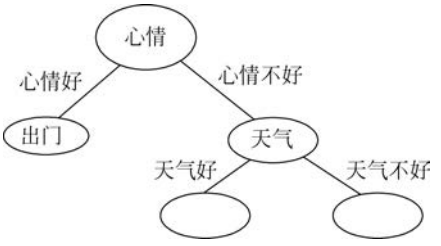


图 5.12　最优属性划分-天气

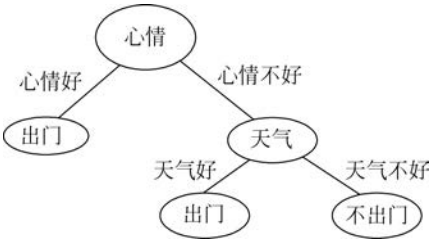


图 5.13　最优属性划分-天气结果

【例 5-10】 对于分支节点下的属性很有可能没有数据，假设训练集变成

data＝[心情好　晴天　出门

心情好　阴天　出门

心情好　雨天　出门

心情好　雾天　出门

心情不好　晴天　出门

心情不好　雨天　不出门

心情不好　阴天　不出门]

求其决策树。

解：心情不好的情况下，天气中并没有雾天，如何判断雾天到底是否出门呢？可以采用该样本最多的分类作为分类，在天气不好的情况下，出门＝1，不出门＝2，那么将不出门作为雾天的分类结果，如图 5.14 所示。

图 5.14　最优属性划分结果

至此，完成了所有属性的划分，结束递归，得到了一棵非常简单的决策树。其 MATLAB 代码实现如下：

```
function[node] = createTree(data,feature)
    type = mostType(data);
    [m,n] = size(data);
    node = struct('value','null','name','null','type','null','children',[]);
    temp_type = data(1,n);
    temp_b = true;
    for i = 1:m
        if temp_type!= data(i,n)
            temp_b = false;
        end
    end
    if temp_b == true
        node.value = data(1,n);
        return;
    end
    if sum(feature) == 0
        node.value = type;
        return;
    end
    feature_bestColumn = bestFeature(data);
    best_feature = getData()(:,feature_bestColumn);
    best_distinct = unique(best_feature);
    best_num = length(best_distinct);
    best_proc = zeros(best_num,2);
    best_proc(:,1) = best_distinct(:,1);
    for i = 1:best_num
        Dv = [];
```

```
            Dv_index = 1;
            bach_node = struct('value','null','name','null','type','null','children',[]);
            for j = 1:m
                if best_proc(i,1) == data(j,feature_bestColumn)
                    Dv(Dv_index,:) = data(j,:);
                    Dv_index = Dv_index + 1;
                end
            end
            if length(Dv) == 0
                bach_node.value = type;
                bach_node.type = best_proc(i,1);
                bach_node.name = feature_bestColumn;
                node.children(i) = bach_node;
                return;
            else
                feature(feature_bestColumn) = 0;
                bach_node = createTree(Dv,feature);
                bach_node.type = best_proc(i,1);
                bach_node.name = feature_bestColumn;
                node.children(i) = bach_node;
            end
        end
    end
    function [column] = bestFeature(data)
        [m,n] = size(data);
        featureSize = n - 1;
        gain_proc = zeros(featureSize,2);
        entropy = getEntropy(data);
        for i = 1:featureSize
            gain_proc(i,1) = i;
            gain_proc(i,2) = getGain(entropy,data,i);
        end
        for i = 1:featureSize
            if gain_proc(i,2) == max(gain_proc(:,2))
                column = i;
                break;
            end
        end
    end
    function [res] = mostType(data)
        [m,n] = size(data);
        res_distinct = unique(data(:,n));
        res_proc = zeros(length(res_distinct),2);
        res_proc(:,1) = res_distinct(:,1);
        for i = 1:length(res_distinct)
            for j = 1:m
                if res_proc(i,1) == data(j,n)
                    res_proc(i,2) = res_proc(i,2) + 1;
                end
            end
```

```
    end
    for i = 1:length(res_distinct)
        if res_proc(i,2) == max(res_proc(:,2))
            res = res_proc(i,1);
            break;
        end
    end
end
function [entropy] = getEntropy(data)
    entropy = 0;
    [m,n] = size(data);
    label = data(:,n);
    label_distinct = unique(label);
    label_num = length(label_distinct);
    proc = zeros(label_num,2);
    proc(:,1) = label_distinct(:,1);
    for i = 1:label_num
        for j = 1:m
            if proc(i,1) == data(j,n)
                proc(i,2) = proc(i,2) + 1;
            end
        end
        proc(i,2) = proc(i,2)/m;
    end
    for i = 1:label_num
        entropy = entropy - proc(i,2) * log2(proc(i,2));
    end
end
function [gain] = getGain(entropy,data,column)
    [m,n] = size(data);
    feature = data(:,column);
    feature_distinct = unique(feature);
    feature_num = length(feature_distinct);
    feature_proc = zeros(feature_num,2);
    feature_proc(:,1) = feature_distinct(:,1);
    f_entropy = 0;
    for i = 1:feature_num
        feature_data = [];
        feature_proc(:,2) = 0;
        feature_row = 1;
        for j = 1:m
            if feature_proc(i,1) == data(j,column)
                feature_proc(i,2) = feature_proc(i,2) + 1;
            end
            if feature_distinct(i,1) == data(j,column)
                feature_data(feature_row,:) = data(j,:);
                feature_row = feature_row + 1;
            end
        end
        f_entropy = f_entropy + feature_proc(i,2)/m * getEntropy(feature_data);
```

```
end
gain = entropy - f_entropy;
```

2. 应用实例 2——汽车特征评估质量

【例 5-11】 给出数据集包括汽车的购买价位、维修成本、车门数量、载客数量、动力性能和安全性能等，对汽车进行分类，确定一辆汽车的质量，其中汽车质量分为 4 种类型：不达标、达标、良好和优秀。数据集形式如图 5.15 所示，其中的每个值都可以看作字符串。

```
med,low,5more,more,big,low,unacc
med,low,5more,more,big,med,good
med,low,5more,more,big,high,vgood
low,vhigh,2,2,small,low,unacc
low,vhigh,2,2,small,med,unacc
low,vhigh,2,2,small,high,unacc
low,vhigh,2,2,med,low,unacc
```

图 5.15 数据集形式

解：考虑数据集中的 6 个属性，其取值范围如下。

(1) 购买价位：取值范围是 vhigh、high、med、low，分别代表很高、高、中等、低。

(2) 维修成本：取值范围是 vhigh、high、med、low，分别代表很高、高、中等、低。

(3) 车门数量：取值范围是 2、3、4、5、5more 等。

(4) 载客数量：取值范围是 2、4、more 等。

(5) 动力性能：取值范围是 small、med、big，分别代表小、中、大。

(6) 安全性能：取值范围是 low、med、high，分别代表低、中、高。

分类的结果，即汽车的质量取值范围是 unacc、acc、good、vgood，分别代表不达标、达标、良好、优秀。

考虑到每一行都具有字符串属性，需要假设所有的特征均是字符串，并在此基础上建立分类器。

首先将数据集中的所有字符串变为数字，方便后面的分类。由于下载的数据集为.data 格式，MATLAB 无法直接读取，已经转换为.xlsx 格式，并且将 vhigh、high、med、low 分别替换为 4、3、2、1，将 small、med、big 替换为 1、2、3，将 low、med、high 替换为 1、2、3，将 unacc、acc、good、vgood 替换为 1、2、3、4。

数据中共有 1728 组，随机从中取出 1500 组作为训练集，剩下的 228 组作为测试集。使用训练集建立决策树，然后使用模型进行预测。分别根据决策树的结果计算出决策树中对车辆各种情况预测的准确率以及全部测试集预测的准确率，然后对决策树进行修剪。

