第 5 章

CHAPTER 5

逻辑回归与最大熵模型

逻辑回归的核心思想是把原线性回归的取值范围通过 Logistic 函数映射到一个概率空间，从而将一个回归模型转换成了一个分类模型。本章将从最简单的二元逻辑回归出发，逐渐拓展到多元逻辑回归。非常重要的一点是，多元逻辑回归同时也是人工神经网络中的 Softmax 模型，或者从信息论的角度来解释，它又被称为最大熵模型。在推导最大熵模型时，还会用到凸优化（包含拉格朗日乘数法）的内容。

5.1 逻辑回归

下面通过一个例子来引出逻辑回归的基本思想。为研究与急性心肌梗死急诊治疗预后情况有关的因素，现收集了 200 个急性心肌梗死的病例如表 5-1 所示。其中，x_1 用于指示救治前是否休克，$x_1=1$ 表示救治前已休克，$x_1=0$ 表示救治前未休克；x_2 用于指示救治前是否心衰，$x_2=1$ 表示救治前已发生心衰，$x_2=0$ 表示救治前未发生心衰；x_3 用于指示 12 小时内有无治疗措施，$x_3=1$ 表示没有，$x_3=0$ 表示有。最后 P 给出了患者的最终结局，当 $P=0$ 时，表示患者生存；当 $P=1$ 时，表示患者死亡。

表 5-1 急性心肌梗死的病例数据

$P=0$				$P=1$			
x_1	x_2	x_3	N	x_1	x_2	x_3	N
0	0	0	35	0	0	0	4
0	0	1	34	0	0	1	10
0	1	0	17	0	1	0	4
0	1	1	19	0	1	1	15
1	0	0	17	1	0	0	6
1	0	1	6	1	0	1	9
1	1	0	6	1	1	0	6
1	1	1	6	1	1	1	6

如果要建立回归模型，进而预测不同情况下患者生存的概率，此时 P 的取值应该为 0～1。考虑用线性回归来建模：

$$P = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

显然将自变量代入上述回归方程，并不能保证概率 P 一定位于 0～1。最直接的想法是使用某个函数将上述线性回归模型的预测值映射至 0～1，逻辑回归模型就是使用 Logistic 函数完成这件事的。Logistic 函数的定义如下：

$$P = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

或

$$z = \ln \frac{P}{1-P}$$

其函数图像如图 5-1 所示。

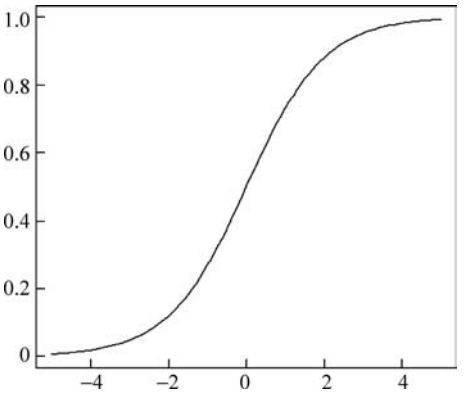

图 5-1　Logistic 函数图形

用上面的 Logistic 函数定义式将多元线性回归中的因变量替换得到

$$\ln \frac{P}{1-P} = w_0 + w_1 x_1 + \cdots + w_n x_n$$

或者

$$P = \frac{e^{w_0 + w_1 x_1 + \cdots + w_n x_n}}{1 + e^{w_0 + w_1 x_1 + \cdots + w_n x_n}}$$

或者

$$P = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + \cdots + w_n x_n)}}$$

可以简记为

$$P(z) = \frac{1}{1+e^{-z}}$$

其中，$z = w_0 + w_1 x_1 + \cdots + w_n x_n$，而且在当前所讨论的例子中 $n=3$。

上式中的 $w_0, w_1, \cdots, w_n$ 正是需要求解的参数，通常采用极大似然估计法对参数进行求解。对于每个观测到的样本，使用变量 y 标记样本的最终状态(0 为生存，1 为死亡)。采用上述模型可以简单得到每个样本生存和死亡的概率分别如下：

$$P(y=1) = \frac{e^z}{1+e^z}, \quad P(y=0) = 1 - P(y=1) = \frac{1}{1+e^z}$$

进而计算似然概率有

$$\mathcal{L} = \left\{ \left[\frac{\exp(w_0)}{1+\exp(w_0)} \right]^{35} \left[\frac{1}{1+\exp(w_0)} \right]^{4} \right\} \times$$
$$\left\{ \left[\frac{\exp(w_0 + w_3)}{1+\exp(w_0 + w_3)} \right]^{34} \left[\frac{1}{1+\exp(w_0 + w_3)} \right]^{10} \right\} \times$$

…

$$\left\{\left[\frac{\exp(w_0+w_1+w_2+w_3)}{1+\exp(w_0+w_1+w_2+w_3)}\right]^6\left[\frac{1}{1+\exp(w_0+w_1+w_2+w_3)}\right]^6\right\}$$

$$=\prod_{i=1}^{k}\left\{\left[\frac{\exp(w_0+w_1x_{1i}+w_2x_{2i}+w_3x_{3i})}{1+\exp(w_0+w_1x_{1i}+w_2x_{2i}+w_3x_{3i})}\right]^{n_{i0}}\left[\frac{1}{1+\exp(w_0+w_1x_{1i}+w_2x_{2i}+w_3x_{3i})}\right]^{n_{i1}}\right\}$$

寻找最适宜的 $\hat{w}_0,\hat{w}_1,\hat{w}_2,\hat{w}_3$ 使得$\mathcal{L}$达到最大,最终得到估计模型为

$$\ln\frac{P}{1-P}=-2.0858+1.1098x_1+0.7028x_2+0.9751x_3$$

或写成

$$P=\frac{1}{1+\mathrm{e}^{-(-2.0858+1.1098x_1+0.7028x_2+0.9751x_3)}}$$

得到上面的公式后,如果再有一组观察样本,将其代入公式,就可以算得患者生存与否的概率。例如,现在有一名患者 A 没有休克,病发 5 小时后送医院,而且已出现了症状,即 $x_1=0,x_2=1,x_3=0$,则可据此计算其生存的概率为

$$P_A=\frac{1}{1+\mathrm{e}^{-(-2.0858+0.7028)}}=0.200$$

同理,若另有一名患者 B 已经出现休克,病发 18 小时后送医院,出现了症状,即 $x_1=1$,$x_2=1,x_3=1$,则可据此计算其生存的概率为

$$P_B=\frac{1}{1+\mathrm{e}^{-(-2.0858+1.1098+0.7028+0.9751)}}=0.669$$

前面在对参数进行估计时,我们其实假设了 $y=1$ 的概率为

$$P(y=1\mid\boldsymbol{x};\boldsymbol{w})=h_{\boldsymbol{w}}(\boldsymbol{x})=g(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})=\frac{\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}}{1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}}$$

于是还有

$$P(y=0\mid\boldsymbol{x};\boldsymbol{w})=1-h_{\boldsymbol{w}}(\boldsymbol{x})=1-g(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}}$$

由此,便可以利用逻辑回归从特征学习中得出一个非 0 即 1 的分类模型。当要判别一个新来的特征属于哪个类时,只需求 $h_{\boldsymbol{w}}(\boldsymbol{x})$即可,若 $h_{\boldsymbol{w}}(\boldsymbol{x})$大于 0.5 就可被归为 $y=1$ 的类;反之就被归为 $y=0$ 类。

以上通过一个例子向读者演示了如何从原始的线性回归演化出逻辑回归。而且不难发现,逻辑回归可以用作机器学习中的分类器。当我们得到一个事件发生与否的概率时,自然就已经得出结论,其到底应该属于“发生”的那一类别,还是属于“不发生”的那一类别。

机器学习最终是希望机器自己学到一个可以用于问题解决的模型。而这个模型本质上是由一组参数(或权值)定义的,也就是前面讨论的 $w_0,w_1,\cdots,w_n$。在得到测试数据时,将这组参数与测试数据线性加和得到

$$z=w_0+w_1x_1+\cdots+w_nx_n$$

这里 $x_1,x_2,\cdots,x_n$ 是每个样本的 n 个特征。之后再按照 Logistic 函数的形式求出

$$P(z)=\frac{1}{1+\mathrm{e}^{-z}}$$

在给定特征向量 $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$ 时，条件概率 $P(y=1|\boldsymbol{x})$ 为根据观测量某事件 y 发生的概率。那么逻辑回归模型可以表示为

$$P(y=1\mid\boldsymbol{x})=\pi(\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{-z}}$$

相对应地，在给定条件 $\boldsymbol{x}$ 时，事件 y 不发生的概率为

$$P(y=0\mid\boldsymbol{x})=1-\pi(\boldsymbol{x})=\frac{1}{1+\mathrm{e}^{z}}$$

而且还可以得到事件发生与不发生的概率之比为

$$\mathrm{odds}=\frac{P(y=1\mid\boldsymbol{x})}{P(y=0\mid\boldsymbol{x})}=\mathrm{e}^{z}$$

这个比值称为事件的发生比。

概率论的知识告诉我们进行参数估计时可以采用最大似然法。假设有 m 个观测样本，观测值分别为 $y_1,y_2,\cdots,y_n$，设 $p_i=P(y_i=1|\boldsymbol{x}_i)$ 为给定条件下得到 $y_i=1$ 的概率。同样地，$y_i=0$ 的概率为 $1-p_i=P(y_i=0|\boldsymbol{x}_i)$，所以得到一个观测值的概率为 $P(y_i)=p_i^{y_i}(1-p_i)^{1-y_i}$。

各个观测样本之间相互独立，那么它们的联合分布为各边缘分布的乘积。得到似然函数为

$$\mathcal{L}(\boldsymbol{w})=\prod_{i=1}^{m}[\pi(\boldsymbol{x}_i)]^{y_i}[1-\pi(\boldsymbol{x}_i)]^{1-y_i}$$

