第3章 机器学习之分类任务

分类任务是机器学习中的常见基本任务，而逻辑回归是分类任务中的经典算法之一。本章将介绍逻辑回归及其在分类任务中的应用，包括模型的构建、训练和评估等关键环节。首先介绍广义线性回归的基本概念，阐述如何通过联系函数将线性模型扩展到更广泛的数据分布。接着深入讲解逻辑回归模型，包括 Sigmoid 函数的作用和如何将线性回归模型转换为分类器。之后讨论交叉熵损失函数的重要性，并比较其与平方损失函数的不同。最后介绍模型评估方法和多分类任务的设计和实现。同时，各小节都给出了具体的 TensorFlow 实现案例，方便读者进行编程实践。

3.1 分类任务与逻辑回归

分类任务是一项需要使用机器学习算法去学习如何根据给定示例预测其类别标签的任务。从建模的角度来看，分类需要一个训练数据集，其中包含许多可供学习的输入和输出示例。模型将会使用训练数据集并计算如何将输入数据映射到最符合的特定类别标签。二分类是指具有两个类别标签的分类任务，如将电子邮件分为“垃圾邮件”或“非垃圾邮件”。可用于二分类的常用算法包括逻辑回归、k 最近邻算法、决策树、支持向量机、朴素贝叶斯等。多类别分类是指具有两个以上类别标签的分类任务，通常使用多元概率分布模型来对多类别分类任务进行建模。可用于多类别分类的流行算法包括 k 最近邻算法、决策树、朴素贝叶斯、随机森林等，逻辑回归也可以通过一些策略方法进行多类别分类。

逻辑回归和线性回归都是常见的回归分析方法，它们都涉及“回归”这一术语，但实际上应用场景和目标却完全不同。线性回归主要用于建立一个连续变量与一个或多个自变量之间的关系模型。逻辑回归主要用于建立一个二分类变量与一个或多个自变量之间的关系模型。线性回归的因变量是连续的，可以取任意实数值。逻辑回归的因变量是二分类的，只能取 0 和 1 两个值。线性回归通过最小二乘法求解参数使得预测值与实际值的残差平方和最小化。逻辑回归则通过最大似然估计法估计参数，使得模型预测概率尽可能接近实际情况。线性回归输出的是连续的数值，可以直接解释为因变量的预测值。逻辑回归输出的是二分类的概率值，需要设置一个阈值作为分类的判断标准。下面从广义线性回归过渡到逻辑回归。

3.1.1 广义线性回归

在线性回归问题中，将自变量和因变量之间的关系用线性模型来表示，从而能够根据已知的样本数据对未知的数据进行估计。这种线性关系在二维空间中是一条直线，在三维空间中是一个平面，在高维空间中是一个超平面。线性模型只能够应用于自变量和因变量是线性或者接近线性的情况。在现实生活中，数据之间存在着大量非线性的关系。为了解决这类问题，就需要对线性模型进行改进。

预测房屋价格的例子中也可以假设 x 和 $\ln y$ 之间是线性关系，就可以得到对数线性回归的函数，可以写成这种形式：

$$y = \mathrm{e}^{wx+b} \tag{3-1}$$

此时，x 和 y 之间是非线性关系，也可以表示为下面的形式：

$$\begin{aligned} g(y) &= wx + b \\ y &= g^{-1}(wx + b) \end{aligned} \tag{3-2}$$

这样的模型称为广义线性模型，这个函数 g()称为联系函数，联系函数可以是任何一个单调可微函数，使用不同的联系函数就可以描述多种不同分布的数据。还可以把广义线性回归推广到高维模型：

$$Y = g^{-1}(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{X}) \tag{3-3}$$

其中，$\boldsymbol{W}=(w_0, w_1, \cdots, w_m)^{\mathrm{T}}$ 和 $\boldsymbol{X}=(x^0, x^1, \cdots, x^m)^{\mathrm{T}}$ 都是 $m+1$ 维的向量，m 是属性的个数，$x^0=1$。线性模型可以通过广义线性回归产生丰富的变化，使它能够描述更加复杂的数据关系，满足实际任务中对非线性关系的需求。


视频讲解

3.1.2 逻辑回归实现二分类

常见分类任务，如图片分类、垃圾短信识别、异常判断等，需要一个分类器自动对输入的数据进行分类，分类器的输入是样本的特征，输出是离散值，表示输入样本属于哪个类别。例如，手写数字识别，手写数字的图片会以向量的形式提供给分类器，经过分类器的计算之后，输出 0～9 共 10 个离散的值。

与回归任务一样，首先需要收集一些有分类标记的训练样本集，然后用这个训练样本集去训练分类器，训练好之后这个分类器就能够接收新的没有标签的样本，并对它做出分类判断。只要对线性回归模型稍加改造就可以实现分类器，例如，预测商品房属于普通住宅还是高档住宅，分别用 0 和 1 来表示二分类。假设房价 100 万以上的属于高档住宅，低于 100 万的属于普通住宅，那么只要在线性回归预测出的房价 $z=wx+b$ 基础上，再增加一个单位阶跃函数就可以判断房屋类型。

$$y = \mathrm{step}(z) = \begin{cases} 0, & z - 1000000 < 0 \\ 1, & z - 1000000 \geqslant 0 \end{cases} \tag{3-4}$$

这就是广义线性回归，这个阶跃函数 step(z)就是联系函数的逆函数，通过它实现了对商品房的二分类，把线性回归模型转换为分类器。阶跃函数存在两个问题：一是不光滑，如果有一套 99.99 万的房子和一套 100.01 万的房子，一刀切地划分成普通住宅和高档住宅，这个结果太过于简单粗暴；二是不连续，函数值存在从 0 到 1 的突变，这在数学计算中会带来很多的不便，在这一点上无法求导数。因此，阶跃函数并不是一个合格的单调可微的联系

函数。

实际上，常用的对数概率函数 $y=\dfrac{1}{1+e^{-z}}$ 是一个更好的选择，如图 3-1 所示，它既能够把线性模型的结果映射到 0 和 1，实现分类，并且是连续光滑、单调上升的，并且任意阶可导，具有很好的数学性质。

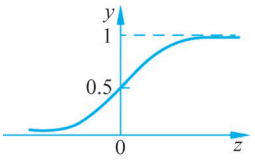


图 3-1　对数概率函数

对数概率函数的形状近似于 s，这类外形的函数称为 Sigmoid 函数。

$$\sigma(z)=\frac{1}{1+e^{-z}}=\frac{1}{1+e^{-(wx+b)}} \tag{3-5}$$

一般情况下，Sigmoid 函数就是指对数概率函数。推广到多元模型中，它的表达式为

$$y=\frac{1}{1+e^{-(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{X})}} \tag{3-6}$$

