第 5 章

CHAPTER 5

非线性判别分析

在很多情况下，数据分布情况复杂，样本集非线性可分，进行线性判别分析会有较高的错误率，可以利用非线性判别实现分类。非线性判别分析并没有特定函数形式，因此不适合采用类似于线性判别函数设计的参数估计方法。不同的设计思路产生不同的方法，本章主要学习常见的几种非线性判别分析方法，包括近邻法、二次判别函数、决策树和 Logistic 回归。

5.1 近邻法

近邻法是一种经典的非线性判别分析方法，采用距离度量作为判别函数，直接根据训练样本对未知类别样本进行分类决策。首先了解利用距离度量进行判别的含义，再介绍近邻法决策规则。

5.1.1 最小距离分类器

距离度量是模式识别中常用的方法，如果类别可分，则同一类样本差别相对较小，不同类的样本差别相对较大，差别可以采用样本间的距离来衡量，因此，可以设计基于距离的分类器。

对于两类情况，设两类 ω_1、ω_2 各自的均值向量为 $\boldsymbol{\mu}_1$、$\boldsymbol{\mu}_2$，待分类样本为 $\boldsymbol{x}$，计算 $\boldsymbol{x}$ 到两类均值的距离，将 $\boldsymbol{x}$ 归为距离小的那一类，即若

$$g(\boldsymbol{x}) = \| \boldsymbol{x} - \boldsymbol{\mu}_1 \|^2 - \| \boldsymbol{x} - \boldsymbol{\mu}_2 \|^2 \lessgtr 0, \quad \boldsymbol{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases} \tag{5-1}$$

则分类决策面为 $\boldsymbol{\mu}_1$、$\boldsymbol{\mu}_2$ 连线的垂直平分面，如图 5-1 所示。

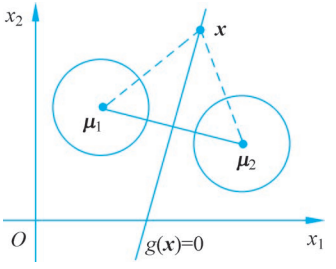


图 5-1 两类的最小距离分类器

对于正态分布模式的贝叶斯决策，在样本集分布呈现特殊情况（$P(\omega_i)=P(\omega_j)$，$\boldsymbol{\Sigma}_j=\sigma^2\boldsymbol{I}$，$i,j=1,2,\cdots,c$）时，设计的判别函数为最小距离分类器（见 2.7 节）。

如果是多类情况，则定义各类的判别函数为

$$g_j(\boldsymbol{x}) = \| \boldsymbol{x} - \boldsymbol{\mu}_j \|^2, \quad j=1,2,\cdots,c \tag{5-2}$$

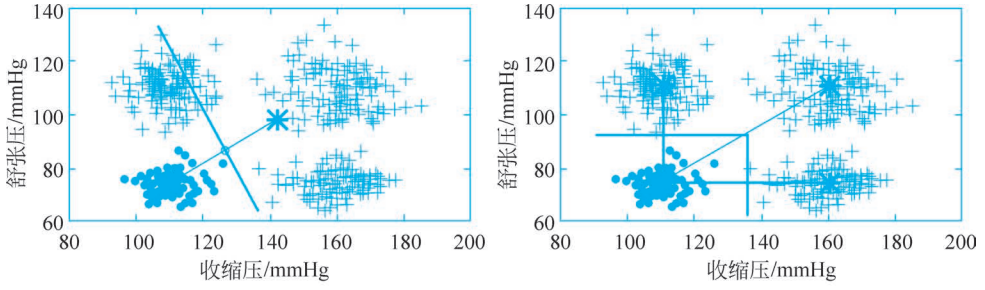
决策时，哪一类的判别函数（距离）最小，则决策为哪一类，即决策规则为

$$若\ g_i(\boldsymbol{x}) < g_j(\boldsymbol{x}),\quad \forall i \neq j,\quad i,j=1,2,\cdots,c,\quad 则\ \boldsymbol{x} \in \omega_i \tag{5-3}$$

很明显，最小距离分类器仅适用于类别线性可分且类间有明显距离的特殊情况，但用均值点作为类的代表点，用距离作为判别函数的思路很有启发性。

5.1.2 分段线性距离分类器

图 5-2 所示为正常血压和高血压数据，采用最小距离分类器，设计的分界面导致很多数据被错误分类，如图 5-2(a)所示。将高血压数据分为 3 个子类，各子类均值点作为该子类的代表点，距离作为判别函数，设计多类分类情况下的最小距离分类器，得到的分界面由多段线性分界面组成，如图 5-2(b)所示。



(a) 最小距离分类器　　(b) 分段线性距离分类器

图 5-2　非线性可分情况下距离分类器示例

对于类别数据呈现多峰分布的情况，将每类划分为若干子类，验证待分类样本 $\boldsymbol{x}$ 到 ω_j 类各子类均值的距离，最小的距离作为该类的判别函数值，将样本归入最近的子类所属的类别，这种分类器称为分段线性距离分类器。

分段线性距离分类器的判别函数表示为

$$g_j(\boldsymbol{x}) = \min_{l=1,2,\cdots,c_j} \| \boldsymbol{x} - \boldsymbol{\mu}_j^l \|,\quad j=1,2,\cdots,c \tag{5-4}$$

c_j 为 ω_j 类的子类数目。

分段线性距离分类器的决策规则为

$$若\ g_i(\boldsymbol{x}) = \min_{j=1,2,\cdots,c} g_j(\boldsymbol{x}),\quad 则\ \boldsymbol{x} \in \omega_i \tag{5-5}$$

分段线性距离分类器分类效果与子类的划分有密切关系，如果对样本没有充足的先验知识，子类划分不一定合理，则会影响分类的效果。

5.1.3 近邻法及仿真实现

1. 最近邻法

将分段线性距离分类器的设计思路发展到极端，把每个样本点都作为一个子类，计算各点与待测样本 $\boldsymbol{x}$ 的距离，将 $\boldsymbol{x}$ 归入距离最近的点所在的类，这种方法称为最近邻法（Nearest Neighbor，NN）。

最近邻法的判别函数如式(5-4)所示，仅将变量 c_j 修改为 ω_j 类的样本数目 N_j，决策规则如式(5-5)所示，此处不再赘述。

式(5-4)计算的是欧氏距离，也可以采用不同的距离度量，或者采用相似性度量（见 7.2 节），

把最小距离变更为最大相似性度量，确定最近邻，实现分类。

由于把每个样本点都作为一个子类，因此最近邻法的决策面由不同类样本点间连线的垂直平分面构成，判别函数是分段线性判别函数(Piecewise Linear Discriminant Function)。

2. k 近邻法

最近邻法根据距离待测样本最近的一个样本的类别进行归类，容易受到样本分布以及噪声的影响，导致决策错误，因此，引入投票机制，选择距离待测样本最近的 k 个样本，将待测样本归入多数近邻所在的类别，称为 k 近邻法(kNN)，最近邻法实际是 $k=1$ 的特例。

k 值需要根据样本情况进行选择，通常选择样本总数的一个很小的比例即可。两类情况下，一般选择 k 为奇数，避免两类得票相等。

k 近邻法(含最近邻法)简单易于决策，根据已知类别的样本直接对待测样本进行决策，不需要事先训练出一个判别函数，在训练样本趋于无穷时接近最优，但计算量大，耗时大，没有考虑决策的风险。

【例 5-1】 对样本集 $\omega_1:\left\{\begin{bmatrix}1\\0\end{bmatrix},\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}0\\-1\end{bmatrix}\right\}$，$\omega_2:\left\{\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}0\\2\end{bmatrix},\begin{bmatrix}0\\-2\end{bmatrix},\begin{bmatrix}-2\\0\end{bmatrix}\right\}$ 进行下列处理。

(1) 采用样本均值作为两类的代表点，按最小距离法分类；

(2) 对样本 $\boldsymbol{x}=[1\quad 2]^{\mathrm{T}}$，按最近邻法分类；

(3) 对 $\boldsymbol{x}=[1\quad 2]^{\mathrm{T}}$ 按 3 近邻法分类。

解： (1) $\boldsymbol{\mu}_1=[1/3\quad 0]^{\mathrm{T}}$、$\boldsymbol{\mu}_2=[-1/2\quad 0]^{\mathrm{T}}$，$\boldsymbol{\mu}_1$ 和 $\boldsymbol{\mu}_2$ 连线的垂直平分线为 $x_1=-1/12$。

(2) $g_1(\boldsymbol{x})=\min\{2,\sqrt{2},\sqrt{10}\}=\sqrt{2}$，$g_2(\boldsymbol{x})=\min\{\sqrt{5},1,\sqrt{17},\sqrt{13}\}=1$。

因为

$$g_1(\boldsymbol{x})>g_2(\boldsymbol{x})$$

所以

$$\boldsymbol{x}\in\omega_2$$

(3) 最近的三个距离为 1、$\sqrt{2}$、2，对应的 3 个近邻为 $[0\quad 2]^{\mathrm{T}}$、$[0\quad 1]^{\mathrm{T}}$、$[1\quad 0]^{\mathrm{T}}$，3 个近邻中有 2 个属于 ω_1 类，所以 $\boldsymbol{x}\in\omega_1$。

【例 5-2】 导入 iris 数据集，根据原理设计程序对样本 $[5.5\quad 2.3\quad 4\quad 1.3]^{\mathrm{T}}$ 进行最近邻法以及 5 近邻法判别。

设计思路：

计算待测样本与所有样本之间的距离，将距离排序，找最小和 5 个最小距离，对应的样本类别即为分类类别。

程序如下：

```
import numpy as np
from sklearn.datasets import load_iris
from numpy.linalg import norm
iris = load_iris()
x, k = np.array([5.5, 2.3, 4, 1.3]), 5
distance = norm(iris.data - x, axis=1) #计算待测样本和所有样本之间的欧氏距离
idx_min = np.argsort(distance)          #对距离升序排序,返回排序后各元素在原数组中的下标
print('最近邻归类:', iris.target_names[iris.target[idx_min[0]]])
```

```
idx_l = iris.target[idx_min[0:k]].tolist()    #将 k 个近邻编号表示为列表类型
label = max(set(idx_l), key = idx_l.count)    #查找列表中出现次数最多的元素
print('k 近邻归类:', iris.target_names[label])
```

运行程序，在输出窗口输出对待测样本归类的结果：

```
最近邻归类: versicolor
k 近邻归类: versicolor
```

3. 近邻法的快速算法

近邻法在样本数目较多时取得好的性能，但计算待测样本与大量训练样本之间的距离，计算量大，导致算法效率降低，因此需要快速算法。KD 树[①](K Dimensional Tree)是一种对 K 维空间中的点进行存储以便对其进行快速搜索的二叉树结构，利用 KD 树可以省去对大部分样本点的搜索，从而减少搜索的计算量。

1）KD 树的构建

取数据的某一维，将数据从小到大排序，以数据点在该维度的中值作为切分超平面，将该维一分为二，小于该中值的数据点挂在其左子树，大于该中值的数据点挂在其右子树。再按同样的方式切分另一维，直到所有维度处理完毕。

【例 5-3】 对二维平面点集合$\{[2\quad 3]^T,[5\quad 4]^T,[9\quad 6]^T,[4\quad 7]^T,[8\quad 1]^T,[7\quad 2]^T\}$构建 KD 树。

解：（1）选择要切分的维度。分别计算 x_1 维和 x_2 维的方差，选择方差大的 x_1 维做切分。

（2）切分 x_1 维。

按数据 x_1 维的数值排序：2、4、5、7、8、9。

确定中值：7(2、4、5、7、8、9 的中值为(5＋7)/2＝6，由于中值需在点集合之内，取后一个 7)。

确定节点：$[7\quad 2]^T$。

分割空间：以 $x_1=7$ 将 x_1 维一分为二，$[2\quad 3]^T$、$[4\quad 7]^T$、$[5\quad 4]^T$ 挂在$[7\quad 2]^T$ 节点的左子树；$[8\quad 1]^T$、$[9\quad 6]^T$ 挂在$[7\quad 2]^T$ 节点的右子树。

（3）切分 x_2 维——$[7\quad 2]^T$ 节点的左子树。

按数据 x_2 维的数值排序：3、4、7。

确定中值：4。

确定节点：$[5\quad 4]^T$。

分割空间：以 $x_2=4$ 将$[7\quad 2]^T$ 节点的左边空间一分为二，$[2\quad 3]^T$ 挂在$[5\quad 4]^T$ 节点的左子树；$[4\quad 7]^T$ 挂在$[5\quad 4]^T$ 节点的右子树。

