哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： 逻辑回归

学号：1190200208
姓名：李旻翀

一、实验目的

理解逻辑回归模型，掌握逻辑回归模型的参数估计算法。

二、实验要求及实验环境

1. 实验要求

实现两种损失函数的参数估计（1，无惩罚项；2.加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等。

验证：

1. 可以手工生成两个分别类别数据（可以用高斯分布），验证你的算法。考察类条件分布不满足朴素贝叶斯假设，会得到什么样的结果。
2. 逻辑回归有广泛的用处，例如广告预测。可以到 UCI 网站上，找一实际数据加以测试。

2. 实验环境

VS code 2021 + Python + numpy + matplotlib

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 算法原理

分类器做分类问题的实质是预测一个已知样本的位置标签，即 $P(Y=1|X)$，根据该概率值来判断样本的类别属性。按照朴素贝叶斯的方法，可以用贝叶斯概率公式将其转化为类条件概率(似然)和类概率的乘积。本实验是直接求该概率[1]。

考虑最基本的 0/1 二分类模型 $f:X\to Y$，其中 X 为实数向量 $X=<X_1,...,X_n>$，$Y\in\{0,1\}$，且根据朴素贝叶斯的假设，有所有 X_i 在给定 Y 的情况下条件独立。另外，有 $P(X_i|Y=y_k)$~$N(\mu_{ik},\sigma_i)$, $P(Y)$~$B(\pi)$ 成立。则基于以上这些假设，并将 $\pi=P(Y=1)$, $1-\pi=P(Y=0)$ 代入，用朴素贝叶斯将其展开，我们有：

$$
\begin{aligned}
P(Y=1|X) &= \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1)+P(Y=0)P(X|Y=0)} \\
&= \frac{1}{1+\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}} \\
&= \frac{1}{1+exp(ln\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})} \\
&= \frac{1}{1+exp(ln\frac{1-\pi}{\pi}+\sum_i ln\frac{P(X_i|Y=0)}{P(X_i|Y=1)})}
\end{aligned}
$$

由于各个维度的条件概率分布符合高斯分布，所以我们可以将各个维度的高斯分布函数$P(X_i|Y=y_k)=\frac{1}{\sigma_i\sqrt{2\pi}}\exp\left(\frac{-(x-\mu_{ik})^2}{2{\sigma_i}^2}\right)$代入：

$$
\begin{aligned}
&= \frac{1}{1+exp(ln(\frac{1-\pi}{\pi})+\sum_i ln(P(X_i|Y=0)-P(X_i|Y=1)))} \\
&= \frac{1}{1+exp(ln(\frac{1-\pi}{\pi})+\sum_i(-ln\sigma_i\sqrt{2\pi}-\frac{(x_i-\mu_{i0})^2}{2\sigma_i^2}-(-ln\sigma_i\sqrt{2\pi}-\frac{(x_i-\mu_{i1})^2}{2\sigma_i^2})))} \\
&= \frac{1}{1+exp(ln(\frac{1-\pi}{\pi})+\sum_i(\frac{(x_i-\mu_{i1})^2}{2\sigma_i^2}-\frac{(x_i-\mu_{i0})^2}{2\sigma_i^2}))} \\
&= \frac{1}{1+exp(ln(\frac{1-\pi}{\pi})+\sum_i(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}x_i+\frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}))}
\end{aligned}
$$

令

$$
w_0=\ln\frac{1-\pi}{\pi}+\sum_i\frac{{\mu_{i1}}^2-{\mu_{i0}}^2}{2{\sigma_i}^2},\ w_i=\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2},\ X=\begin{bmatrix}1\\X_1\\\vdots\\X_n\end{bmatrix},\ w=\begin{bmatrix}w_0\\w_1\\\vdots\\w_n\end{bmatrix}\quad，\quad 则
$$