其中 $\boldsymbol{W}=(w_0,w_1,\cdots,w_m)^{\mathrm{T}}$，$\boldsymbol{X}=(x^0,x^1,\cdots,x^m)^{\mathrm{T}}$，$x^0=1$。

对数概率回归也称作逻辑回归。逻辑回归使用线性回归的结果作为对数概率函数的自变量，它的名字是回归，但是实现的是一个分类器。它不仅可以预测类别，而且还可以预测出输入样本属于某个类别的概率，这对于很多需要利用概率来辅助决策的任务来说非常有用。例如，在商品房评估时，可以输出房屋属于高档住宅的概率，当房价是 99.9 万或者 100.1 万时，属于高档住宅的概率都在 50%左右，也就是说属于高档住宅和普通住宅的可能性差不多，没有明显的差别。进行分类时，我们可以把这个概率值转换为类别输出。例如，将阈值设置为 0.5，当概率值大于 0.5 时，就是高档住宅；当概率值小于 0.5 时，就是普通住宅。这就是用逻辑回归实现二分类的例子。

3.1.3　交叉熵损失函数


视频讲解

交叉熵损失函数(Cross-Entropy Loss)是机器学习和深度学习中常用的一种损失函数，特别适用于分类问题。交叉熵来源于信息论，用来衡量两个概率分布之间的差异。在机器学习中，交叉熵损失函数用于评估模型的预测概率分布和真实分布之间的差异，能够衡量模型预测的准确性。在分类问题中，交叉熵损失函数将模型的输出概率分布与真实标签的概率分布进行比较，通过最小化交叉熵损失来优化模型参数，从而提高模型的预测准确性。此外，交叉熵损失函数常与 softmax 函数配合使用，softmax 函数可以将模型的输出转换为概率分布形式，从而与交叉熵损失函数相结合进行模型训练。下面详细介绍交叉熵损失函数的计算。

逻辑回归在线性模型之上再增加一个 Sigmoid 函数，把线性模型的输出映射到 0～1 范围，输出一个概率值，并根据这个概率值实现分类。为了衡量模型，需要使用损失函数。在线性回归模型中，通常使用平方损失函数，可以写出逻辑回归的平方损失函数：

$$\mathrm{Loss}=\frac{1}{2}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2=\frac{1}{2}\sum_{i=1}^{n}(y_i-\sigma(wx_i+b))^2 \tag{3-7}$$

采用梯度下降法来更新 w 和 b 时，需要计算损失函数对 w 和 b 的偏导数，需要对 Sigmoid 函数求导数，Sigmoid 函数在大部分的时候导数非常接近于 0，损失函数对 w 和 b 的偏导数非常小，导致迭代更新 w 和 b 时，步长非常小，更新非常缓慢。另外，在线性回归

中，平方损失函数是一个凸函数，只有一个极小值点。但是在逻辑回归中，它的平方损失函数是一个复杂的非凸函数，有多个局部极小值点，使用梯度下降法有可能会陷入局部极小值中。为了解决这些问题，在逻辑回归中通常采用交叉熵损失函数来代替平方损失函数，交叉熵损失函数的表达式为

$$\text{Loss}=-\sum_{i=1}^{n}[y_i\ln\hat{y}_i+(1-y_i)\ln(1-\hat{y}_i)] \tag{3-8}$$

其中，y_i 是第 i 个样本的标签，$\hat{y}_i=\sigma(wx_i+b)$ 是 Sigmoid 函数的输出，是第 i 个样本的预测概率。这个 Loss 函数是所有样本的交叉熵，除以样本总数 n 就可以得到平均交叉熵损失函数。

交叉熵损失函数对模型参数 w 和 b 的偏导数：

$$\begin{aligned}\frac{\partial\text{Loss}}{\partial w}&=\frac{1}{n}\sum_{i=1}^{n}x_i(\hat{y}_i-y_i)\\ \frac{\partial\text{Loss}}{\partial b}&=\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i-y)\end{aligned} \tag{3-9}$$

可以看到不需要对 Sigmoid 函数求导数，因此它能够有效地克服平方损失函数应用于逻辑回归时更新模型参数过慢的问题。这个偏导数的值只受到预测值和真实值之间的误差的影响，当误差比较大时，偏导数也比较大，模型参数的更新速度就比较快。当误差比较小的时候，更新速度也就比较慢，这也符合训练模型的要求。除此之外，交叉熵损失函数是凸函数，因此使用梯度下降法得到的极小值就是全局最小值。

假设某个逻辑回归任务的训练集有 4 个样本，每个样本的标签值列在表 3-1 中。假设模型 A 对最后一个样本的预测概率是 0.45，那么这个样本就会被归为第 0 类，出现分类错误，现在这个模型的准确率是 75%。假设还有一个模型 B，他对这个训练集的预测结果也只出现了一个错误，这个模型的准确率也是 75%。比较这两个模型，模型 A 对于前 3 个样本的判断非常准确，对于样本 4 虽然判断错误，但是也没有错得太离谱。模型 B 虽然把前 3 个样本都预测正确了，但是正好在阈值附近、比较悬，而对于样本 4 则错得非常离谱。显然在这两个模型中，虽然分类准确率都是 75%，但是模型 A 的性能更好，可见仅仅通过准确率是无法细分出模型的优劣的，需要计算它们的交叉熵损失。

表 3-1 模型对比数据

模型 A				模型 B			
样本	标记	预测值	结果判断	样本	标记	预测值	结果判断
样本 1	0	0.2	正确	样本 1	0	0.44	正确
样本 2	0	0.3	正确	样本 2	0	0.46	正确
样本 3	1	0.9	正确	样本 3	1	0.58	正确
样本 4	1	0.45	错误	样本 4	1	0.2	错误