（4）切分 x_2 维——$[7\quad 2]^T$ 节点的右子树。

按数据 x_2 维的数值排序：1、6。

确定中值：6。

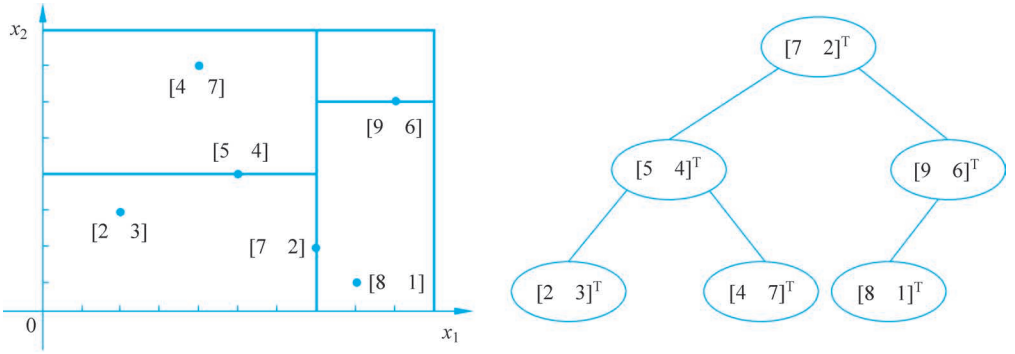
确定节点：$[9\quad 6]^T$。

---

① 本书统一用 n 表示样本维数，而 KD 树的 K 表示维数，此处与资料上的名称保持一致，注意与 kNN 中 k 的区别。

分割空间：以 $x_2=6$ 将[7　2]^T 节点的右边空间一分为二，[8　1]^T 挂在[9　6]^T 节点的左子树。

KD 树构建完成，对二维空间的切分如图 5-3(a)所示；构建的 KD 树如图 5-3(b)所示。



(a) 空间切分　　(b) KD树

图 5-3　KD 树的构建

2）最近邻搜索

给定点 $\boldsymbol{x}$，查询数据集中与其距离最近点的过程即为最近邻搜索，采用 KD 树实现。过程如下。

(1) 在 KD 树中找出包含目标点 $\boldsymbol{x}$ 的叶节点。

从根节点出发，向下访问 KD 树，如果 $\boldsymbol{x}$ 当前维的坐标小于切分点的坐标，则移动到左子节点，否则移动到右子节点，直到叶节点为止，以此叶节点为“当前最近点”。顺序存储经过的节点。

(2) 按存储顺序向上回溯，在每个节点处进行以下操作。

① 如果当前节点比当前最近点距离搜索点更近，则更新当前最近点为当前节点。

② 以搜索点为圆心，以当前最近点到搜索点的距离为半径，确定圆，判断该圆是否与父节点的超平面相交；如果相交，则进入父节点的另外一侧搜索最近邻节点；如果不相交，则继续向上回溯。

当搜索到根节点时，搜索完成，得到最近邻节点。

【例 5-4】　利用例 5-3 建好的 KD 树搜索[2　4.5]^T 的最近邻。

解：(1) 在 KD 树中找出包含目标点 $\boldsymbol{x}$ 的叶节点。从根节点[7　2]^T 出发，当前维为 x_1，2<7，移动到[7　2]^T 的左子节点[5　4]^T；当前维更新为 x_2，4.5>4，移动到[5　4]^T 的右子节点[4　7]^T；[4　7]^T 为叶节点，设为当前最近点，到待搜索点[2　4.5]^T 的距离为 $\sqrt{10.25}$。存储经过的节点，即搜索路径为[7　2]^T、[5　4]^T、[4　7]^T。

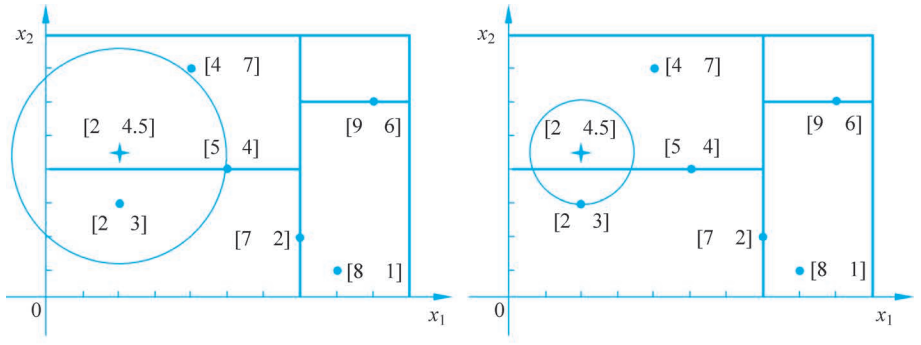
(2) 向上回溯到[5　4]^T，[5　4]^T 到待搜索点[2　4.5]^T 的距离为 $\sqrt{9.25}$，$\sqrt{9.25}<\sqrt{10.25}$，更新当前最近点为[5　4]^T。

(3) 以[2　4.5]^T 为圆心、以 $\sqrt{9.25}$ 为半径的圆和父节点[5　4]^T 的超平面 $x_2=4$ 相交，如图 5-4(a)所示，进入[5　4]^T 的另一侧查找，修改当前点为[2　3]^T；更新搜索路径为[7　2]^T、[2　3]^T。

(4) $[2 \quad 3]^T$ 到待搜索点 $[2 \quad 4.5]^T$ 的距离为 $\sqrt{2.25}$，$\sqrt{2.25} < \sqrt{9.25}$，更新当前最近点为 $[2 \quad 3]^T$。

(5) 向上回溯到 $[7 \quad 2]^T$，$[7 \quad 2]^T$ 到待搜索点 $[2 \quad 4.5]^T$ 的距离大于 $\sqrt{2.25}$，不修改当前最近点。

(6) 以 $[2 \quad 4.5]^T$ 为圆心、以 $\sqrt{2.25}$ 为半径的圆和父节点 $[7 \quad 2]^T$ 的切分平面 $x_1 = 7$ 不相交，如图 5-4(b)所示，搜索完成，最近邻为当前最近点 $[2 \quad 3]^T$。



(a) 第一次回溯　　(b) 第二次回溯

图 5-4　利用 KD 树搜索最近邻

4. 仿真实现

scikit-learn 库中 neighbors 模块提供了实现近邻法分类的相关模型及方法，对其进行简要介绍。

1) NearestNeighbors 模型

NearestNeighbors(* , n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=None)，其实现近邻搜索。NearestNeighbors 模型的主要参数和方法如表 5-1 所示。

表 5-1　NearestNeighbors 的主要参数和方法

名　　称		功　　能
参数	n_neighbors	近邻个数，即 k，整数，默认为 5
	radius	指定搜索近邻的范围，到中心点的距离，默认为 1
	algorithm	搜索近邻采用的算法，可以取'auto'(默认)、'ball_tree'、'kd_tree'、'brute'(穷举)，根据传递给 fit 方法的参数选择最合适的算法
	leaf_size	指定搜索树叶节点中最大数据点数量，BallTree 和 KDTree 的参数，默认为 30
	metric	距离度量方法，默认为 'minkowski'，在 $p=2$ 时即是欧氏距离
	p	距离度量为'minkowski'时的参数，各种距离的定义见第 7 章
	metric_params	字典，距离度量的额外参数

续表

名　　称		功　　能
方法	fit(X[, y])	根据样本集 X 拟合近邻法模型
	kneighbors(X = None, n_neighbors = None, return_distance=True)	搜索 X 的 n_neighbors 个近邻点，返回近邻点的索引；如果参数 return_distance=True，也返回邻点和 X 之间的距离
	kneighbors_graph(X=None, n_neighbors=None, mode='connectivity')	计算 X 中点的加权图。如果没有给出 X，所有训练点都是查询点。参数 mode 取'connectivity'(默认)和'distance'，返回图中节点权值分别为 0/1 或距离
	radius_neighbors(X = None, radius = None, return_distance = True, sort_results=False)	在给定半径 radius 范围内搜索 X 的近邻，返回近邻点的索引、近邻点和 X 之间的距离(return_distance=True)，如果 sort_results=True，返回值按升序排列。注意：return_distance=False 且 sort_results=True 将报错
	radius_neighbors_graph(X=None, radius=None, mode='connectivity', sort_results=False)	计算 X 样本集中点的加权图

2) KDTree 模型

KDTree(X, leaf_size=40, metric='minkowski', ** kwargs)，针对样本集 X 中的样本，构建 KD 树并用于搜索近邻点。常用的方法如表 5-2 所示。

表 5-2　KDTree 模型的主要方法

名　　称	功　　能
query(X, k=1, return_distance=True, dualtree=False, breadth_first=False)	查找 X 的 k 个近邻点。dualtree 和 breadth_first 是树搜索方面的参数
query_radius(X, r, return_distance = False, count_only=False, sort_results=False)	查找 X 的在半径 r 范围内的近邻点。如果 count_only=True，只返回近邻点数目；否则，返回近邻点的索引。return_distance 和 count_only 不能同时为 True
two_point_correlation(X, r, dualtree=False)	计算两点的相关系数
kernel_density(X, h, kernel='gaussian', atol=0, rtol=1E-8, breadth_first=True, return_log=False)	利用给定的核 kernel 和创建树时采用的距离度量方法，估计点 X 处的概率密度

3) KNeighborsClassifier 模型

KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)，实现 k 近邻分类，参数 weights 表示近邻点的加权值，可取'uniform'(默认)、'distance'，分别表示权值相等、按照距离的倒数加权，也可以不加权或者由用户指定。主要方法如表 5-3 所示。

表 5-3　KNeighborsClassifier 的主要方法

名　　称	功　　能
fit(X, y)	根据样本集 X 拟合近邻法模型
kneighbors(X = None, n_neighbors = None, return_distance=True)	搜索 X 的近邻，返回邻点索引(以及距离)

续表

名　　称	功　　能
kneighbors_graph(X = None, n_neighbors = None, mode='connectivity')	计算 X 数据集中点的加权图
predict(X)	利用模型对样本矩阵 X 中的样本进行决策
predict_proba(X)	预测 X 的归类概率
score(X, y, sample_weight=None)	返回对 X 预测的平均正确率

4）RadiusNeighborsClassifier 模型

RadiusNeighborsClassifier(radius=1.0, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', outlier_label=None, metric_params=None, n_jobs=None),在给定范围内进行 k 近邻分类。其主要方法有 fit、predict、predict_proba、radius_neighbors、radius_neighbors_graph、score 等。

【例 5-5】　导入 iris 数据集，随机选择 147 个训练样本，3 个测试样本，采用 NearestNeighbors 模型进行近邻搜索，使用 RadiusNeighborsClassifier 模型对测试样本进行分类。

程序如下：

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import RadiusNeighborsClassifier
from matplotlib import pyplot as plt
iris = load_iris()
X = iris.data[:, 2:4]
N, n = np.shape(X)
    #以下生成测试样本和训练样本
np.random.seed(1)
test_id = np.random.randint(1, N, 3)
testing, test_label = X[test_id, :], iris.target[test_id]
train_id = np.delete(np.array(range(N)), test_id)
training, train_label = X[train_id, :], iris.target[train_id]
    #以下是利用 NearestNeighbors 模型确定各测试样本的最近邻
nnb = NearestNeighbors(n_neighbors = 1).fit(training)
D1, Idx1 = nnb.kneighbors(testing, return_distance = True)
    #以下是训练 RadiusNeighborsClassifier 模型,并对测试样本进行分类
rnc = RadiusNeighborsClassifier(radius = 0.2)
rnc.fit(training, train_label)
test_result = rnc.predict(testing)
print("三个测试样本分别为", iris.target_names[test_result])
    #以下为绘制样本点、待测样本及各自的最近邻点
se = X[iris.target == 0, :]
ve = X[iris.target == 1, :]
vi = X[iris.target == 2, :]
plt.scatter(se[:, 0], se[:, 1], c = 'r', marker = '+')
plt.scatter(ve[:, 0], ve[:, 1], c = 'g', marker = '.')
plt.scatter(vi[:, 0], vi[:, 1], c = 'b', marker = 'x')
plt.scatter(testing[:, 0], testing[:, 1], c = 'k', s = 256, marker = '+')
plt.scatter(training[Idx1, 0], training[Idx1, 1], c = 'm', s = 92, marker = '*')
plt.rcParams['font.sans-serif'] = ['SimSun']
```

```
plt.legend(['setosa', 'versicolor', 'virginica', '待测样本', '最近邻点'])
plt.xlabel('花瓣长/cm')
plt.ylabel('花瓣宽/cm')
plt.show()
```

运行程序，绘制的样本点、近邻点如图 5-5 所示，并在命令窗口输出 3 个待测样本的归类结果。