我们的目标是求出使这一似然函数值最大的参数估计，于是对函数取对数得到

$$\begin{aligned}\ln\mathcal{L}(\boldsymbol{w})&=\sum_{i=1}^{m}\{y_i\ln[\pi(\boldsymbol{x}_i)]+(1-y_i)\ln[1-\pi(\boldsymbol{x}_i)]\}\\&=\sum_{i=1}^{m}\ln[1-\pi(\boldsymbol{x}_i)]+\sum_{i=1}^{m}y_i\ln\frac{\pi(\boldsymbol{x}_i)}{1-\pi(\boldsymbol{x}_i)}\\&=\sum_{i=1}^{m}\ln[1-\pi(\boldsymbol{x}_i)]+\sum_{i=1}^{m}y_iz_i\\&=\sum_{i=1}^{m}-\ln[1+\mathrm{e}^{z_i}]+\sum_{i=1}^{m}y_iz_i\end{aligned}$$

其中，$z_i=\boldsymbol{w}\cdot\boldsymbol{x}_i=w_0+w_1x_{i1}+\cdots+w_nx_{in}$，根据多元函数求极值的方法，为了求出使得 $\ln\mathcal{L}(\boldsymbol{w})$ 最大的向量 $\boldsymbol{w}=(w_0,w_1,\cdots,w_n)$，对上述的似然函数求偏导后得到

$$\begin{aligned}\frac{\partial\ln\mathcal{L}(\boldsymbol{w})}{\partial w_k}&=\sum_{i=1}^{m}y_ix_{ik}-\sum_{i=1}^{m}\frac{1}{1+\mathrm{e}^{z_i}}\mathrm{e}^{z_i}\cdot x_{ik}\\&=\sum_{i=1}^{m}x_{ik}[y_i-\pi(\boldsymbol{x}_i)]\end{aligned}$$

现在，我们所要做的就是通过上面已经得到的结论来求解使得似然函数最大化的参数向

量。在实际中有很多方法可供选择,其中比较常用的有梯度下降法、牛顿法和拟牛顿法等。

5.2 牛顿法解逻辑回归

现代计算中涉及大量的工程计算问题,这些计算问题往往很少采用我们通常在求解计算题甚至是考试时所采用的方法,因为计算机最擅长的无非就是“重复执行大量的简单任务”,所以数值计算方面的迭代法在计算机时代便有了很大的作用。一个典型的例子就是利用牛顿迭代法近似求解方程的方法。牛顿迭代又称为牛顿-拉夫逊(Newton-Raphson)方法。

有时候某些方程的求根公式可能很复杂(甚至有些方程可能没有求根公式),导致求解困难,这时便可利用牛顿法进行迭代求解。

如图 5-2 所示,假设要求解方程 $f(x)=0$ 的根,首先随便找一个初始值 x_0,如果 x_0 不是解,做一个经过$(x_0,f(x_0))$这个点的切线,与 x 轴的交点为 x_1。同样的道理,如果 x_1 不是解,做一个经过$(x_1,f(x_1))$这个点的切线,与 x 轴的交点为 x_2。以此类推。以这样的方式得到的 x_i 会无限趋近于 $f(x)=0$ 的解。

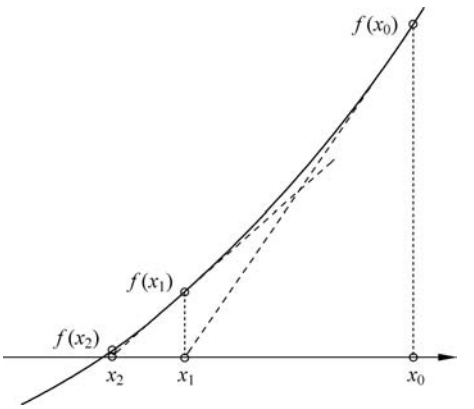

图 5-2 牛顿迭代法求方程的根

判断 x_i 是否是 $f(x)=0$ 的解有两种方法:一是直接计算 $f(x_i)$的值并判断其是否为 0,二是判断前后两个解 x_i 和 x_{i-1} 是否无限接近。经过$(x_i,f(x_i))$这个点的切线方程为(注意这也是一元函数的一阶泰勒展开式)

$$f(x)=f(x_i)+f'(x_i)(x-x_i)$$

其中,$f'(x)$为 $f(x)$的导数。令切线方程等于 0,即可求出

$$x_{i+1}=x_i-\frac{f(x_i)}{f'(x_i)}$$

于是得到了一个迭代公式,而且它必然在 $f(x^*)=0$ 处收敛,其中 x^* 就是方程的根,由此便可对方程进行迭代求根。

接下来讨论牛顿法在最优化问题中的应用。假设当前任务是优化一个目标函数 f,也就是求该函数的极大值或极小值问题,可以转化为求解函数 f 的导数 $f'=0$ 的问题,这样就可以把优化问题看成方程 $f'=0$ 的求解问题。剩下的问题就和前面提到的牛顿迭代法求解很相似了。

这次为了求解方程 $f'=0$ 的根,把原函数 $f(x)$做泰勒展开,展开到二阶形式(注意之前是一阶)

$$f(x+\Delta x)=f(x)+f'(x)\Delta x+\frac{1}{2}f''(x)\Delta x^2$$

当且仅当 Δx 无限趋近于 0 时，(可以舍去后面的无穷小项)使得等式成立。此时，上式等价于

$$f'(x)+\frac{1}{2}f''(x)\Delta x=0$$

注意因为 Δx 无限趋近于 0，前面的的常数$\frac{1}{2}$将不再起作用，可以将其一并忽略，即

$$f'(x)+f''(x)\Delta x=0$$

求解

$$\Delta x=-\frac{f'(x)}{f''(x)}$$

得出迭代公式

$$x_{n+1}=x_n-\frac{f'(x)}{f''(x)},\quad n=0,1,\cdots$$

最优化问题除了用牛顿法来解之外，还可以用梯度下降法来解。但是通常来说，牛顿法可以利用到曲线本身的信息，比梯度下降法更容易收敛，即迭代更少次数。

再次联系到 Hessian 矩阵与多元函数极值，对于一个多维向量 $\boldsymbol{x}$，以及在点 $\boldsymbol{x}_0$ 的邻域内有连续二阶偏导数的多元函数 $f(\boldsymbol{x})$，可以写出该函数在点 $\boldsymbol{x}_0$ 处的(二阶)泰勒展开式

$$f(\boldsymbol{x})=f(\boldsymbol{x}_0)+(\boldsymbol{x}-\boldsymbol{x}_0)^{\mathrm{T}}\nabla f(\boldsymbol{x}_0)+\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_0)^{\mathrm{T}}\boldsymbol{H}f(\boldsymbol{x}_0)(\boldsymbol{x}-\boldsymbol{x}_0)+o(\|\boldsymbol{x}-\boldsymbol{x}_0\|^2)$$

其中，$o(\|\boldsymbol{x}-\boldsymbol{x}_0\|^2)$是高阶无穷小表示的皮亚诺余项。而 $\boldsymbol{H}f(\boldsymbol{x}_0)$是一个 Hessian 矩阵。依据之前的思路，忽略掉无穷小项，写出迭代公式即为

$$\boldsymbol{x}_{n+1}=\boldsymbol{x}_n-\frac{\nabla f(\boldsymbol{x}_n)}{\boldsymbol{H}f(\boldsymbol{x}_n)},\quad n\geqslant 0$$

由此可知，高维情况依然可以用牛顿迭代求解，但是问题是 Hessian 矩阵引入的复杂性，使得牛顿迭代求解的难度大大增加。所以人们又提出了所谓的拟牛顿法(Quasi-Newton method)，不再直接计算 Hessian 矩阵，而是每一步的时候使用梯度向量更新 Hessian 矩阵的近似。

现在尝试用牛顿法来求解逻辑回归。我们已经求出了逻辑回归的似然函数的一阶偏导数

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{w})}{\partial w_k}=\sum_{i=1}^{m}x_{ik}[y_i-\pi(\boldsymbol{x}_i)]$$

由于 $\ln \mathcal{L}(\boldsymbol{w})$是一个多元函数，变量是 $\boldsymbol{w}=(w_0,w_1,\cdots,w_n)$，所以根据多元函数求极值问题的规则，易知极值点处的导数一定均为零，所以一共需要列出 $n+1$ 个方程，联立解出所有的参数。下面列出方程组如下：

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{w})}{\partial w_0}=\sum_{i=1}^{m}x_{i0}[y_i-\pi(\boldsymbol{x}_i)]=0$$

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{w})}{\partial w_1}=\sum_{i=1}^{m}x_{i1}[y_i-\pi(\boldsymbol{x}_i)]=0$$

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{w})}{\partial w_2}=\sum_{i=1}^{m} x_{i2}[y_i-\pi(\boldsymbol{x}_i)]=0$$

$$\vdots$$

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{w})}{\partial w_n}=\sum_{i=1}^{m} x_{in}[y_i-\pi(\boldsymbol{x}_i)]=0$$