这是第一个样本的损失：

$$\text{样本 1}：-(0\times\ln0.2+1\times\ln0.8)=-\ln0.8=0.2231\ldots$$

采用同样的方法可以计算出模型 A 所有样本的损失：

$$-(0\times\ln0.2+1\times\ln0.8)=-\ln0.8=0.2231\ldots$$

$$-(0\times\ln0.3+1\times\ln0.7)=-\ln0.7=0.3566\ldots$$

$$-(1\times\ln 0.9+0\times\ln 0.1)=-\ln 0.9=0.1053\ldots$$

$$-(1\times\ln 0.45+0\times\ln 0.55)=-\ln 0.45=0.7985\ldots$$

得到模型 A 的平均交叉熵损失为 0.3708。

采用同样的方法可以计算出模型 B 所有样本的损失：

$$-(0\times\ln 0.44+1\times\ln 0.56)=-\ln 0.56=0.5798\ldots$$

$$-(0\times\ln 0.46+1\times\ln 0.54)=-\ln 0.54=0.6161\ldots$$

$$-(1\times\ln 0.58+0\times\ln 0.42)=-\ln 0.58=0.5447\ldots$$

$$-(1\times\ln 0.2+0\times\ln 0.8)=-\ln 0.2=1.6094\ldots$$

得到模型 B 的平均交叉熵损失为 0.8375。模型 A 的损失明显低于模型 B，可见交叉熵损失函数能够很好地反映概率之间的误差，是训练分类器时的重要依据。

3.1.4　TensorFlow 实现一元逻辑回归

下面使用一元逻辑回归实现对商品房的分类，假设一组商品房类型的数据集列在表 3-2 中，包括房屋面积和对应的类型，0 代表普通住宅，1 代表高档住宅。下面使用房屋面积和对应类别来训练模型。

表 3-2　商品房屋类型

序　号	面积/平方米	类　型	序　号	面积/平方米	类　型
1	80.00	0	9	60.00	0
2	120.00	1	10	110.00	1
3	150.00	1	11	180.00	1
4	200.00	1	12	165.00	1
5	75.00	0	13	140.00	1
6	90.00	0	14	95.00	0
7	130.00	1	15	70.00	0
8	140.00	1	16	125.00	1

第一步，导入需要的库，加载数据集，房屋类别只有 0 和 1 两个值。

```
import tensorflow as tf
import numpy as tf
import matplotlib.pyplot as plt

x = np.array([80.00, 120.00, 150.00, 200.00, 75.00, 90.00, 130.00, 140.00, 60.00, 110.00,
180.00, 165.00, 140.00, 95.00, 70.00, 125.00])
y = np.array([0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1])
```

第二步，数据处理。因为房屋面积都是比较大的正数，而 Sigmoid 函数是以 0 点为中心的，因此需要对这些点进行中心化，每个样本点都减去它们的平均值，这样整个数据集的均值就等于 0，相当于这些点被整体平移了，但是相对位置不变。

```
x_train = xnp,mean(x)
y_train = y
```

第三步，设置超参数和显示间隔。

```
learn_rate = 0.005
iter = 5
display_step = 1
```

第四步，设置模型变量的初始值。

```
np.randon.seed(612)
w = tf.Variable(np.random.randn())
b = tf.Variable(np.random.randn())
```

第五步，训练模型的代码如下。

```
cross_train = []
acc_train = []
for i in range(0,iter + 1):

    with tf.GradientTape() as tape:
        pred_train = 1/(1 + tf.exp( - (w * x_train + b)))
        Loss_train = - tf.reduce_mean(y_train * tf.math.log(pred_train) + (1 - y_train) *
tf.math.log(1 - pred_train))
        Accuracy_train = tf.reduce_mean(tf.cast(tf.equal(tf.where(pred_train < 0.5, 0, 1),
y_train), tf.float32))

    cross_train.append(Loss_train)
    acc_train.append(Accuracy_train)

dL_dw,dL_db = tape.gradient(Loss_train,[w,b])

    w.assign_sub(learn_rate * dL_dw)
    b.assign_sub(learn_rate * dL_db)

if i % display_step == 0:
        print("i: % i,TrainLoss: % f,Accuracy: % f" % (i,Loss_train,Accuracy_train))
```

其中，列表 cross_train 用来存放训练集的交叉熵损失，acc_train 用来存放训练集的分类准确率。

在实现逻辑回归时，需要使用 TensorFlow 计算 Sigmoid 函数、准确率、交叉熵损失函数。在 TensorFlow 中，使用 exp( )函数来实现 e 指数，所以 Sigmoid 函数的代码实现就是：

```
1/(1 + tf.exp( - (w * x_train + b)))
```

实现交叉熵损失函数的代码块是：

```
- tf.reduce_mean(y_train * tf.math.log(pred_train) + (1 - y_train)) * tf.math.log(1 - pred_
train)
```

其中，y_train 是样本标签，pred_train 是预测概率，使用 math.log( )函数实现以 e 为底的对数运算，y_train 和 pred_train 都是一维数组，1-y_train 和 1-pred_train 做广播运算，结果也是一维数组。对每一个样本的交叉熵损失求和，执行 reduce_mean( ) 函数就可以得到

所有样本的平均交叉熵损失，按照 Loss 函数的定义，前面还有一个负号。

准确率是正确分类的样本数除以样本总数，实现准确率统计的代码块是：

```
tf.reduce_mean(tf.cast(tf.equal(tf.where(pred_train<0.5,0,1),y_train),tf.float32))
```

其中，预测值 pred_train 是通过 Sigmoid 函数得到的是一个概率值，通过 where( )函数转化为类别 0 或 1，判断的阈值为 0.5。然后使用 equal( )函数逐元素的比较预测值和标签值，得到的结果是一个一维张量，再使用 cast( )函数把这个结果转换为整数，然后使用 reduce_mean( )函数对所有元素求平均值，就可以得到正确样本在所有样本中的比例。

使用 append( )方法将新元素添加到已创建的列表中，记录每一次迭代的平均交叉熵损失和准确率。通过 tape.gradient( )获得损失函数对 w 和 b 的偏导数。最后使用 assign_sub()方法更新模型参数，打印输出训练过程中的损失和准确率，运行结果如图 3-2 所示。

```
i: 0,Train Loss: 6.682626, Accuracy: 0.062500
i: 8,Train Loss: 0.381193, Accuracy: 0.750000
i: 16,Train Loss: 0.337738, Accuracy: 0.812500
i: 24,Train Loss: 0.324547, Accuracy: 0.812500
i: 32,Train Loss: 0.318235, Accuracy: 0.812500
i: 40,Train Loss: 0.314597, Accuracy: 0.812500
```

图 3-2　运行结果

可以看到损失一直在下降，准确率在提高并稳定在 0.8125。下面就可以使用这个模型对新的商品房进行分类了，我们用以下面积的房屋做测试。

```
x_test = [87.34, 120.56, 65.78, 103.21, 98.45, 56.12, 145.67, 73.89, 129.01, 112.48]
```

根据面积计算预测概率，这里使用训练数据的平均值对测试数据进行中心化处理：

```
pred_test = 1/(1 + tf.exp( - (w * (x_test - np.mean(x)) + b)))
```