分类的 MATLAB 代码实现如下：

```
clear all;
clc;
close all;
%% 导入数据
load car;
a = randperm(1728);
%训练集
Train_Data = data(a(1:1500),1:6);
Train_Label = data(a(1:1500),7);
%测试集
```

```
Test_Data = data(a(1501:1728),1:6);
Test_Label = data(a(1501:1728),7);
%% 创建决策树分类器
Tree = ClassificationTree.fit(Train_Data,Train_Label);
%% 查看决策树视图
view(Tree);
view(Tree,'mode','graph');
%% 预测分类
Tree_pre = predict(Tree,Test_Data);
%% 结果分析
count_train_1 = length(find(Train_Label == 1));        %训练集中车辆质量不达标个数
count_train_2 = length(find(Train_Label == 2));        %训练集中车辆质量达标个数
count_train_3 = length(find(Train_Label == 3));        %训练集中车辆质量良好个数
count_train_4 = length(find(Train_Label == 4));        %训练集中车辆质量优秀个数
rate_train_1 = count_train_1 / 1500;                   %训练集中车辆质量不达标占的比例
rate_train_2 = count_train_2 / 1500;                   %训练集中车辆质量达标占的比例
rate_train_3 = count_train_3 / 1500;                   %训练集中车辆质量良好占的比例
rate_train_4 = count_train_4 / 1500;                   %训练集中车辆质量优秀占的比例
total_1 = length(find(data(:,7) == 1));                %总数据中车辆质量不达标个数
total_2 = length(find(data(:,7) == 2));                %总数据中车辆质量达标个数
total_3 = length(find(data(:,7) == 3));                %总数据中车辆质量良好个数
total_4 = length(find(data(:,7) == 4));                %总数据中车辆质量优秀个数
count_test_1 = length(find(Test_Label == 1));          %测试集中车辆质量不达标个数
count_test_2 = length(find(Test_Label == 2));          %测试集中车辆质量达标个数
count_test_3 = length(find(Test_Label == 3));          %测试集中车辆质量良好个数
count_test_4 = length(find(Test_Label == 4));          %测试集中车辆质量优秀个数
count_right_1 = length(find(Tree_pre == 1 & Test_Label == 1));
                                                       %测试集中预测车辆质量不达标正确的个数
count_right_2 = length(find(Tree_pre == 2 & Test_Label == 2));
                                                       %测试集中预测车辆质量达标正确的个数
count_right_3 = length(find(Tree_pre == 3 & Test_Label == 3));
                                                       %测试集中预测车辆质量良好正确的个数
count_right_4 = length(find(Tree_pre == 4 & Test_Label == 4));
                                                       %测试集中预测车辆质量优秀正确的个数
rate_right = (count_right_1 + count_right_2 + count_right_3 + count_right_4)/228;
%% 显示部分结果
disp(['车辆总数:1728'...
     '不达标:' num2str(total_1)...
     '达标:' num2str(total_2)...
     '良好:' num2str(total_3)...
     '优秀:' num2str(total_4)]);
disp(['训练集车辆数:1500'...
     '不达标:' num2str(count_train_1)...
     '达标:' num2str(count_train_2)...
     '良好:' num2str(count_train_3)...
     '优秀:' num2str(count_train_4)]);
disp(['测试集车辆数:228'...
```

```
        '不达标:' num2str(count_test_1)...
        '达标:' num2str(count_test_2)...
        '良好:' num2str(count_test_3)...
        '优秀:' num2str(count_test_4)]);
    disp(['决策树判断结果:'...
        '不达标正确率:' sprintf('%2.2f%%', count_right_1/count_test_1*100)...
        '达标正确率:' sprintf('%2.2f%%', count_right_2/count_test_2*100)...
        '良好正确率:' sprintf('%2.2f%%', count_right_3/count_test_3*100)...
        '优秀正确率:' sprintf('%2.2f%%', count_right_4/count_test_4*100)]);
    disp(['总正确率:'...
        sprintf('%2.2f%%', rate_right*100)]);
    %% 优化前决策树的重采样误差和交叉验证误差
    resubDefault = resubLoss(Tree);
    lossDefault = kfoldLoss(crossval(Tree));
    disp(['剪枝前决策树的重采样误差:'...
        num2str(resubDefault)]);
    disp(['剪枝前决策树的交叉验证误差:'...
        num2str(lossDefault)]);
    %% 剪枝
    [~,~,~,bestlevel] = cvLoss(Tree,'subtrees','all','treesize','min');
    cptree = prune(Tree,'Level',bestlevel);
    view(cptree,'mode','graph')
    %% 剪枝后决策树的重采样误差和交叉验证误差
    resubPrune = resubLoss(cptree);
    lossPrune = kfoldLoss(crossval(cptree));
    disp(['剪枝后决策树的重采样误差:'...
        num2str(resubPrune)]);
    disp(['剪枝后决策树的交叉验证误差:'...
        num2str(resubPrune)]);
```

分类结果如下。

(1) 车辆总数为 1728,不达标：1210,达标：384,良好：69,优秀：65。

(2) 训练集车辆数为 1500,不达标：1046,达标：338,良好：56,优秀：60。

(3) 测试集车辆数为 228,不达标：164,达标：46,良好：13,优秀：5。

决策树判断结果如下。

(1) 不达标正确率为 97.56%；达标正确率为 95.65%；良好正确率为 84.62%；优秀正确率为 100.00%；总正确率为 96.49%。

(2) 剪枝前决策树的重采样误差为 0.026。

(3) 剪枝前决策树的交叉验证误差为 0.048667。

(4) 剪枝后决策树的重采样误差为 0.026667。

(5) 剪枝后决策树的交叉验证误差为 0.026667。

MATLAB 代码实现结果如图 5.16 所示。

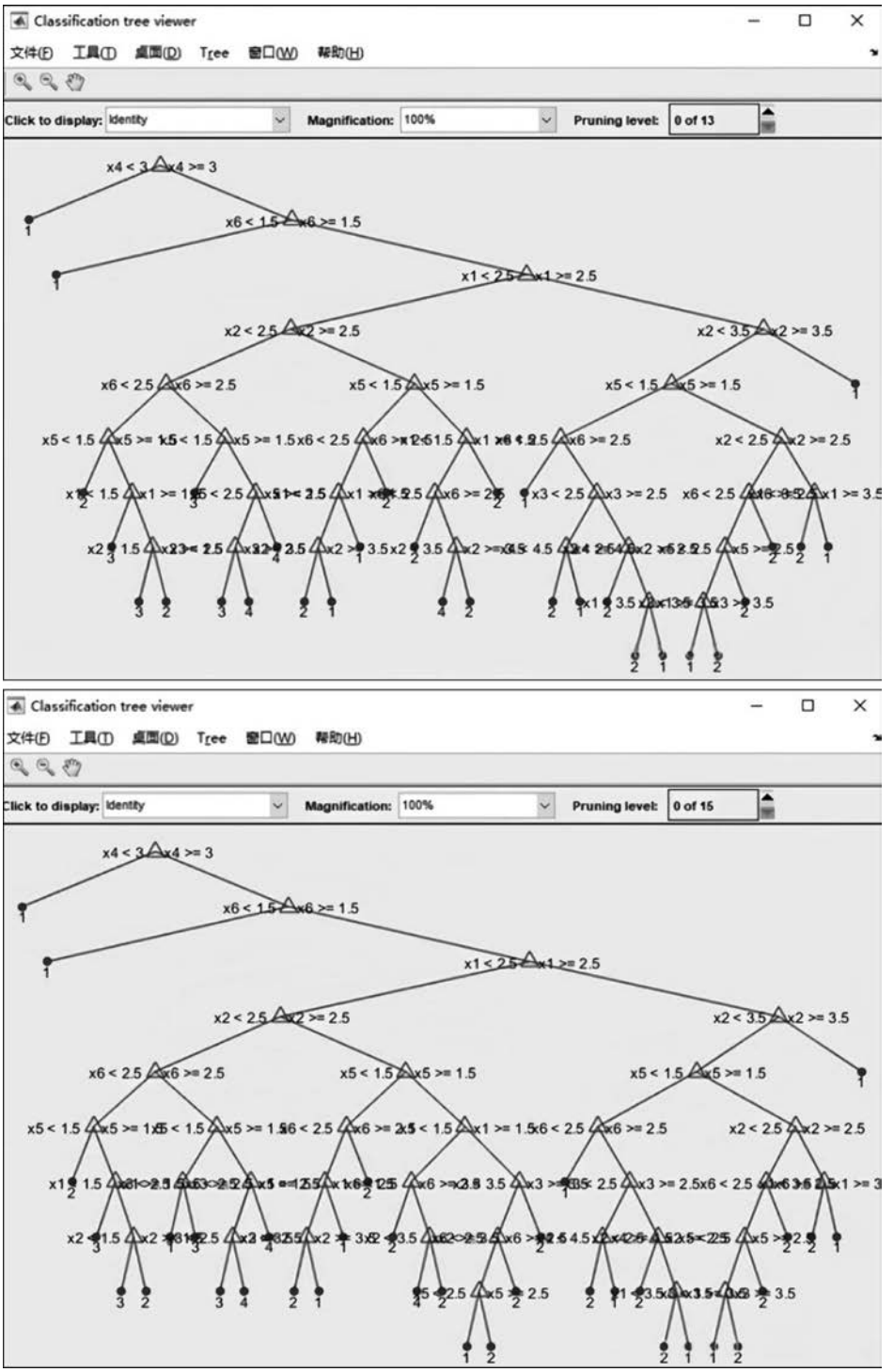


图 5.16　MATLAB 代码实现结果

3. 应用实例 3——恶性乳腺肿瘤判断

具体 MATLAB 代码实现如下：

```
clear; all;
clc;
warning off
load data.mat
a = randperm(569);
Train = data(a(1:500),:);
Test = data(a(501:end),:);
P_train = Train(:,3:end);
T_train = Train(:,2);
P_test = Test(:,3:end);
T_test = Test(:,2);
ctree = ClassificationTree.fit(P_train,T_train);
view(ctree);
view(ctree,'mode','graph');
T_sim = predict(ctree,P_test);
count_B = length(find(T_train == 1));
count_M = length(find(T_train == 2));
rate_B = count_B / 500;
rate_M = count_M / 500;
total_B = length(find(data(:,2) == 1));
total_M = length(find(data(:,2) == 2));
number_B = length(find(T_test == 1));
number_M = length(find(T_test == 2));
number_B_sim = length(find(T_sim == 1 & T_test == 1));
number_M_sim = length(find(T_sim == 2 & T_test == 2));
disp(['病例总数:' num2str(569)...
      '良性:' num2str(total_B)...
      '恶性:' num2str(total_M)]);
disp(['训练集病例总数:' num2str(500)...
      '良性:' num2str(count_B)...
      '恶性:' num2str(count_M)]);
disp(['测试集病例总数:' num2str(69)...
      '良性:' num2str(number_B)...
      '恶性:' num2str(number_M)]);
disp(['良性乳腺肿瘤确诊:' num2str(number_B_sim)...
      '误诊:' num2str(number_B - number_B_sim)...
      '确诊率 p1 = ' num2str(number_B_sim/number_B * 100) '%']);
disp(['恶性乳腺肿瘤确诊:' num2str(number_M_sim)...
      '误诊:' num2str(number_M - number_M_sim)...
      '确诊率 p2 = ' num2str(number_M_sim/number_M * 100) '%']);
[~,~,~,bestlevel] = cvLoss(ctree,'subtrees','all','treesize','min')
cptree = prune(ctree,'Level',bestlevel);
view(cptree,'mode','graph')
```

