第3章 逻辑回归

智者察同，愚者察异。 ——《黄帝内经》

Prediction is very difficult, especially about the future.（预测不是一件容易的事儿，尤其是预测未来。） ——Niels Bohr

第2章讲述了线性回归模型，它用于预测连续值，如体重、跳高高度等实数域中的变量，然而，在许多现实任务中，我们关心的目标变量是离散值，例如颜色、品种或等级。这类问题属于分类(Classification)任务，而不同于回归。

分类问题中的离散标签可以分为有序和无序两类，例如，学习成绩的“优、良、中、差”或天气预报中的风力级别具有自然的排序关系，而颜色类别(如红色、蓝色、黑色)则无法进行大小比较。此外，在现实世界中，不同类别之间的边界既可能清晰，也可能模糊，分类模型的核心任务就是学习决策边界(Decision Boundary)，以便准确地区分不同类别的样本。

本章将介绍一种由线性回归自然演化而来的分类模型——逻辑回归。作为二分类模型，逻辑回归通过将线性回归的输出转换到(0,1)区间，从而计算样本属于某个类别的概率，实现分类任务。与线性回归类似，逻辑回归的损失函数可以通过最大似然估计推导得到，因此它本质上也是一种概率模型。逻辑回归在机器学习中占据重要地位，因为它不仅是一种经典的分类方法，还可以看作单个神经元的数学模型，而神经元，正是构建大规模神经网络和深度学习模型的基石。

3.1 用 sklearn 实现逻辑回归

在学习逻辑回归模型之前，用 sklearn 提供的逻辑回归模型对一个简单的模拟数据集进行分类，其目的是了解分类的基本概念和术语，为深入理解逻辑回归模型做准备。

3.1.1 生成用于分类的模拟数据集

用 Python 语言生成一个用于分类的模拟数据集，其样本为二维空间中两类点的集合，并对样本进行可视化。虽然数据集是二维的，但很容易将其推广到 D 维空间的逻辑回归模型，代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification

#生成模拟二分类数据集,100个二维平面的样本点
X, labels = make_classification(
    n_samples=100,                          #总样本数
    n_features=2,                           #总特征数
    n_redundant=0,                          #冗余特征数
    n_informative=2,                        #有效特征数
    random_state=8,                         #随机数种子
    n_classes=2,                            #分类数
    n_clusters_per_class=1)                 #每个类别的簇数

#绘制模拟数据的散点图
unique_labels = set(labels)                 #获取标签类别
plt.figure(figsize=(8,6))
m=['ro','bs']
for k in unique_labels:
    x_k = X[labels==k]
plt.plot(x_k[:,0], x_k[:,1], m[k], markersize=8,
        markeredgecolor="w", alpha=0.6)
plt.title('A binary dataset for classification')
plt.show()
```

输出的散点图可视化结果如图 3-1 所示。图中包含两类样本点,分别用圆和方块表示。容易发现,存在一条直线将两类样本点分隔。利用 sklearn 提供的逻辑回归模型,即可得到这条直线,从而实现对未来样本的分类。

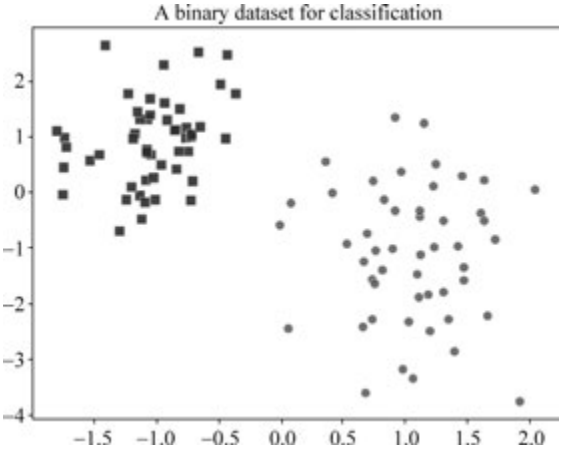

图 3-1　模拟生成的二分类数据集

3.1.2 用 sklearn 对样本进行分类

调用 sklearn 中的逻辑回归包建立逻辑回归模型，使用模拟数据集对模型进行训练和预测，代码如下：

```
from sklearn.linear_model import LogisticRegression

#默认的 solver='lbfgs'
clf = LogisticRegression(solver='lbfgs', random_state=0).fit(X, labels)
print('模型参数:', clf.intercept_, clf.coef_)

y_pred = clf.predict(X)
print('预测完全准确？ ', (y_pred == labels).all())
```

运行结果如下：

```
模型参数: [-0.36834112] [[-2.85447232  1.14461041]]
预测完全准确？ True
```

与线性回归类似，逻辑回归模型的参数也包括截距（包含在 intercept_属性中）和系数（包含在 coef_属性中）。这里用整个数据集进行训练和预测，分类结果完全准确。

最后对模型进行可视化，根据模型输出的截距和系数，可以画出一条直线，即为两种样本的决策边界，代码如下：

```
#绘制模拟数据的散点图
plt.figure(figsize=(8,6))
m=['ro','bs']
for k in unique_labels:
    x_k = X[labels==k]
    plt.plot(x_k[:,0], x_k[:,1], m[k], markeredgecolor="w", markersize=8, alpha=
0.6)
plt.title('A binary dataset for classification')