然后根据概率进行分类：

```
y_test = tf.where(pred_test < 0.5,0,1)
```

打印测试面积、预测的概率和分类：

```
for i in range(len(x_test)):
    print(x_test[i],"\t",pred_test[i].numpy(),"\t",y_test[i].numpy(),"\t")
```

运行结果如图 3-3 所示。

```
87.34    0.008614448    0
120.56   0.3286283      0
65.78    0.000634536    0
103.21   0.056262016    0
98.45    0.032375015    0
56.12    0.00019658318  0
145.67   0.91155076     1
73.89    0.0016959267   0
129.01   0.5771335      1
112.48   0.1551314      0
```

图 3-3　运行结果

可以把分类的结果可视化输出。首先根据分类结果绘制散点图，然后绘制预测概率曲线，代码如下：

```
plt.scatter(x_test,y_test)

x_ = np.array(range( - 80,80))
y_ = 1/(1 + tf.exp( - (w * x_ + b)))
plt.plot(x_ + np.mean(x),y_)
plt.show()
```

得到的绘制图像如图 3-4 所示，通过散点图可以更加直观地看到商品房类型和面积之间的关系，几套被划分为高档住宅，几套被划分为普通住宅。

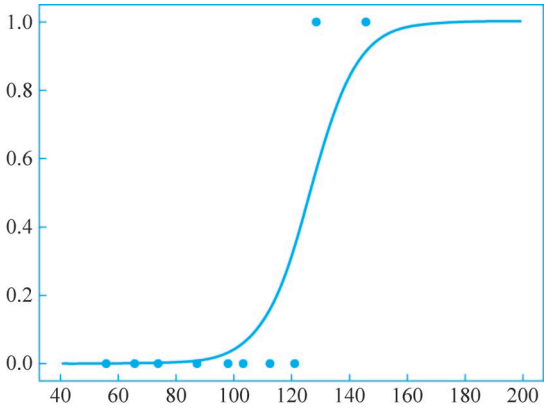


图 3-4 散点图

3.1.5 TensorFlow 实现多元逻辑回归

下面使用逻辑回归实现对鸢尾花的分类。鸢尾花训练集有 120 条样本，测试集有 30 条样本，每条样本有 4 个属性：花萼长度、花萼宽度、花瓣长度、花瓣宽度，鸢尾花分为 3 个类别：山鸢尾、变色鸢尾、弗吉尼亚鸢尾。我们用逻辑回归做二分类，因此选取山鸢尾和变色鸢尾两种类型的样本，使用花萼长度和花萼宽度这两个属性。

第一步，导入需要的库，加载数据集。

使用 keras 中的 get_file( )函数加载数据集，使用 pandas 读取 CSV 文件，结果是一个 pandas 二维数据表。

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1],TRAIN_URL)

df_iris = pd.read_csv(train_path, header=0)
```

第二步，处理数据。首先把 pandas 二维数据表转换为 numpy 数组：

```
iris=np.array(df_iris)
```

训练集中有 120 条样本，每条样本有 5 列数据，前 4 列是属性值，第 5 列是标签值。只取出前两列属性：花萼长度和花萼宽度来训练模型，取出第 5 列标签值。

```
train_x=iris[:,0:2]
train_y=iris[:,4]
```

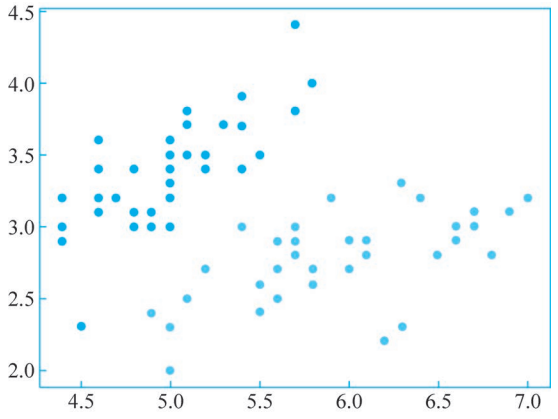
选取山鸢尾和变色鸢尾两种类型的样本，也就是提取出标签值为 0 和 1 的样本。

```
x_train = train_x[train_y<2]
```

```
y_train = train_y[train_y < 2]
```

为了样本数据可视化，对提取出的样本和属性绘制散点图（如图 3-5 所示）。使用花萼长度和花萼宽度作为样本点的横坐标和纵坐标，根据样本点的标签值确定散点的颜色，蓝色点是山鸢尾，红色点是山鸢尾和变色鸢尾。

```
cm_pt = mpl.colors.ListedColormap(["blue","red"])
plt.scatter(x_train[:,0],x_train[:,1],c = y_train,cmap = cm_pt)
plt.show()
```



彩色图片

图 3-5　散点图

可以看到这两个属性的尺度相同，因此不用进行归一化，可以直接对它们进行中心化处理。对每个属性中心化，也就是要按列数据中心化。结果如图 3-6 所示。

```
x_train = x_train - np.mean(x_train,axis = 0)
plt.scatter(x_train[:,0],x_train[:,1],c = y_train,cmap = cm_pt)
plt.show()
```

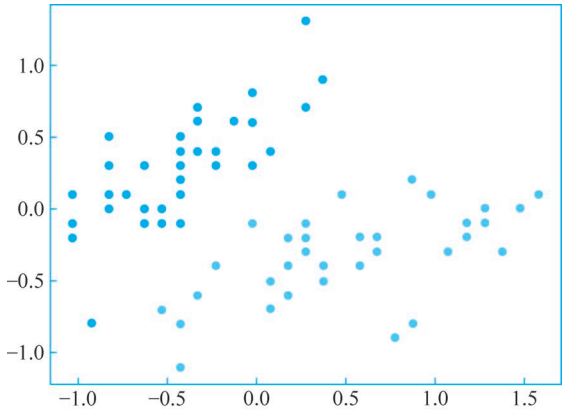


图 3-6　中心化散点图

可以看到中心化之后样本点被整体平移，样本点的横坐标和纵坐标的均值都是 0。下面生成多元逻辑回归模型需要的属性矩阵 X 和标签列向量 Y，这和多元线性回归是完全一

样的。

```
num = len(x_train)
x0_train = np.ones(num).reshape(-1,1)
X = tf.cast(tf.concat((x0_train,x_train),axis = 1),tf.float32)
Y = tf.cast(y_train.reshape(-1,1),tf.float32)
```

第三步，设置超参数和显示间隔。

```
learn_rate = 0.2
iter = 120

display_step = 30
```

第四步，设置模型参数的初始值。W 是一个列向量，用 3 行 1 列的二维数组来表示。

```
np.random.seed(612)
W = tf.Variable(np.random.randn(3,1),dtype = tf.float32)
```

第五步，训练模型。