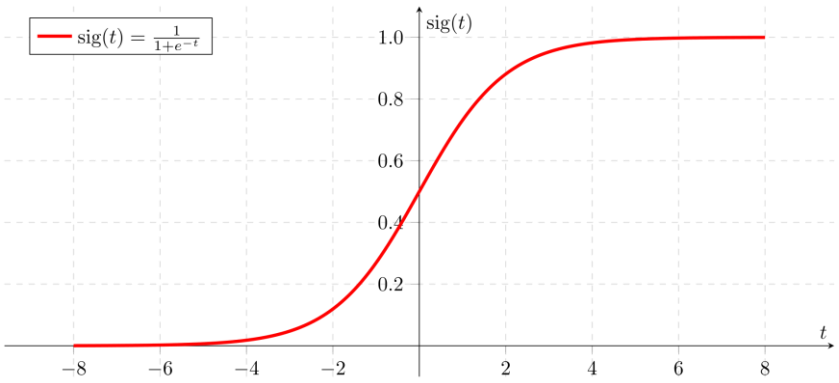
有：

$$
P(Y=1|X)=\frac{1}{1+exp(w_0+\sigma_{i=1}^n w_iX_i)}
$$

转化为矩阵形式：

$$
P(Y=1|X)=\frac{1}{1+\exp(w^TX)}
$$

形如$y=\frac{1}{1+e^{-z}}$的函数被称作$sigmoid$函数，它具有将实值映射到 0 到 1 之间的某个值的特性。观察其函数图像可知，$sigmoid(0)=0.5$，且从$x=0$出发，无论是向右或向左，y 值都以很快的速度向 1 和 0 逼近。因此可以看作这个函数将我们的线性模型预测的连续值映射到 0 到 1 的概率上，从而得到 0/1 标签离散值[2]。



对于 Logistic 回归问题，参数即为w。设损失函数为$l(w)$，要计算$l(w)$，通过极大似然估计（MLE）难以解决（因为需要计算$P(<X,Y>|w)$），我们可以将其转化为计算极大似然条件估计（MCLE）的问题[3]，在这种情况下，只需要计算$P(X|Y,w)$，我们有：

$$P(Y=0|X,W)=\frac{1}{1+exp(w_0+\sum_i w_iX_i)}$$

$$P(Y=1|X,W)=\frac{exp(w_0+\sum_i w_iX_i)}{1+exp(w_0+\sum_i w_iX_i)}$$

进而：

$$\begin{aligned}l(W)&=\sum_l Y^l lnP(Y^l=1|X^l,W)+(1-Y^l)lnP(Y^l=0|X^l,W)\\&=\sum_l Y^l ln\frac{P(Y^l=1|X,W)}{P(Y^l=0|X^l,W)}+lnP(Y^l=0|X^l,W)\\&=\sum_l Y^l(w_0+\sum_i^n w_iX_i^l)-ln(1+exp(w_0+\sum_i^n w_iX_i^l))\end{aligned}$$

其中n为样本维度，在二维情况下为 2，之后将其记为d。l代表当前样本点组数。共$\{<X_1^1,X_2^1,...,X_d^1>,<X_1^2,X_2^2,...,X_d^2>,\cdots,<X_1^n,X_2^n,...,X_d^n>\}$组样本点。$X_d^n$代表第$n$组样本点的第$d$个坐标。

用梯度下降法求得$w = argmax_w\ l(w)$，而我们常常将 $-l(w)$ 作为损失函数，等价于求其最小值。

$$\frac{\partial l(w)}{\partial w_i} = \sum_l X_i^l (Y^l - \frac{1}{1 + exp(-(w_0 + \sum_i^n w_i X_i^l))})$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \frac{1}{1 + exp(-(w_0 + \sum_i^n w_i X_i^l))})$$

为控制过拟合问题，我们还可以加入正则项。在 Logistic 回归分类问题中，决策面是线性的，有无数个线性决策面可以有效地区分出两类数据，而我们希望所选取的决策面，可以使“最短距离”最大化，以达到最好的分类效果。这里的最短距离，指的是两类中距离线性决策面最近的距离之和。