```
三个测试样本分别为 ['setosa' 'virginica' 'versicolor']
```

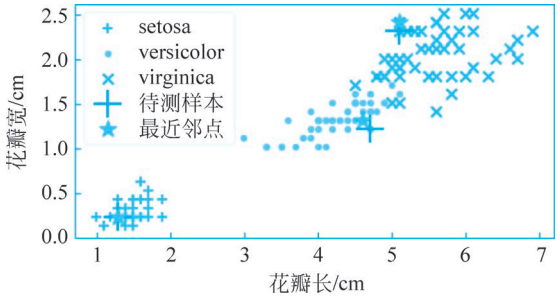


图 5-5　iris 数据集的近邻搜索

5.2　二次判别函数

二次判别(Quadratic Discriminant)也是一种比较常用的固定函数类型的分类方法，一般形式为

$$\begin{aligned} g(\boldsymbol{x}) &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{x} + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + w_0 \\ &= \sum_{i=1}^{n} w_{ii}x_i^2 + 2\sum_{i=1}^{n-1}\sum_{k=i+1}^{n} w_{ik}x_i x_k + \sum_{i=1}^{n} w_i x_i + w_0 \end{aligned} \tag{5-6}$$

其中，$\boldsymbol{W}$ 是 $n\times n$ 的实对称矩阵，$\boldsymbol{w}$ 是 n 维列向量。

由式(5-6)可以看出，二次判别函数中有太多的参数需要确定，如果采用类似于线性判别函数设计的方法，则计算复杂，在样本数不足的情况下，不能保证结果的可靠性和推广能力。因此，需要采用其他的设计方法。

在 2.7 节了解到，在一般正态分布情况下，贝叶斯决策判别函数为二次函数，如果可以用正态分布模拟样本分布，则直接定义 ω_j 类的二次判别函数为

$$g_j(\boldsymbol{x}) = C_j^2 - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^{\mathrm{T}}\boldsymbol{\Sigma}_j^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j) \tag{5-7}$$

其中，C_j^2 是一个调节项，受协方差矩阵和先验概率的影响，可以通过调节该参数以调整类错误率。

例如，在例 2-15 中，针对两类正态分布的样本，利用 QDA 模型设计了二次判别函数。

如果只有一类样本可以用正态分布模拟，另一类比较均匀地分布在第一类附近，则可以只对第一类求解二次判别函数，决策规则为

$$\text{若 } g(\boldsymbol{x}) \gtrless 0, \quad \text{则决策 } \boldsymbol{x} \in \begin{cases} \omega_1 \\ \omega_2 \end{cases} \tag{5-8}$$

【例 5-6】 导入 iris 数据集，选择其中的两类，采用正态分布模拟样本分布，设计二次判别函数，绘制二次决策面。

程序如下：

```
import numpy as np
from sklearn.datasets import load_iris
from numpy.linalg import inv
from matplotlib import pyplot as plt
iris = load_iris()
X = iris.data[:, 2:4]
N, n = np.shape(X)
    #以下为获取 versicolor 和 virginica 的样本、均值和协方差矩阵
id_ve, id_vi = iris.target == 1, iris.target == 2
ve, vi = X[id_ve, :], X[id_vi, :]
training = np.concatenate((ve, vi), axis = 0)
mu_ve, mu_vi = np.mean(ve, 0), np.mean(vi, 0)
sigma_ve, sigma_vi = np.cov(ve.T), np.cov(vi.T)
    #以下计算训练样本取值范围内平面点对应的二次判别函数值
min_x, max_x, step = np.min(training, 0), np.max(training, 0), 0.1
x1, x2 = np.mgrid[min_x[0]:max_x[0]:step, min_x[1]:max_x[1]:step]
x_test = np.concatenate((x1.reshape(-1, 1), x2.reshape(-1, 1)), 1)
x_ve, x_vi = x_test - mu_ve, x_test - mu_vi
gx = -np.sum(np.dot(x_ve, inv(sigma_ve)) * x_ve, 1) \
     + np.sum(np.dot(x_vi, inv(sigma_vi)) * x_vi, 1)
    #绘制样本点和分界线
plt.rcParams['font.sans-serif'] = ['Times New Roman']
plt.plot(ve[:, 0], ve[:, 1], 'b.', label = 'versicolor')
plt.plot(vi[:, 0], vi[:, 1], 'r+', label = 'virginica')
plt.contour(x1, x2, gx.reshape(x1.shape), linestyles = '--', levels = [0])
plt.legend(loc = 'upper left')
plt.xlabel('花瓣长', fontproperties = 'SimSun')
plt.ylabel('花瓣宽', fontproperties = 'SimSun')
plt.show()
```

程序运行结果如图 5-6 所示。程序中两类判别函数中调节项设为 0。

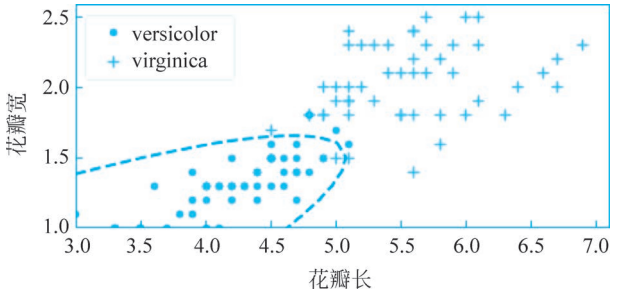


图 5-6 二次判别函数设计

本例也可以直接利用进行二次判别分析的 QuadraticDiscriminantAnalysis 模型，设计二次判别函数并分类，只需要对例 5-6 中的程序进行简单修改（用此段后所示语句替换程序中蓝色的几行代码），即可得到图 5-6 的结果。修改内容如下。

替换导入求逆矩阵函数 inv 的语句为导入 QuadraticDiscriminantAnalysis 模型：

```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

替换计算均值和协方差矩阵的语句为

```
y = np.delete(iris.target, iris.target == 0)
clf = QuadraticDiscriminantAnalysis().fit(training, y)
```

替换计算判别函数的语句为

```
gx = clf.decision_function(x_test)
```

5.3　决策树

人们在决策时，往往会针对多个方面依次判断，模式识别中也可以采用类似的多级决策方式，利用一定的训练样本，依次分类，直到获得最终可以接受的类，用树形表示决策的过程，并形成决策规则。这种从样本中“学习”出决策规则的方法称为决策树(Decision Tree)，是一种非线性分类器。

5.3.1　基本概念

一般而言，一棵决策树包含一个根节点、若干内部节点和若干叶节点。根节点包含所要进行分类的样本集，按照某种规则(常用某一种特征测试)，将样本集分为几个子集，对应几个子节点，称为内部节点；子节点再分，直到叶节点，对应决策结果。

叶节点有三种情形：

(1) 当前节点所含样本全属于同一类别，不需要再划分，该节点为叶节点。

(2) 当前特征为空，或所有样本当前特征取值相同，无法划分，该节点为叶节点，标记类别为该节点所含样本最多的类别。

(3) 当前节点所含样本集为空，无法划分，该节点为叶节点，标记类别为其父节点所含样本最多的类别。


第 9 集
微课视频


第 10 集
微课视频

以判断是否是高血压为例，说明决策树的生成以及决策。用 x_1 表示收缩压，用 x_2 表示舒张压，血压为二维样本 $\boldsymbol{x}=[x_1 \quad x_2]^{\mathrm{T}}$。图 5-7(a)所示为正常血压 ω_1 和高血压 ω_2 二维样本集$\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$；生成的决策树如图 5-7(b)所示。根节点①包括所有样本，考查第一个特征 x_1，根据是否满足 $x_1<140$ 将样本集一分为二，即节点①有两个子节点②和③；节点③中所有样本全部属于高血压 ω_2 类，为叶节点，标记类别为 ω_2 类；节点②非叶节点，考查第二个特征 x_2，根据是否满足 $x_2<90$ 将样本集一分为二，即节点②有两个子节点④和⑤；节点④中所有样本全部属于正常血压 ω_1 类，为叶节点，标记类别为 ω_1 类；节点⑤中所有样本全部属于高血压 ω_2 类，为叶节点，标记类别为 ω_2 类。从根节点①到叶节点③、④、⑤存在 3 条通路，通路上的判断条件即为决策规则，即：

若 $x_1\geqslant 140$，则 $\boldsymbol{x}\in\omega_2$。

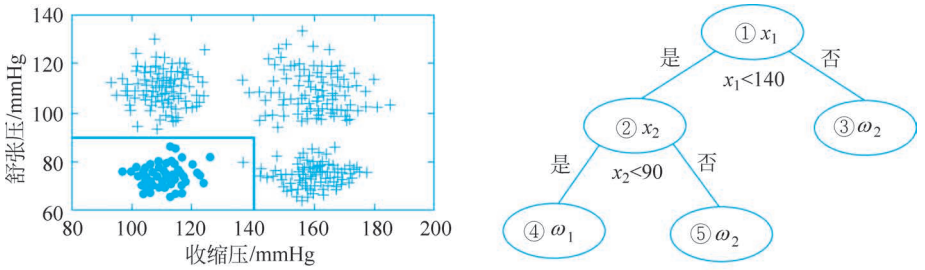
若 $x_1<140$ 且 $x_2<90$，则 $\boldsymbol{x}\in\omega_1$。

若 $x_1<140$ 且 $x_2\geqslant 90$，则 $\boldsymbol{x}\in\omega_2$。

图 5-7 所示的决策树生成中，节点采用的分枝准则为“特征 $x_i<\alpha$ 或者 $x_i\geqslant\alpha$”，这种决策树称为普通二进制分类树(Ordinary Binary Classification Tree，OBCT)，也可以根据不同的条件生成不同的决策树。

5.3.2　决策树的构建

从图 5-7 所示的决策树生成示例可以看出，决策树的构建就是选取特征和确定分枝准



(a) 血压(样本)　　(b) 生成的决策树

图 5-7　决策树示例

则的过程，每一个分枝应该产生比父节点样本集更有利于分类的样本子集，也就是新子集中的样本都更好地属于特定的类。因此，决策树的构建首先要有能衡量分类有利性的指标量，以便合理选择分枝采用的某个特征，以及确定分枝准则。

1. 信息增益

熵(Entropy)用来度量对某个事件进行观察后得到的信息量，设该事件有 c 种可能，每种可能对应的概率为 P_j，$j=1,2,\cdots,c$，则熵为

$$H=-(P_1\mathrm{lb}P_1+P_2\mathrm{lb}P_2+\cdots+P_c\mathrm{lb}P_c)=-\sum_{j=1}^{c}P_j\mathrm{lb}P_j \tag{5-9}$$

例如，一个四类分类问题，各类样本数相同，根节点包括所有样本，样本属于某一类的不确定性最大，即纯度最低，对应的熵为

$$H=-4\times 0.25\times \mathrm{lb}0.25=2$$

如果某个内部节点只包含其中两类的样本且数量相等，则样本属于某一类的不确定性相对于根节点要低，即纯度有所提高，对应的熵为

$$H=-2\times 0.5\times \mathrm{lb}0.5=1$$

熵的值较根节点有所下降。而对于叶节点，只包含同一类样本，纯度最高，此时的熵为

$$H=-1\times \mathrm{lb}1=0$$

熵最小，不确定性最小，纯度最高。

因此，对于某个节点上的样本，熵也称为熵不纯度，反映了该节点上的特征对样本分类的不纯度，取值越小，不纯度越低。

在决策树分枝的时候，希望分枝后的子集相对于原来的子集纯度有所提高，或说不纯度下降，因此，定义信息增益(Information Gain)为不纯度减少量。

如果某个节点上，把 N 个样本划分成 m 组，每组 N_m 个样本，则信息增益为

$$\Delta H(N)=H(N)-[P_1H(N_1)+P_2H(N_2)+\cdots+P_mH(N_m)] \tag{5-10}$$

其中，$P_m=N_m/N$。

2. ID3 方法

交互式二分(Interactive Dichotomizer-3，ID3)法是最早比较著名的决策树构建方法，对于每个非叶节点，计算比较采用不同特征进行分枝的信息增益，选择带来最大信息增益的特征作为分枝特征，以实现决策树的生成与构建，方法步骤如下。

(1) 计算当前节点包含的所有样本的熵不纯度。

(2) 比较采用不同特征进行分枝将会得到的信息增益，选取具有最大信息增益的特征

赋予当前节点。特征的取值个数决定了该节点下的分枝数目。

（3）如果后继节点只包含一类样本，则停止该枝的生长，该节点成为叶节点。

（4）如果后继节点仍包含不同类样本，则再次进行以上步骤，直到每一枝的节点都到达叶节点。

【例 5-7】 以表 5-4 中的 1～14 号样本为训练样本，15～19 号样本为测试样本，采用 ID3 方法设计决策树，并对测试样本进行决策。

表 5-4　血压数据表

序号	年龄/岁	体重	饮食偏好	父辈血压	高血压
1	34	超重	油腻	高	是
2	28	正常	均衡	高	是
3	42	偏重	清淡	高	是
4	45	超重	均衡	正常	是
5	50	偏重	油腻	正常	是
6	61	偏重	油腻	正常	是
7	65	正常	清淡	高	是
8	36	超重	均衡	正常	否
9	31	正常	清淡	高	否
10	27	偏重	油腻	高	否
11	44	偏重	清淡	正常	否
12	43	偏重	均衡	正常	否
13	61	正常	清淡	正常	否
14	66	正常	清淡	正常	否
15	25	正常	均衡	正常	否
16	35	偏重	均衡	正常	是
17	48	超重	油腻	高	是
18	40	正常	清淡	正常	否
19	63	偏重	均衡	高	是
20	62	正常	均衡	正常	否

解：（1）连续特征数据调整。表 5-4 中的年龄取值太多，如果直接按年龄分枝，将导致太多的子节点。因此把年龄分为三个级别，40 岁以下为青年；40～59 岁为中年；60 岁及以上为老年，如表 5-5 所示。

表 5-5　血压数据表

序号	年龄	体重	饮食偏好	父辈血压	高血压
1	青年	超重	油腻	高	是
2	青年	正常	均衡	高	是
3	中年	偏重	清淡	高	是
4	中年	超重	均衡	正常	是
5	中年	偏重	油腻	正常	是
6	老年	偏重	油腻	正常	是
7	老年	正常	清淡	高	是
8	青年	超重	均衡	正常	否
9	青年	正常	清淡	高	否