```
ce = []
acc = []

for i in range(0,iter + 1):
    with tf.GradientTape() as tape:
        PRED = 1/(1 + tf.exp(-tf.matmul(X,W)))
        Loss = -tf.reduce_mean(Y * tf.math.log(PRED) + (1 - Y) * tf.math.log(1 - PRED))

    accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.where(PRED.numpy()< 0.5,0.,1.),Y),tf.
float32))
    ce.append(Loss)
    acc.append(accuracy)

    dL_dW = tape.gradient(Loss,W)
    W.assign_sub(learn_rate * dL_dW)

    if i % display_step == 0:
        print("i: %i, ACC: %f, Loss: %f" % (i,accuracy,Loss))
```

列表 ce 用来保存每一次迭代的交叉熵损失，列表 acc 用来保存准确率，多元模型的 Sigmoid 函数要用到属性矩阵 X 和参数向量 W 相乘，其结果是一个列向量，因此计算得到的 PRED 也是一个列向量，是每个样本的预测概率。多元模型交叉熵损失的结果也是一个列向量，包括每个样本的损失，使用 reduce_mean( )函数求它们的平均值得到平均交叉熵损失。准确率 accuracy 不需要求导，所以把它放在 with 语句的外面。使用 append( )方法记录每一次迭代的平均交叉熵损失和准确率。通过 tape.gradient( )获得损失函数对 W 的偏导数，最后使用 assign_sub()方法更新模型参数，打印输出训练过程中的损失和准确率，运行结果如图 3-7 所示。

```
i: 0, ACC:0.230769, Loss: 0.994269
i: 30, ACC:0.961538, Loss: 0.481892
i: 60, ACC:0.987179, Loss: 0.319128
i: 90, ACC:0.987179, Loss: 0.246626
i: 120, ACC:1.000000, Loss: 0.204982
```

图 3-7 运行结果

可以看到一开始的准确率很低，随着迭代次数的增加，准确率不断提高，最后达到了 100%，同时交叉熵损失在不断

下降。

第六步，可视化绘制交叉熵损失和准确率的变化曲线图（见图 3-8）。

```
plt.figure(figsize = (5,3))
plt.plot(ce,color = "blue",label = "Loss")
plt.plot(acc,color = "red",label = "acc")
plt.legend()
plt.show()
```

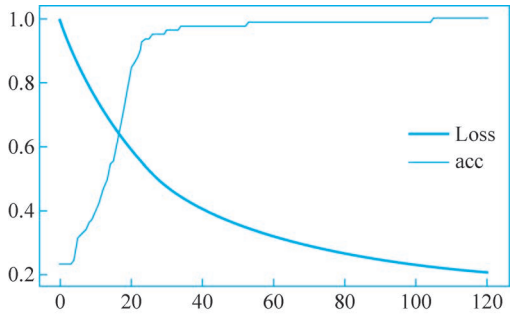


图 3-8　变化曲线图

3.2　模型评估

鸢尾花数据集被划分为训练集和测试集，训练集用来训练模型的样本，测试集用来评价这个模型的性能。模型预测输出和样本真实标签之间的差异称为误差，训练集上的误差称为训练误差，在新样本上的误差称为泛化误差。

在前面的程序中，计算的损失都是在训练集上的损失，因此都是训练误差。在训练模型时，如果一味地追求使训练误差尽量达到最小，就有可能会出现一种现象，在训练集上表现得很好的模型，在新样本上的泛化误差却很大，这种现象就叫作过拟合。训练样本的数量有限，如果模型的学习能力很强，就会发生过度学习，把训练样本中某些特性学习为所有样本都必须具备的普遍特征，导致泛化能力降低。例如，某个猫狗识别模型，如果提供的训练样本中，猫的朝向都是头在图像左边，过度学习就会学习到朝向特征，认为所有的猫都是头在图像左边，必须具备这样的特征才是猫。如果给模型一张猫头在图像右边的图，它就识别不出来，因此在训练集上的正确率接近 100%，但是当出现新的、没有见过的样本时，却出现很多错误。与过拟合相对的是欠拟合，产生欠拟合的原因是模型的学习能力低，没有学习到样本中的通用特征。

机器学习的目标不仅是让模型的训练误差小，更是希望泛化误差小。但是新的样本是无限多的，实际上无法得到真正的泛化误差。为了评价学习算法在新的数据上的效果，一般把数据样本集划分为训练集和测试集。首先使用训练集训练模型，训练完成之后再在测试集上运行模型，测试模型对新样本的预测或者判断能力，使用这个测试集上的测试误差来作为泛化误差的近似。通过模型在测试集上的表现来评价模型的性能。为了得到泛化性能强的模型，测试集中的样本最好没有在训练集中出现过，也就是说测试集应该尽可能和训练集是互斥的，同时要求测试集和训练集是独立同分布的，也就是说它们有着相同的均值和方

差，这样的测试才有意义。为了方便，在使用公共数据集时，最好尽量使用划分好的训练集和测试集。如果需要自己划分的话，也要注意这个独立同分布的约束。

在机器学习和模式识别等领域中，一般需要将样本分成独立同分布的 3 部分：训练集、验证集和测试集。其中训练集用来估计模型，验证集用来确定网络结构或者控制模型复杂程度的参数，而测试集则检验最终选择最优模型的性能。在训练模式时，一般需要在训练集中再分出一部分作为验证集，用于评估模型的训练效果和调整模型的超参数。验证集获得的评估结果不是模型的最终效果，而是基于当前数据的调优结果，如果不需要调整超参数，则可以不用验证集。通俗地讲，训练集等同于学习知识，验证集等同于课后测验检测学习效果并且查漏补缺，测试集是最终考试评估这个模型到底怎样。对于小规模样本集（几万量级），常用的分配比例是 60%训练集、20%验证集、20%测试集。对于大规模样本集（百万级以上），只要验证集和测试集的数量足够即可（例如 1 万条数据）。超参数越少，或者超参数很容易调整，那么可以减少验证集的比例，更多的分配给训练集。

公共数据集一般由研究机构或大型公司创建和维护，可以直接从这些机构提供的链接中下载。由于它们的格式各不相同，需要针对不同的数据集去编写代码解析和读取它们。下面在训练模型的同时，使用测试集来评价模型的性能。

第一步，导入需要的库，加载数据集。

使用 keras 中的 get_file( )函数加载数据集。

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1],TRAIN_URL)

TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"
test_path = tf.keras.utils.get_file(TEST_URL.split('/')[-1],TEST_URL)
```

第二步，处理数据。使用 pandas 分别读取训练集和测试集 CSV 文件，产生 pandas 二维数据表，然后把 pandas 二维数据表转换为 numpy 数组：

```
df_iris_train = pd.read_csv(train_path, header=0)
df_iris_test = pd.read_csv(test_path, header=0)

iris_train=np.array(df_iris_train)
iris_test=np.array(df_iris_test)
```

可以查看一下训练集和测试集的形状：

```
print(iris_train.shape,iris_test.shape)
```

输出：(120, 5) (30, 5)。

可以看到训练集有 120 条样本，测试集有 30 条样本.