#根据模型参数绘制分隔直线
x = np.arange(-1.5, 1.5, 0.1)
y = -(clf.intercept_ + clf.coef_[0][0] *x) / clf.coef_[0][1]

#与利用最大似然估计得到的参数对应的直线进行比较
#y2 = -(-2.89082246 + (-11.00914254 )*x) / 3.72070713

plt.plot(x,y,c='red',label='sklearn')
#plt.plot(x,y2,ls='dotted',c='blue',label='Maximum likelihood')

plt.legend()
plt.show()
```

代码输出的样本和决策边界可视化如图 3-2 所示。从图中可以进一步确认，所有样本分类均正确，这是由于这个数据集是线性可分的。然而，实际数据往往是线性不可分的，此时，在无法保证分类完全准确的条件下，也可以找到一条最优分类直线。读者可以修改上述代码中 make_classification() 函数的 random_state 参数值，从而产生线性不可分的模拟数据集，然后观察分类结果。

不难发现，线性回归和逻辑回归都是在确定一条直线（或超平面），然而直线的选取目的有所不同。在线性回归中，希望直线与样本点尽量接近，而在分类问题中，希望直线与样本点尽量远离，从而得到最优的分类边界。

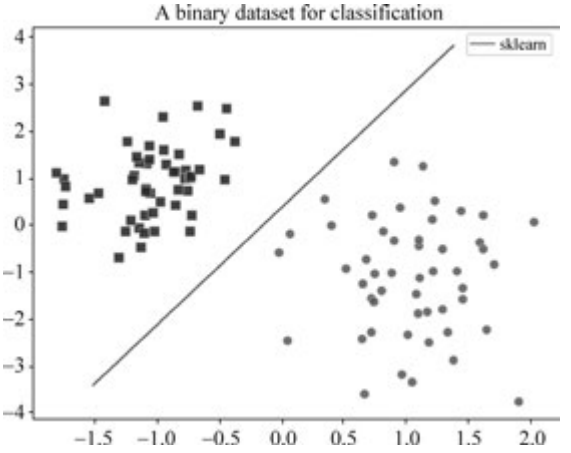

图 3-2　用 sklearn 中的逻辑回归实现二分类，得到分类决策边界

3.2　逻辑回归的模型结构和预测

3.2.1　数据集表示

逻辑回归本质上是一种分类模型，要求样本具备标签，以便进行有监督学习。首先考虑一个最基本的二分类问题，即数据集仅包含两类样本。设训练集包含 N 个数据对象，记为 $\boldsymbol{X}=\{\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_n,\cdots,\boldsymbol{x}_N\}$，其对应的标签集为 $\boldsymbol{y}=\{y_1,y_2,\cdots,y_n,\cdots,y_N\}$。

与线性回归类似，在训练集的特征矩阵 $\boldsymbol{X}$ 中，每个样本 $\boldsymbol{x}_n$ 均表示一个 D 维向量，为了方便计算，我们在每个向量前额外添加一个元素 1 作为偏置项，从而将其扩展为 $D+1$ 维向量：

$$\boldsymbol{x}_n=[1,x_{n1},x_{n2},\cdots,x_{nd},\cdots,x_{nD}]^{\mathrm{T}}$$

然而，与线性回归不同的是，逻辑回归的标签必须是离散值。对于二分类问题，我们通常约定类别的取值为 0 和 1，即 $y_n\in\{0,1\}$。这种表示方式使逻辑回归能够学习到样本属于某类别的概率，并通过概率阈值来完成分类任务。

3.2.2 模型结构和预测

1. 模型结构

根据3.1节用sklearn实现的逻辑回归，读者应能感受到逻辑回归与线性回归在形式上存在诸多相似之处，例如，两者都有截距项和系数项。实际上，逻辑回归正是基于线性回归模型改造而来的，其模型参数也为 $\boldsymbol{w}=[w_0,w_1,\cdots,w_d,\cdots,w_D]^{\mathrm{T}}$。对于输入样本 $\boldsymbol{x}=[1,x_1,x_2,\cdots,x_d,\cdots,x_D]^{\mathrm{T}}$，逻辑回归的预测值 $\hat{y}$ 是关于输入 $\boldsymbol{x}$ 和参数 $\boldsymbol{w}$ 的非线性函数，具体形式如下：

$$\hat{y}=\frac{1}{1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})} \tag{3-1}$$

从式(3-1)可以看出，逻辑回归的本质是将线性回归的输出作为Sigmoid函数的输入，从而得到一个新的输出。Sigmoid函数的数学表达式为

$$\sigma(z)=\frac{1}{1+\mathrm{e}^{-z}} \tag{3-2}$$

Sigmoid函数的输出是一个位于区间(0,1)的实数。若令 $z=wx+b$，Sigmoid函数的图像如图3-3所示。从图中可以看出，不同的参数 w 和 b 会影响曲线的位置和陡峭程度，使其呈现不同的S形，从而赋予模型更强的可训练潜力。

此外，逻辑回归的输出 $\hat{y}$ 可以被解释为输入样本 $\boldsymbol{x}$ 属于类别1的概率，而 $1-\hat{y}$ 为 $\boldsymbol{x}$ 属于类别0的概率，即

$$\begin{cases}P(y_n=1\mid\boldsymbol{x}_n)=\hat{y}_n\\P(y_n=0\mid\boldsymbol{x}_n)=1-\hat{y}_n\end{cases} \tag{3-3}$$

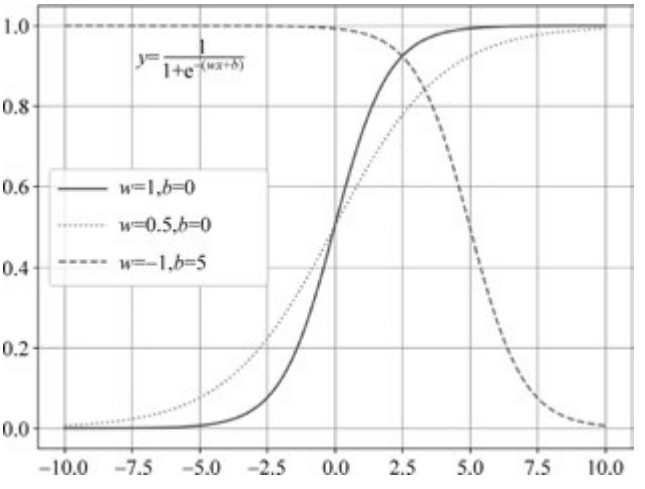

图3-3 Sigmoid函数曲线(对于不同的参数 w 和 b)

2. 模型预测

在训练阶段，给定训练数据 $\boldsymbol{X}$ 及其标签 $\boldsymbol{y}$，通过构造合适的损失函数，优化模型参数，得到模型的最优参数 $\hat{\boldsymbol{w}}$。在预测阶段，对于一个待分类样本 $\boldsymbol{x}_n$，其属于类别 1 概率为