当然,在具体解方程组之前需要用 Hessian 矩阵来判断极值的存在性。求 Hessian 矩阵就得先求二阶偏导,即

$$\begin{aligned}\frac{\partial^2 \ln \mathcal{L}(\boldsymbol{w})}{\partial w_k \partial w_j}&=\frac{\partial \sum_{i=1}^{m} x_{ik}[y_i-\pi(\boldsymbol{x}_i)]}{\partial w_j}\\&=\sum_{i=1}^{m} x_{ik}\frac{\partial\left(-\frac{e^{z_i}}{1+e^{z_i}}\right)}{\partial w_j}\\&=-\sum_{i=1}^{m} x_{ik}\frac{\partial(1+e^{-z_i})^{-1}}{\partial w_j}\\&=-\sum_{i=1}^{m} x_{ik}\left[-(1+e^{-z_i})^{-2}\cdot e^{-z_i}\cdot\left(-\frac{\partial z_i}{\partial w_j}\right)\right]\\&=-\sum_{i=1}^{m} x_{ik}[(1+e^{-z_i})^{-2}\cdot e^{-z_i}\cdot x_{ij}]\\&=-\sum_{i=1}^{m} x_{ik}\cdot\frac{1}{1+e^{z_i}}\cdot\frac{e^{z_i}}{1+e^{z_i}}\cdot x_{ij}\\&=\sum_{i=1}^{m} x_{ik}\cdot\pi(\boldsymbol{x}_i)\cdot[\pi(\boldsymbol{x}_i)-1]\cdot x_{ij}\end{aligned}$$

显然可以用 Hessian 矩阵来表示以上多元函数的二阶偏导数,于是有

$$\boldsymbol{X}=\begin{bmatrix}x_{10} & x_{11} & \cdots & x_{1n}\\ x_{20} & x_{21} & \cdots & x_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ x_{m0} & x_{m1} & \cdots & x_{mn}\end{bmatrix}$$

$$\boldsymbol{A}=\begin{bmatrix}\pi(\boldsymbol{x}_1)\cdot[\pi(\boldsymbol{x}_1)-1] & 0 & \cdots & 0\\ 0 & \pi(\boldsymbol{x}_2)\cdot[\pi(\boldsymbol{x}_2)-1] & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & \pi(\boldsymbol{x}_m)\cdot[\pi(\boldsymbol{x}_m)-1]\end{bmatrix}$$

所以得到 Hessian 矩阵 $\boldsymbol{H}=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{X}$,可以看出矩阵 $\boldsymbol{A}$ 是负定的。线性代数的知识告诉我们,如果 $\boldsymbol{A}$ 是负定的,那么 Hessian 矩阵 $\boldsymbol{H}$ 也是负定的。也就是说,多元函数存在局部极大值,这刚好与我们要求的最大似然估计相吻合。于是确信可以用牛顿迭代法来继续求解

最优化问题。

对于多元函数求解零点，同样可以用牛顿迭代法，对于当前讨论的逻辑回归，可以得到如下迭代式

$$\boldsymbol{X}_{\text{new}}=\boldsymbol{X}_{\text{old}}-\frac{\boldsymbol{U}}{\boldsymbol{H}}=\boldsymbol{X}_{\text{old}}-\boldsymbol{H}^{-1}\boldsymbol{U}$$

其中 $\boldsymbol{H}$ 是 Hessian 矩阵，$\boldsymbol{U}$ 的表达式如下：

$$\boldsymbol{U}=\begin{bmatrix} x_{10} & x_{11} & \cdots & x_{1n} \\ x_{20} & x_{21} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m0} & x_{m1} & \cdots & x_{mn} \end{bmatrix}\begin{bmatrix} y_1-\pi(\boldsymbol{x}_1) \\ y_2-\pi(\boldsymbol{x}_2) \\ \vdots \\ y_m-\pi(\boldsymbol{x}_m) \end{bmatrix}$$

由于 Hessian 矩阵 $\boldsymbol{H}$ 是对称负定的，将矩阵 $\boldsymbol{A}$ 提取一个负号出来，得到

$$\boldsymbol{A}'=\begin{bmatrix} \pi(\boldsymbol{x}_1)\cdot[1-\pi(\boldsymbol{x}_1)] & 0 & \cdots & 0 \\ 0 & \pi(\boldsymbol{x}_2)\cdot[1-\pi(\boldsymbol{x}_2)] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \pi(\boldsymbol{x}_m)\cdot[1-\pi(\boldsymbol{x}_m)] \end{bmatrix}$$

然后 Hessian 矩阵 $\boldsymbol{H}$ 变为 $\boldsymbol{H}'=\boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}'\boldsymbol{X}$，这样 $\boldsymbol{H}'$ 就是对称正定的了。那么现在牛顿迭代公式变为

$$\boldsymbol{X}_{\text{new}}=\boldsymbol{X}_{\text{old}}+(\boldsymbol{H}')^{-1}\boldsymbol{U}$$

现在我们需要考虑如何快速地算得 $(\boldsymbol{H}')^{-1}\boldsymbol{U}$，即解方程组 $(\boldsymbol{H}')\boldsymbol{X}=\boldsymbol{U}$。通常的做法是直接用高斯消元法求解，但是这种方法的效率一般比较低。而当前我们可以利用的一个有利条件是 $\boldsymbol{H}'$ 是对称正定的，所以可以用 Cholesky 矩阵分解法来解。

到这里，牛顿迭代法求解逻辑回归的原理已经介绍完了，但正如前面所提过的，在这个过程中因为要对 Hessian 矩阵求逆，计算量还是很大。于是研究人员又提出了拟牛顿法，它是针对牛顿法的弱点进行了改进，更具实践应用价值。

5.3　应用实例：二分类问题

本节将基于 Python 的机器学习包 scikit-learn，给出一个使用逻辑回归模型处理二分类问题的实例。在此基础之上，会使用 matplotlib 和 seaborn 这两个常用的包对数据进行一定程度的可视化，方便读者了解数据理解模型。

5.3.1　数据初探

一组来自世界银行的数据统计了 30 个国家的两项指标，第三产业增加值占 GDP 的比重和年龄大于或等于 65 岁的人口（也就是老龄人口）占总人口的比重。使用 Pandas 包的 read_csv()方法，以 DataFrame 的格式读取数据。简单地打印前几行数据如下：

```
import pandas as pd
# 读取数据
data = pd.read_csv("countries_data.csv")

# 以下为打印内容
        countries    Services_of_GDP    ages65_of_total    label
    0   Belgium                 76.7                 18        1
    1   France                  78.9                 18        1
    2   Denmark                 76.2                 18        1
    3   Spain                   73.9                 18        1
    4   Japan                   72.6                 25        1
```

数据集包含四列：第一列是国家名，第二列是“第三产业增加值占 GDP 比重”，第三列是“老龄人口占总人口的比重”，最后一列是数据的标注。简单根据数据集提供的两个特征绘制散点图，可以发现两个类别的数据之间区分度还是很明显的。此处引入 matplotlib 包，并使用 seaborn 包提供的 scatterplot()函数，对数据集进行可视化。代码如下：

```
import matplotlib.pyplot as plt
import seaborn as sns

# 绘制散点图
sns.scatterplot(data['Services_of_GDP'],data['ages65_of_total'],hue = data['label'],style =
data['label'],data = data)

# 打印绘制的散点图
plt.show()
```

上述代码中 scatterplot()函数接收了 5 个参数，前两个参数分别指定了散点图的 x 轴和 y 轴所使用的数据；第三和第四个参数分别指定了用以区分点的颜色和样式所依据的数据，此处使用 label 字段区分点的颜色和样式；最后一个参数 data 指定绘图依赖的 DataFrame，即 data。

观察图 5-3 不难发现，数据集只包含两种标注 0 和 1。右上方橙色的数据点第三产业增加值占比和老龄化人口占比都较高，应该是发达国家；左下方蓝色的数据点，第三产业增加值占比和老龄化人口占比相对较低，应该是发展中国家。事实上，右边数据点代表的国家都是 OECD(经济合作与发展组织)成员国，而左边数据点代表的国家则是发展中国家(包括一些东盟国家、南亚国家和拉美国家)。

5.3.2 建模

在了解数据的大致分布后，使用 scikit-learn 下 linear_model 提供的 LogisticRegression 类直接根据数据集拟合一个逻辑回归模型。

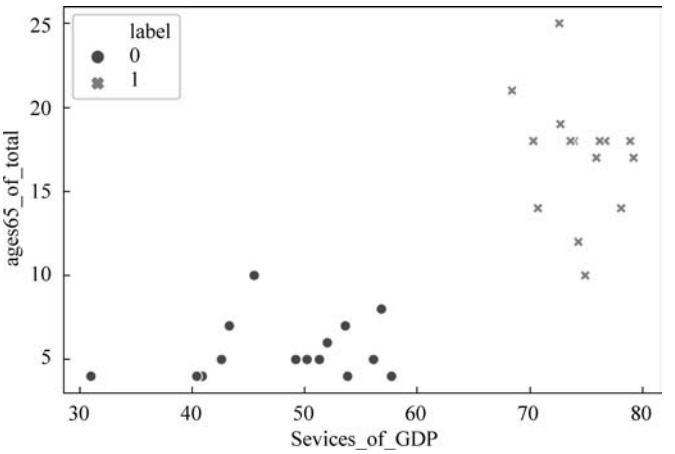

图 5-3 彩图

图 5-3　数据集的简单可视化

```
from sklearn.linear_model import LogisticRegression

# 区分特征和标注
X = data[['Services_of_GDP','Services_of_GDP']]
Y = data['label']

# 训练模型
clf = LogisticRegression(penalty = 'l2',fit_intercept = True,solver = 'liblinear',max_iter =
100).fit(X, Y)

# 使用训练得到的模型预测训练数据集
prediction = clf.predict(X)
```