续表

序号	年龄	体重	饮食偏好	父辈血压	高血压
10	青年	偏重	油腻	高	否
11	中年	偏重	清淡	正常	否
12	中年	偏重	均衡	正常	否
13	老年	正常	清淡	正常	否
14	老年	正常	清淡	正常	否
15	青年	正常	均衡	正常	否
16	青年	偏重	均衡	正常	是
17	中年	超重	油腻	高	是
18	中年	正常	清淡	正常	否
19	老年	偏重	均衡	高	是
20	老年	正常	均衡	正常	否

(2) 生成决策树。

首先,计算不考虑任何特征时熵不纯度。14 个人中高血压 7 人,正常血压 7 人,根节点熵不纯度为

$$H(14,7)=-[0.5\times\log_2(0.5)+0.5\times\log_2(0.5)]=1$$

其次,考虑根节点分枝。考查采用不同特征划分样本后的信息增益。

采用“年龄”特征划分样本集:青年组 5 人,其中高血压 2 人;中年组 5 人,其中高血压 3 人;老年组 4 人,其中高血压 2 人。熵不纯度为

$$H_{\text{age}}=\frac{5}{14}H(5,2)+\frac{5}{14}H(5,3)+\frac{4}{14}H(4,2)=0.9793$$

信息增益为

$$\Delta H_{\text{age}}(14)=H(14,7)-H_{\text{age}}=0.0207$$

采用“体重”特征划分样本集:正常组 5 人,其中高血压 2 人;偏重组 6 人,其中高血压 3 人;超重组 3 人,其中高血压 2 人。熵不纯度为

$$H_{\text{weight}}=\frac{5}{14}H(5,2)+\frac{6}{14}H(6,3)+\frac{3}{14}H(3,2)=0.9721$$

信息增益为

$$\Delta H_{\text{weight}}(14)=H(14,7)-H_{\text{weight}}=0.0279$$

采用“饮食偏好”特征划分样本集:油腻组 4 人,其中高血压 3 人;清淡组 6 人,其中高血压 2 人;均衡组 4 人,其中高血压 2 人。熵不纯度为

$$H_{\text{diet}}=\frac{4}{14}H(4,3)+\frac{6}{14}H(6,2)+\frac{2}{14}H(4,2)=0.9111$$

信息增益为

$$\Delta H_{\text{diet}}(14)=H(14,7)-H_{\text{diet}}=0.0889$$

采用“父辈血压”特征划分样本集:高血压组 6 人,其中高血压 4 人;正常血压组 8 人,其中高血压 3 人。熵不纯度为

$$H_{\text{parent}}=\frac{6}{14}H(6,4)+\frac{8}{14}H(8,3)=0.9389$$

信息增益为

$$\Delta H_{\text{parent}}(14)=H(14,7)-H_{\text{parent}}=0.0611$$

通过对比，采用“饮食偏好”特征划分样本集的信息增益最大，用该特征将根节点一分为三，如图 5-8(a)所示。

再次，考虑下一级节点分枝。

如图 5-8(a)中，节点②有 4 个样本，为序号 1、5、6、10，其中高血压 3 人，熵不纯度为

$$H(4,3)=-\left(\frac{3}{4}\times\log_2\frac{3}{4}+\frac{1}{4}\times\log_2\frac{1}{4}\right)=0.8113$$

依次采用“年龄”“体重”“父辈血压”特征划分样本集，各自的熵不纯度和信息增益为

$$H_{\text{age}}=\frac{2}{4}H(2,1)+\frac{1}{4}H(1,1)+\frac{1}{4}H(1,1)=0.5000$$

$$\Delta H_{\text{age}}(4)=H(4,3)-H_{\text{age}}=0.311$$

$$H_{\text{weight}}=\frac{3}{4}H(3,2)+\frac{1}{4}H(1,1)=0.6887$$

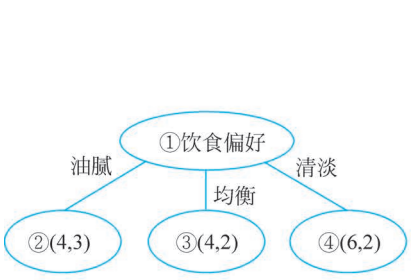
$$\Delta H_{\text{weight}}(4)=H(4,3)-H_{\text{weight}}=0.1226$$

$$H_{\text{parent}}=\frac{2}{4}H(2,1)+\frac{2}{4}H(2,2)=0.5000$$

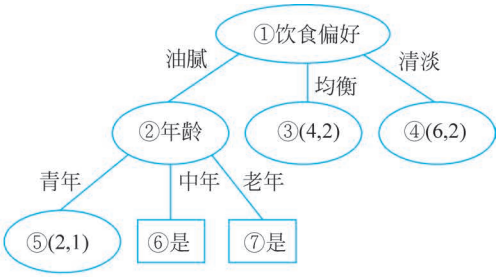
$$\Delta H_{\text{parent}}(4)=H(4,3)-H_{\text{parent}}=0.311$$

采用“年龄”和“父辈血压”划分样本集信息增益一样大，选择“年龄”特征将节点②一分为三，如图 5-8(b)所示。其中，节点⑥和⑦中各有一个高血压类样本，为叶节点，标记为高血压类。

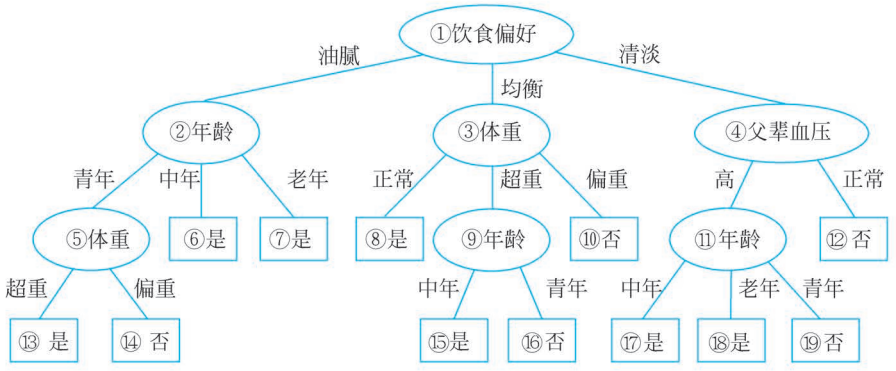
然后，同样的方法依次处理直到叶节点，生成的决策树共有 4 级，如图 5-8(c)所示，方框表示叶节点。



(a) 决策树的第一级



(b) 决策树的继续生长



(c) 完整的决策树

图 5-8　用 ID3 方法对表 5-4 中数据判读是否为高血压的决策树

(3) 利用决策树对测试样本进行决策。

首先,确定决策规则。生成的决策树有 12 个叶节点,从根节点到叶节点的 12 条通路,对应 12 条决策规则,例如:

若(饮食偏好==油腻)且(年龄==青年)且(体重==超重),则(血压=高血压)。

若(饮食偏好==油腻)且(年龄==青年)且(体重==偏重),则(血压=正常血压)。

……

若(饮食偏好==清淡)且(父辈血压==正常),则(血压=正常血压)。

其次,将待测样本代入决策规则进行判断,例如:

15 号样本,饮食偏好==均衡,体重==正常,节点⑧,判断为高血压,判断错误。

16 号样本,饮食偏好==均衡,体重==偏重,节点⑩,判断为正常血压,判断错误。

17 号样本,饮食偏好==油腻,年龄==中年,节点⑥,判断为高血压,判断正确。

18 号样本,饮食偏好==清淡,父辈血压==正常,节点⑫,判断为正常血压,判断正确。

19 号样本,饮食偏好==均衡,体重==偏重,节点⑩,判断为正常血压,判断错误。

20 号样本,饮食偏好==均衡,体重==正常,节点⑧,判断为高血压,判断错误。

例题中样本过少,叶节点往往只有一个样本,可能会抓住由于偶然性带来的假象,分类器判断准确率较差。可以采用剪枝的方式提高泛化能力,该内容将在下节学习。

ID3 方法虽然名为交互式二分法,但实际根据特征的取值可以划分为多个子节点。

3. C4.5 算法

C4.5 算法采用信息增益率(Gain Ratio)代替信息增益来选择最优划分特征,增益率的定义为

$$\Delta H_{\mathrm{R}}(N)=\frac{\Delta H(N)}{H(N)} \tag{5-11}$$

C4.5 算法增加了处理连续特征的功能,采用了二分法将连续特征离散化为二值特征。设特征 x_i,$i=1,2,\cdots,n$,在训练样本上共包含了 m 个值,将这些值按照从小到大的顺序排列,记为$\{x_i^1,x_i^2,\cdots,x_i^m\}$;设定一个阈值 T,将样本分为两个子集 x_i^-、x_i^+;不同的阈值将导致不同的分割子集,对每种情况计算信息增益率,选择信息增益率最大的划分方案实现特征二值化,再进行决策树的生成。

阈值 T 可以取相邻两个取值的中值,即

$$T=\frac{x_i^j+x_i^{j+1}}{2},\quad j=1,2,\cdots,m-1 \tag{5-12}$$

【例 5-8】 以表 5-4 中的 1～14 号样本为训练样本,采用 C4.5 算法生成决策树。

解: (1) 根节点熵不纯度为 $H(14,7)=1$。

(2) 根节点分枝。考查采用不同特征划分样本后的信息增益率。

采用“体重”“饮食偏好”“父辈血压”特征划分样本集,信息增益率分别为 0.0279、0.0889、0.0611。

对于“年龄”特征,根节点包含的样本在该特征上的取值排序为{27,28,31,34,36,42,43,44,45,50,61,61,65,66};阈值 T 取值为{27.5,29.5,32.5,35,39,42.5,43.5,44.5,47.5,55.5,63,65.5},当 $T=27.5$ 时,“年龄$<T$”组 1 人,其中高血压 0 人;“年龄$>T$”组 13 人,其中高血压 7 人。熵不纯度为

$$H_{\text{age}}^{1}=\frac{1}{14}H(1,0)+\frac{13}{14}H(13,7)=0.9246$$

信息增益率为

$$\Delta H_{\text{R,age}}(14)=\frac{H(14,7)-H_{\text{age}}^{1}}{H(14,7)}=0.0754$$

依次修改阈值 T，计算“年龄”特征最大信息增益率为 0.0754，对应阈值 $T=27.5$。

采用“饮食偏好”特征划分样本集信息增益率最大，因此，用该特征将根节点一分为三，如图 5-9(a)所示。

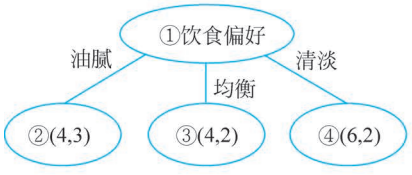
(3) 下一级节点分枝。

如图 5-9(a)所示，节点②，有 4 个样本，高血压 3 人，熵不纯度为 $H(4,3)=0.8113$。采用“体重”“父辈血压”特征划分样本集，信息增益率分别为 0.1511 和 0.3837。

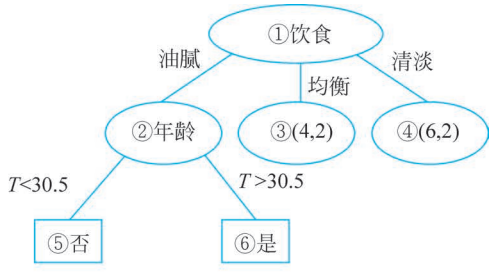
对于“年龄”特征，节点②包含的样本在该特征上的取值排序为{27,34,50,61}，阈值 T 取值为{30.5,42,55.5}；当 $T=30.5$ 时，“年龄$<T$”组 1 人，其中高血压 0 人；“年龄$>T$”组 3 人，其中高血压 3 人，熵不纯度为 0，信息增益率为 1；当 $T=42$，$T=55.5$ 时，信息增益率分别为 0.3837 和 0.1511。采用“年龄 $T=30.5$”划分样本集信息增益率最大，选择“年龄”特征将节点②一分为二，子节点⑤中只有一个正常血压样本，为叶节点，标记为正常血压类；子节点⑥中有 3 个高血压类样本，为叶节点，标记为高血压类，如图 5-9(b)所示。

(4) 同样的方法依次处理直到叶节点，生成的决策树如图 5-9(c)所示。

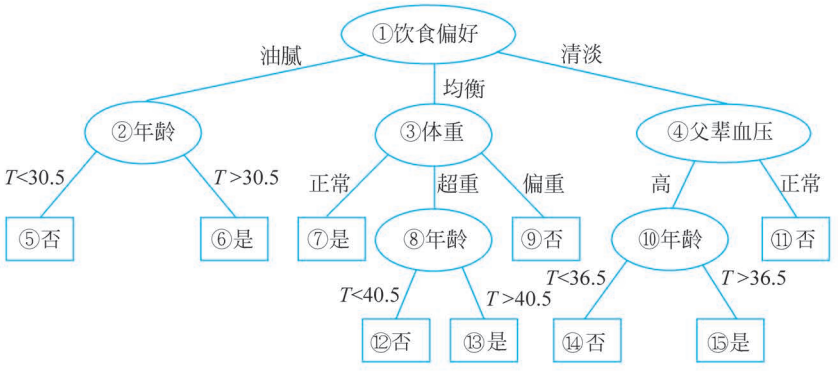
C4.5 算法不需要事先将连续特征离散化为少量级别，但在生成决策树的过程中，取不同阈值将连续特征二值化，划分方案增多。另外，C4.5 算法还可以处理部分特征缺失的样本。