$$\hat{y}_n = \frac{1}{1+\exp(-\hat{\boldsymbol{w}}^{\mathrm{T}}\boldsymbol{x}_n)} \tag{3-4}$$

逻辑回归巧妙地把分类问题转换为计算样本属于某类别的概率问题，并基于此概率进行决策。通常会设置一个概率阈值 δ 来确定最终的分类结果。在默认情况下，取阈值 $\delta=0.5$，即如果一个样本属于类别 1 的概率超过 0.5，则将其分为类别 1，否则分为类别 0。此时，在决策边界处，有 $\hat{y}_n=0.5$，即 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n=0$，该方程描述了逻辑回归的决策边界，实际上对应一个超平面，该超平面将数据空间划分为两类，实现样本的分类。

3.3 逻辑回归模型的损失函数

给定训练集 $\boldsymbol{X}$ 及其标签 $\boldsymbol{y}$，如何确定逻辑回归模型的参数呢？关键问题仍是确定损失函数的形式。一旦确定了损失函数，就能够通过最优化方法求得最优参数。回想在线性回归模型的求解过程中，我们采用了残差平方和作为损失函数。那么，在逻辑回归中，是否也可以采用类似的方式呢？即使用如下的损失函数：

$$L(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n-\hat{y}_n)^2 = \sum_{n=1}^{N}\left(y_n - \frac{1}{1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)}\right)^2 \tag{3-5}$$

这个损失函数看起来是合理的，因为它的目标是使样本的真实标签值 y_n 与样本的预测概率 $\hat{y}_n$ 尽可能地接近，从而约束模型的参数 $\boldsymbol{w}$。然而，可以证明，当线性模型经非线性的 Sigmoid 函数映射后，再以平方误差形式与标签计算损失时，所得损失函数可能出现多个局部极小点，因而失去凸性，导致逻辑回归的训练过程不稳定。

回想第 2 章中，我们利用最大似然估计推导了线性回归模型的损失函数，并得到了与残差平方和一致的形式，因此接下来尝试从概率的角度，利用最大似然估计求解逻辑回归模型。

3.3.1 确定似然函数

在逻辑回归模型中，对于每个样本，模型预测的是该样本属于类别 1 的概率，而不是直接预测该样本的类别是 1 还是 0。类似于线性回归，逻辑回归也属于判别式模型，认为样本特征向量 $\boldsymbol{X}=(X_1,X_2,\cdots,X_D)$ 已经给定，并以此为条件直接对样本的标签随机变量 Y 建模，即研究 $P(Y|\boldsymbol{X})$ 的分布。

令样本 $\boldsymbol{x}_n$ 对应的特征随机变量为 $\boldsymbol{X}_n$，引入 $\boldsymbol{X}_n$ 对应的标签随机变量 Y_n，其值域为 $\{0,1\}$。根据式(3-3)，可以看出每个随机变量 Y_n 服从参数为 $\hat{y}_n$ 的伯努利分布，即 $Y_n \sim \mathrm{Bern}(\hat{y}_n)$，因此 Y_n 的分布可以表示为

$$P(Y_n = y_n \mid \boldsymbol{x}_n, \boldsymbol{w}) = \hat{y}_n^{y_n}(1-\hat{y}_n)^{1-y_n} \tag{3-6}$$

对于由 N 个随机变量构成的多维随机变量 $\boldsymbol{Y}=(Y_1, Y_2, \cdots, Y_N)$，当参数 $\boldsymbol{w}$ 固定后，所有 Y_n 均相互独立(但不是同分布)，故其联合概率质量函数，也就是似然函数，可表示为

$$\begin{aligned}\mathcal{L}(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) &= P(y_1, y_2, \cdots, y_N \mid \boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N, \boldsymbol{w}) \\ &= \prod_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n, \boldsymbol{w}) \\ &= \prod_{n=1}^{N} \hat{y}_n^{y_n}(1-\hat{y}_n)^{1-y_n}\end{aligned} \tag{3-7}$$

3.3.2 确定损失函数

为了便于模型求解，通常采用负对数似然，将优化问题从最大似然估计转换为最小损失问题。整理后，负对数似然损失函数为

$$L(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) = -\log \mathcal{L}(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) = \sum_{n=1}^{N}[-y_n \log(\hat{y}_n) - (1-y_n)\log(1-\hat{y}_n)] \tag{3-8}$$

至此，我们会惊奇地发现，式(3-8)右端正是每个样本损失之和，而它与之前假定的残差平方损失 $\sum_{n=1}^{N}(y_n-\hat{y}_n)^2$ 并不相同。实际上，每个样本$(\boldsymbol{x}_n, y_n)$的损失为

$$L(\boldsymbol{w} \mid \boldsymbol{x}_n, y_n) = -y_n \log(\hat{y}_n) - (1-y_n)\log(1-\hat{y}_n) \tag{3-9}$$