在上面的示例代码中，首先声明了一个 LogisticRegression 类并接收了 4 个参数。从左到右，第一个参数 penalty 指定模型的正则化方式，并提供 l1 正则、l2 正则、elasticnet 正则和 none 共 4 个选项，示例代码中使用了 l2 正则。第二个参数 fit_intercept，配置决策函数中是否需要截距项，示例中设置为 True 表示需要截距项。第三个参数 solver，用于指定模型的优化算法，有 5 个选项：newton-cg、lbfgs、liblinear、sag 和 saga。其中 liblinear 是基于 C++的 LIBLINEAR 包实现的坐标下降法(coordinate descent)，对于小数据集效果较好，上述示例中便使用了此优化算法。另外几种优化算法中，newton-cg 是牛顿法的一种，lbfgs 是拟牛顿法的一种，sag 是随机平均梯度下降法，saga 是 sag 的一种变体算法。其中 newton-cg、lbfgs 和 sag 算法，只能配合 l1 正则或无正则项；liblinear 和 saga 均可配合 l1 和 l2 正则项，saga 算法还能处理 elasticnet 正则项。对于小数据集的场景，liblinear、lbfgs 和 newton-cg 都是不错的选择，而 sag 和 saga 则更适合大数据集的场景。第四个参数 max_iter 设置了优化算法的最大迭代次数，采用默认值 100。更多模型参数，请查阅 scikit-learn 的官方文档。紧接着，使用了 fit()方法，将训练数据 X 和标注数据 Y 传递给了方法，得到

训练好的模型。

使用如下代码,将模型在训练集上的预测结果绘制在图 5-4 中。

```
# 将预测结果添加为 data 的一列
data['prediction'] = prediction

# 使用颜色表示预测结果,样式表示真实标注
sns.scatterplot(data['Services_of_GDP'],data['ages65_of_total'],hue = data['prediction'],
style = data['label'],data = data)
plt.show()
```


图 5-4 彩图

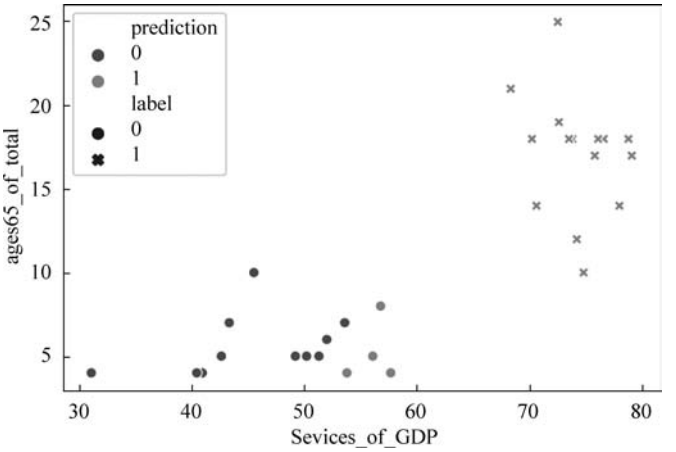

图 5-4 模型分类结果

图 5-4 中使用颜色标识了模型的预测结果,而数据点的样式标识了数据的真实标注。圆点代表真实标注为“发展中国家”,十字代表真实标注为“发达国家”;蓝色代表模型判断为“发展中国家”,橙色代表模型判断为“发达国家”。显然,一些发展中国家(圆点)被识别为了发达国家(橙色),即模型并未完全拟合训练集。若使用 score()函数,输入待评估数据 X 和真实标注 Y,则可以得到模型在训练数据集上的平均准确率。

```
# 使用 score 方法获取平均准确率
clf.score(X,Y)

# 以下为打印结果
0.8666666666666667
```

为了让模型更好地拟合训练数据集,可以尝试增加模型的表征能力,即模型复杂度。具体到调参上,可以尝试增大模型的迭代次数;通过参数 C 降低正则化项的权重;还可以尝试使用收敛速度更快的优化算法。这个例子中,将参数 C 增大到 2.5 或者换用 newton-cg 和 lbfgs 算法均可在其他参数不变的情况下使模型的平均准确率达到 1。感兴趣的读者可自行尝试。

此外 scikit-learn 还提供了另一种内置交叉验证的逻辑回归模型 LogisticRegressionCV，其中 CV 代表交叉验证(cross-validation)。这个方法在训练模型时会使用交叉验证根据 score() 方法的评估结果寻找最优的 C 和 l1_ratio 参数。使用同样的参数，这次换用 LogisticRegressionCV 训练模型并打印平均准确率。

```
from sklearn.linear_model import LogisticRegressionCV
clf2 = LogisticRegressionCV(penalty='l2',fit_intercept=True,solver='liblinear',max_iter
=100).fit(X, Y)
clf2.score(X,Y)

# 打印结果
1.0
```

换用 LogisticRegressionCV 建模后，几乎无须调参便完美拟合了训练数据集。实际应用中使用 LogisticRegressionCV 方法建模，可以在调参上节省不少的工作。此方法的参数列表和 LogisticRegression 方法大同小异，感兴趣的读者可自行查阅 scikit-learn 官方文档，了解更多细节。

5.4　多元逻辑回归

在本章前面的介绍中，逻辑回归通常只用来处理二分类问题，所以又称这种逻辑回归为 Binary(或 Binomial)Logistic Regression。本节以此为基础，讨论如何处理多分类问题，这也就是通常所说的 Multinomial Logistic Regression。MLR 还有很多可能听起来更熟悉的名字，例如，在神经网络中，称其为 Softmax；从最大熵原理出发，又称其为最大熵模型。

人们之所以会提出逻辑回归的方法，其实是从线性回归中得到的启发。通常，线性回归可以表示为

$$P = \boldsymbol{w} \cdot \boldsymbol{x}$$

其中，$\boldsymbol{w}$ 是参数向量；$\boldsymbol{x}$ 是样本观察值向量。对于一个二元分类器问题而言，我们希望 P 表示一个概率，也就是由 $\boldsymbol{x}$ 所表征的样本属于两个分类之一的概率。既然是概率，所以也就要求 P 介于 0 和 1 之间。但上述方程左侧的 $\boldsymbol{w} \cdot \boldsymbol{x}$ 可以取任何实数值。

为此，定义

$$\text{odds} = \frac{P}{1-P}$$

为概率 P 对它的补 $1-P$ 的比率，或正例对负例的比率。

然后对上式两边同时取对数，得到 logit 或 log-odds 如下：

$$y = \text{logit}(P) = \log \frac{P}{1-P} = \boldsymbol{w} \cdot \boldsymbol{x}$$

取反函数可得

$$P=\frac{e^{y}}{1+e^{y}}$$

此时，概率 P 就介于 0 到 1 之间了。

在其他一些文献中，也有人从另外一个角度对逻辑回归进行了解读。既然我们最终是想求条件概率 $P(y|\boldsymbol{x})$，其中 y 是目标的类别，$\boldsymbol{x}$ 是特征向量(或观察值向量)。那么根据贝叶斯定理，则有

$$\begin{aligned}P(y=1\mid\boldsymbol{x})&=\frac{P(\boldsymbol{x}\mid y=1)P(y=1)}{P(\boldsymbol{x}\mid y=1)P(y=1)+P(\boldsymbol{x}\mid y=0)P(y=0)}\\&=\frac{e^{y}}{1+e^{y}}=\frac{1}{1+e^{-y}}\end{aligned}$$

其中

$$y=\log\frac{P(\boldsymbol{x}\mid y=1)P(y=1)}{P(\boldsymbol{x}\mid y=0)P(y=0)}=\log\frac{P(y=1\mid\boldsymbol{x})}{P(y=0\mid\boldsymbol{x})}=\log\frac{P}{1-P}$$

这与本书前面的解读是一致的。

逻辑回归遵循一个(单次的)二项分布，所以可以写出下列 PMF：

$$P(y\mid\boldsymbol{x})=\begin{cases}P_{w}(\boldsymbol{x}), & y=1\\1-P_{w}(\boldsymbol{x}), & y=0\end{cases}$$

其中，$\boldsymbol{w}$ 是参数向量，且

$$P_{w}(\boldsymbol{x})=\frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{x}}}$$

上述 PMF 也可写为如下形式：

$$P(y\mid\boldsymbol{x})=[P_{w}(\boldsymbol{x})]^{y}[1-P_{w}(\boldsymbol{x})]^{1-y}$$

然后就可以使用 N 次观测样本的最大对数似然来对参数进行估计：

$$\begin{aligned}\ln\mathcal{L}(\boldsymbol{w})&=\ln\Big\{\prod_{i=1}^{N}[P_{w}(\boldsymbol{x}_{i})]^{y_{i}}[1-P_{w}(\boldsymbol{x}_{i})]^{1-y_{i}}\Big\}\\&=\sum_{i=1}^{N}\{y_{i}\log P_{w}(\boldsymbol{x}_{i})+(1-y_{i})\log[1-P_{w}(\boldsymbol{x}_{i})]\}\end{aligned}$$

根据多元函数求极值的方法，为了求出使得 $\ln\mathcal{L}(\boldsymbol{w})$ 最大的向量 $\boldsymbol{w}=(w_{0},w_{1},\cdots,w_{n})$，对上述的似然函数求偏导后得到

$$\begin{aligned}\frac{\partial\ln L(\boldsymbol{w})}{\partial w_{k}}&=\frac{\partial}{\partial w_{k}}\sum_{i=1}^{N}\{y_{i}\log P_{w}(\boldsymbol{x}_{i})+(1-y_{i})\log[1-P_{w}(\boldsymbol{x}_{i})]\}\\&=\sum_{i=1}^{N}x_{ik}[y_{i}-P_{w}(\boldsymbol{x}_{i})]\end{aligned}$$

因此，所要做的就是通过上面已经得到的结论来求解使得似然函数最大化的参数向量。我们已经在本章前面给出了基于牛顿迭代法求解的步骤。

现在考虑当目标类别数超过 2 时的情况，这时就需要使用多元逻辑回归(Multinomial

Logistic Regression)。总的来说，

- 二元逻辑回归是一个针对二项分布样本的概率模型；
- 多元逻辑回归则将同样的原则扩展到多项分布的样本上。

在这个从二分类向多分类演进的过程中，参考之前的做法无疑会带给我们许多启示。由于参数向量 $\boldsymbol{w}$ 和特征向量 $\boldsymbol{x}$ 的线性组合 $\boldsymbol{w}\cdot\boldsymbol{x}$ 的取值范围是任意实数，为了能够引入概率值 P，在二分类中我们的做法是将两种分类情况的概率之比(取对数后)作为与 $\boldsymbol{w}\cdot\boldsymbol{x}$ 相对应的取值，因为只有两个类别，所以类别 1 比上类别 2 也就等于类别 1 比上类别 2 的补，即

$$\log\frac{P(y=1\mid\boldsymbol{x})}{P(y=0\mid\boldsymbol{x})}=\log\frac{P}{1-P}=\boldsymbol{w}\cdot\boldsymbol{x}$$