分别取出训练集的前两列属性（花萼长度和花萼宽度）和标签值，取出测试集的前两列

属性(花萼长度和花萼宽度)和标签值:

```
train_x = iris_train[:,0:2]
train_y = iris_train[:,4]

test_x = iris_test[:,0:2]
test_y = iris_test[:,4]
```

从训练集和测试集中取出标签值为 0 和 1 的样本,也就是山鸢尾和变色鸢尾两种类型的样本。分别记录训练集和测试集中的样本数,并绘制它们的散点图(如图 3-9 所示),代码如下:

```
x_train = train_x[train_y < 2]
y_train = train_y[train_y < 2]

x_test = test_x[test_y < 2]
y_test = test_y[test_y < 2]

num_train = len(x_train)
num_test = len(x_test)

plt.figure(figsize = (10,3))
cm_pt = mpl.colors.ListedColormap(["blue","red"])

plt. subplot(121)
plt.scatter(x_train[:,0],x_train[:,1],c = y_train, cmap = cm_pt)

plt.subplot(122)
plt.scatter(x_test[:,0],x_test[:,1],c = y_test,cmap = cm_pt)

plt. show()
```

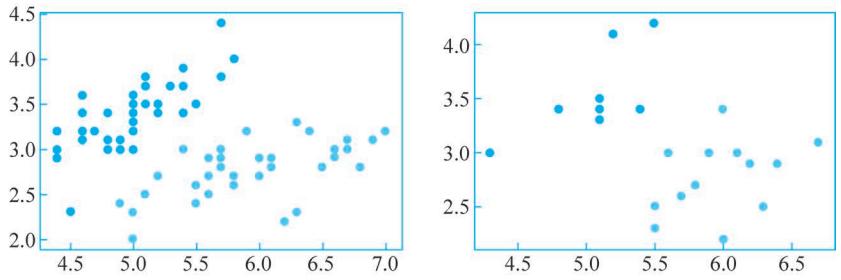


图 3-9　散点图

从图 3-9 可以看到在训练集和测试集中,这两类鸢尾花都能够被很好地区分开。

下面分别对训练集和测试集中的数据按列做中心化,并绘制中心化之后的散点图(如图 3-10 所示)。

```
x_train = x_train - np.mean(x_train,axis = 0)
x_test = x_test - np.mean(x_test,axis = 0)
```

```
plt.figure(figsize = (10,3))

plt.subplot(121)
plt.scatter(x_train[:,0],x_train[:,1],c = y_train,cmap = cm_pt)

plt.subplot(122)
plt.scatter(x_test[:,0],x_test[:,1],c = y_test,cmap = cm_pt)

plt.show()
```

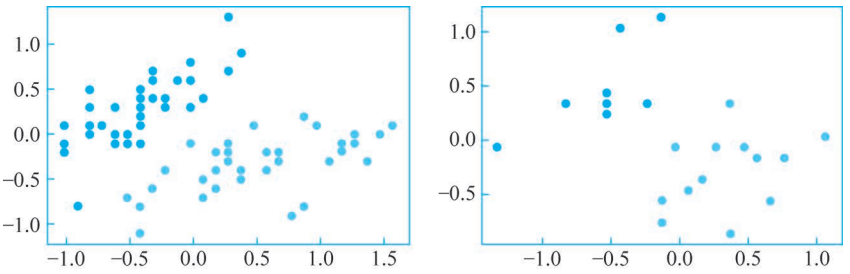


图 3-10　中心化散点图

从图 3-10 可以看到，训练集和测试集中的数据都被中心化了，全体样本点被整体平移，样本点的横坐标和纵坐标的均值接近 0。在机器学习中，要求训练集和测试集是独立同分布的，也就是说它们具有相同的均值和方差，在样本数量有限的情况下，可能无法做到完全相等，可以看到两者虽然有偏差，但是非常接近，不会影响模型的训练和测试效果。

下面构造多元逻辑回归模型需要的属性矩阵 X 和标签列向量 Y，包括训练集和测试集。

```
x0_train = np.ones(num_train).reshape(-1,1)
X_train = tf.cast(tf.concat((x0_train,x_train), axis = 1),dtype = tf.float32)
Y_train = tf.cast(y_train.reshape(-1,1),dtype = tf.float32)

x0_test = np.ones(num_test).reshape(-1,1)
X_test = tf.cast(tf.concat((x0_test,x_test), axis = 1),dtype = tf.float32)
Y_test = tf.cast(y_test.reshape(-1,1),dtype = tf.float32)
```

第三步，设置超参数，设置模型参数的初始值。

```
learn_rate = 0.2
iter = 120
display_step = 10

np.random.seed(600)
W = tf.Variable(np.random.randn(3,1),dtype = tf.float32)
```

第四步，训练模型。

```
ce_train = []
ce_test = []
acc_train = []
acc_test = []
```

```
for i in range(0, iter + 1):
    with tf.GradientTape() as tape:
        PRED_train = 1/(1 + tf.exp( - tf.matmul(X_train, W)))

Loss_train = - tf.reduce_mean(Y_train * tf.math.log(PRED_train) + (1 - Y_train) * tf.math.log
(1 - PRED_train))
        PRED_test = 1/(1 + tf.exp( - tf.matmul(X_test,W)))

Loss_test = - tf.reduce_mean(Y_test * tf.math.log(PRED_test) + (1 - Y_test) * tf.math.log(1 -
PRED_test))
   accuracy_train = tf.reduce_mean(tf.cast(tf.equal(tf.where(PRED_train.numy() < 0.5, 0., 1.), Y_
train),tf.float32))
    accuracy_test = tf.reduce_mean(tf.cast(tf.equal(tf.where(PRED_test.numpy() < 0.5, 0., 1.), Y_
test), tf.float32))

    ce_train.append(Loss_train)
    ce_test.append(Loss_test)
    acc_train.append(accuracy_train)
    acc_test.append(accuracy_test)

    dL_dW = tape.gradient(Loss_train, W)
    W.assign_sub(learn_rate * dL_dW)

    if i % display_step == 0:
        print("i: % i, TrainAcc:% f, TrainLoss: % f ,TestAcc:% f, Testloss: % f" % (i,
accuracy_train,Loss_train,accuracy_test,Loss_test))
```