MATLAB 代码实现结果如图 5.17 所示。

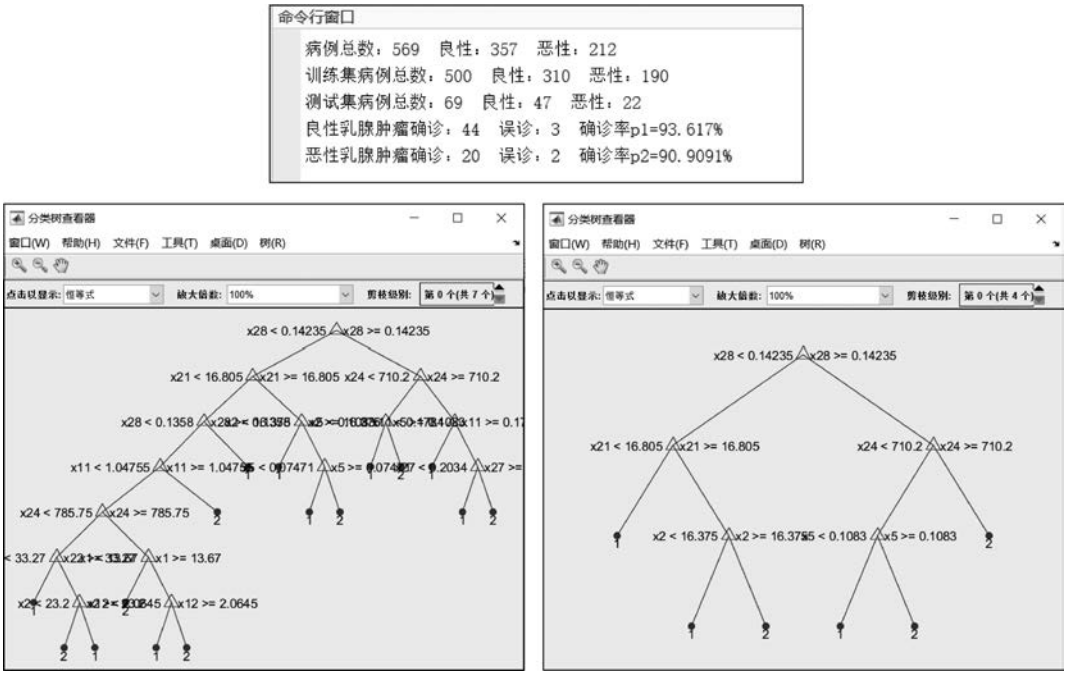


图 5.17　MATLAB 代码实现结果

4. 应用实例 4——商店销量与天气的关系

【例 5-12】 构建如下数据集的决策树模型。

序号	天气	是否周末	是否有促销	销量
1	坏	是	是	高
2	坏	是	是	高
3	坏	是	是	高
4	坏	否	是	高
5	坏	是	是	高
6	坏	否	是	高
7	坏	是	否	高
8	好	是	是	高
9	好	是	否	高
10	好	是	是	高
11	好	是	是	高
12	好	是	是	高
13	好	是	是	高
14	坏	是	是	低
15	好	否	是	高

16	好	否	是	高
17	好	否	是	高
18	好	否	是	高
19	好	否	否	高
20	坏	否	否	低
21	坏	否	是	低
22	坏	否	是	低
23	坏	否	是	低
24	坏	否	否	低
25	坏	是	否	低
26	好	否	是	低
27	好	否	是	低
28	坏	否	否	低
29	坏	否	否	低
30	好	否	否	低
31	坏	是	否	低
32	好	否	是	低
33	好	否	否	低
34	好	否	否	低

解：采用 ID3 算法构建决策树模型的具体步骤如下。

(1) 计算总的信息熵，其中数据中总记录数为 34，而销售数量为“高”的数据有 18，“低”的数据有 16。

$$I(18,16)=-\frac{18}{34}\log_2\frac{18}{34}-\frac{16}{34}\log_2\frac{16}{34}=0.997503$$

(2) 根据

$$I(s_1,s_2,\cdots,s_m)=-\sum_{i=1}^{m}P_i\log_2(P_i),\quad E(A)=\sum_{j-1}^{k}\frac{s_{1j}+s_{2j}+\cdots+s_{mj}}{s}I(s_{1j},s_{2j},\cdots,s_{mj})$$

计算每个测试属性的信息熵。对于天气属性，其属性值有“好”和“坏”两种。其中天气为“好”的条件下，销售数量为“高”的记录为 11，销售数量为“低”的记录为 6，可表示为(11,6)；天气为“坏”的条件下，销售数量为“高”的记录为 7，销售数量为“低”的记录为 10，可表示为(7,10)。则天气属性的信息熵计算过程如下：

$$I(11,6)=-\frac{11}{17}\log_2\frac{11}{17}-\frac{6}{17}\log_2\frac{6}{17}=0.936667$$

$$I(7,10)=-\frac{7}{17}\log_2\frac{7}{17}-\frac{10}{17}\log_2\frac{10}{17}=0.977418$$

$$E(\text{天气})=\frac{17}{34}I(11,6)+\frac{17}{34}I(7,10)=0.957403$$

对于是否周末属性，其属性值有“是”和“否”两种。其中是否周末属性为“是”的条件下，销售数量为“高”的记录为11，销售数量为“低”的记录为3，可表示为(11,3)；周末属性为“否”的条件下，销售数量为“高”的记录为7，销售数量为“低”的记录为13，可表示为(7,13)。则节假日属性的信息熵计算过程如下：

$$I(11,3)=-\frac{11}{14}\log_2\frac{11}{14}-\frac{3}{14}\log_2\frac{3}{14}=0.749595$$

$$I(7,13)=-\frac{7}{20}\log_2\frac{7}{20}-\frac{13}{20}\log_2\frac{13}{20}=0.934068$$

$$E(\text{是否周末})=\frac{14}{34}I(11,3)+\frac{20}{34}I(7,13)=0.858109$$

对于是否有促销属性，其属性值有“是”和“否”两种。其中是否有促销属性为“是”的条件下，销售数量为“高”的记录为15，销售数量为“低”的记录为7，可表示为(15,7)；有促销属性为“否”的条件下，销售数量为“高”的记录为3，销售数量为“低”的记录为9，可表示为(3,9)。则是否有促销属性的信息熵计算过程如下：

$$I(15,7)=-\frac{15}{22}\log_2\frac{15}{22}-\frac{7}{22}\log_2\frac{7}{22}=0.902393$$

$$I(3,9)=-\frac{3}{12}\log_2\frac{3}{12}-\frac{9}{12}\log_2\frac{9}{12}=0.811278$$

$$E(\text{是否有促销})=\frac{22}{34}I(15,7)+\frac{12}{34}I(3,9)=0.870235$$

根据

$$\text{Gain}(A)=I(s_1,s_2,\cdots,s_m)-E(A)$$

计算天气、是否周末和是否有促销属性的信息增益值：

$\text{Gain}(\text{天气})=I(18,16)-E(\text{天气})=0.997503-0.957043=0.04046$

$\text{Gain}(\text{是否周末})=I(18,16)-E(\text{是否周末})=0.997503-0.858109=0.139394$

$\text{Gain}(\text{是否有促销})=I(18,16)-E(\text{是否有促销})=0.997503-0.870235=0.127268$

(3) 由计算结果可以知道是否周末属性的信息增益值最大，它的两个属性值“是”和“否”作为该根节点的两个分支。然后按照上面的步骤继续对该根节点的两个分支进行节点的划分，针对每一个分支节点继续进行信息增益的计算，如此循环反复，直到没有新的节点分支，最终构成一棵决策树。生成的决策树模型如图5.18所示。

根据决策树模型，得到如下分类结果：若周末属性为“是”，天气为“好”，则销售数量为“高”；若周末属性为“是”，天气为“坏”，促销属性为“是”，则销售数量为“高”；若周末属性为“是”，天气为“坏”，促销属性为“否”，则销售数量为“低”；若周末属性为“否”，促销属性为“否”，则销售数量为“低”；若周末属性为“否”，促销属性为“是”，天气为“好”，则销售数量为“高”；若周末属性为“否”，促销属性为“是”，天气为“坏”，则销售数量为“低”。