事实上，我们知道，如果 $P(y=1|\boldsymbol{x})$ 大于 $P(y=0|\boldsymbol{x})$，自然就表示特征向量 $\boldsymbol{x}$ 所标准的样本属于 $y=1$ 这一类别的概率更高。所以 odds 这个比值是相当有意义的。同理，如果现在的目标类别超过 2 个，即 $y=1,y=2,\cdots,y=j,\cdots,y=J$，那么我们就应该从这些目标类别中找一个类别来作为基线，然后计算其他类别相对于这个基线的 log-odds。并让这个 log-odds 作为线性函数的自变量。通常，我们选择最后一个类别作为基线。

在多元逻辑回归模型中，假设每一个响应(response)的 log-odds 遵从一个线性模型，即

$$y_j=\log\frac{P_j}{P_J}=\boldsymbol{w}_j\cdot\boldsymbol{x}$$

其中，$j=1,2,\cdots,J-1$。这个模型与之前的二元逻辑回归模型本质上是一致的。只是响应 Y 的分布由二项分布变成了多项分布。而且我们不再只有一个方程，而是 $J-1$ 个。例如，当 $J=3$ 时，我们有两个方程，分别比较 $j=1$ 对 $j=3$ 和 $j=2$ 对 $j=3$。你可能会疑惑是不是缺失了一个比较。缺失的那个比较是在类别 1 和类别 2 之间进行的，而它很容易从另外两个比较的结果中获得，因为 $\log(P_1/P_2)=\log(P_1/P_3)-\log(P_2/P_3)$。

多元逻辑回归也可以写成原始概率 P_j(而不是 log-odds)的形式。从二项分布的逻辑回归中出发，有

$$y=\log\frac{P(y=1)}{P(y=0)}\Rightarrow P=\frac{\mathrm{e}^y}{1+\mathrm{e}^y}$$

类似地，在多项式分布的情况中，我们还可以得到

$$y_j=\log\frac{P_j}{P_J}=\boldsymbol{w}_j\cdot\boldsymbol{x}\Rightarrow P_j=\frac{\mathrm{e}^{y_j}}{\sum_{k=1}^{J}\mathrm{e}^{y_k}}$$

其中，$j=1,2,\cdots,J$。

下面我们来证明它。这里需要用到的一个事实是 $\sum_{j=1}^{J}P_j=1$。首先证明

$$P_J=\frac{1}{\sum_{j=1}^{J}\mathrm{e}^{y_j}}$$

如下：

$$
\begin{aligned}
P_J &= P_J \cdot \frac{\sum_{j=1}^{J} P_j}{\sum_{j=1}^{J} P_j} = \frac{P_J(P_1+P_2+\cdots+P_J)}{\sum_{j=1}^{J} P_j} \\
&= \frac{P_1}{\frac{P_1+P_2+\cdots+P_J}{P_J}} + \frac{P_2}{\frac{P_1+P_2+\cdots+P_J}{P_J}} + \cdots + \frac{P_J}{\frac{P_1+P_2+\cdots+P_J}{P_J}} \\
&= \frac{P_1}{e^{y_1}+e^{y_2}+\cdots+e^{y_J}} + \frac{P_2}{e^{y_1}+e^{y_2}+\cdots+e^{y_J}} + \cdots + \frac{P_J}{e^{y_1}+e^{y_2}+\cdots+e^{y_J}} \\
&= \frac{P_1+P_2+\cdots+P_J}{e^{y_1}+e^{y_2}+\cdots+e^{y_J}} = \frac{\sum_{j=1}^{J} P_j}{\sum_{j=1}^{J} e^{y_j}} = \frac{1}{\sum_{j=1}^{J} e^{y_j}}
\end{aligned}
$$

又因为 $P_j = P_J \cdot e^{y_j}$，而且 $y_J = 0$，综上便证明了之前的结论。

此外，如果从贝叶斯定理的角度出发，可得

$$
P(y=j \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid y=j)P(y=j)}{\sum_{k=1}^{J} P(\boldsymbol{x} \mid y=k)P(y=k)} = \frac{y_j}{\sum_{k=1}^{J} e^{y_k}}
$$

其中，$y_k = \log[P(\boldsymbol{x} \mid y=k)P(y=k)]$。

前面的推导就告诉我们，在多元逻辑回归中，响应变量 $\boldsymbol{y}$ 遵循一个多项分布。可以写出下列 PMF(假设有 J 个类别)：

$$
P(y \mid \boldsymbol{x}) = \begin{cases}
\frac{e^{y_1}}{\sum_{k=1}^{J} e^{y_k}} = \frac{e^{\boldsymbol{w}_1 \cdot \boldsymbol{x}}}{\sum_{k=1}^{J} e^{\boldsymbol{w}_k \cdot \boldsymbol{x}}}, & y=1 \\
\frac{e^{y_2}}{\sum_{k=1}^{J} e^{y_k}} = \frac{e^{\boldsymbol{w}_2 \cdot \boldsymbol{x}}}{\sum_{k=1}^{J} e^{\boldsymbol{w}_k \cdot \boldsymbol{x}}}, & y=2 \\
\vdots & \\
\frac{e^{y_j}}{\sum_{k=1}^{J} e^{y_k}} = \frac{e^{\boldsymbol{w}_j \cdot \boldsymbol{x}}}{\sum_{k=1}^{J} e^{\boldsymbol{w}_k \cdot \boldsymbol{x}}}, & y=j \\
\vdots & \\
\frac{e^{y_J}}{\sum_{k=1}^{J} e^{y_k}} = \frac{e^{\boldsymbol{w}_J \cdot \boldsymbol{x}}}{\sum_{k=1}^{J} e^{\boldsymbol{w}_k \cdot \boldsymbol{x}}}, & y=J
\end{cases}
$$

二元逻辑回归和多元逻辑回归本质上是统一的，二元逻辑回归是多元逻辑回归的特殊

情况。我们将这两类机器学习模型统称为逻辑回归。更进一步地，逻辑回归和“最大熵模型”本质上也是一致的、等同的。或者说最大熵模型是多元逻辑回归的另外一个称谓，我们将在 5.5 节讨论最大熵模型和多元逻辑回归的等同性。

5.5　最大熵模型

我们在 5.4 节中导出的多元逻辑回归之一般形式为

$$P(y=j \mid \boldsymbol{x})=\frac{\mathrm{e}^{\boldsymbol{w}_j \cdot \boldsymbol{x}}}{\sum_{k=1}^{J}\mathrm{e}^{\boldsymbol{w}_k \cdot \boldsymbol{x}}}$$

而本节将要介绍的最大熵模型的一般形式(其中的 f 为特征函数)为

$$P_{\boldsymbol{w}}(y \mid \boldsymbol{x})=\frac{1}{Z_{\boldsymbol{w}}(\boldsymbol{x})}\exp\left[\sum_{k=1}^{J}\boldsymbol{w}_k f_k(\boldsymbol{x},y)\right]$$

其中，

$$Z_{\boldsymbol{w}}(\boldsymbol{x})=\sum_{y}\exp\left[\sum_{k=1}^{J}\boldsymbol{w}_k f_k(\boldsymbol{x},y)\right]$$

此处，$\boldsymbol{x}\in\mathbb{R}^J$，为输入，$y=\{1,2,\cdots,K\}$为输出，$\boldsymbol{w}=\mathbb{R}^J$，为权值向量，$f_k(\boldsymbol{x},y)$为任意实值特征函数，其中 $k=1,2,\cdots,J$。

可见，多元逻辑回归和最大熵模型在形式上是统一的。事实上，尽管采用的方法不同，二者最终是殊途同归、万法归宗的。因此，无论是多元逻辑回归，还是最大熵模型，又或者是 Softmax，它们本质上都是统一的。本节就将从最大熵原理这个角度来推导上述最大熵模型的一般形式。

5.5.1　最大熵原理

本书前面已经讨论过熵的概念了。简单地说，假设离散随机变量 X 的概率分布是 $P(X)$，则其熵是

$$H(P)=-\sum_{x}P(x)\log P(x)$$

而且熵满足下列不等式：

$$0\leqslant H(P)\leqslant \log|X|$$

其中，$|X|$是 X 的取值个数，当且仅当 X 的分布是均匀分布时右边的等号成立。也就是说，当 X 服从均匀分布时，熵最大。

直观地，最大熵原理认为要选择的概率模型首先必须满足已有的事实，即约束条件。在没有更多信息的情况下，那些不确定的部分都是“等可能的”。最大熵原理通过熵的最大化来表示等可能性。“等可能性”不容易操作，而熵则是一个可以优化的数值指标。

吴军博士在其所著的《数学之美》一书中曾经谈到：“有一次，我去 AT&T 实验室作关

最大熵模型的报告，随身带了一个骰子。我问听众‘每个面朝上的概率分别是多少’，所有人都说是等概率，即各种点数的概率均为 1/6。这种猜测当然是对的。我问听众为什么，得到的回答是一致的：对这个‘一无所知’的骰子，假定它每一面朝上概率均等是最安全的做法（你不应该主观假设它像韦小宝的骰子一样灌了铅）。从投资的角度看，就是风险最小的做法。从信息论的角度讲，就是保留了最大的不确定性，也就是说让熵达到最大。接着我又告诉听众，我的这个骰子被我特殊处理过，已知四点朝上的概率是 1/3，在这种情况下，每个面朝上的概率是多少？这次，大部分人认为除去四点的概率是 1/3，其余的均是 2/15，也就是说，已知的条件（四点概率为 1/3）必须满足，而对于其余各点的概率因为仍然无从知道，因此只好认为它们均等。注意，在猜测这两种不同情况下的概率分布时，大家都没有添加任何主观的假设，诸如四点的反面一定是三点等（事实上，有的骰子四点的反面不是三点而是一点）。这种基于直觉的猜测之所以准确，是因为它恰好符合了最大熵原理。”