(a) 决策树的第一级



(b) 决策树的继续生长



(c) 完整的决策树

图 5-9　用 C4.5 算法对表 5-4 中数据判读是否为高血压的决策树

4. CART 算法

分类和回归树(Classification And Regression Tree,CART)算法也是一个著名的决策树算法,核心思想和 ID3 方法相同。主要不同之处在于:CART 算法在每一个节点上都采用二分法,最终构成一棵二叉树;CART 算法既可用于分类,也可以用于构造回归树对连续变量进行回归。下面主要介绍 CART 算法构建用于分类的决策树。

CART 算法使用基尼系数(Gini Index)来选择节点分枝特征。设样本集$\mathscr{X}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$有 c 个类别,其中第 j 类的概率为 P_j,基尼系数定义为

$$\mathrm{Gini}(\mathscr{X})=\sum_{j=1}^{c}P_j(1-P_j)=1-\sum_{j=1}^{c}P_j^2 \tag{5-13}$$

从定义可以看出,基尼系数反映了随机抽取的两个样本,其类别标记不一样的概率。因此,基尼系数越小,两个样本来自不同类的概率越小,样本集不纯度越低。

如果各类的样本数为 N_j,则样本集 $\mathscr{X}$ 的基尼系数为

$$\mathrm{Gini}(\mathscr{X})=1-\sum_{j=1}^{c}\left(\frac{N_j}{N}\right)^2 \tag{5-14}$$

决策树生成中根据某个特征 x_i,$i=1,2,\cdots,n$ 将样本集划分为两部分$\mathscr{X}_1$ 和 $\mathscr{X}_2$,各自的样本数为 N_1 和 N_2,按照 x_i 划分的样本集的基尼系数为

$$\mathrm{Gini}(\mathscr{X},x_i)=\frac{N_1}{N}\mathrm{Gini}(\mathscr{X}_1)+\frac{N_2}{N}\mathrm{Gini}(\mathscr{X}_2) \tag{5-15}$$

使用基尼系数代替信息熵存在一定的误差,是熵模型的一个近似替代。但是,其避免了大量的对数运算,因此减少了计算量,简化了模型的运用。

【例 5-9】 以表 5-4 中的 1～14 号样本为训练样本,采用 CART 算法生成分类决策树。

解:(1) 根节点分枝。

考查采用不同特征划分样本后的基尼系数。

采用“体重”特征划分样本集:正常组 5 人,其中高血压 2 人,非正常组 9 人,其中高血压 5 人;偏重组 6 人,其中高血压 3 人,非偏重组 8 人,其中高血压 4 人;超重组 3 人,其中高血压 2 人,非超重组 11 人,其中高血压 5 人。三种划分情况的基尼系数分布为

$$\mathrm{Gini}^1_{\mathrm{weight}}=\frac{5}{14}\left[1-\left(\frac{2}{5}\right)^2-\left(\frac{3}{5}\right)^2\right]+\frac{9}{14}\left[1-\left(\frac{5}{9}\right)^2-\left(\frac{4}{9}\right)^2\right]=0.4889$$

$$\mathrm{Gini}^2_{\mathrm{weight}}=\frac{6}{14}\left[1-\left(\frac{3}{6}\right)^2-\left(\frac{3}{6}\right)^2\right]+\frac{8}{14}\left[1-\left(\frac{4}{8}\right)^2-\left(\frac{4}{8}\right)^2\right]=0.5$$

$$\mathrm{Gini}^3_{\mathrm{weight}}=\frac{3}{14}\left[1-\left(\frac{2}{3}\right)^2-\left(\frac{1}{3}\right)^2\right]+\frac{11}{14}\left[1-\left(\frac{5}{11}\right)^2-\left(\frac{6}{11}\right)^2\right]=0.4848$$

最小的基尼系数为 0.4848。

同理,采用“饮食偏好”特征划分样本集,按照油腻和非油腻划分,基尼系数最小,为 0.4500。

采用“父辈血压”特征划分样本集,基尼系数为 0.4583。

对于“年龄”特征,根节点包含的样本在该特征上的取值排序为{27,28,31,34,36,42,43,44,45,50,61,61,65,66};阈值 T 取值为{27.5,29.5,32.5,35,39,42.5,43.5,44.5,47.5,55.5,63,65.5},当 $T=27.5$ 时,“年龄$<T$”组 1 人,其中高血压 0 人;“年龄$>T$”组

13 人，其中高血压 7 人。基尼系数为

$$\mathrm{Gini}_{\mathrm{age}}^{1}=\frac{1}{14}\left[1-1^{2}\right]+\frac{13}{14}\left[1-\left(\frac{7}{13}\right)^{2}-\left(\frac{6}{13}\right)^{2}\right]=0.4615$$

阈值 T 取其他值情况下，基尼系数依次为 0.5、0.4848、0.5、0.4889、0.5、0.4898、0.4583、0.4889、0.5、0.5、0.4615，最小基尼系数为 0.4583，对应划分阈值 $T=44.5$。

采用“饮食偏好”特征划分样本集基尼系数最小，用该特征将根节点一分为二。

（2）下一级节点分枝。

如图 5-10 所示，节点②，有 4 个样本，为序号 1、5、6、10，其中高血压 3 人，采用“体重”特征划分样本集：偏重组 3 人，其中高血压 2 人；超重组 1 人，其中高血压 1 人。基尼系数为 0.3333。

采用“父辈血压”特征划分样本集，基尼系数为 0.25。

对于“年龄”特征，根节点包含的样本在该特征上的取值排序为{27,34,50,61}；阈值 T 取值为{30.5,42,55.5}，当 $T=30.5$ 时，基尼系数为最小值 0。

采用“年龄”特征划分样本集基尼系数最小，用年龄≤30.5 将根节点一分为二。

（3）同样的方法依次处理直到叶节点，生成的决策树如图 5-10 所示，方框表示叶节点。节点⑧采用“饮食偏好”和“年龄”划分基尼系数相等，此处采用“饮食偏好”划分。最终的决策树有 8 个叶节点，对应 8 条决策规则。

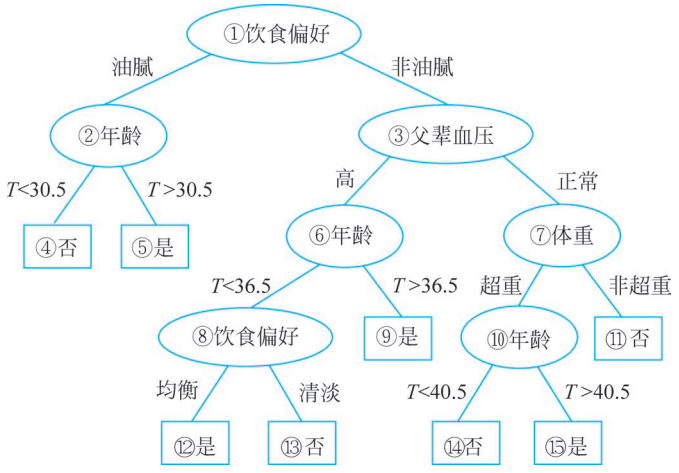


图 5-10　用 CART 算法对表 5-4 中数据判读是否为高血压的决策树

5.3.3　过学习与决策树的剪枝

从 5.3.2 节的例子可以看出，决策树构建的同时，建立了决策规则，可以利用决策规则对未知类别的样本进行分类。依据有限样本全部正确划分为准则建立决策规则，需要考虑为未来数据分析时的成功率，即分类器的泛化性能。如果一个算法在训练样本上表现良好，但在测试样本或未来的新样本上的表现与在训练样本上差别很大，称该算法遇到了过学习或过适应。

在有限的样本下，如果决策树生长得很大（分枝多、级别多），则可能会抓住由于偶然性或噪声带来的假象，导致过学习。例如例 5-7 利用 ID3 算法生成的决策树，仅 14 个训练样本，却生成了 12 条决策规则，每条决策规则对应的样本数很少，很有可能是偶然现象，导致

了对于5个待测样本进行判断时错误率较高。

因此，要控制决策树规模，即剪枝(Pruning)，防止出现过学习。决策树剪枝的方法有两种基本策略，即先剪枝和后剪枝。

1. 先剪枝

先剪枝指的是在决策树生长过程中判断节点是继续分枝还是直接作为叶节点，某些节点不再分枝，将减小决策树的规模。

先剪枝的关键在于确定节点是否继续分枝的判断条件，有多种方法，比如设定信息增益阈值，若小于该值，则停止生长，但该阈值不易设定；统计已有节点的信息增益分布，若继续生长得到的信息增益与该分布相比不显著，则停止生长等。剪枝主要是为了提高决策树泛化性能，可以通过实时检测当前决策树对于测试样本集的决策性能来判断节点是否继续分枝：性能有所提高，则继续分枝；性能没有提高，则直接作为叶节点。

因此，生成先剪枝决策树时，对于每个节点，进行如下操作：

(1) 将节点视为叶节点，标记为该节点训练样本中数目最多的类别。

(2) 计算当前决策树对于测试样本集的分类精度。

(3) 寻找进行分枝的最优特征并进行分枝。

(4) 对分枝后的子节点进行标记。

(5) 验证分枝后的决策树分类精度是否有提高：若有，则进行分枝；若没有，则剪枝(即禁止分枝)，当前节点作为叶节点。

【例5-10】 以表5-5中的1～14号样本为训练样本，15～20号样本为测试样本，采用ID3算法生成先剪枝决策树。

解：(1) 对根节点是否分枝进行判断。

首先，标记。当前节点14个样本，7个高血压，7个正常血压，标记为“高血压”。

其次，计算分类精度。当前为根节点，认为所有的样本均为“高血压”，对于测试样本，6个样本中3个判断正确，分类精度为3/6。

再次，寻找进行分枝的最优特征并分枝。由例5-7可知，采用“饮食偏好”特征划分样本集信息增益最大，用该特征将根节点划分为三个子节点②、③、④，各自的样本数及高血压样本数分别为油腻(4,3)、均衡(4,2)、清淡(6,2)。

然后，将节点②、③、④分别标记为“高血压”“高血压”和“正常血压”。

最后，验证当前决策树的分类精度。测试样本集中的16、17、18、19号样本分类正确，分类精度为4/6，有提高，进行分枝。

(2) 对节点②是否分枝进行判断。

首先，寻找进行分枝的最优特征并分枝。采用“年龄”特征划分样本集信息增益最大，用该特征将根节点划分为三个子节点，各自的样本数及高血压样本数分别为青年(2,1)、中年(1,1)、老年(1,1)。

其次，标记。三个子节点均被标记为“高血压”。

最后，验证当前决策树的分类精度。测试样本集中的16、17、18、19号样本被正确分类，分类精度为4/6，没有提高，禁止该分枝，节点②为叶节点，标记为“高血压”。

(3) 对节点③是否分枝进行判断。

首先，寻找进行分枝的最优特征并分枝。采用“体重”特征划分样本集信息增益最大，用

该特征将根节点划分为三个子节点，各自的样本数及高血压样本数分别为正常(1,1)、超重(2,1)、偏重(1,0)。

其次，标记。子节点分别被标记为“高血压”“高血压”和“正常血压”。

最后，验证当前决策树的分类精度。测试样本集中的17、18号样本被正确分类，分类精度为2/6，降低了，禁止该分枝，节点③为叶节点，标记为“高血压”。

(4) 对节点④是否分枝进行判断。

首先，寻找进行分枝的最优特征并分枝。采用“父辈血压”特征划分样本集信息增益最大，用该特征将根节点划分为两个子节点，各自的样本数及高血压样本数分别为高血压(3,2)和正常血压(3,0)。

其次，标记。子节点分别被标记为“高血压”和“正常血压”。

最后，验证当前决策树的分类精度。测试样本集中的16、17、18、19号样本被正确分类，分类精度为4/6，没有提高，禁止分枝，节点④为叶节点，标记为“正常血压”。

没有可分枝的节点，决策树生成到此为止，最终形成的先剪枝决策树如图5-11所示。

图5-11　基于ID3算法利用表5-5中数据生成先剪枝决策树

由例5-10可知，构建决策树时进行先剪枝，很多分枝未展开，降低了过学习的风险，减少了训练时间、测试时间开销；有些分枝虽不能提升泛化性能，但在其基础上的后续划分有可能提高性能，因此，先剪枝有可能带来欠学习的风险。

2. 后剪枝

后剪枝指的是在决策树充分生长后对其进行修剪，核心思想是合并分枝，从叶节点出发，如果消除具有相同父节点的节点后能够提高决策树的泛化性能则消除，并以其父节点作为新的叶节点；不断地从叶节点往上回溯，直到合并操作不再合适为止。