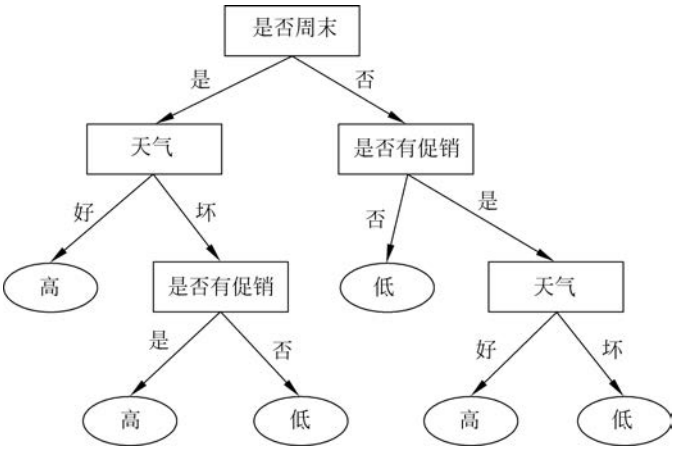


图 5.18　决策树模型

决策树模型的 MATLAB 程序实现代码如下：

```
clear ;
function [ matrix,attributes,activeAttributes ] = id3_preprocess( )
attributes = txt(1,2:end);
activeAttributes = ones(1,length(attributes) - 1);
data = txt(2:end,2:end);
[rows,cols] = size(data);
matrix = zeros(rows,cols);
for j = 1:cols
    matrix(:,j) = cellfun(@trans2onezero,data(:,j));
end
end
function flag = trans2onezero(data)
    if strcmp(data,'坏') ||strcmp(data,'否')...
        ||strcmp(data,'低')
        flag = 0;
        return ;
    end
    flag = 1;
end
function [ tree ] = id3( examples, attributes, activeAttributes )
if (isempty(examples));
    error('必须提供数据!');
end
numberAttributes = length(activeAttributes);
numberExamples = length(examples(:,1));
tree = struct('value', 'null', 'left', 'null', 'right', 'null');
lastColumnSum = sum(examples(:, numberAttributes + 1));
if (lastColumnSum == numberExamples);
    tree.value = 'true';
    return
end
```

```
    if (lastColumnSum == 0);
    tree.value = 'false';
    return
end
if (sum(activeAttributes) == 0);
    if (lastColumnSum >= numberExamples / 2);
        tree.value = 'true';
    else
        tree.value = 'false';
    end
    return
end
p1 = lastColumnSum / numberExamples;
if (p1 == 0);
    p1_eq = 0;
else
    p1_eq = -1 * p1 * log2(p1);
end
p0 = (numberExamples - lastColumnSum) / numberExamples;
if (p0 == 0);
    p0_eq = 0;
else
    p0_eq = -1 * p0 * log2(p0);
end
currentEntropy = p1_eq + p0_eq;
gains = -1 * ones(1,numberAttributes);
for i = 1:numberAttributes;
    if (activeAttributes(i))
        s0 = 0; s0_and_true = 0;
        s1 = 0; s1_and_true = 0;
        for j = 1:numberExamples;
            if (examples(j,i));
                s1 = s1 + 1;
                if (examples(j, numberAttributes + 1));
                    s1_and_true = s1_and_true + 1;
                end
            else
                s0 = s0 + 1;
                if (examples(j, numberAttributes + 1));
                    s0_and_true = s0_and_true + 1;
                end
            end
        end
        if (~s1);
            p1 = 0;
        else
            p1 = (s1_and_true / s1);
        end
        if (p1 == 0);
            p1_eq = 0;
```

```
        else
            p1_eq = -1 * (p1) * log2(p1);
        end
        if (~s1);
            p0 = 0;
        else
            p0 = ((s1 - s1_and_true) / s1);
        end
        if (p0 == 0);
            p0_eq = 0;
        else
            p0_eq = -1 * (p0) * log2(p0);
        end
        entropy_s1 = p1_eq + p0_eq;
        if (~s0);
            p1 = 0;
        else
            p1 = (s0_and_true / s0);
        end
        if (p1 == 0);
            p1_eq = 0;
        else
            p1_eq = -1 * (p1) * log2(p1);
        end
        if (~s0);
            p0 = 0;
        else
            p0 = ((s0 - s0_and_true) / s0);
        end
        if (p0 == 0);
            p0_eq = 0;
        else
            p0_eq = -1 * (p0) * log2(p0);
        end
        entropy_s0 = p1_eq + p0_eq;
        gains(i) = currentEntropy - ((s1/numberExamples) * entropy_s1) - ((s0/
numberExamples) * entropy_s0);
    end
end
[~, bestAttribute] = max(gains);
tree.value = attributes{bestAttribute};
activeAttributes(bestAttribute) = 0;
examples_0 = examples(examples(:,bestAttribute) == 0,:);
examples_1 = examples(examples(:,bestAttribute) == 1,:);
if (isempty(examples_0));
    leaf = struct('value', 'null', 'left', 'null', 'right', 'null');
    if (lastColumnSum >= numberExamples / 2); % for matrix examples
        leaf.value = 'true';
    else
        leaf.value = 'false';
```

```
    end
    tree.left = leaf;
else
    tree.left = id3(examples_0, attributes, activeAttributes);
end
if (isempty(examples_1));
    leaf = struct('value', 'null', 'left', 'null', 'right', 'null');
    if (lastColumnSum >= numberExamples / 2);
        leaf.value = 'true';
    else
        leaf.value = 'false';
    end
    tree.right = leaf;
else
    tree.right = id3(examples_1, attributes, activeAttributes);
end
return
End
function [nodeids_,nodevalue_] = print_tree(tree)
global nodeid nodeids nodevalue;
nodeids(1) = 0; % 根节点的值为 0
nodeid = 0;
nodevalue = {};
if isempty(tree)
    disp('空树!');
    return ;
end
queue = queue_push([],tree);
while ~isempty(queue)
    [node,queue] = queue_pop(queue);
    visit(node,queue_curr_size(queue));
    if ~strcmp(node.left,'null')
        queue = queue_push(queue,node.left);
    end
    if ~strcmp(node.right,'null')
        queue = queue_push(queue,node.right);
    end
end
nodeids_ = nodeids;
nodevalue_ = nodevalue;
end
function visit(node,length_)
    global nodeid nodeids nodevalue;
    if isleaf(node)
        nodeid = nodeid + 1;
        fprintf('叶子节点,node: %d\t,属性值: %s\n', ...
        nodeid, node.value);
        nodevalue{1,nodeid} = node.value;
    else
        nodeid = nodeid + 1;
```

```
            nodeids(nodeid + length_ + 1) = nodeid;
            nodeids(nodeid + length_ + 2) = nodeid;
            fprintf('node: %d\t 属性值: %s\t,左子树为节点:node%d,右子树为节点:node%d\n'
, ...
            nodeid, node.value,nodeid + length_ + 1,nodeid + length_ + 2);
            nodevalue{1,nodeid} = node.value;
        end
    end
    function flag = isleaf(node)
        if strcmp(node.left,'null') && strcmp(node.right,'null')
            flag = 1;
        else
            flag = 0;
        end
    End
```

结果如图 5.19 所示。

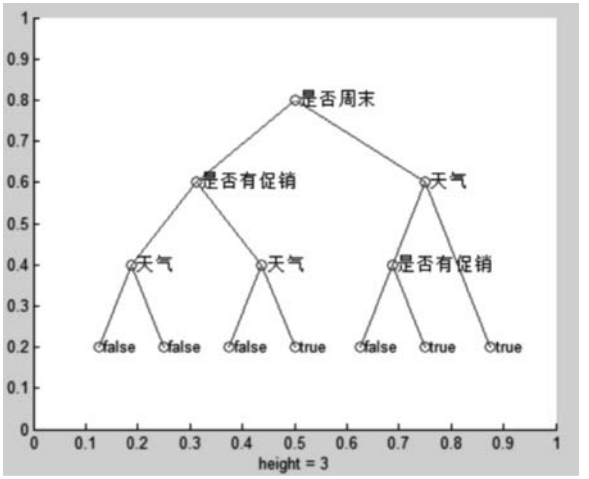


图 5.19 MATLAB 程序仿真结果

5.3 随机森林

5.2 节研究了决策树的相关内容,本节将详细研究随机森林(Random Forest,RF)。本节内容主要包括:

- 随机森林的定义及分类原理;
- 随机森林的收敛性;
- 随机森林的特性;
- 随机森林的构造方法;
- 随机森林的推广。

5.3.1 随机森林的基本概念

从决策树算法的介绍中可以发现，决策树有些与生俱来的缺点。

(1) 分类规则复杂。

(2) 收敛到非全局的局部最优解。

(3) 容易出现过度拟合。

因此，很多学者通过聚集多个模型提高预测精度，这些方法称为组合(ensemble)或分类器组合(classifier combination)方法。组合方法首先利用训练数据构建一组基分类模型(base classifier)，然后通过对每个基分类模型的预测值进行投票(因变量为分类变量时)或取平均值(因变量为连续数值变量时)决定最终预测值。