通过上面这个关于骰子的例子，我们对最大熵原理应该已经有了一个基本的认识，即“对已有知识进行建模，对未知内容不做任何假设（model all that is known and assume nothing about that which is unknown）”。

5.5.2 约束条件

最大熵原理是统计学习理论中的一般原理，将它应用到分类任务上就会得到最大熵模型。假设分类模型是一个条件概率分布 $P(Y|\boldsymbol{X})$，$\boldsymbol{X}\in \text{Input}\subseteq\mathbb{R}^n$ 表示输入（特征向量），$Y\in \text{Output}$ 表示输出（分类标签），Input 和 Output 分别是输入和输出的集合。这个模型表示的是对于给定的输入 $\boldsymbol{X}$，输出为 Y 的概率是 $P(Y|\boldsymbol{X})$。

给定一个训练数据集

$$\mathcal{T}=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$$

我们现在的目标是利用最大熵原理来选择最好的分类模型。首先考虑模型应该满足的条件。给定训练数据集，便可以据此确定联合分布 $P(\boldsymbol{X},Y)$的经验分布 $\widetilde{P}(\boldsymbol{X},Y)$，以及边缘分布 $P(\boldsymbol{X})$的经验分布。此处，则有

$$\widetilde{P}(\boldsymbol{X}=\boldsymbol{x},Y=y)=\frac{v(\boldsymbol{X}=\boldsymbol{x},Y=y)}{N}$$

$$\widetilde{P}(\boldsymbol{X}=\boldsymbol{x})=\frac{v(\boldsymbol{X}=\boldsymbol{x})}{N}$$

其中，$v(\boldsymbol{X}=\boldsymbol{x},Y=y)$ 表示训练数据集中样本$(\boldsymbol{x},y)$出现的频率（也就是计数）；$v(\boldsymbol{X}=\boldsymbol{x})$ 表示训练数据集中输入 $\boldsymbol{x}$ 出现的频率（也就是计数），N 是训练数据集的大小。

举个例子，在英汉翻译中，take 有多种解释，例如下文中存在 7 种。

(t_1) 抓住：The mother takes her child by the hand. 母亲抓住孩子的手。

(t_2) 拿走：Take the book home. 把书拿回家。

(t_3) 乘坐：I take a bus to work. 我乘坐公交车上班。

(t_4) 量：Take your temperature. 量一量你的体温。

(t_5) 装：The suitcase wouldn't take another thing. 这个手提箱不能装下别的东西了。

(t_6) 花费：I takes a lot of money to buy a house. 买一座房子要花很多钱。

(t_7) 理解：How do you take this passage? 你怎么理解这段话？

在没有任何限制的条件下，最大熵原理认为翻译成任何一种解释都是等概率的，即

$$P(t_1 \mid \boldsymbol{x}) = P(t_2 \mid \boldsymbol{x}) = \cdots = P(t_7 \mid \boldsymbol{x}) = \frac{1}{7}$$

实际中总有或多的限制条件，例如 t_1、t_2 比较常见，假设满足

$$P(t_1 \mid \boldsymbol{x}) + P(t_2 \mid \boldsymbol{x}) = \frac{2}{5}$$

同样根据最大熵原理，可以得出

$$P(t_1 \mid \boldsymbol{x}) = P(t_2 \mid \boldsymbol{x}) = \frac{1}{5}$$

$$P(t_3 \mid \boldsymbol{x}) = P(t_4 \mid \boldsymbol{x}) = P(t_5 \mid \boldsymbol{x}) = P(t_6 \mid \boldsymbol{x}) = P(t_7 \mid \boldsymbol{x}) = \frac{3}{25}$$

通常可以用特征函数 $f(\boldsymbol{x}, y)$来描述输入 $\boldsymbol{x}$ 和输出 y 之间的某一个事实。一般来说，特征函数可以是任意实值函数，下面我们采用一种最简单的二值函数来定义我们的特征函数

$$f(\boldsymbol{x}, y) = \begin{cases} 1, & \boldsymbol{x}, y \text{ 满足某一事实} \\ 0, & \text{其他} \end{cases}$$

它表示当 $\boldsymbol{x}$ 和 y 满足某一事实时，函数取值为 1，否则取值为 0。

在实际的统计模型中，我们通过引入特征（以特征函数的形式）提高准确率。例如 take 翻译为乘坐的概率小，但是当后面跟着交通工具的名词 bus，概率就变得非常大。于是有

$$f(\boldsymbol{x}, y) = \begin{cases} 1, & y = \text{乘坐，并且 next}(\boldsymbol{x}) = \text{bus} \\ 0, & \text{其他} \end{cases}$$

特征函数 $f(\boldsymbol{x}, y)$关于经验分布 $\widetilde{P}(\boldsymbol{X}, Y)$的期望值，用 $E_{\widetilde{P}}(f)$表示：

$$E_{\widetilde{P}}(f) = \sum_{\boldsymbol{x}, y} \widetilde{P}(\boldsymbol{x}, y) f(\boldsymbol{x}, y)$$

同理，$E_P(f)$表示 $f(\boldsymbol{x}, y)$在模型上关于实际联合分布 $P(\boldsymbol{X}, Y)$的数学期望，类似地有

$$E_P(f) = \sum_{\boldsymbol{x}, y} P(\boldsymbol{x}, y) f(\boldsymbol{x}, y)$$

注意，$P(\boldsymbol{x}, y)$是未知的，而建模的目标是生成 $P(y \mid \boldsymbol{x})$，因此我们希望将 $P(\boldsymbol{x}, y)$表示成 $P(y \mid \boldsymbol{x})$的函数。因此，利用贝叶斯公式，有 $P(\boldsymbol{x}, y) = P(\boldsymbol{x}) P(y \mid \boldsymbol{x})$，但 $P(\boldsymbol{x})$仍然是未知的。此时，只得利用 $\widetilde{P}(\boldsymbol{x})$来近似。于是便可以将 $E_P(f)$重写成

$$E_P(f) = \sum_{\boldsymbol{x}, y} \widetilde{P}(\boldsymbol{x}) P(y \mid \boldsymbol{x}) f(\boldsymbol{x}, y)$$

以上公式中的求和号 $\sum_{x,y}$ 均是对 $\sum_{(x,y)\in\tau}$ 的简写，下同。

对于概率分布 $P(y|\boldsymbol{x})$，我们希望特征函数 f 的期望应该与从训练数据集中得到的特征期望值相一致，因此提出约束：

$$E_P(f)=E_{\widetilde{P}}(f)$$

即

$$\sum_{x,y}\widetilde{P}(\boldsymbol{x})P(y\mid\boldsymbol{x})f(\boldsymbol{x},y)=\sum_{x,y}\widetilde{P}(\boldsymbol{x},y)f(\boldsymbol{x},y)$$

把上式作为模型学习的约束条件。假如有 J 个特征函数 $f_k(\boldsymbol{x},y)$，$i=1,2,\cdots,J$，那么相应就有 J 个约束条件。

5.5.3 模型推导

给定训练数据集，我们的目标是：利用最大熵原理选择一个最好的分类模型，即对于任意给定的输入 $\boldsymbol{x}\in\text{Input}$，可以以概率 $P(y|\boldsymbol{x})$ 输出 $y\in\text{Output}$。要利用最大熵原理，而目标是获取一个条件分布，因此要采用相应的条件熵 $H(Y|\boldsymbol{X})$，或者记作 $H(P)$。

$$\begin{aligned}H(Y\mid\boldsymbol{X})&=-\sum_{x,y}P(\boldsymbol{x},y)\log P(y\mid\boldsymbol{x})\\&=-\sum_{x,y}\widetilde{P}(\boldsymbol{x})P(y\mid\boldsymbol{x})\log P(y\mid\boldsymbol{x})\end{aligned}$$

至此，就可以给出最大熵模型的完整描述了。对于给定的训练数据集以及特征函数 $f_k(\boldsymbol{x},y)$，$k=1,2,\cdots,J$，最大熵模型就是求解

$$\begin{aligned}&\max_{P\in C}H(P)=-\sum_{x,y}\widetilde{P}(\boldsymbol{x})P(y\mid\boldsymbol{x})\log P(y\mid\boldsymbol{x})\\&\text{s.t.}\quad E_P(f_k)=E_{\widetilde{P}}(f_k),\quad k=1,2,\cdots,J\\&\sum_y P(y\mid\boldsymbol{x})=1\end{aligned}$$

或者按照最优化问题的习惯，可以将上述求最大值的问题等价地转化为下面这个求最小值的问题：

$$\begin{aligned}&\min_{P\in C}-H(P)=\sum_{x,y}\widetilde{P}(\boldsymbol{x})P(y\mid\boldsymbol{x})\log P(y\mid\boldsymbol{x})\\&\text{s.t.}\quad E_P(f_k)-E_{\widetilde{P}}(f_k)=0,\quad k=1,2,\cdots,J\\&\sum_y P(y\mid\boldsymbol{x})=1\end{aligned}$$

其中的约束条件 $\sum_y P(y|\boldsymbol{x})=1$ 是为了保证 $P(y|\boldsymbol{x})$ 是一个合法的条件概率分布。注意，上面的式子其实还隐含一个不等式约束，即 $P(y|\boldsymbol{x})\geqslant 0$，尽管无须显式地讨论它。现在便得到了一个带等式约束的最优化问题，求解这个带约束的最优化问题，所得之解即为最大熵模型学习的解。