因此，生成后剪枝决策树时，进行如下操作：

(1) 生成一棵完整的决策树，并计算决策树对于测试样本集的分类精度。

(2) 从最高级开始合并节点，对其父节点进行标记。

(3) 计算合并分枝后的决策树的分类精度，并判断是否有提高：若有，则剪枝(即进行合并)；若没有，则不剪枝(保留分枝)。

(4) 向上回溯，重复(2)、(3)的合并、判断操作，直到不再需要合并为止。

下面介绍一种经典的后剪枝算法，即代价复杂度剪枝(Cost-Complexity Pruning，CCP)。CCP算法将决策树分为一棵棵子树，给每棵子树定义了代价和复杂度，计算将每棵子树剪枝后的代价增量与复杂度减量的比值，选择最小比值(代价增加小，复杂度降低大，即最小代价增量率)的子树进行剪枝；再对其子树重复以上过程，逐步向上，直到根节点，构建了剪枝决策树序列以及最小代价增量率序列；再从构建的剪枝决策树序列中选择决策树，可以根据交叉验证方法选择最佳决策树，或者根据要求的代价增量率选择对应的剪枝决策树，或者根据剪枝决策树序列中的节点号、树的不纯度等进行剪枝。

决策树中每个内部节点 t 对应一棵子树 T_t(该节点看作子树的根节点)，子树 T_t 的代价可以定义为

$$R(T_t)=\sum_{i\in L(T_t)}\frac{e(i)}{N(i)}\times\frac{N(i)}{N} \tag{5-16}$$

其中，$L(T_t)$是子树 T_t 的叶节点集合，$e(i)$、$N(i)$是子树中各个叶节点分类错误样本数和叶节点样本数，N 是总样本数。如果要剪掉这棵子树，也就是内部节点 t 变成叶节点，则计算这一个叶节点的误差代价 $R(t)$，剪枝前后代价的增量为 $R(t)-R(T_t)$。

除了叶节点分类错误率外，代价也有其他的定义方式，比如使用叶节点的不纯度度量（如 Gini 值、熵值等）。

子树 T_t 的复杂度定义为该子树叶节点数目，表示为 $|T_t|$。如果剪掉子树后，该子树变为一个叶节点，复杂度的减量为 $|T_t|-1$。所以，剪枝前后代价增量与复杂度减量的比值为

$$s_t=\frac{R(t)-R(T_t)}{|T_t|-1} \tag{5-17}$$

根据以上描述整理 CCP 算法过程如下。

（1）生成完整的决策树，将其作为剪枝决策树 PT^i，$i=0$，当前最小代价增量率 $\alpha^0=0$。

（2）$i=i+1$，计算各子树剪枝时的代价增量率 s_t，确定并记录最小代价增量率 $\alpha^i=\min s_t$。

（3）将 α^i 对应的子树剪枝得 PT^i，加入剪枝决策树序列。

（4）判断是否已到根节点，如果是，则执行第(5)步；否则，返回第(2)步。

（5）从剪枝决策树序列中找出最终剪枝决策树。

【例 5-11】 以表 5-4 中的 1～14 号样本为训练样本，15～20 号样本为测试样本，采用 CCP 算法生成后剪枝决策树。

解：（1）利用 CART 算法生成完整的决策树如图 5-12(a)所示，表示为 PT^0。椭圆节点为内部节点，里面标记了节点编号、“高血压”和“正常血压”样本数。如根节点编号为①，样本集包含 7 个高血压和 7 个正常血压；长方形节点为叶节点，里面标记了类别标签、两类样本数，如最左侧的叶节点，标记为“否 0∶1”，即正常血压，有一个正常血压样本。当前树可以分为 7 棵子树，用 7 个带编号的节点表示。

（2）$i=1$，计算 PT^0 中各棵子树的代价增量、复杂度减量及二者比值。如节点①，对应子树 T_1，所有叶节点都没有错分样本，$R(T_1)=0$，复杂度 $|T_1|=8$；子树 T_1 剪枝后，节点①为叶节点，错分比例为 $R(1)=7/14$；剪枝前后代价增量与复杂度减量的比值为 $s_1=[R(1)-R(T_1)]/(|T_1|-1)=1/14$。7 棵子树的计算如表 5-6 所示。

表 5-6 CCP 算法运算过程 1

节点序号 t	$R(t)$	$R(T_t)$	复杂度 $\lvert T_t\rvert$	$s_t=[R(t)-R(T_t)]/(\lvert T_t\rvert-1)$
①	7/14	0	8	1/14
②	1/14	0	2	1/14
③	4/14	0	6	0.8/14
④	1/14	0	3	0.5/14
⑤	1/14	0	3	0.5/14
⑥	1/14	0	2	1/14
⑦	1/14	0	2	1/14

s_4、$s_5=\min s_t=0.5/14$，选择 T_4 剪枝，并将 $PT^1=PT^0-T_4$ 放入子树序列，如图 5-12(b)所示，记录 $\alpha^1=s_4=0.5/14$。

(3) $i=2$，计算 PT^1 中各棵子树的代价增量、复杂度减量及二者比值，如表 5-7 所示。

表 5-7　CCP 算法运算过程 2

节点序号 t	$R(t)$	$R(T_t)$	复杂度 $\lvert T_t \rvert$	$s_t=[R(t)-R(T_t)]/(\lvert T_t \rvert-1)$
①	7/14	1/14	6	1.2/14
②	1/14	0	2	1/14
③	4/14	1/14	4	1/14
⑤	1/14	0	3	0.5/14
⑦	1/14	0	2	1/14

$s_5=\min s_t=0.5/14$，选择 T_5 剪枝，并将 $PT^2=PT^1-T_5$ 放入子树序列，如图 5-12(c)所示，记录 $\alpha^2=s_5=0.5/14$。

(4) $i=3$，计算 PT^2 中各棵子树的代价增量、复杂度减量及二者比值，如表 5-8 所示。

表 5-8　CCP 算法运算过程 3

节点序号 t	$R(t)$	$R(T_t)$	复杂度 $\lvert T_t \rvert$	$s_t=[R(t)-R(T_t)]/(\lvert T_t \rvert-1)$
①	7/14	2/14	4	1.7/14
②	1/14	0	2	1/14
③	4/14	2/14	2	2/14

$s_2=\min s_t=1/14$，选择 T_2 剪枝，并将 $PT^3=PT^2-T_2$ 放入子树序列，如图 5-12(d)所示，记录 $\alpha^3=s_2=1/14$。

(5) $i=4$，计算 PT^3 中各棵子树的代价增量、复杂度减量及二者比值，如表 5-9 所示。

表 5-9　CCP 算法运算过程 4

节点序号 t	$R(t)$	$R(T_t)$	复杂度 $\lvert T_t \rvert$	$s_t=[R(t)-R(T_t)]/(\lvert T_t \rvert-1)$
①	7/14	3/14	3	2/14
③	4/14	2/14	2	2/14

s_1、$s_3=\min s_t=2/14$，选择 T_1 剪枝，并将 $PT^4=PT^3-T_1$ 放入子树序列，如图 5-12(e)所示，记录 $\alpha^4=s_1=2/14$。

(6) 已从根节点进行了剪枝，循环结束。构建了剪枝决策树序列{PT^0、PT^1、PT^2、PT^3、PT^4}、最小代价增量率序列{α^0、α^1、α^2、α^3、α^4}，选择剪枝决策树。本例利用测试样本计算各树的分类精度，分别为 5/6、5/6、5/6、5/6、3/6，选择分类精度最高、复杂度最低的 PT^3 作为剪枝后的决策树。

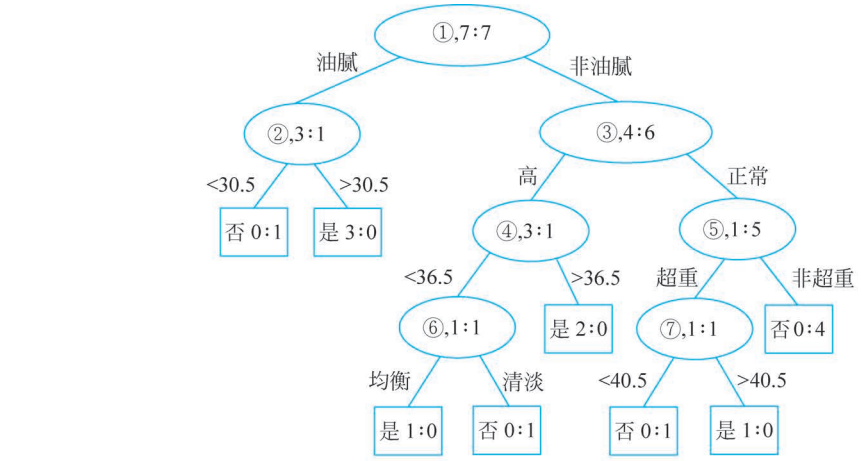
5.3.4　仿真实现

scikit-learn 库中 tree 模块提供了决策树分类的相关函数，对其进行简要介绍。

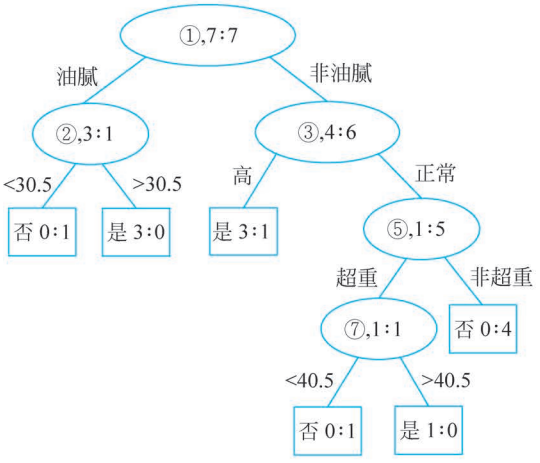
1) DecisionTreeClassifier(决策树分类)模型

DecisionTreeClassifier 模型定义如下：

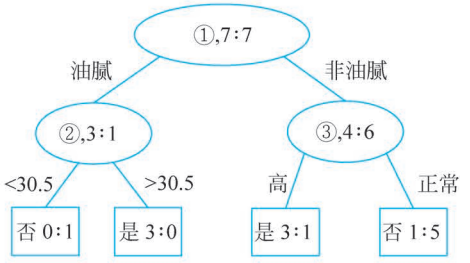
```
DecisionTreeClassifier( * , criterion = 'gini', splitter = 'best', max_depth = None, min_samples_
split = 2, min_samples_leaf = 1, min_weight_fraction_leaf = 0.0, max_features = None, random_
state = None, max_leaf_nodes = None, min_impurity_decrease = 0.0, class_weight = None, ccp_
alpha = 0.0)
```



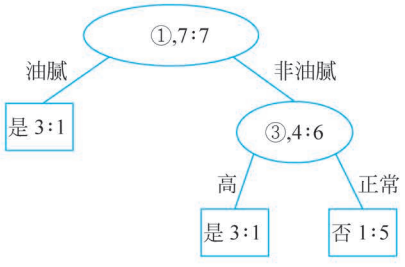
(a) PT^0决策树



(b) PT^1决策树



(c) PT^2决策树



(d) PT^3决策树

(e) PT^4决策树

图 5-12　基于 CCP 算法利用表 5-4 中数据生成后剪枝决策树

定义中给出了各参数的默认取值，具体含义如表 5-10 所示。

表 5-10　DecisionTreeClassifier 模型的参数

名　称	含　义
criterion	分枝依据，可取'gini'（默认）、'entropy'和'log_loss'
splitter	分枝策略，可取'best'（默认）或'random'

续表

名　　称	含　　义
max_depth	最大深度，整数，默认叶节点全部为同一类或者叶节点样本数少于 min_samples_split
min_samples_split	内部节点最少包含的样本数，可取整数或浮点数，后者表示节点样本数占样本总数的比例
min_samples_leaf	叶节点最少包含的样本点数，可取整数或浮点数，含义同上
max_features	进行分枝运算时搜索的最多特征数，可取整数、浮点数、'auto'、'sqrt'、'log2'或者 None
random_state	控制估计器的随机性，取整数时每次运算结果不变
max_leaf_nodes	最多叶节点数，可取整数或 None
min_impurity_decrease	用于判断节点是否继续分枝，不纯度的减少大于或等于这个值则继续分枝
class_weight	各类的权重，取 None 时表示各类权重都是 1
ccp_alpha	非负浮点数，最小代价-复杂度剪枝的复杂度参数，默认情况下不进行剪枝

DecisionTreeClassifier 模型主要方法有 fit、predict、predict_log_proba、predict_proba、score 等，含义和其他模型对应方法一样。另有两个决策树模型特有的方法如下。

(1) cost_complexity_pruning_path(X, y, sample_weight=None)：计算 CCP 剪枝路径，返回最小代价增量率序列 ccp_alpha 以及与 ccp_alpha 相应的子树叶节点不纯度之和。

(2) decision_path(X, check_input=True)：获取 X 的决策路径，返回矩阵中的非零元素，指明样本经过的节点。

2) plot_tree(绘制决策树)函数

plot_tree 函数定义如下：

```
plot_tree(decision_tree, *, max_depth = None, feature_names = None, class_names = None, label = 'all', filled = False, impurity = True, node_ids = False, proportion = False, rounded = False, precision = 3, ax = None, fontsize = None)
```