式(3-9)表示随机变量 Y_n 的交叉熵(Cross-entropy)。交叉熵是信息论中的一个概念，用于衡量两个概率分布的差异程度。令 $p(x)$表示真实分布，$q(x)$表示预测分布，则 p 与 q 的交叉熵 $H(p,q)$定义为

$$H(p,q) = -\sum_i p(x_i)\log q(x_i) \tag{3-10}$$

在逻辑回归的损失函数中，样本的真实标签 y_n 和 $1-y_n$ 可视为随机变量 Y_n 的真实分布，而 $\hat{y}_n$ 和 $1-\hat{y}_n$ 可视为随机变量 Y_n 的预测分布，二者构成了如式(3-9)所示的交叉熵。综上所述，我们把式(3-8)表示的损失函数称为交叉熵损失(Cross-entropy Loss)，简称为对数损失(Log-loss)。

为比较交叉熵损失与残差平方和损失的差异，绘制了当 $y_n=1$ 时，对于不同的预测值 $\hat{y}_n$，两种损失函数的曲线对比，如图 3-4 所示。从图中可以看出，与残差平方和损失相比，交叉熵损失的值普遍较大，并且在不同位置的差异明显，而残差平方和损失的变化幅度较小。特别地，在预测接近正确时(例如 $\hat{y}_n \geqslant 0.8$)，交叉熵损失略高于残差平方损失；当预测接近错误时(例如 $\hat{y}_n \leqslant 0.2$)，交叉熵损失比残差平方损失高得多。这种现象显然是符合直觉的。

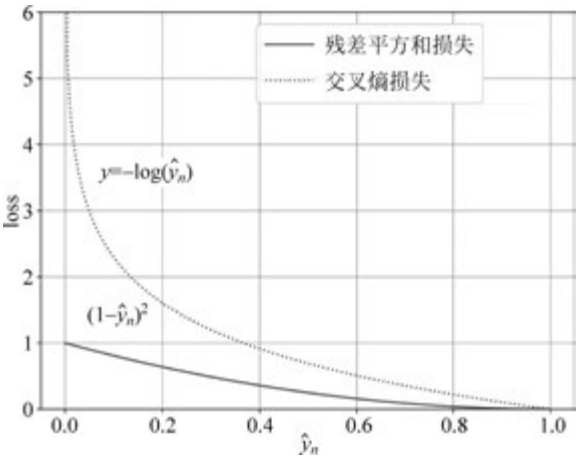

图 3-4　残差平方损失和交叉熵损失函数比较(当样本标签 $y_n=1$ 时)

3.4　逻辑回归的模型训练

在确定逻辑回归模型采用交叉熵损失函数后,模型求解即为寻找最优参数 $\boldsymbol{w}$,使式(3-8)取得最小值,然而,与线性回归不同,该优化目标函数不存在解析解,因此只能依赖梯度下降法,通过迭代的方式近似求解。下面详细介绍这一过程。

首先,需要计算损失函数的梯度。在式(3-8)中对参数 $\boldsymbol{w}$ 求导,得

$$\frac{\partial L(\boldsymbol{w}\mid\boldsymbol{X},\boldsymbol{y})}{\partial \boldsymbol{w}}=\sum_{n=1}^{N}\left(\frac{1-y_n}{1-\hat{y}_n}\frac{\partial \hat{y}_n}{\partial \boldsymbol{w}}-\frac{y_n}{\hat{y}_n}\frac{\partial \hat{y}_n}{\partial \boldsymbol{w}}\right) \tag{3-11}$$

由式(3-4),可得

$$\begin{aligned}\frac{\partial \hat{y}_n}{\partial \boldsymbol{w}}&=-\frac{1}{(1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n))^2}\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)(-\boldsymbol{x}_n)\\&=\frac{\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)}{(1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n))^2}\boldsymbol{x}_n\\&=\frac{1}{1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)}\frac{\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)}{1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)}\boldsymbol{x}_n\\&=\hat{y}_n(1-\hat{y}_n)\boldsymbol{x}_n\end{aligned}$$

将其代入公式(3-11),整理后,得

$$\frac{\partial L(\boldsymbol{w}\mid\boldsymbol{X},\boldsymbol{y})}{\partial \boldsymbol{w}}=\sum_{n=1}^{N}\boldsymbol{x}_n(\hat{y}_n-y_n)=\boldsymbol{X}^{\mathrm{T}}(\hat{\boldsymbol{y}}-\boldsymbol{y}) \tag{3-12}$$

这就是逻辑回归模型损失函数的梯度。对比式(2-15),可以发现线性回归和逻辑回归损失函数的梯度在形式上完全相同,只是计算$\hat{\boldsymbol{y}}$ 的方法不同。但这只是一种巧合,因为两种模型用于建立似然函数的概率分布完全不同(一个是高斯,另一个是 Bernoulli)。不同的似

然函数又导致两种模型采用的损失函数也完全不同(一个是残差平方和损失,另一个是交叉熵损失)。如果尝试在逻辑回归中采用残差平方和损失,则不仅会使损失函数的形式变得复杂,甚至其梯度也不再具有上述简单的形式。

得到逻辑回归模型损失函数的梯度后,可以采用与线性回归类似的方法,基于式(2-16)更新参数 $\boldsymbol{w}$。下面用 Python 实现梯度下降逻辑回归模型求解,代码如下:

```
numDims = X.shape[1]
numInstance = X.shape[0]

one = np.ones_like(labels).reshape(-1, 1)
X2= np.hstack((one, X))
y = labels
w = np.random.randn((numDims + 1))                    #用高斯分布初始化权重参数
print('w0=', w)

eta = 0.01 #learning rate
loss = []
i = 0
while i<10000:
    i+=1
    w_old = w.copy()
    y_hat = 1/(1 + np.exp(-np.dot(X2, w)))            #计算预测值

    dw = np.dot(X2.T, y_hat - y)                      #计算梯度

    w -= eta *dw                                      #更新参数

    if np.any(np.isnan(w)):
        print(i, 'isinan')
        break;

    if i%100 == 0:
        loss.append(-np.mean(y *np.log(y_hat) + (1 - y) *np.log(1 - y_hat)))
        print(i, loss[-1])

    if np.linalg.norm(w-w_old) / np.linalg.norm(w_old) <= 1e-6:
                                                      #如果参数变化足够小,则结束迭代
        break

print('w=', w)
```

运行结果如下:

```
w0=[ 1.13478349 -0.50473325  0.82110716]
100 0.02073641267354986
200 0.012501455284030906
300 0.009217830891141063
```

```
400 0.00739033040444143
500 0.006208417359716129
(中间略)
9700 0.0005015925101606459
9800 0.0004969927718618606
9900 0.0004924798487556406
10000 0.00048805126480187187
w=[ -2.89078506 -11.00891541   3.72092855]
```

迭代结束后，得到的参数为[−2.89，−11.01，3.72]（由于初始值是随机的，所以每次运行结果可能会有所不同）。这与使用 sklearn 包给出的参数[−0.37，−2.85，1.15]差别很大，然而，用上述参数代入决策边界公式 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}=0$ 并绘制分类直线，如图 3-5 所示，可以发现它们都能完美地分隔两类样本。这是因为，决策边界的方向不受参数的尺度影响，即 $\boldsymbol{w}$ 和 $k\boldsymbol{w}$ 确定的是相同的直线。也就是说，$\boldsymbol{w}$ 的取值不是唯一的。另一方面，两条直线的斜率并不完全相同，这是因为 sklearn 提供的逻辑回归默认加入了正则化处理。

此外，观察迭代过程可以发现，基于梯度下降的参数更新收敛速度很慢。存在一些显著的提高梯度下降的收敛速度的方法，例如，第 9 章将介绍贝叶斯逻辑回归和正则化技术，第 14 章将介绍动量法、Adam 等迭代优化算法。

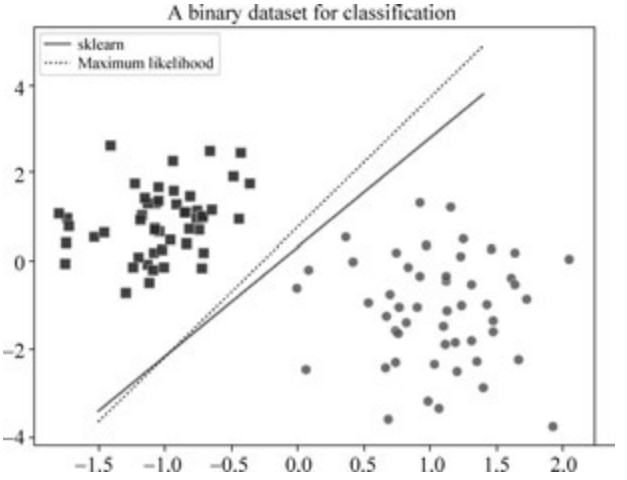

图 3-5　使用 sklearn 和最大似然估计求解逻辑回归的分类决策边界对比

3.5　多分类逻辑回归

以上，我们讨论了二分类逻辑回归模型。实际上，逻辑回归可以自然地扩展到多分类问题，这一模型被称为多分类逻辑回归（Multiclass/Multinomial Logistic Regression）。

在多分类任务中，假设样本的类别总数为 C，则模型需要为每个类别分配一套权重参数。如果输入样本的维度为 D，则所有类别的权重参数可以构成一个 $C\times D$ 的矩阵 $\boldsymbol{W}$。多

分类逻辑回归本质上相当于一个简单的单层人工神经网络，可以通过多个感知机单元并行计算，并在输出层使用 Softmax 激活函数进行归一化，因此该方法也被称为 Softmax 回归。

第 14 章将详细介绍 Softmax 回归的数学推导、模型结构及其训练方法。此外，sklearn 库提供的 LogisticRegression 类支持多分类逻辑回归，其具体使用方式可参考 3.8 节的实践案例。

3.6 线性回归模型的扩展

逻辑回归可以看作线性回归的扩展，将预测值从任意实数压缩到(0,1)区间的概率值。这种扩展具有通用性，可以根据需要将线性回归扩展到特定的应用环境，例如，多项式回归可以看作基本线性回归模型的扩展。

3.6.1 对数概率回归

我们知道，逻辑回归本质上是一种分类模型，能够预测样本属于某类别的概率。对于给定输入样本 $\boldsymbol{x}$，预测其标签随机变量 Y 属于不同类别的概率，即

$$P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x})=\frac{1}{1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})}$$