计算每一次迭代后训练集和测试集的损失和准确率，并在列表 ce_train、ce_test、acc_train、acc_test 中记录它们。注意：只使用训练集来更新模型参数。运行的结果如图 3-11 所示。

```
i: 0, TrainAcc:0.602564, TrainLoss: 0.658168 ,TestAcc:0.590909, Testloss: 0.613218
i: 10, TrainAcc:0.833333, TrainLoss: 0.519591 ,TestAcc:0.863636, Testloss: 0.510719
i: 20, TrainAcc:0.961538, TrainLoss: 0.428297 ,TestAcc:0.954545, Testloss: 0.442076
i: 30, TrainAcc:0.987179, TrainLoss: 0.366116 ,TestAcc:0.863636, Testloss: 0.394263
i: 40, TrainAcc:0.987179, TrainLoss: 0.321825 ,TestAcc:0.863636, Testloss: 0.359277
i: 50, TrainAcc:0.987179, TrainLoss: 0.288864 ,TestAcc:0.863636, Testloss: 0.332476
i: 60, TrainAcc:0.987179, TrainLoss: 0.263381 ,TestAcc:0.863636, Testloss: 0.311151
i: 70, TrainAcc:0.987179, TrainLoss: 0.243044 ,TestAcc:0.863636, Testloss: 0.293665
i: 80, TrainAcc:0.987179, TrainLoss: 0.226385 ,TestAcc:0.863636, Testloss: 0.278984
i: 90, TrainAcc:0.987179, TrainLoss: 0.212446 ,TestAcc:0.863636, Testloss: 0.266426
i: 100, TrainAcc:1.000000, TrainLoss: 0.200574 ,TestAcc:0.863636, Testloss: 0.255522
i: 110, TrainAcc:1.000000, TrainLoss: 0.190315 ,TestAcc:0.863636, Testloss: 0.245940
i: 120, TrainAcc:1.000000, TrainLoss: 0.181341 ,TestAcc:0.863636, Testloss: 0.237433
```

图 3-11　运行结果

可以看到，虽然训练集的准确率达到了 100%，但是测试集的准确率只有 86%。训练集和测试集的损失仍然在持续下降，可以继续迭代训练这个模型，使得测试集的准确率进一步提高。

分别绘制训练集和测试集的损失曲线与准确率曲线（如图 3-12 所示）：

```
plt.figure(figsize = (10,3))

plt.subplot(121)
plt.plot(ce_train,color = "blue",label = "train")
```

```
plt.plot(ce_test,color = "red",label = "test")
plt.ylabel("Loss")
plt.legend()

plt.subplot(122)
plt.plot(acc_train,color = "blue",label = "train")
plt.plot(acc_test,color = "red",label = "test")
plt.ylabel("Accuracy")

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

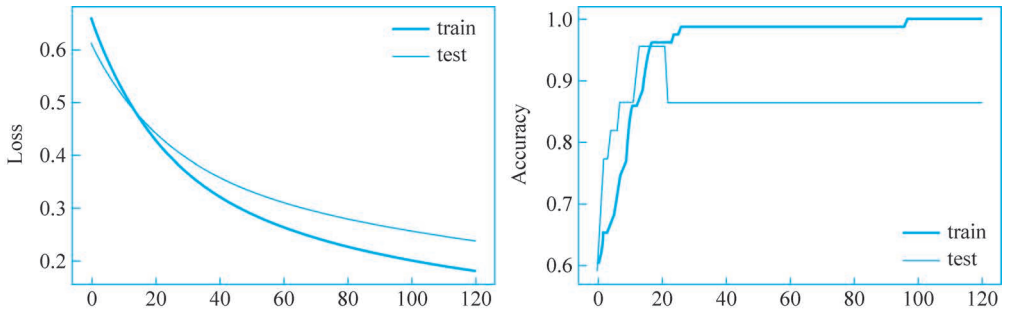


图 3-12　训练集和测试集的损失曲线与准确率曲线图

从图 3-12 可以看到训练集的损失下降更快，分类准确率也更高。至此，我们得到了一个能够区分山鸢尾和变色鸢尾两种类型的分类器，使用花萼长度和花萼宽度这两个属性（或特征）来完成对鸢尾花的分类。

3.3　多分类任务

逻辑回归能够很好地解决二分类的问题，但是现实中遇到更多任务是需要实现多分类问题的。例如，图像识别、车牌识别、人脸识别、水果分类、动物识别等，都需要把输入的样本划分为多个类别。对于多分类任务来说，主要有两种处理方法：直接作为多分类任务和转化为二分类任务。直接作为多分类任务时，使用一个模型同时处理所有类别，模型的输出通常是一个包含所有类别的预测概率分布，从中选择最高概率的类别作为预测结果。另一种处理方法是把多分类任务转化为二分类任务，有两种策略：一对多策略与一对一策略。在进行分类任务时，通常需要进行数据预处理、特征工程、模型选择和评估等步骤。选择合适的算法和调优参数对分类任务的表现至关重要。

下面还是以鸢尾花数据集为例，实现多分类的任务。在鸢尾花数据集中，山鸢尾、变色鸢尾、弗吉尼亚鸢尾分别被标记为 0、1、2，这种编码方式称为自然顺序码。使用自然顺序码运算时出现的问题是，山鸢尾和弗吉尼亚鸢尾的平均值就是变色鸢尾，而且它们之间的距离也不同，2 到 0 的距离要远一些，1 到 0 的距离要近一些。为了避免自然顺序码的数值大小可能在机器学习过程中造成偏差，可以采用独热编码来表示鸢尾花的类别，每种类别用一个一维向量来表示，其中的 3 个元素分别对应 3 个类别，(1,0,0)表示山鸢尾，(0,1,0)表示变色鸢尾，(0,0,1)表示弗吉尼亚鸢尾。显然，采用独热编码需要占用更多的空间，但是它能够

更加合理地表示数据之间的关系，而且 3 个种类对应三维空间中的点到原点的距离都是相等的。

在多分类问题中，当样本的标签被表示成独热编码的形式时，模型的输出也被表示为向量的形式，其中的每个元素是样本属于每个类别的概率。如图 3-13 所示，模型输出表明样本分别属于各种类别的概率，根据概率看出这个样本属于第三类，与标签(0,0,1)一致。可以采用 softmax 回归来实现多分类问题，例如输入鸢尾花的花瓣长度和花瓣宽度，经过线性运算后，再使用 softmax( )函数作为激活函数就可以得到这个样本的预测概率，即 $Y=\text{softmax}(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{X})$，其中是输入的属性矩阵 $\boldsymbol{X}$，$\boldsymbol{W}$ 是权值参数矩阵。softmax( )函数是对数概率函数在多分类问题上的推广，也是一种广义线性回归，用来完成分类任务。

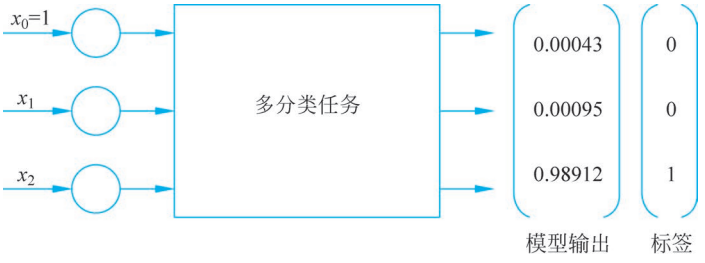


图 3-13　多分类任务模型

多分类任务是对二分类任务的扩展，多分类交叉熵损失函数可以定义为

$$\text{Loss}=-\sum_{i=1}^{n}\sum_{p=1}^{C}y_{i,p}\ln(\hat{y}_{i,p}) \tag{3-10}$$

其中，n 为样本总数；C 为类别总数，当 $C=2$ 时，就是二元交叉熵损失函数；$y_{i,p}$ 是第 i 个样本属于第 p 类的标签值；$\hat{y}_{i,p}$ 是 softmax( )函数输出的预测概率。通过交叉熵损失函数可以计算预测概率和实际分类标签的误差。在使用 softmax( )函数实现的多分类模型中，每个样本只能属于一个类别，这称为互斥的多分类问题。

下面还是以鸢尾花分类为例，使用 TensorFlow 编程实现多分类任务。

第一步，导入需要的库，加载数据。使用 keras 中的 get_file( )函数下载数据集，使用 pandas 读取 CSV 文件，结果是一个 pandas 二维数据表。