定义中给出了各参数的默认取值，具体含义如表 5-11 所示。

表 5-11　plot_tree 函数的部分参数

名　　称	含　　义
max_depth	整数，表示绘制的最大深度，默认为 None 时表示完全绘制
feature_names	列表或字符串，表示特征名，设置为 None 时用 $x[0]$、$x[1]$等代替
class_names	列表或字符串，表示类名，设置为 None 时不显示
label	设置在哪些节点显示各项数据标签，可取'all'(默认)、'root'、'none'
filled	设置为 True 时，根据节点中多数样本所属的类给节点填色
impurity	设置为 True 时，在节点中显示不纯度
proportion	设置为 True 时，value 和 samples 分别按比例和百分比显示
precision	整数，表示各项浮点数的精度位数
fontsize	字体大小，设置为 None 时自动调节字体大小适应图形窗口

【例 5-12】 利用 iris 数据集设计决策树，生成剪枝决策树，并利用不同的剪枝决策树对样本 $[4.8 \quad 3.5 \quad 1.5 \quad 0.2]^{T}$ 进行判别。

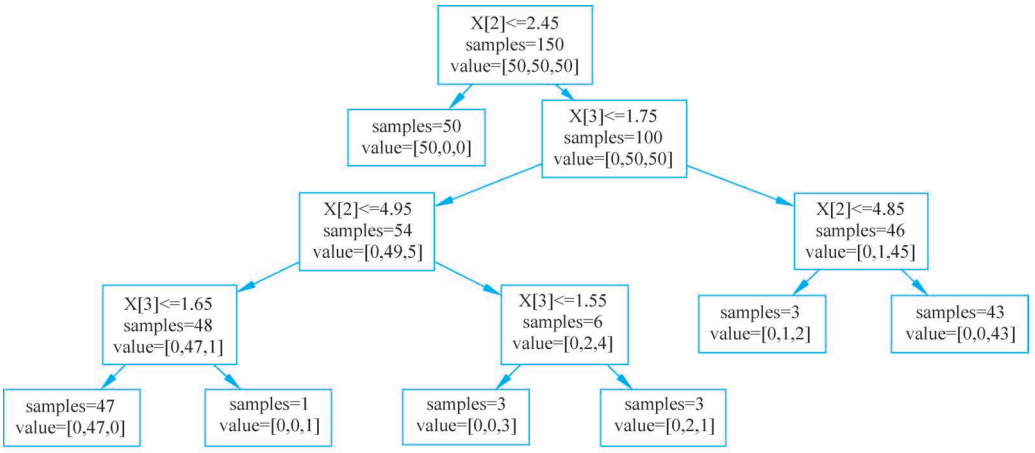
程序如下：

```
from sklearn import tree
```

```
import numpy as np
from sklearn.datasets import load_iris
from matplotlib import pyplot as plt
    #以下为导入数据集,训练决策树模型,获取 CCP 剪枝路径以及 ccp_alpha 序列长度
iris = load_iris()
clf = tree.DecisionTreeClassifier(min_samples_split = 5)
clf.fit(iris.data, iris.target)
ccp_path = clf.cost_complexity_pruning_path(iris.data, iris.target)
num_alpha = len(ccp_path.ccp_alphas)
    #以下分别从 ccp_alpha 序列中获取两个 ccp_alpha 值,并重新训练,获取剪枝决策树模型
alpha1 = ccp_path.ccp_alphas[max(num_alpha - 2, 1)]
clf1 = tree.DecisionTreeClassifier(min_samples_split = 5, ccp_alpha = alpha1)
clf1.fit(iris.data, iris.target)
alpha2 = ccp_path.ccp_alphas[max(num_alpha - 4, 1)]
clf2 = tree.DecisionTreeClassifier(min_samples_split = 5, ccp_alpha = alpha2)
clf2.fit(iris.data, iris.target)
    #设定待测样本,利用两个剪枝决策树模型进行决策、输出,并绘制不剪枝及两个剪枝决策树
sample = np.array([[4.8, 3.5, 1.5, 0.2]])
result1 = iris.target_names[clf1.predict(sample)]
result2 = iris.target_names[clf2.predict(sample)]
print('样本[4.8, 3.5, 1.5, 0.2]分别被判断为', result1, result2)
plt.rcParams['font.sans-serif'] = ['Times New Roman']
fig1 = plt.figure()
tree.plot_tree(clf, impurity = False, fontsize = 9)
fig2 = plt.figure()
tree.plot_tree(clf1, impurity = False, fontsize = 9)
fig3 = plt.figure()
tree.plot_tree(clf2, impurity = False, fontsize = 9)
plt.show()
```

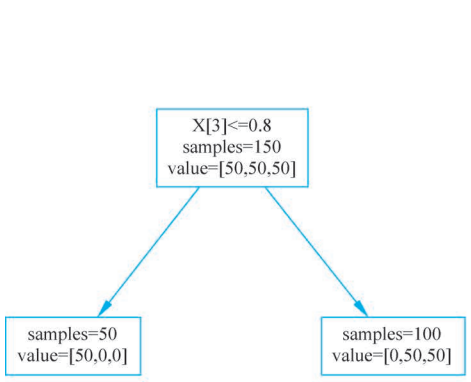
程序运行,生成的决策树如图 5-13 所示,树中各节点标明了分枝条件、节点样本数以及各类样本数。程序在输出窗口输出决策结果:

```
样本[4.8, 3.5, 1.5, 0.2]分别被判断为 ['setosa'] ['setosa']
```

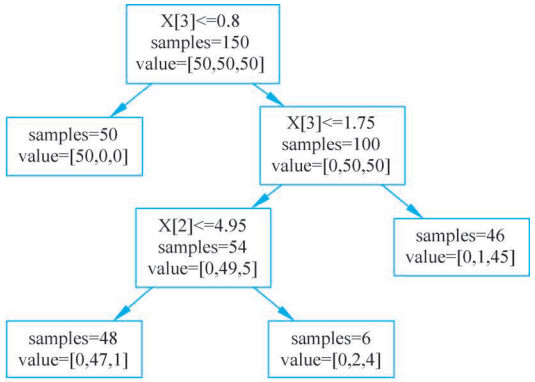


(a) 不剪枝决策树

图 5-13 利用 iris 数据集生成的决策树



(b) 较大α_1对应的剪枝决策树



(c) 较小α_2对应的剪枝决策树

图 5-13　（续）

5.4　Logistic 回归

Logistic 回归(Logistic Regression)是一种经典分类方法，也称为对数几率回归(也有文献译为“逻辑回归”“罗杰斯特回归”等)，是采用对数几率模型描述样本属于某类的可能性与样本特征之间的关系，用训练样本估计模型中的系数，进而实现分类的方法。

5.4.1　基本原理

考虑二分类任务，采用线性判别函数 $g(\boldsymbol{x})=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0$ 对样本 $\boldsymbol{x}$ 进行归类判别，设类别标签 $y\in\{0,1\}$，将样本归类，需要根据 $g(\boldsymbol{x})$的值判断 y 的值，因此，有

$$y=f[g(\boldsymbol{x})] \tag{5-18}$$

$f(\cdot)$称为 Link 函数(Link Function)。

例如：当 $g(\boldsymbol{x})>0$ 时，$\boldsymbol{x}\in\omega_1$，$y=1$；当 $g(\boldsymbol{x})<0$ 时，$\boldsymbol{x}\in\omega_2$，$y=0$；当 $g(\boldsymbol{x})=0$ 时，可以任意判别。Link 函数 $f(\cdot)$实际是单位阶跃函数，即

$$y=\begin{cases}0, & g(\boldsymbol{x})<0\\ 0.5, & g(\boldsymbol{x})=0\\ 1, & g(\boldsymbol{x})>0\end{cases} \tag{5-19}$$

函数如图 5-14 所示。

单位阶跃函数不连续，用单调可微的 Logistic 函数近似表达单位阶跃函数为

$$y=\frac{1}{1+\mathrm{e}^{-g(\boldsymbol{x})}} \tag{5-20}$$

Logistic 函数将 $g(\boldsymbol{x})$的值转换为一个接近 0 或 1 的 y 值，在 $g(\boldsymbol{x})=0$ 附近变化曲线很陡，图形如图 5-14 中虚线所示。

由式(5-19)可知，y 可以看作将样本 $\boldsymbol{x}$ 归为 ω_1 类的可能性，则 $1-y$ 是将样本归为 ω_2 类的可能性，当 $y>1-y$ 时，$\boldsymbol{x}\in\omega_1$；当 $y<1-y$ 时，$\boldsymbol{x}\in\omega_2$。定义概率为

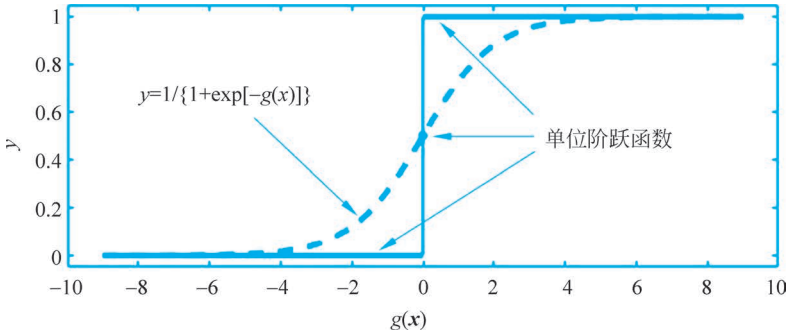


图 5-14　单位阶跃函数和 Logistic 函数

$$\frac{y}{1-y}=\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0} \tag{5-21}$$

反映了 $\boldsymbol{x}$ 归为 ω_1 类的相对可能性。

对概率取对数得到 logit 函数

$$\mathrm{logit}(\boldsymbol{x})=\ln\left(\frac{y}{1-y}\right)=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0 \tag{5-22}$$

表明样本属于某类的可能性与样本之间呈线性关系。很明显，

$$\text{当 } \mathrm{logit}(\boldsymbol{x})\gtrless 0 \text{ 时，}\quad \boldsymbol{x}\in\begin{cases}\omega_1\\ \omega_2\end{cases} \tag{5-23}$$

如果能够确定 $\boldsymbol{w}$ 和 w_0，则可以确定 logit 函数，从而实现分类。

进一步，样本属于某一类的可能性用后验概率表示，logit 函数重写为

$$\ln\left[\frac{P(y=1\mid\boldsymbol{x})}{P(y=0\mid\boldsymbol{x})}\right]=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0 \tag{5-24}$$

其中，

$$P(y=1\mid\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0)}}=\frac{\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0}}{1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0}} \tag{5-25}$$

$$P(y=0\mid\boldsymbol{x})=1-P(y=1\mid\boldsymbol{x})=\frac{\mathrm{e}^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0)}}{1+\mathrm{e}^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0)}}=\frac{1}{1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+w_0}} \tag{5-26}$$

采用最大似然估计的方法确定 $\boldsymbol{w}$ 和 w_0，给定数据集 $\mathscr{X}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$，对应的类别标号为 $\mathscr{Y}=\{y_1,y_2,\cdots,y_N\}$，对于每一个样本 $\boldsymbol{x}_i$，$i=1,2,\cdots,N$，有 $y_i\in\{0,1\}$，则

$$P(y_i\mid\boldsymbol{x}_i;\boldsymbol{w},w_0)=[P(y_i=1\mid\boldsymbol{x}_i;\boldsymbol{w},w_0)]^{y_i}[1-P(y_i=1\mid\boldsymbol{x}_i;\boldsymbol{w},w_0)]^{1-y_i} \tag{5-27}$$

定义对数似然函数为

$$h(\boldsymbol{w},w_0)=\sum_{i=1}^{N}\ln P(y_i\mid\boldsymbol{x}_i;\boldsymbol{w},w_0) \tag{5-28}$$

将式(5-25)、式(5-26)和式(5-27)代入式(5-28)，得

$$h(\boldsymbol{w},w_0)=\sum_{i=1}^{N}\{-y_i\ln[1+\mathrm{e}^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+w_0)}]-(1-y_i)\ln(1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+w_0})\} \tag{5-29}$$

对数似然函数求最大值得到最优的 $\boldsymbol{w}$ 和 w_0，或者对负对数似然函数求最小值，即

$$-h(\boldsymbol{w},w_0)=\sum_{i=1}^{N}\{y_i\ln[1+\mathrm{e}^{-(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+w_0)}]+(1-y_i)\ln(1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i+w_0})\} \tag{5-30}$$

可以通过迭代策略求解。

SciPy 库中 optimize 模块提供了进行优化计算的 minimize 函数，其调用格式如下。

```
minimize(fun, x0, args = (), method = None, jac = None, hess = None, hessp = None, bounds = None,
constraints = (), tol = None, callback = None, options = None)
```

其中，参数 fun 是要进行最小化的函数，其输入变量是一维数组；x0 是初始值；args 是其他需要的参数；method 指定优化算法，可以选'Nelder-Mead'、'Powell'、'CG' 、'BFGS'等(可以查阅优化算法类参考资料)；jac、hess、heep、bounds、constraints 是不同优化算法需要的参数；tol 是终止计算的容差量；options 是运算时的相关设置，包括最大迭代次数、是否显示优化计算信息等。返回一个 OptimizeResult 模型实例，属性 x 是计算出的优化值，根据采用的优化算法，有其他不同的属性。