$$P(Y=0 \mid \boldsymbol{X}=\boldsymbol{x})=\frac{\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})}{1+\exp(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})}$$

经过整理变形，可得

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}=\log\frac{P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x})}{1-P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x})}=\log\frac{\hat{y}}{1-\hat{y}} \tag{3-13}$$

对比线性回归模型 $\hat{y}=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$，可以发现逻辑回归是对线性回归的输出 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ 进行了函数变换，把实数转变成了概率值。我们把这个变换称为 logit 函数，即

$$\mathrm{logit}(\hat{y})=\log\frac{\hat{y}}{1-\hat{y}} \tag{3-14}$$

其中，$\frac{\hat{y}}{1-\hat{y}}$表示两种类别出现概率的比值，称为概率(Odds)。$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ 也称为 logit，意为“对数概率”(Log-odds)，因此逻辑回归也称为“对数概率回归”。通过这个变换，巧妙地将线性回归的预测值范围由实数域 $\mathbf{R}$ 转换为概率的取值范围(0,1)，从而实现分类任务。

3.6.2 多项式回归

同理，在线性回归中通过将 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ 替换为其他函数 $f(\boldsymbol{x},\boldsymbol{w})$，也可得到新的回归模型，例如，对于一元变量 x，取

$$\hat{y}=w_0+w_1x+w_2x^2+\cdots+w_Dx^D \tag{3-15}$$

即可得到一元 D 次多项式回归(Unary Polynomial Regression of Degree D)模型。由于这

里只改变了关于输入特征 $\boldsymbol{x}_n$ 的定义，其参数求解方法与多元线性回归类似，所以只需将训练集特征矩阵由

$$\boldsymbol{X}=\begin{bmatrix}1 & x_{11} & \cdots & x_{1D}\\ 1 & x_{21} & \cdots & x_{2D}\\ \vdots & \vdots & \ddots & \vdots\\ 1 & x_{N1} & \cdots & x_{ND}\end{bmatrix}$$

改为

$$\boldsymbol{X}=\begin{bmatrix}1 & x_1 & \cdots & x_1^D\\ 1 & x_2 & \cdots & x_2^D\\ \vdots & \vdots & \ddots & \vdots\\ 1 & x_N & \cdots & x_N^D\end{bmatrix}$$

为观察多项式回归模型的效果，从身高、体重数据文件 data.txt 中读入数据，用身高预测体重，实现多项式回归模型，并对其可视化，代码如下：

```
import numpy as np

#载入数据集
data = np.loadtxt('data.txt', delimiter='\t', skiprows=1)
x = data[:,0][:,None]
y = data[:,1][:,None]

#构造多项式回归模型训练集
maxorder = 3
X = np.ones_like(x)
for i in range(1, maxorder + 1):
    X = np.hstack((X, x **i))

#用最小二乘法求解模型参数
w = np.dot(np.linalg.inv(np.dot(X.T, X)),np.dot(X.T, y))

#生成曲线拟合数据
x_test = np.linspace(150, 200, 100)[:, None]
X_test = np.ones_like(x_test)
for i in range(1, maxorder + 1):
    X_test = np.hstack((X_test, x_test **i))
f_test = np.dot(X_test, w)

from sklearn import linear_model
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

#用 sklearn 训练线性回归模型
regr = linear_model.LinearRegression()
regr.fit(data[:,0].reshape(-1, 1), data[:,1].reshape(-1, 1))
```

```
#可视化
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.figure(figsize=(8,6))
plt.scatter(data[:,0], data[:,1], c='blue', alpha=0.6)
plt.xlabel('身高 (cm)')
plt.ylabel('体重 (kg)')