这里需要使用拉格朗日乘数法，并将带约束的最优化之原始问题转换为无约束的最优化之对偶问题，并通过求解对偶问题来求解原始问题。首先，引入拉格朗日乘子 $w_0, w_1, \cdots, w_J$，并定义拉格朗日函数

$$\begin{aligned}\mathcal{L}(P,\boldsymbol{w}) &= -H(P) + w_0\left[1-\sum_y P(y\mid \boldsymbol{x})\right] + \sum_{k=1}^{J} w_k\left[E_P(f_k) - E_{\widetilde{P}}(f_k)\right] \\ &= \sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P(y\mid \boldsymbol{x})\log P(y\mid \boldsymbol{x}) + w_0\left[1-\sum_y P(y\mid \boldsymbol{x})\right] + \\ &\quad \sum_{k=1}^{J} w_k\left[\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)f_k(\boldsymbol{x},y) - \sum_{x,y}\widetilde{P}(\boldsymbol{x})P(y\mid \boldsymbol{x})f_k(\boldsymbol{x},y)\right]\end{aligned}$$

为了找到该最优化问题的解，我们将诉诸 Kuhn-Tucker 定理，该定理表明可以先将 P 看成是待求解的值，然后求解 $L(P,\boldsymbol{w})$ 得到一个以 $\boldsymbol{w}$ 为参数形式的 P^*，然后把 P^* 代回 $\mathcal{L}(P,\boldsymbol{w})$ 中，这时再求解 $\boldsymbol{w}^*$。本书后面介绍 SVM 时，还会再遇到这个问题。

最优化的原始问题是

$$\min_{P\in C}\max_{\boldsymbol{w}} \mathcal{L}(P,\boldsymbol{w})$$

通过交换极大和极小的位置，可以得到如下这个对偶问题：

$$\max_{\boldsymbol{w}}\min_{P\in C} \mathcal{L}(P,\boldsymbol{w})$$

由于拉格朗日函数 $\mathcal{L}(P,\boldsymbol{w})$ 是关于 P 的凸函数，原始问题与对偶问题的解是等价的。这样便可以通过求解对偶问题来求解原始问题。

对偶问题内层的极小问题 $\min_{P\in C}\mathcal{L}(P,\boldsymbol{w})$ 是关于参数 $\boldsymbol{w}$ 的函数，将其记为

$$\Psi(\boldsymbol{w}) = \min_{P\in C}\mathcal{L}(P,\boldsymbol{w}) = \mathcal{L}(P_{\boldsymbol{w}},\boldsymbol{w})$$

同时将其解记为

$$P_{\boldsymbol{w}} = \operatorname*{argmin}_{P\in C}\mathcal{L}(P,\boldsymbol{w}) = P_{\boldsymbol{w}}(y\mid \boldsymbol{x})$$

接下来，根据费马定理，求 $\mathcal{L}(P,\boldsymbol{w})$ 对 $P(y|\boldsymbol{x})$ 的偏导数

$$\begin{aligned}\frac{\partial L(P,\boldsymbol{w})}{\partial P(y\mid \boldsymbol{x})} &= \sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})[\log P(y\mid \boldsymbol{x})+1] - \sum_y w_0 - \sum_{\boldsymbol{x},y}\left[\widetilde{P}(\boldsymbol{x})\sum_{k=1}^{J} w_k f_k(\boldsymbol{x},y)\right] \\ &= \sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})[\log P(y\mid \boldsymbol{x})+1] - \sum_{\boldsymbol{x}}\widetilde{P}(\boldsymbol{x})\sum_y w_0 - \sum_{\boldsymbol{x},y}\left[\widetilde{P}(\boldsymbol{x})\sum_{k=1}^{J} w_k f_k(\boldsymbol{x},y)\right] \\ &= \sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})\left[\log P(y\mid \boldsymbol{x})+1-w_0-\sum_{k=1}^{J} w_k f_k(\boldsymbol{x},y)\right]\end{aligned}$$

注意上述推导中运用了下面这个事实：

$$\sum_{\boldsymbol{x}}\widetilde{P}(\boldsymbol{x}) = 1$$

进一步地，令

$$\frac{\partial\,\mathcal{L}(P,\boldsymbol{w})}{\partial P(y\mid \boldsymbol{x})} = 0$$

又因为 $\widetilde{P}(\boldsymbol{x})>0$，于是有

$$\log P(y \mid \boldsymbol{x}) + 1 - w_0 - \sum_{k=1}^{J} w_k f_k(\boldsymbol{x}, y) = 0$$

进而有

$$P(y \mid \boldsymbol{x}) = \exp\left[w_0 - 1 + \sum_{k=1}^{J} w_k f_k(\boldsymbol{x}, y)\right] = \exp(w_0 - 1) \cdot \exp\left[\sum_{k=1}^{J} w_k f_k(\boldsymbol{x}, y)\right]$$

又因为

$$\sum_{y} P(y \mid \boldsymbol{x}) = 1$$

所以可得

$$\sum_{y} P(y \mid \boldsymbol{x}) = \exp[w_0 - 1] \cdot \sum_{y} \exp\left[\sum_{k=1}^{J} w_k f_k(\boldsymbol{x}, y)\right] = 1$$

即

$$\exp(w_0 - 1) = 1 \Big/ \sum_{y} \exp\left[\sum_{k=1}^{J} w_k f_k(\boldsymbol{x}, y)\right]$$

将上面的式子代回前面 $P(y|\boldsymbol{x})$的表达式,则得到

$$P_{\boldsymbol{w}} = \frac{1}{Z_{\boldsymbol{w}}(\boldsymbol{x})} \exp\left[\sum_{k=1}^{J} w_k f_k(\boldsymbol{x}, y)\right]$$

其中,

$$Z_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{y} \exp\left[\sum_{k=1}^{J} w_k f_k(\boldsymbol{x}, y)\right]$$

$Z_{\boldsymbol{w}}(\boldsymbol{x})$称为规范化因子;$f_k(\boldsymbol{x}, y)$是特征函数;$w_k$ 是特征的权值。由上述两式所表示的模型 $P_{\boldsymbol{w}} = P_{\boldsymbol{w}}(y|\boldsymbol{x})$就是最大熵模型。这里,$\boldsymbol{w}$ 是最大熵模型中的参数向量。注意,我们之前曾经提过,特征函数可以是任意实值函数,如果 $f_k(\boldsymbol{x}, y) = x_k$,那么这其实也就是 5.5.2 节中所说的多元逻辑回归模型,即

$$P_j = P(y = j \mid \boldsymbol{x}) = \frac{\mathrm{e}^{\boldsymbol{w}_j \cdot \boldsymbol{x}}}{\sum_{k=1}^{J} \mathrm{e}^{\boldsymbol{w}_k \cdot \boldsymbol{x}}}$$

5.5.4 极大似然估计

下面,需要求解对偶问题中外部的极大化问题

$$\max_{\boldsymbol{w}} \Psi(\boldsymbol{w})$$

将其解记为 $\boldsymbol{w}^*$,即

$$\boldsymbol{w}^* = \arg\max_{\boldsymbol{w}} \Psi(\boldsymbol{w})$$

这就是说,可以应用最优化算法求对偶函数 $\Psi(\boldsymbol{w})$的极大化,得到 $\boldsymbol{w}^*$,用来表示 $P^* \in C$。这里,$P^* = P_{\boldsymbol{w}^*} = P_{\boldsymbol{w}^*}(y|\boldsymbol{x})$是学习到的最优模型(最大熵模型)。于是,最大熵模型的学习算法现在就归结为对偶函数 $\Psi(\boldsymbol{w})$的极大化问题上来。

前面已经给出了 $\Psi(\boldsymbol{w})$的表达式:

$$\Psi(\boldsymbol{w})=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P(y\mid\boldsymbol{x})\log P(y\mid\boldsymbol{x})+w_0\Big[1-\sum_{y}P(y\mid\boldsymbol{x})\Big]+$$
$$\sum_{k=1}^{J}w_k\Big[\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)f_k(\boldsymbol{x},y)-\sum_{x,y}\widetilde{P}(\boldsymbol{x})P(y\mid\boldsymbol{x})f_k(\boldsymbol{x},y)\Big]$$

由于,其中

$$\sum_{y}P_w(y\mid\boldsymbol{x})=1$$

于是将 $P_w(y|\boldsymbol{x})$代入 $\Psi(\boldsymbol{w})$,可得

$$\begin{aligned}\Psi(\boldsymbol{w})&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P_w(y\mid\boldsymbol{x})\log P_w(y\mid\boldsymbol{x})+\\&\sum_{k=1}^{J}w_k\Big[\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)f_k(\boldsymbol{x},y)-\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P_w(y\mid\boldsymbol{x})f_k(\boldsymbol{x},y)\Big]\\&=\sum_{k=1}^{J}w_k\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)f_k(\boldsymbol{x},y)+\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P_w(y\mid\boldsymbol{x})\log P_w(y\mid\boldsymbol{x})-\\&\sum_{k=1}^{J}w_k\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P_w(y\mid\boldsymbol{x})f_k(\boldsymbol{x},y)\\&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)+\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P_w(y\mid\boldsymbol{x})\Big[\log P_w(y\mid\boldsymbol{x})-\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)\Big]\\&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)-\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x})P_w(y\mid\boldsymbol{x})\log Z_w(\boldsymbol{x})\\&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)-\sum_{\boldsymbol{x}}\widetilde{P}(\boldsymbol{x})\log Z_w(\boldsymbol{x})\sum_{y}P_w(y\mid\boldsymbol{x})\\&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)-\sum_{\boldsymbol{x}}\widetilde{P}(\boldsymbol{x})\log Z_w(\boldsymbol{x})\end{aligned}$$