为了生成这些组合模型，通常需要生成随机向量控制组合中每个决策树的生长。Bagging 是早期组合树方法之一，又称自助聚集(bootstrap aggregating)，是一种从训练集中随机抽取部分样本(不一定有放回抽样)生成决策树的方法。另外一种方法是随机分割选取，该方法是在每个节点从 k 个最优分割中随机选取一种分割。Breiman 通过从原始训练集中随机抽取输出变量得到新的训练集。Ho 对随机子空间(random subspace)方法做了很多研究，该方法通过对特征变量随机选取子集生成每棵决策树。Amit 和 Geman 定义了很多几何属性以及从这些随机选择属性中寻找每个节点的最优分割。该方法对 Breiman 在 1996 年提出随机森林起了很大的启发作用。

1. 随机森林的定义及分类原理

以上这些方法的一个共同特征是为第 k 棵决策树生成随机向量θ_k，且θ_k 独立同分布于前面的随机向量θ_1，θ_2，…，θ_{k-1}。利用训练集和随机向量θ_k 生成一棵决策树，得到分类模型 $h(X,\theta_k)$，其中 X 为输入变量(自变量)。比如，在 bagging 方法中随机向量θ 可以理解为是通过随机扔 N 把飞镖在 N 个箱子上扔中的结果生成，其中 N 是训练集中的样本记录数。在生成许多决策树后，通过投票方法或取平均值作为最后结果，称这个为随机森林方法。

随机森林的组成元素为 n 棵决策树，每棵树的判断原理与单棵决策树决策并无较大差别，但是每棵树之间并不是完全相同。从原始训练样本集 N 中有放回地重复随机抽取 k 个样本生成新的训练样本集合，然后根据自助样本集生成 k 个分类树组成随机森林，新数据的分类结果按分类树投票多少形成的分数而定。其实质是对决策树算法的一种改进，将多个决策树合并在一起，每棵树的建立依赖于一个独立抽取的样品，森林中的每棵树具有相同的分布，分类误差取决于每一棵树的分类能力和它们之间的相关性。特征选择采用随机的方法分裂每一个节点，然后比较不同情况下产生的误差。能够检测到的内在估计误差、分类能力和相关性决定选择特征的数目。单棵树的分类能力可能很小，但在随机产生大量的决策树后，一个测试样品可以通过每一棵树的分类结果经统计后选择最可能的分类，如图 5.20 所示。

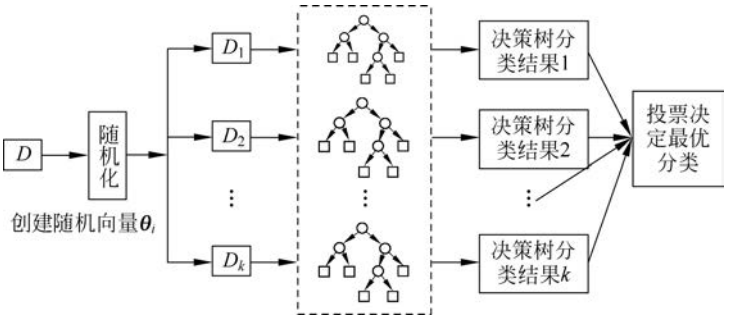


图 5.20　随机森林结构示意图

2. 随机森林的收敛性

给定一组分类模型$\{h_1(\boldsymbol{X}),h_2(\boldsymbol{X}),\cdots,h_k(\boldsymbol{X})\}$，每个分类魔性的训练集都是从原始数据集$(\boldsymbol{X},\boldsymbol{Y})$随机抽样所得。则可以得到其余量函数(margin function)：

$$\mathrm{mg}(\boldsymbol{X},\boldsymbol{Y})=av_kI[h_k(\boldsymbol{X})=\boldsymbol{Y}]-\max_{j\neq k}av_kI[h_k(\boldsymbol{X})=j]$$

余量函数用来度量平均正确分类数据超过平均错误分类数的程度。余量值越大，分类预测就越可靠。外推误差(泛化误差)可写成：

$$\mathrm{PE}^*=P_{\boldsymbol{X},\boldsymbol{Y}}[\mathrm{mg}(\boldsymbol{X},\boldsymbol{Y})<0]$$

当决策树分类模型足够多，$h_k(\boldsymbol{X})=h(\boldsymbol{X},\theta_k)$服从于强大数定律。

【定理 5.1】证明，随着决策树分类模型的增加，所有序列$\theta_1,\theta_2,\cdots,\theta_k$，$\mathrm{PE}^*$几乎处处收敛于

$$P_{\boldsymbol{X},\boldsymbol{Y}}\{P_{\theta}[h(\boldsymbol{X},\theta)=\boldsymbol{Y}]-\max_{j\neq k}P_{\theta}[h(\boldsymbol{X},\theta)=j]<0\}$$

证明：

可以证明对于所有的$\boldsymbol{X}$，在序列空间$\theta_1,\theta_2,\cdots$上，存在一个零概率集合C(也就是在集合C外)使得：

$$\frac{1}{N}\sum_{n=1}^{N}I[h(\theta_n,\boldsymbol{X})=j]\rightarrow P_{\theta}[h(\theta_n,\boldsymbol{X})=j]$$

对于给定的训练集和给定的参数θ下，所有满足$h(\theta_n,\boldsymbol{X})=j$的$X$集合是超矩阵(hyper-rectangles)的并。对于所有的$h(\theta,\boldsymbol{X})$，只存在一个有限数为K的超矩阵，记为$S_1S_2\cdots S_k$。假如$\{\boldsymbol{X}:h(\theta,\boldsymbol{X})=j\}=S_k$，定义$\varphi(\theta)=k$。令$N_k$为前$N$个实验中$\varphi(\theta_n)=k$的次数。则：

$$\frac{1}{N}\sum_{n=1}^{N}I[h(\theta_n,\boldsymbol{X})=j]=\frac{1}{N}\sum_{k}N_kI(\boldsymbol{X}\in S_k)$$

通过大数定理得：

$$N_k=\frac{1}{N}\sum_{n=1}^{N}I[\varphi(\theta_n)=k]$$

几乎处处收敛于 $P_\theta[\varphi(\theta_n)=k]$。在所有集合的并上，给定一个零概率集合 C（也就是在集合 C 外）下，对所有的 k，收敛性不一定都存在。

$$\frac{1}{N}\sum_{n=1}^{N}I[h(\theta_n,\boldsymbol{X})=j]\rightarrow\sum_{k}P_\theta[\phi(\theta_n)=k]I(\boldsymbol{X}\in S_k)$$

右边是 $P_\theta[h(\theta_n,\boldsymbol{X})=j]$，所以得证。

这个定理说明了为什么 RFC 方法不会随着决策树的增加而产生过度拟合的问题，但要注意的是可能会产生一定限度内的泛化误差。

3. 随机森林的特性

随机森林具有以下优点。

(1) 在现有算法中，随机森林算法的精度是无可比拟的。

(2) 随机森林能够高效处理大数据集。

(3) 随机森林可以处理成千上万的输入属性。

(4) 随机森林在分类的应用中可以计算出不同变量属性的重要性。

(5) 在构建随机森林的过程中可以产生一个关于泛化误差的内部无偏估计。

(6) 当大量数据缺失的时候，随机森林采用高效的方法估计缺失的数据并保持着准确率。

(7) 在不平衡的数据集中，随机森林可以提供平衡误差的方法。

(8) 可以保存已经生成的随机森林，方便解决以后的问题。

(9) 原型(Prototype)的计算可以给出属性变量本身和分类的相关性。

(10) 计算样本实例之间的近似性(Proximity)，可以用于聚类分析、异常分析或者数据的其他有趣的视图。

随机森林存在以下缺点。

(1) 在某些噪音比较大的样本集上，随机森林模型容易陷入过拟合。

(2) 取值划分比较多的特征容易对随机森林的决策产生更大的影响，从而影响拟合的模型的效果。

5.3.2　随机森林的构造方法

随机森林的构造方法具体步骤如下。

步骤 1：假如有 N 个样本，则有放回地随机选择 N 个样本(每次随机选择一个样本，然后返回继续选择)。选择好了的 N 个样本用来训练一个决策树，作为决策树根节点处的样本。

步骤 2：当每个样本有 M 个属性时，在决策树的每个节点需要分裂时，随机从这 M 个

属性中选取出 m 个属性，满足条件 $m<<M$。然后从这 m 个属性中采用某种策略（比如说信息增益）来选择一个属性作为该节点的分裂属性。

步骤 3：决策树形成过程中每个节点都要按照步骤 2 进行分裂（很容易理解，如果下一次该节点选出来的那个属性是刚刚其父节点分裂时用过的属性，则该节点已经达到了叶节点，无须继续分裂了），一直到不能够再分裂为止。注意整个决策树形成过程中没有进行剪枝。