#画线性回归直线
h= np.arange(150,200,0.1)
w = h *regr.coef_[0] + regr.intercept_
plt.plot(h, w, 'r-', label='Linear Regression by sklearn')
#画多项式回归曲线
plt.plot(x_test, f_test,'b-.', label='Polynomial Regression of Degree 3 by least
square')
plt.legend()

plt.show()
```

代码中使用最小二乘法实现 3 次多项式回归模型拟合身高、体重数据集，得到模型参数 $\boldsymbol{w}$。对比用 sklearn 实现的线性回归模型，结果如图 3-6 所示。由于在曲线末端缺乏足够多的样本，所以曲线的曲率和方向难以把控。

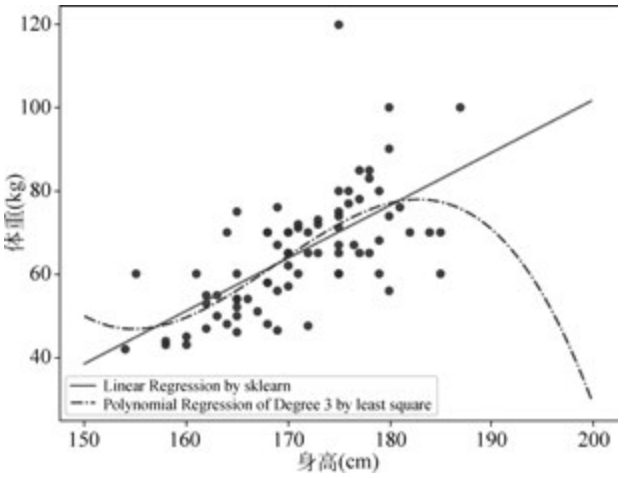

图 3-6 身高、体重 3 次多项式回归曲线

3.7 分类性能评估指标

本节介绍两种分类模型性能评估指标。对于多分类问题，一般使用准确率(Accuracy)或混淆矩阵(Confusion Matrix)；对于二分类问题，常用的评估指标包括精度(Precision)、

召回率(Recall)和 F1 分数(F1 score)等。

3.7.1 多分类性能评估指标

假定类别个数为 C,其类别标签集合为$\mathcal{D}=\{0,1,2,\cdots,C-1\}$。分类器对 N 个样本进行预测,对于第 n 个样本,其真实标签为 $y_n\in\mathcal{D}$,分类器预测标签为 $y_n^{\mathrm{p}}\in\mathcal{D}$。下面定义两种分类指标。

(1) 准确率:也称为 0/1 损失,表示分类器正确分类的样本数占所有样本数的比例,即

$$\mathrm{Accuracy}=\frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(y_n=y_n^{\mathrm{p}}) \tag{3-16}$$

其中,$\mathbb{I}(y_n=y_n^{\mathrm{p}})$表示指示函数,当条件成立时值为 1,否则为 0。

准确度对于严重不均衡数据将失去意义,例如,对于一个二分类问题,类别 0 出现的概率为 95%,类别 1 出现的概率为 5%,则分类器只需把所有样本都预测为类别 0,便可达到 95%的准确度。此时,可以采用更复杂的评估指标,如 AUC 等,这里就不再赘述了。

(2) 混淆矩阵:与准确率相比,混淆矩阵可以更加全面地反映分类器对于不同类别的预测表现。混淆矩阵是一个 $C\times C$ 的矩阵 $\boldsymbol{M}$,其第 i 行、第 j 列元素 m_{ij} 定义为

$$m_{ij}=\sum_{n=1}^{N}\mathbb{I}(y_n=i,y_n^{\mathrm{p}}=j) \tag{3-17}$$

即元素 m_{ij} 等于真实值为类别 i 的样本被预测为类别 j 的次数。显然,对角线元素为正确预测的样本数,其他位置均为预测错误的样本数。

3.7.2 二分类性能评估指标

对于二分类问题,也可以采用准确度和混淆矩阵度量指标,然而,由于二分类的普遍性和简单性,人们又提出了针对二分类的性能度量指标。

在二分类中,我们将类别 0 对应的值设置为 False,类别 1 对应的值设置为 True,对于每个样本,其真实标签和预测标签可以有以下 4 种组合:

(1) TP(True Positive)表示实际标签为 1,预测标签也为 1,即 $y_n=1,y_n^{\mathrm{p}}=1$。

(2) FP(False Positive)表示实际标签为 0,预测标签为 1,即 $y_n=0,y_n^{\mathrm{p}}=1$。

(3) TN(True Negative)表示实际标签为 0,预测标签也为 0,即 $y_n=0,y_n^{\mathrm{p}}=0$。

(4) FN(False Negative)表示实际标签为 1,预测标签为 0,即 $y_n=1,y_n^{\mathrm{p}}=0$。

由此,可以得到下列二分类度量指标(其中 TP、FP、TN、FN 表示相应样本的数量)。

(1) 精度:表示模型分类为 True 的样本数被正确分类的比例,即

$$\mathrm{Precision}=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{3-18}$$

(2) 召回率:表示实际标签为 True 的样本被模型正确找到的比例,即

$$\mathrm{Recall}=\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{3-19}$$

(3) 灵敏度：含义为阳性样本召回率，即

$$\text{Sensitivity}=\frac{\text{TP}}{\text{TP}+\text{FN}} \tag{3-20}$$

(4) 特异度：含义为阴性样本召回率，即

$$\text{Specificity}=\frac{\text{TN}}{\text{TN}+\text{FP}} \tag{3-21}$$

(5) F1 分数：显然，希望精度和召回率越高越好，然而，对于一个确定的分类器来讲，二者不可兼得。因为提高精度的代价是增加严格的条件限制，这会导致召回率降低；反之，提高召回率意味着放宽条件限制，所以精度就会降低。为了权衡二者，可采用 F1 分数指标，即

$$\text{F1 score}=2\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}} \tag{3-22}$$

对于给定测试集，可以通过调整模型参数权衡模型的精度和召回率，从而得到最佳 F1 分数。

以上 5 种二分类指标的取值范围均为 [0,1]。

3.8 逻辑回归实践：用 sklearn 实现鸢尾花分类

本节利用鸢尾花数据集实现逻辑回归分类，并计算其分类性能指标。鸢尾花数据集是一个著名的机器学习数据集，常用于各种分类模型的评估。首先，导入鸢尾花数据集，查看其描述，代码如下：