注意其中倒数第 4 行至倒数第 3 行运用了下面这个推导:

$$P_w(y\mid\boldsymbol{x})=\frac{1}{Z_w(\boldsymbol{x})}\exp\Big[\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)\Big]\Rightarrow\log P_w(y\mid\boldsymbol{x})=\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)-\log Z_w(\boldsymbol{x})$$

下面来证明对偶函数的极大化等价于最大熵模型的极大似然估计。已知训练数据的经验概率分布 $\widetilde{P}(\boldsymbol{X},Y)$,条件概率分布 $P(Y|\boldsymbol{X})$的对数似然函数表示为

$$\mathcal{L}_{\widetilde{P}}(P_w)=\log\prod_{\boldsymbol{x},y}P(y\mid\boldsymbol{x})^{\widetilde{P}(\boldsymbol{x},y)}=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\log P(y\mid\boldsymbol{x})$$

当条件概率分布 $P(y|\boldsymbol{x})$是最大熵模型时,对数似然函数为

$$\begin{aligned}\mathcal{L}_{\widetilde{P}}(P_w)&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\log P_w(y\mid\boldsymbol{x})\\&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\Big[\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)-\log Z_w(\boldsymbol{x})\Big]\\&=\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\sum_{k=1}^{J}w_kf_k(\boldsymbol{x},y)-\sum_{\boldsymbol{x},y}\widetilde{P}(\boldsymbol{x},y)\log Z_w(\boldsymbol{x})\end{aligned}$$

对比之后，不难发现

$$\Psi(\boldsymbol{w}) = \mathcal{L}_{\underset{P}{\sim}}(P_{\boldsymbol{w}})$$

既然对偶函数 $\Psi(\boldsymbol{w})$ 等价于对数似然函数，也就证明了最大熵模型学习中的对偶函数极大化等价于最大熵模型的极大似然估计这一事实。由此，最大熵模型的学习问题就转换为具体求解“对数似然函数极大化的问题”或者“对偶函数极大化的问题”。

5.6 应用实例：多分类问题

本节将演示使用多元逻辑回归模型对手写数字进行识别。作为例子，这里要完成的任务是对 0～9 这 10 个手写数字进行分类。示例中使用到的数据集是 scikit-learn 内置的数据集。此数据集是创建于 1998 年的 UCI 手写数字数据集中测试集的复制，包含 1797 个样本，每个样本是一张 8×8 的手写数字图片。因此每个样本也可以看作有 8×8 = 64 维特征，每维特征是个 0～16 的整数。

5.6.1 数据初探

首先加载数据，并可视化数据集中前 50 个样本，结果见图 5-5。

```
from sklearn.datasets import load_digits

# 加载内置数据集
digits = load_digits()

# 可视化数据集的前几项
_, axes = plt.subplots(5, 10)

axes = axes.reshape(1, -1)
for ax, image in zip(axes[0], digits.images[:50]):
  ax.set_axis_off()
  ax.imshow(image, cmap = plt.cm.gray_r)

plt.show()
```

上面的代码中，首先使用 load_digits()函数加载了 scikit-learn 内置的手写数字数据集。然后，使用 matplotlib 包中 pyplot 的 subplots()方法生成一个 5 行 10 列的布局，返回的是每张子图的坐标。最后，将每个样本填入对应的坐标中，并使用 imshow()方法将数据绘制为一张二维图片，便得到了图 5-5 的效果。显然，数据集的确是 0～9 等数字的手写图片，每个数字都有多张图片作为样

图 5-5 手写数字数据集的样例

本，数字本身则是标注。

5.6.2　建模

显然这是个多分类问题，数据集一共有 10 个类别。仍然使用 scikit-learn 下 linear_model 模块中提供的 LogisticRegression 类建模，不过在参数的设置上需要格外注意。

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_raw = digits.images
Y = digits.target

# 将原始三维矩阵转化成二维矩阵
X = X_raw.reshape(X_raw.shape[0], -1)

# 将原始数据划分成训练和测试集两个部分
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

# 归一化原始数据
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 训练模型
clf = LogisticRegression(penalty='l1',fit_intercept=True,solver='saga',tol=0.001,max_
iter=1000)
clf.fit(X_train, Y_train)
```

上述代码中，首先将原本三维的数据转化成了二维，方便后续处理。注意，这个操作和机器学习中的降维算法是截然不同的。此处只是将原本 1797×8×8 的三维数组变形为 1797×64 的格式，原本每个样本是 64 个特征，变化后还是 64 个特征，并不改变原始的特征取值。而后续章节将介绍的降维算法，属于机器学习中的流型学习(manifold learning)，一般会改变数据的原始取值。

然后采用 train_test_split()函数将数据分成训练集和测试集。函数的第一个参数是特征数据 X，第二个参数是标注数据 Y，第三个参数设定测试集的占比。此处划分比例被设置为 0.2，即原始数据的 80%将用于模型训练，而剩余 20%的数据则是测试数据集。紧接着，使用 StandardScaler 类的 fit_transform()方法，对训练集进行 z-score 标准化处理。fit_transform()方法等价于先调用 fit()方法得到标准化模型 scaler，再用这个实例的 transform()方法对训练集进行标准化。然后，调用基于训练集得到的预处理模型 scaler 的 transform()方法，对测试集的特征进行同样的标准化处理。标准化的目的是防止特征各维度值域的差

异对模型造成影响，注意训练集和测试集的标准化计算必须一致。

最后，声明一个 LogisticRegression 类的实例，设置必要参数后，调用 fit()方法基于标准化后的训练集和标注训练模型。这次建模采用的优化算法是 saga，正则项为 L1；此外，模型的最大迭代次数提升到了 1000 次，而模型的停止迭代误差被提升到了 0.001。为什么要这样调整模型的参数设置呢？这就不得不深入到优化算法各自的特点了。

之前二分类的示例中，采用的优化算法是在小数据集上效果不错的 liblinear 算法。虽然 liblinear 算法也可以处理多分类任务，但是其实现的坐标下降法并不能学习到一个真正的多分类模型。liblinear 算法实际上使用的是“一对多”(one-vs-rest)框架处理多分类问题，即将每个类别和剩余所有类别看成一个二分类问题，然后训练多个二分类模型集成后处理多分类问题。第 7 章将详细介绍这种框架。

剩余的 newton-cg、lbfgs、sag 和 saga 均能学习到真正的多分类模型。我们知道，L1 正则具有将参数稀疏化的能力，从而简化学习到的模型参数。观察图 5-3 的任一个数字样本，能发现虽然手写数字共 64 个特征，而实际上包含了有效信息的特征并不多。有效信息是图中黑色笔画的部分，白色部分并没有有效信息。因此，这个问题中采用 L1 正则是更好的选择，又因为剩余的算法中只有 saga 算法能处理 L1 正则，所以优化算法采用 saga。剩余的 max_iter 和 tol 参数都是为了保证模型更快的收敛而设置的，感兴趣的读者可自行尝试。

最后，简单使用 score()方法观察学习到的模型的平均精确度。

```
score = clf.score(X_test, Y_test)
print("Test score with L1 penalty: %.4f" % score)

# 以下为打印内容
Test score with L1 penalty: 0.9556
```

每个类别的详细分类效果，可使用 metrics 模块下的 classification_report()方法，只需要提供预测结果和真实标注即可。

```
from sklearn import metrics

predicted = clf.predict(X_test)
print("Classification report for classifier %s:\n%s\n" % (clf, metrics.classification_
report(Y_test, predicted)))

# 打印结果如下:
                    precision    recall      f1-score    support

             0        1.00        1.00        1.00         29
             1        0.86        0.92        0.89         39
             2        0.98        0.98        0.98         44
             3        1.00        0.93        0.96         40
             4        1.00        0.97        0.98         30
```

```
              5       0.98      0.98      0.98        42
              6       1.00      0.97      0.99        36
              7       0.97      1.00      0.98        30
              8       0.94      0.89      0.92        38
              9       0.86      0.94      0.90        32

    accuracy                            0.96       360
    macro avg       0.96      0.96      0.96       360
    weighted avg    0.96      0.96      0.96       360
```

上述打印内容中，第一列为类别名称，后面 3 列为 precision、recall、f1 值等指标，最后一列是测试集中该类别的数量。可以发现模型对数字 0 的识别效果最好，对数字 1 的识别效果最差。此外，metrics 模块还提供了一个函数 plot_confusion_matrix()能绘制模型的混淆矩阵，只需提供分类器、测试集和标注。

```
disp = metrics.plot_confusion_matrix(clf, X_test, Y_test)
disp.figure_.suptitle("Confusion Matrix") #设置图片名称
plt.show()
```

运行上述代码，结果见图 5-6。其中纵坐标为真实标注，横坐标为预测类别。显然，对角线上的数字代表分类正确的样本数，非对角线上的数字代表了误分类样本数。例如，真实标注为 1 的第二行，可以发现有两个样本被误识别为了数字“8”，还有一个样本被误识别为了数字“9”。此外，图 5-6 还使用不同的颜色代表样本的数量，能够非常直观地发现各类别误分类的具体情况。

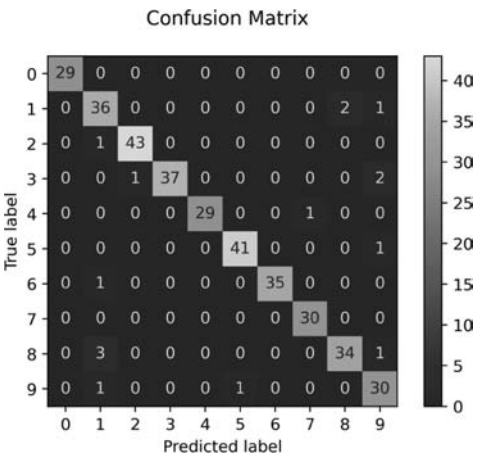

图 5-6　模型的混淆矩阵