步骤 4：按照步骤 1～3 建立大量的决策树，这样就构成了随机森林了。

图 5.21 为重抽样的示意图。

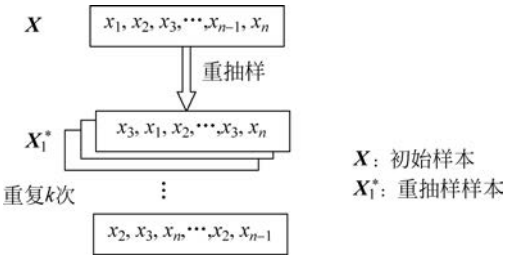


图 5.21　Bootstrap 重抽样示意图

从上面的步骤可以看出，随机森林的随机性体现在每棵树的训练样本是随机的，树中每个节点的分类属性也是随机选择的。有了这两个随机的保证，随机森林就不会产生过拟合的现象了。随机森林有两个参数需要人为控制，一个是森林中树的数量，一般建议取很大；另一个是 m 的大小，推荐 m 的值为 M 的均方根。

如果不进行随机抽样，每棵树的训练集都一样，那么最终训练出的树分类结果也是完全一样的。

如果不是有放回的抽样，那么每棵树的训练样本都是不同的，都是没有交集的，这样每棵树都是“有偏的”，都是绝对“片面的”（当然这样说可能不对），也就是说每棵树训练出来都是有很大的差异的。而随机森林最后分类取决于多棵树（弱分类器）的投票表决，这种表决应该是“求同”，因此使用完全不同的训练集来训练每棵树这样对最终分类结果是没有帮助的，这样无异于是“盲人摸象”。

随机森林分类效果（错误率）与两个因素有关。

（1）森林中任意两棵树的相关性：相关性越大，错误率越大。

（2）森林中每棵树的分类能力：每棵树的分类能力越强，整个森林的错误率越低。

减小特征选择个数 m，树的相关性和分类能力也会相应降低；增大 m，两者也会随之增大。所以关键问题是如何选择最优的 m（或者是范围），这也是随机森林唯一的一个参数。

构建随机森林的关键问题就是如何选择最优的 m，要解决这个问题主要依据计算袋外（Out-Of-Bag，OOB）错误率。

随机森林有一个重要的优点是，没有必要对它进行交叉验证或者用一个独立的测试集

来获得误差的一个无偏估计。它可以在内部进行评估，也就是说在生成的过程中就可以对误差建立一个无偏估计。

在构建每棵树时，对训练集使用了不同的 bootstrap sample(随机且有放回地抽取)。所以对于每棵树而言(假设对于第 k 棵树)，大约有 1/3 的训练实例没有参与第 k 棵树的生成，它们称为第 k 棵树的 OOB 样本。

而这样的采样特点就允许我们进行 OOB 估计，它的计算方式如下：

(1) 对每个样本，计算它作为 OOB 样本的树对它的分类情况(约 1/3 的树)。

(2) 以简单多数投票作为该样本的分类结果。

(3) 用误分个数占样本总数的比率作为随机森林的 OOB 误分率。

5.3.3 随机森林的推广

1. ET

ET(Extra Trees)是随机森林的一个变种，原理几乎和随机森林一模一样，仅有区别如下。

(1) 对于每个决策树的训练集，随机森林采用的是随机采样 bootstrap 选择采样集作为每个决策树的训练集，而 ET 一般不采用随机采样，即每个决策树采用原始训练集。

(2) 在选定了划分特征后，随机森林的决策树会基于基尼系数或均方差等原则，选择一个最优的特征值划分点，这和传统的决策树相同。但是 ET 比较激进，会随机地选择一个特征值划分决策树。

以二叉树为例，当特征属性是类别的形式时，随机选择具有某些类别的样本为左分支，而把具有其他类别的样本作为右分支；当特征属性是数值的形式时，随机选择一个处于该特征属性的最大值和最小值之间的任意数，当样本的该特征属性值大于该值时，作为左分支，当小于该值时，作为右分支。这样就实现了在该特征属性下把样本随机分配到两个分支上的目的。然后计算此时的分叉值(如果特征属性是类别的形式，可以应用基尼指数；如果特征属性是数值的形式，可以应用均方误差)。遍历节点内的所有特征属性，按上述方法得到所有特征属性的分叉值，选择分叉值最大的形式实现对该节点的分叉。从上面的介绍可以看出，这种方法比随机森林的随机性更强。

由于随机选择了特征值的划分点位，而不是最优点位，这样会导致生成的决策树的规模一般会大于随机森林所生成的决策树。也就是说，模型的方差相对于随机森林进一步减少，在某些时候，ET 的泛化能力比随机森林更好。

对于某棵决策树，由于它的最佳分叉属性是随机选择的，因此使用它的预测结果往往是不准确的，但多棵决策树组合在一起，就可以达到很好的预测效果。

当 ET 构建好以后，也可以应用全部的训练样本得到该 ET 的预测误差。这是因为尽管构建决策树和预测应用的是同一个训练样本集，但由于最佳分叉属性是随机选择的，所以

仍然会得到完全不同的预测结果，用该预测结果就可以与样本的真实响应值比较，从而得到预测误差。如果与随机森林相类比的话，在 ET 中，全部训练样本都是 OOB 样本，所以计算 ET 的预测误差，也就是计算 OOB 误差。

这里仅仅介绍了 ET 算法与随机森林的不同之处，ET 算法的其他内容（如预测、OOB 误差的计算）与随机森林是完全相同的，具体内容可查看关于随机森林的介绍。

2. TRTE

TRTE（Totally Random Trees Embedding）是一种非监督学习的数据转化方法。它将低维的数据集映射到高维，从而让映射到高维的数据更好地运用于分类回归模型。在支持向量机中运用了核方法将低维的数据集映射到高维，此处 TRTE 提供了另外一种方法。

TRTE 在数据转化的过程也使用了类似于 RF 的方法，建立 T 棵决策树来拟合数据。当决策树建立完毕以后，数据集里的每个数据在 T 棵决策树中叶节点的位置也确定。如果有 3 棵决策树，每棵决策树有 5 个叶节点，某个数据特征 x 划分到第一棵决策树的第 2 个叶节点，第二棵决策树的第 3 个叶节点，第三棵决策树的第 5 个叶节点。则 x 映射后的特征编码为（0，1，0，0，0， 0，0，1，0，0， 0，0，0，0，1），有 15 维的高维特征。这里特征维度之间加上空格是为了强调 3 棵决策树各自的子编码。

映射到高维特征后，可以继续使用监督学习的各种分类回归算法了。

3. IForest

IForest（Isolation Forest）是一种异常点检测的方法。它也使用了类似随机森林的方法检测异常点。

1）IForest 算法原理

IForest 属于非参数化（non-parametric）和无监督（unsupervised）的方法，既不用定义数学模型也不需要有标记的训练。对于如何查找哪些点是否容易被孤立（isolated），IForest 使用了一套非常高效的策略。假设用一个随机超平面切割数据空间（data space），切一次可以生成两个子空间。继续用一个随机超平面切割每个子空间，循环下去，直到每个子空间里面只有一个数据点为止。直观上讲，可以发现那些密度很高的簇被切分很多次才会停止切割，但是密度很低的点很早就停到一个子空间了。

IForest 算法得益于随机森林的思想，与随机森林由大量决策树组成一样，IForest 森林也由大量的二叉树组成，IForest 中的树称为 ITree（Isolation Tree），ITree 和决策树不太一样，其构建过程也比决策树简单，是一个完全随机的过程。

假设数据集有 N 条数据，构建一棵 ITree 时，从 N 条数据中均匀抽样（一般是无放回抽样）出 n 个样本出来，作为这棵树的训练样本。在样本中，随机选出一个特征，并在这个特征的所有值范围内（最小值和最大值之间）随机选一个值，对样本进行二叉划分，将样本中小于该值的划分到节点的左边，大于或等于该值的划分到节点的右边。由此得到一个分裂条件和左右两边的数据集，然后分别在左右两边的数据集上重复上面的过程，直到数据集只

有一条记录或者达到了树的限定高度。

由于异常数据较小且特征值和正常数据差别很大。因此，构建 ITree 的时候，异常数据离根更近，而正常数据离根更远。一棵 ITree 的结果往往不可信，IForest 算法通过多次抽样，构建多棵二叉树。最后整合所有树的结果，并取平均深度作为最终的输出深度，由此计算数据点的异常分支。

2) Isolation Forest 算法步骤

怎么切割这个数据空间是 IForest 的设计核心思想，此处仅介绍最基本的方法，由于切割是随机的，所以需要用 ensemble 方法得到一个收敛值(蒙特卡洛方法)，即反复从头开始切，然后平均每次切的结果。IForest 由 t 个 ITree 组成，每个 ITree 是一个二叉树结构，所以下面先介绍 ITree 的构建，然后再看 IForest 树的构建。

3) ITree 的构建

ITree 是一种随机二叉树，每个节点要么有两个孩子(即叶节点)，要么一个孩子都没有。给定一堆数据集 D，这里 D 的所有属性都是连续型的变量，ITree 的构成过程如下。

(1) 随机选择一个属性 Attr。

(2) 随机选择该属性的一个值 Value。

(3) 根据 Attr 对每条记录进行分类，把 Attr 小于 Value 的记录放在左孩子，把大于或等于 Value 的记录放在右孩子。

(4) 然后递归地构造左孩子和右孩子，直到满足以下条件：传入的数据集只有一条记录或者多条一样的记录；树的高度达到了限定高度。

ITree 构建好之后，就可以对数据进行预测了。预测的过程就是把测试记录在 ITree 上，看看测试记录在哪个叶节点。ITree 能有效检测异常的假设是：异常点一般都是稀有的，在 ITree 中会很快被划分到叶节点。注意，异常点一般来说是稀疏的，因此可以用较小次划分把它们归结到单独的区域中，或者说只包含它的空间的面积较大。