【例 5-13】　三维空间两类分类问题，样本集为

$$\omega_1:\{[0\quad 0\quad 0]^{\mathrm{T}},[1\quad 0\quad 1]^{\mathrm{T}},[1\quad 0\quad 0]^{\mathrm{T}},[1\quad 1\quad 0]^{\mathrm{T}}\}$$

$$\omega_2:\{[0\quad 0\quad 1]^{\mathrm{T}},[0\quad 1\quad 1]^{\mathrm{T}},[0\quad 1\quad 0]^{\mathrm{T}},[1\quad 1\quad 1]^{\mathrm{T}}\}$$

试用 Logistic 回归求解判别函数权向量，并对样本 $[0\quad 0.6\quad 0.8]^{\mathrm{T}}$ 进行判别。

程序如下：

```
import numpy as np
from scipy.optimize import minimize

def nll_fun(w, *args):                                    #负对数似然函数
    data, y = args[0], args[1]
    return -np.sum(y * np.log(logistic_fun(w, data) + 0.001) +
                   (1 - y) * np.log(1 - logistic_fun(w, data) + 0.001))

X = np.array([[0, 0, 0], [1, 0, 0], [1, 0, 1], [1, 1, 0],
              [0, 0, 1], [0, 1, 1], [0, 1, 0], [1, 1, 1]])
y = np.array([1, 1, 1, 1, 0, 0, 0, 0])
gx = lambda w, x: w[0] * x[:, 0] + w[1] * x[:, 1] + w[2] * x[:, 2] + w[3]
logistic_fun = lambda w, x: 1 / (1 + np.exp(-gx(w, x)))
W0 = np.array([0, 0, 0, 0])                               #迭代求解初始值
opt = {'maxiter': 1000, 'disp': True}
res = minimize(nll_fun, W0, args = (X, y), method = 'Nelder-Mead', options = opt)
    #利用 minimize 函数、采用 Nelder-Mead 单纯形算法求无约束多变量函数的最小值
sample = np.array([0, 0.6, 0.8])
logitX = res.x[0] * sample[0] + res.x[1] * sample[1] + res.x[2] * sample[2] + res.x[3]
result = 1 if logitX > 0 else 2                           #对待测样本计算 logit 函数值并判断类别
np.set_printoptions(precision = 2)
print("最优权向量为", res.x)
print("样本[0, 0.6, 0.8]归为第 %d 类" % result)
```

运行程序，将在输出窗口输出：

```
Optimization terminated successfully.
         Current function value: -0.007996
         Iterations: 83
```

```
        Function evaluations: 217
最优权向量为 [ 159.7  -146.1  -114.91 60.57]
样本[0, 0.6, 0.8]归为第 2 类
```

需要注意：程序中最小值对应的权向量受初值 W0 的影响会发生变化。

5.4.2 多类分类任务

设训练样本集 $\mathcal{X}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_N\}$，各自的类别标号 $y_i\in\{1,2,\cdots,c\}$，$c>2$ 为类别数。将样本 $\boldsymbol{x}$ 标记为 j 的概率 $P_j=P(y=j\mid\boldsymbol{x})$，$j=1,2,\cdots,c$，$\sum_{j=1}^{c}P_j=1$，改写式(5-24)为

$$\ln\left(\frac{P_j}{\sum_{l\neq j}P_l}\right)=\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}+w_{j0} \tag{5-31}$$

$$P_j=\frac{1}{1+\mathrm{e}^{-(\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}+w_{j0})}}=\frac{\mathrm{e}^{\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}+w_{j0}}}{1+\mathrm{e}^{\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}+w_{j0}}} \tag{5-32}$$

$$\sum_{l\neq j}P_l=1-P_j=\frac{\mathrm{e}^{-(\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}+w_0)}}{1+\mathrm{e}^{-(\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}+w_0)}}=\frac{1}{1+\mathrm{e}^{\boldsymbol{w}_j^{\mathrm{T}}\boldsymbol{x}+w_0}} \tag{5-33}$$

同样采用最大似然估计的方法确定 $\boldsymbol{w}_j^{\mathrm{T}}$ 和 w_{j0}。

5.4.3 仿真实现

scikit-learn 库中 linear_model 模块提供了 LogisticRegression 模型，其调用格式如下。

```
LogisticRegression(penalty = 'l2', *, dual = False, tol = 0.0001, C = 1.0, fit_intercept = True,
intercept_scaling = 1, class_weight = None, random_state = None, solver = 'lbfgs', max_iter =
100, multi_class = 'auto', verbose = 0, warm_start = False, n_jobs = None, l1_ratio = None)
```

其中，参数 penalty 是额外增加的约束，可取 None、'l2'、'l1'或'elasticnet'；solver 指定优化算法，不同的优化算法能添加的约束也不一样；multi_class 指明在多类情况下的分类策略(在后续版本中会去掉)，可选'auto'、'ovr'和'multinomial'。

LogisticRegression 模型属性中，coef_存放权向量 $\boldsymbol{w}$；intercept_存放权向量常数 w_0。

LogisticRegression 模型主要方法有 decision_function、fit、predict、predict_log_proba、predict_proba、score 等，含义和其他模型对应方法一样。

【例 5-14】 利用 LogisticRegression 模型对例 5-13 中的数据进行 Logistic 回归分析。程序如下：

```
import numpy as np
from sklearn.linear_model import LogisticRegression

def gx(w, x):
    return w[0] * x[:, 0] + w[1] * x[:, 1] + w[2] * x[:, 2] + w[3]

def logistic_fun(w, x):                                  # Logistic 函数 P(y = 1|X)
    return 1 / (1 + np.exp( - gx(w, x)))

training = np.array([[0, 0, 0], [1, 0, 0], [1, 0, 1], [1, 1, 0],
```

```
                    [0, 0, 1], [0, 1, 1], [0, 1, 0], [1, 1, 1]])
y = np.array([1, 1, 1, 1, 0, 0, 0, 0])
clf = LogisticRegression(penalty = None, random_state = 0)
clf.fit(training, y)                          #拟合模型
W = np.append(clf.coef_, clf.intercept_)      #线性函数权向量
sample = np.array([[0, 0.6, 0.8]])
Py1 = logistic_fun(W, sample)                 #根据式(5-25)计算 P(y=1|X)
logitX = np.log(Py1/(1-Py1))                  #计算 logit 函数,或 logitX = gx(W, sample)
result = 1 if logitX > 0 else 2
np.set_printoptions(precision = 2)
print("最优权向量为", W)
print("样本[0, 0.6, 0.8]归类概率为", Py1, 1 - Py1)
print("样本[0, 0.6, 0.8]归为第 %d 类" % result)
```

运行程序,在输出窗口输出:

```
最优权向量为 [ 21.78 -21.46 -21.46 10.38]
样本[0, 0.6, 0.8]归类概率为 [2.87e-09] [1.]
样本[0, 0.6, 0.8]归为第 2 类
```

【例 5-15】 利用 iris 数据集拟合 Logistic 回归模型,并对样本 $[6.3 \quad 2.8 \quad 4.9 \quad 1.7]^{T}$ 进行归类。

程序如下:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
iris = load_iris()
clf = LogisticRegression(penalty = None, random_state = 2)
clf.fit(iris.data, iris.target)
W = np.concatenate((clf.coef_, clf.intercept_.reshape(-1, 1)), axis = 1)
sample = np.array([[6.3, 2.8, 4.9, 1.7]])
result = iris.target_names[clf.predict(sample)]
Py = clf.predict_proba(sample)
np.set_printoptions(precision = 2)
print("三类判别函数的权向量为", W)
print("样本[6.3, 2.8, 4.9, 1.7]归类概率为", Py)
print("样本[6.3, 2.8, 4.9, 1.7]归为", result, "类")
```

运行程序,在输出窗口输出:

```
三类判别函数的权向量为 [[ 7.35 20.4 -30.26 -14.14 3.98]
                         [ -2.44 -6.86 10.42 -2.07 19.33]
                         [ -4.91 -13.54 19.85 16.21 -23.31]]
样本[6.3 2.8 4.9 1.7]归类概率为 [[2.50e-43  3.98e-01  6.02e-01]]
样本[6.3 2.8 4.9 1.7]归为 ['virginica'] 类
```

5.5　非线性判别分析的实例

【例 5-16】 对由不同字体的数字图像构成的图像集,实现基于最近邻法和 Logistic 回归的分类器设计及判别。

1. 设计思路

两种分类方法均采用和例 2-16 相同的预处理方法,提取同样的特征。在最近邻法中,

测试样本在训练样本集中寻找最近邻，根据近邻的类别归类。设计方案如图 5-15 所示。Logistic 回归分类要先拟合多分类 Logistic 回归模型，再根据概率实现分类决策，设计方案如图 5-16 所示。

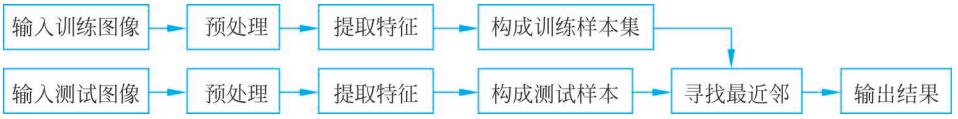


图 5-15　最近邻归类设计方案框图

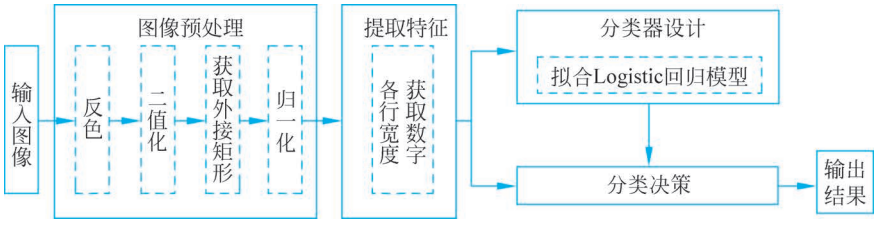


图 5-16　Logistic 回归分类设计方案框图

2. 程序设计

1）导入 KNeighborsClassifier 和 LogisticRegression 模型

在程序最开始导入两个模型。

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

2）生成训练样本

读取训练图像文件，经过图像预处理，提取特征，生成训练样本。同例 2-16。

3）拟合 KNeighborsClassifier 和 LogisticRegression 模型

生成训练样本后，拟合模型。

```
knn = KNeighborsClassifier(weights = 'distance')
knn.fit(training, y_train)
clf = LogisticRegression(penalty = 'none', random_state = 2)
clf.fit(training, y_train)
```

4）生成测试样本

读取测试图像文件，经过图像预处理，提取特征，生成测试样本。同例 2-16。

5）对测试样本进行决策归类

利用训练好的两个模型对测试样本进行预测，并测算检测率。

```
y_knn = knn.predict(testing)
y_log = clf.predict(testing)
ratio_knn = np.sum(y_test == y_knn)/y_test.size
ratio_log = np.sum(y_test == y_log)/y_test.size
print("最近邻法识别正确率：%.4f" % ratio_knn)
print("Logistic 回归识别正确率：%.4f" % ratio_log)
```

3. 实验结果

采用 10 个数字的共 50 幅图像进行训练，采用 30 幅图像进行测试，运行程序输出：

```
最近邻法识别正确率：0.8000
Logistic 回归识别正确率：0.8000
```

习题

1. 简述最小距离分类器的分类方法及特点。

2. 简述近邻法的分类思路及特点。

3. 简述决策树方法的分枝过程和决策过程，并说明决策树剪枝优化的思路。

4. 对例 5-9 中 CART 算法生成的决策树进行剪枝。

5. 已知二维空间三类样本均服从正态分布$\boldsymbol{\mu}_1=[1\quad 1]^{\mathrm{T}}$，$\boldsymbol{\mu}_2=[4\quad 4]^{\mathrm{T}}$，$\boldsymbol{\mu}_3=[8\quad 1]^{\mathrm{T}}$，$\boldsymbol{\Sigma}_1=\boldsymbol{\Sigma}_2=\boldsymbol{\Sigma}_3=2\boldsymbol{I}$，编写程序，基于这三类生成 1000 个二维向量的数据集，分别采用欧氏距离和马氏距离，利用数据集设计最小距离分类器。

6. 编写程序，随机生成二维向量作为待测样本，利用上题的训练样本，采用最近邻和 5 近邻方法对待测样本进行归类。

7. 利用 3 类样本 ω_1：$\left\{\begin{bmatrix}-5\\-5\end{bmatrix},\begin{bmatrix}-5\\-4\end{bmatrix},\begin{bmatrix}-4\\-5\end{bmatrix},\begin{bmatrix}-5\\-6\end{bmatrix},\begin{bmatrix}-6\\-5\end{bmatrix}\right\}$，$\omega_2$：$\left\{\begin{bmatrix}5\\5\end{bmatrix},\begin{bmatrix}5\\4\end{bmatrix},\begin{bmatrix}4\\5\end{bmatrix},\begin{bmatrix}5\\6\end{bmatrix},\begin{bmatrix}6\\5\end{bmatrix}\right\}$，$\omega_3$：$\left\{\begin{bmatrix}-5\\5\end{bmatrix},\begin{bmatrix}-5\\4\end{bmatrix},\begin{bmatrix}-4\\5\end{bmatrix},\begin{bmatrix}-5\\6\end{bmatrix},\begin{bmatrix}-6\\5\end{bmatrix}\right\}$，拟合 Logistic 回归模型，并对数据 $[-2\quad -2]^{\mathrm{T}}$ 进行归类。