```
import numpy as np
from collections import Counter
from sklearn import datasets
iris = datasets.load_iris()
print(iris.DESCR)
```

部分运行结果如下：

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
```

```
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.
(余略)
```

可以看到，鸢尾花数据集共有 150 个样本，包括 4 个数值型特征，分别表示花萼长度(Sepal Length)、花萼宽度(Sepal Width)、花瓣长度(Petal Length)和花瓣宽度(Petal Width)，单位为厘米。鸢尾花分 3 种类别，分别为 Iris-Setosa、Iris-Versicolour、Iris-Virginica，分别用 0、1、2 表示。三类样本的数量相同，均为 50 个，所以是一个类别均衡的数据集。数据集由 R. A. Fisher 于 1988 年创建。三种鸢尾花的照片如图 3-7 所示(图片来自维基百科)。

图 3-7　鸢尾花照片，左起 Setosa(山鸢尾)、Versicolor(变色鸢尾)和 Virginica(维吉尼亚鸢尾)

然后将数据集分为训练集和测试集，并使用逻辑回归模型建立分类器，在训练集上训练分类器，并在测试集上预测每个样本，并计算其准确度，代码如下：

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

X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
print('训练集大小:', X_train.shape, y_train.shape)
print('测试集大小:', X_test.shape, y_test.shape)
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(y_pred)
print(y_test)
print('Accuracy=',np.sum(y_pred==y_test)/len(y_test))
```

运行结果如下：

```
训练集大小: (105, 4) (105,)
测试集大小: (45, 4) (45,)
[0 0 1 0 2 2 1 1 0 0 0 1 0 2 0 0 1 0 1 2 1 2 0 2 1 2 1 0 2 2 2 2 2 0 2 0 1 2 0 0 2 1 2 1 1]
[0 0 1 0 2 1 1 1 0 0 0 1 0 2 0 0 1 0 1 2 1 2 0 2 1 2 1 0 2 2 2 2 2 0 2 0 1 2 0 0 2 1 2 1 1]
Accuracy= 0.9777777777777777
```

最后，利用 sklearn 包对预测结果进行性能指标计算，包括准确度和混淆矩阵，代码如下：

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

print('Accuracy by sklearn=', accuracy_score(y_test, y_pred))
print('Confusion matrix=\n', confusion_matrix(y_test, y_pred))
```

输出的结果如下：

```
Accuracy by sklearn= 0.9777777777777777
Confusion matrix=
[[16  0  0]
 [ 0 13  1]
 [ 0  0 15]]
```

3.9 练习题

(1) 为什么逻辑回归不使用残差平方和作为损失函数？

(2) 如果逻辑回归模型将分类概率的阈值设为 0.6，试计算此时的分类决策边界。

(3) 设计基于随机梯度下降的逻辑回归求解算法，并与梯度下降方法比较参数的一致

性及收敛速度。

(4) 用不同的参数调用 make_classification()函数,用 random_state 控制随机程度,产生多个分类,每个分类有多个簇,用逻辑回归模型对其进行训练和测试,比较分类性能变化情况。

(5) 在身高、体重、跳高数据集 HWJ 中,以性别为标签,建立利用身高、体重、跳高预测性别的逻辑回归模型。建立训练集和测试集,画出性别分类决策边界,并计算模型的性能指标。如果性能不理想,则思考还需要加入什么特征才能提高性能。

(6) 人的身高随年龄变化,但变化不是线性的。通过网络搜索或实际调查获取 6～25 岁的人群身高数据(10～30 人),建立多项式回归模型用于分析和预测身高随年龄增长的规律。分析样本数量对回归性能的影响,并说明几次曲线拟合效果最佳。

(7) 已知某测试集包含 100 个样本,其中 97 个为阳性,3 个为阴性。某愚蠢分类器将所有样本均预测为阳性,试计算该分类器的准确度、精度、召回率和 F1 分数。如果都预测为阴性呢?