因此可以利用叶节点到根节点的路径 $h(x)$ 长度判断一条记录 x 是否是异常点(也就是根据 $h(x)$ 判断 x 是否是异常点)。对于一个包含 n 条记录的数据集，其构造的树的高度最小值为 $\log(n)$，最大值为 $n-1$，用 $\log(n)$ 和 $n-1$ 归一化不能保证有界和不方便比较，所以用复杂一点的归一化公式：

$$S(x,n)=2^{-\frac{h(x)}{c(n)}}$$

$$c(n)=2H(n-1)-[2(n-1)/n]$$

$$H(k)=\ln(k)+\zeta$$

其中，$S(x,n)$ 为记录 x 在 n 个样本的训练数据构成的 ITree 的异常指数，$S(x,n)$ 取值范围为 $[0,1]$；$\zeta=0.5772156649$。

记录 x 在由 n 个样本的训练数据构成 ITree 的异常指数，取值范围为 $[0,1]$，异常情况

的判断分以下几种情况。

(1) 越接近 1 表示是异常点的可能性高。

(2) 越接近 0 表示是正常点的可能性高。

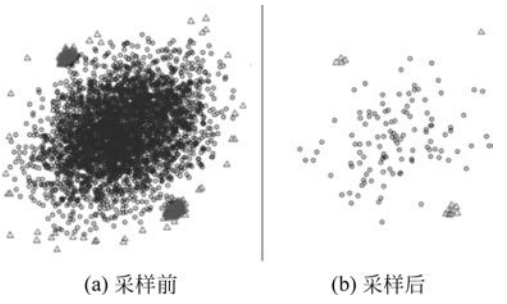
(3) 如果大部分的训练样本的 $S(x,n)$都接近于 0.5,说明整个数据集都没有明显的异常值。

如果是随机选属性,随机选属性值,一棵树这么随机选肯定不行,但是把多棵树结合起来就变得强大了。

4) IForest 的构建

给定一个包含 n 条记录的数据集 D,构造 IForest 和随机森林的方法有点类似,都是随机采样一部分数据集去构造一棵树,保证不同树之间的差异性,不过 IForest 与随机森林不同,采样的数据量 P_{si} 不需要等于 n,可以远远小于 n。

图 5.22(a)是原始数据,图 5.22(b)是采样数据,蓝色是正常样本,红色是异常样本。可以看到,在采样之前,正常样本和异常样本出现重叠,因此很难分开。但采样之后,异常样本和正常样本可以明显地分开。



(a) 采样前　　(b) 采样后

图 5.22　采样前后示意图

构造 IForest 的步骤如下。

(1) 从训练数据中随机选择 n 个点样本作为子采样(subsample),放入树的根节点。

(2) 随机指定一个维度(attribute),在当前节点数据中随机产生一个切割点 p,切割点产生于当前节点数据中指定维度的最大值和最小值之间。

(3) 以此切割点生成了一个超平面,然后将当前节点数据空间划分为 2 个子空间:把指定维度里面小于 p 的数据放在当前节点的左孩子,把大于或等于 p 的数据放在当前节点的右孩子。

(4) 在孩子节点中递归步骤(2)和(3),不断构造新的叶节点,直到叶节点中只有一个数据(无法再继续切割)或者叶节点已达限定高度。

除了限制采样大小之外,还要给每棵 ITree 设置最大高度,这是因为异常数据记录都比较少,其路径长度也比较低,因此只需要把正常记录和异常记录区分开来,关心低于平均高

度的部分就好，这样算法效率更高。这样调整之后，计算需要一点点改进。

获得 t 个 ITree 之后，IForest 训练就结束，然后用生成的 IForest 评估测试数据。对于一个训练数据 X，令其遍历每一棵 ITree，然后计算 X 最终落在每棵树第几层（X 在树的高度）。可以得到 X 在每棵树的高度平均值（average path length over the ITree）。

5）IForest 特性

（1）IForest 具有线性时间复杂度，因为是随机森林的方法，所以可以用于含有海量数据的数据集，通常树的数量越多，算法越稳定。由于每棵树都是互相独立生成的，因此可以部署在大规模分布式系统上来加速运算。

（2）IForest 仅对即全局稀疏点（global anomaly）敏感，不擅长处理局部的相对稀疏点（local anomaly），这样在某些局部异常点较多时检测可能不是很准。同时，IForest 不适用于特别高维的数据，由于每次切割数据空间都是随机选取一个维度和该维度的随机一个特征，建完树后仍然有大量的维度没有被使用，导致算法可靠性降低。

（3）IForest 不适用于特别高维的数据。由于每次切割数据空间都是随机选取一个维度，建完树后仍然有大量的维度信息没有被使用，导致算法可靠性降低。高维空间还可能存在大量噪音维度或者无关维度（irrelevant attributes），影响树的构建。

（4）IForest 推动了重心估计（mass estimation）理论，目前在分类聚类和异常检测中都取得显著效果。

（5）IForest 算法主要有两个参数：二叉树的个数；训练单棵 ITree 时抽取样本的数目。实验表明，当设定为 100 棵树，抽样样本为 256 条的时候，IForest 在大多数情况下都可以取得不错的效果，这也体现了算法的简单高效。

（6）IForest 是无监督的检测算法，目前是异常点检测最常用的算法之一。在实际应用中，并不需要黑白标签。需要注意的是：如果训练样本中异常样本的比例比较高，违背了先前提到的异常检测的基本假设，可能会影响最终的效果；异常检测与具体的应用场景紧密相关，算法检测出的“异常”不一定是实际想要的，所以在特征选择时，需要过滤不太相关的特征，以免识别出一些不相关的“异常”。

5.3.4 随机森林的相关应用与 MATLAB 算例

1. 应用实例 1

```
clear;
clc;
close all;
load imports - 85;
Y = X(:,1);
X = X(:,2:end);
isCategorical = [zeros(15,1);ones(size(X,2) - 15,1)]; % Categorical variable flag
tic
```

```
leaf = 5;
ntrees = 200;
fboot = 1;
disp('Training the tree bagger')
b = TreeBagger(ntrees, X,Y, 'Method','regression', 'oobvarimp','on', 'surrogate', 'on', 'minleaf',
leaf,'FBoot',fboot);
toc
disp('Estimate Output using tree bagger')
x = Y;
y = predict(b, X);
toc
cct = corrcoef(x,y);
cct = cct(2,1);
disp('Create a scatter Diagram')
plot(x,x,'LineWidth',3);
hold on
scatter(x,y,'filled');
hold off
grid on
set(gca,'FontSize',18)
xlabel('Actual','FontSize',25)
ylabel('Estimated','FontSize',25)
title(['Training Dataset, R^2 = ' num2str(cct^2,2)],'FontSize',30)
drawnow
fn = 'ScatterDiagram';
fnpng = [fn,'.png'];
print('-dpng',fnpng);
tic
disp('Sorting importance into descending order')
weights = b.OOBPermutedVarDeltaError;
[B,iranked] = sort(weights,'descend');
toc
disp(['Plotting a horizontal bar graph of sorted labeled weights.'])
figure
barh(weights(iranked),'g');
xlabel('Variable Importance','FontSize',30,'Interpreter','latex');
ylabel('Variable Rank','FontSize',30,'Interpreter','latex');
title(...
    ['Relative Importance of Inputs in estimating Redshift'],...
    'FontSize',17,'Interpreter','latex'...
    );
hold on
barh(weights(iranked(1:10)),'y');
barh(weights(iranked(1:5)),'r');
grid on
xt = get(gca,'XTick');
xt_spacing = unique(diff(xt));
xt_spacing = xt_spacing(1);
yt = get(gca,'YTick');
ylim([0.25 length(weights) + 0.75]);
```

```
xl = xlim;
xlim([0 2.5 * max(weights)]);
for ii = 1:length(weights)
    text(...
        max([0 weights(iranked(ii)) + 0.02 * max(weights)]),ii,...
        ['Column ' num2str(iranked(ii))],'Interpreter','latex','FontSize',11);
end
set(gca,'FontSize',16)
set(gca,'XTick',0:2 * xt_spacing:1.1 * max(xl));
set(gca,'YTick',yt);
set(gca,'TickDir','out');
set(gca, 'ydir', 'reverse' )
set(gca,'LineWidth',2);
drawnow
fn = 'RelativeImportanceInputs';
fnpng = [fn,'.png'];
print('-dpng',fnpng);
disp('Ploting out of bag error versus the number of grown trees')
figure
plot(b.oobError,'LineWidth',2);
xlabel('Number of Trees','FontSize',30)
ylabel('Out of Bag Error','FontSize',30)
title('Out of Bag Error','FontSize',30)
set(gca,'FontSize',16)
set(gca,'LineWidth',2);
grid on
drawnow
fn = 'EroorAsFunctionOfForestSize';
fnpng = [fn,'.png'];
print('-dpng',fnpng);
```