```
import tensorflow as tf
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1],TRAIN_URL)
df_iris_train = pd.read_csv(train_path, header=0)
```

第二步，处理数据。首先把 pandas 二维数据表转换为 numpy 数组：

```
iris_train=np.array(df_iris_train)
```

训练集中有 120 条样本，每条样本有 5 列数据，前 4 列是属性值，第 5 列是标签值。只

取出后两列属性：花瓣长度和花瓣宽度作为鸢尾花分类的依据，取出数据集第 5 列标签值，也就是鸢尾花的类别，并记录样本总数，以便之后求损失的平均值。

```
x_train = iris_train[:,2:4]
y_train = iris_train[:,4]
num_train = len(x_train)
```

下面构造多元逻辑回归模型需要的属性矩阵 X 和标签列向量 Y，使用 one_hot()函数将标签值转换为独热编码的形式：

```
x0_train = np.ones(num_train).reshape(-1,1)
X_train = tf.cast(tf.concat([x0_train,x_train],axis = 1),tf.float32)
Y_train = tf.one_hot(tf.constant(y_train,dtype = tf.int32),3)
```

第三步，设置超参数，设置模型参数初始值。

```
learn_rate = 0.2
iter = 500
display_step = 100

np.random.seed(612)
W = tf.Variable(np.random.randn(3,3),dtype = tf.float32)
```

第四步，训练模型。

```
acc = []
cce = []

for i in range(0,iter + 1):
    with tf.GradientTape() as tape:

        PRED_train = tf.nn.softmax(tf.matmul(X_train,W))
        Loss_train = -tf.reduce_sum(Y_train * tf.math.log(PRED_train))/num_train

    accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_train.numpy(), axis = 1),
y_train), tf.float32))

    acc.append(accuracy)
    cce.append(Loss_train)

    dL_dW = tape.gradient(Loss_train, W)
    W.assign_sub(learn_rate * dL_dW)
    if i % display_step == 0:
        print("i: %i, Acc: %f, Loss: %f" % (i, accuracy, Loss_train))
```

列表 acc 用来保存每一次迭代的准确率，列表 cce 用来保存每一次迭代的交叉熵损失。实现预测概率的代码为

```
PRED_train = tf.nn.softmax(tf.matmul(X_train,W))
```

这里直接使用了 TensorFlow 的神经网络模块提供的 tf.nn.softmax()函数，得到预测

值的分类概率，代码中的 X_train 是属性矩阵，W 是模型参数矩阵。

实现交叉熵损失函数的代码为

```
Loss_train = - tf.reduce_sum(Y_train * tf.math. log(PRED_train))/num_train
```

首先根据公式计算每个样本的交叉熵，然后通过 reduce_sum()函数对矩阵中的所有值求和，得到所有样本的交叉熵损失之和，再除以样本数 num_train 得到平均交叉熵损失。注意，按照定义，交叉熵损失的表达式前面有一个负号。

统计准确率的代码为

```
accuracy = tf. reduce_mean(tf. cast (tf. equal (tf. argmax(PRED_train. numpy (), axis = 1), y_
train), tf. float32))
```

使用 TensorFlow 中的 argmax()函数得到一个数组中最大元素的索引，这里设置 axis＝1 表示对每一行元素求最大，得到的结果会把样本的预测值转换成自然顺序码的形式，然后使用 equal()函数逐元素地比较预测值和标签值，结果是一个布尔类型的一维张量，之后使用 cast()函数把布尔值转化为数值 0 和 1，最后用 reduce_mean()函数对数组中的所有元素求平均值，就可以得到预测准确率。准确率 accuracy 不需要求导，所以把它放在 with 语句的外面。使用 append( )函数记录每一次迭代的平均交叉熵损失和准确率。通过 tape. gradient( ) 获得损失函数对 W 的偏导数，最后使用 assign_sub()函数更新模型参数，打印输出训练过程中的准确率和平均交叉熵损失，运行的结果如图 3-14 所示。

```
i:0, Acc:0.350000, Loss:4.510763
i:100, Acc:0.808333, Loss:0.503537
i:200, Acc:0.883333, Loss:0.402912
i:300, Acc:0.891667, Loss:0.352650
i:400, Acc:0.941667, Loss:0.319779
i:500, Acc:0.941667, Loss:0.295599
```

图 3-14　运行结果

可以看到一开始的准确率很低，随着迭代次数的增加，准确率不断提高，同时损失也在不断下降。

本章小结

本章介绍了广义线性回归、逻辑回归、交叉熵损失函数、模型评估方法和多分类任务的设计和实现。同时给出了具体的 TensorFlow 实现案例，方便读者进行编程实践。