结果如图 5.23 所示。

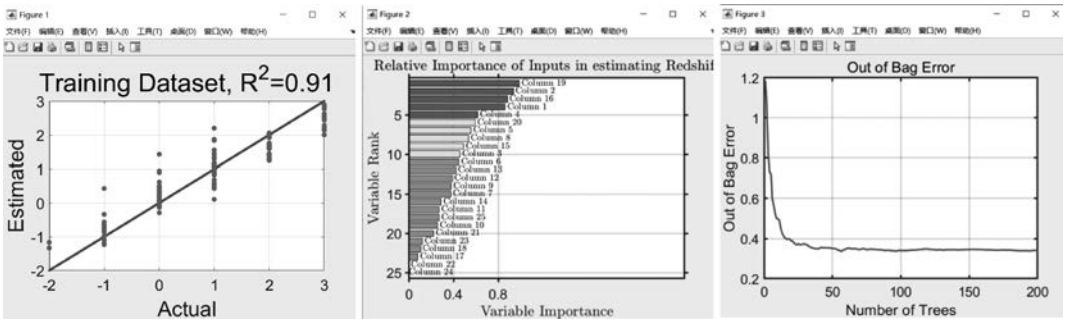


图 5.23　MATLAB 代码实现结果图

2. 应用实例 2

```
clear all;
rnode = cell(3,1);
sn = 300;
```

```
tn = 10;
load('aaa.mat');
n = size(r,1);
discrete_dim = [];
for j = 1:tn
    Sample_num = randi([1,n],1,sn);
    SData = r(Sample_num,:);
    [tree,discrete_dim] = train_C4_5(SData, 5, 10, discrete_dim);
    rnode{j,1} = tree;
end
load('aaa.mat');
T = r;
TData = roundn(T, -1);
result = statistics(tn, rnode, TData, discrete_dim);
gd = T(:,end);
len = length(gd);
count = sum(result == gd);
fprintf('共有 %d 个样本,判断正确的有 %d\n',len,count);
function [tree,discrete_dim] = train_C4_5(S, inc_node, Nu, discrete_dim)
    train_patterns = S(:,1:end-1)';
    train_targets = S(:,end)';
    [Ni, M] = size(train_patterns);
    inc_node = inc_node * M/100;
    if isempty(discrete_dim)
corresponding dimension on the test patterns
        discrete_dim = zeros(1,Ni);
        for i = 1:Ni
            Ub = unique(train_patterns(i,:));
            Nb = length(Ub);
            if (Nb <= Nu)
            end
        end
    end
flag = [];
tree = make_tree(train_patterns, train_targets, inc_node, discrete_dim, max(discrete_dim),
0, flag);
function tree = make_tree(patterns, targets, inc_node, discrete_dim, maxNbin, base, flag)
[N_all, L] = size(patterns);
    if isempty(flag)
        N_choose = randi([1,N_all],1,0.5 * sqrt(N_all));
        Ni_choose = length(N_choose);
        flag.N_choose = N_choose;
        flag.Ni_choose = Ni_choose;
    else
        N_choose = flag.N_choose;
        Ni_choose = flag.Ni_choose;
    end
    Uc = unique(targets);
    tree.dim = 0;
    tree.split_loc = inf;
```

```
    if isempty(patterns)
        return
    end
        if ((inc_node > L) | (L == 1) | (length(Uc) == 1))
        H = hist(targets, length(Uc));
        [m, largest] = max(H);
        tree.Nf = [];
        tree.split_loc = [];
        tree.child = Uc(largest);
        return
    end
    for i = 1:length(Uc)
        Pnode(i) = length(find(targets == Uc(i))) / L;
    end
log2(9/14) - 5/14 * log2(5/14) = 0.940
    Inode = -sum(Pnode. * log(Pnode)/log(2));
        delta_Ib     = zeros(1, Ni_choose);
        split_loc    = ones(1, Ni_choose) * inf;
    for i_idx = 1:Ni_choose
        i = N_choose(i_idx);
        data     = patterns(i,:);
        Ud       = unique(data);
        Nbins    = length(Ud);
        if (discrete_dim(i))
            P    = zeros(length(Uc), Nbins);
            for j = 1:length(Uc)
                for k = 1:Nbins
                    indices = find((targets == Uc(j)) & (patterns(i,:) == Ud(k)));
                    P(j,k) = length(indices);
                end
            end
            Pk = sum(P);
            P1 = repmat(Pk, length(Uc), 1);
            P1 = P1 + eps * (P1 == 0);
            P = P./P1;
            Pk = Pk/sum(Pk);
            info = sum(-P. * log(eps + P)/log(2));
            delta_Ib(i_idx) = (Inode - sum(Pk. * info))/(-sum(Pk. * log(eps + Pk)/log(2)));
        else
            P = zeros(length(Uc), 2);
            [sorted_data, indices] = sort(data);
            sorted_targets = targets(indices);
              I = zeros(1,Nbins);
              delta_Ib_inter     = zeros(1, Nbins);
              for j = 1:Nbins - 1
                  P(:, 1) = hist(sorted_targets(find(sorted_data <= Ud(j))) , Uc);
                  P(:, 2) = hist(sorted_targets(find(sorted_data > Ud(j))) , Uc);
                  Ps = sum(P)/L;
                  P = P/L;
                  Pk = sum(P);
```

```
                    P1 = repmat(Pk, length(Uc), 1);
                    P1 = P1 + eps * (P1 == 0);
                    info = sum( - P./P1. * log(eps + P./P1)/log(2));
                    I(j) = Inode - sum(info. * Ps);
                    delta_Ib_inter(j) = I(j)/( - sum(Ps. * log(eps + Ps)/log(2)));
                end
            [~, s] = max(I);
            delta_Ib(i_idx) = delta_Ib_inter(s);
            split_loc(i_idx) = Ud(s);
        end
    end
[m, dim]     = max(delta_Ib);
dims         = 1:Ni_choose;
dim_all = 1:N_all;
dim_to_all = N_choose(dim);
tree.dim = dim_to_all;
Nf         = unique(patterns(dim_to_all,:));
Nbins     = length(Nf);
tree.Nf = Nf;
tree.split_loc = split_loc(dim);
if (Nbins == 1)
    H = hist(targets, length(Uc));
    [m, largest] = max(H);
    tree.Nf = [];
    tree.split_loc = [];
    tree.child = Uc(largest);
    return
end
    if (discrete_dim(dim_to_all))
        for i = 1:Nbins
            indices = find(patterns(dim_to_all, :) == Nf(i));
            tree.child(i) = make_tree(patterns(dim_all, indices), targets(indices), inc_
node, discrete_dim(dim_all), maxNbin, base, flag);
    else
        indices1 = find(patterns(dim_to_all,:) <= split_loc(dim));
        indices2 = find(patterns(dim_to_all,:) > split_loc(dim));
        if ~(isempty(indices1) | isempty(indices2))
            tree.child(1) = make_tree(patterns(dim_all, indices1), targets(indices1), inc_
node, discrete_dim(dim_all), maxNbin, base + 1, flag);
            tree.child(2) = make_tree(patterns(dim_all, indices2), targets(indices2), inc_
node, discrete_dim(dim_all), maxNbin, base + 1, flag);
        else
            H = hist(targets, length(Uc));
            [m, largest] = max(H);
            tree.child = Uc(largest);
            tree.dim = 0;
        end
end
function [result] = statistics(tn, rnode, PValue, discrete_dim)
    TypeName = {'1','2'};
```

```
    TypeNum = [0 0];
    test_patterns = PValue(:,1:end-1)';
    class_num = length(TypeNum);
    type = zeros(tn,size(test_patterns,2));
    for i = 1:tn
        type(tn,:) = vote_C4_5(test_patterns, 1:size(test_patterns,2), rnode{i,1},
discrete_dim, class_num);
    end
    result = mode(type,1)';
end
function targets = vote_C4_5(patterns, indices, tree, discrete_dim, Uc)
    targets = zeros(1, size(patterns,2));
    if (tree.dim == 0)
        % Reached the end of the tree
        targets(indices) = tree.child;
        return
    end
    dim = tree.dim;
    dims = 1:size(patterns,1);
    if (discrete_dim(dim) == 0)
        in = indices(find(patterns(dim, indices) <= tree.split_loc));
        targets = targets + vote_C4_5(patterns(dims, :), in, tree.child(1), discrete_dim
(dims), Uc);
        in = indices(find(patterns(dim, indices) > tree.split_loc));
        targets = targets + vote_C4_5(patterns(dims, :), in, tree.child(2), discrete_dim
(dims), Uc);
    else
        Uf = unique(patterns(dim,:));
        for i = 1:length(Uf)
            if any(Uf(i) == tree.Nf)
                in = indices(find(patterns(dim, indices) == Uf(i)));
                targets = targets + vote_C4_5(patterns(dims, :), in, tree.child(find(Uf
(i) == tree.Nf)), discrete_dim(dims), Uc);
            end
        end
    end
```

结果如图 5.24 所示。

```
共有464个样本，判断正确的有427
>> 427/464

ans =

    0.9203
```

图 5.24　算法样本数据判断正确率