记线性决策面方程为$w^T x + b = 0$，设常数c，有$(w^T x_i + b) y_i > c$。其中，对于$\forall x_i \in$第一类，$y_i = 1$，$\forall x_i \in$第二类，$y_i = -1$。

由于$w^T x$是点到线性决策面的法向量的投影，那么记x_i^*, y_j^*分别是类别内距离决策面的最近点，记m为x_i^*, y_j^*到单位长的$\vec{w}$的投影值的差值，则有：

$$m = \frac{w^T}{||w||}(x_i^* - x_j^*) = \frac{2c}{||w||}$$

从而我们的优化问题，变成了在约束条件$y_i(w^T x_i + b) \geqslant 1, \forall i$下，求$\min w^T w$。从而加上正则项的梯度下降为：

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - \frac{1}{1 + exp(-(w_0 + \sum_i^n w_i X_i^l))})$$

2. 步骤讲解

1) 生成数据

若要生成类条件分布满足朴素贝叶斯假设的数据，那么就对每一个类别的每一个维度都用一个独立的高斯分布生成。若要生成类条件分布不满足朴素贝叶斯假设的数据，那么就对每一个类别的两个维度用一个二维高斯分布生成。

需要注意的是，由于高斯分布具有的特性，多维高斯分布不相关可以推出独立性，因此，可以用二维高斯分布生成数据，如果是满足朴素贝叶斯假设的，那么协方差矩阵的非对角线元素均为 0，如果是不满足朴素贝叶斯假设的，那么协方差矩阵的非对角线元素不为 0（协方差矩阵应该是对称阵）

2) 梯度下降法

按照前述算法设计梯度下降法的具体实现，通过三个函数构成：

gradient_descent()函数是梯度下降法的主函数，通过调节参数可以实现

带惩罚项与不带惩罚项的迭代。
gradient()函数按照算法计算梯度。
loss()函数计算损失函数
以下是梯度下降部分的核心函数：

```
def loss(X, train_y, w):
    """损失函数"""
    loss = 0
    size = float(train_y.shape[0])
    for i in range(0, X.shape[1]):
        wXT = w @ X[:, i]
        loss += (train_y[i] * wXT / size - np.log(1 + np.exp(wXT)) / size)
    return loss
```

```
def gradient(X, train_y, w):
    """计算梯度"""
    grad = np.zeros((1, X.shape[0]))
    for i in range(0, X.shape[1]):
        wXT = w @ X[:, i]
        grad += X[:, i].T * (train_y[i] - sigmoid(wXT))
    return grad
```

```
def gradient_descent(train_x, train_y):
    """梯度下降法"""
    eta_temp = eta
    w = np.ones((1, train_x.shape[1] + 1))  # (1,3)
    X = genX(train_x)
    grad = gradient(X, train_y, w)
    loss_list = []
    loss0 = 0
    loss1 = loss(X, train_y, w)
    count = 0
    while count < epoch:  # 误差值，小于该值时停止迭代
        count += 1
        w = w - eta_temp * lamda * w + eta_temp * grad
        loss0 = loss1
        loss1 = loss(X, train_y, w)
        if(np.abs(loss1) > np.abs(loss0)):  # loss不降反增，则减半学习率
            eta_temp *= 0.5
        grad = gradient(X, train_y, w)
        loss_list.append(np.abs(loss0))
        print(count)
    return w, count, loss_list
```

3) 对 UCI 数据集进行分类实验并做可视化展示

我们选取 UCI 的 Skin Segmentation Data Set，其中包含 20w+三维分类指标与对应的标签[4]，我们将自己的模型应用于这组数据进行分类实验，抽取 80%

的数据作为训练集，20%的数据作为测试集。

具体而言，我们通过 skin_read 函数读入数据集，通过 skin_plot_show 函数对结果进行可视化展示，skin_show 函数则作为主函数。

由于样本数量太大难以训练，所以我们按步长为 30 抽取样本进行训练，训练的详细结果见第四点：实验结果与分析。

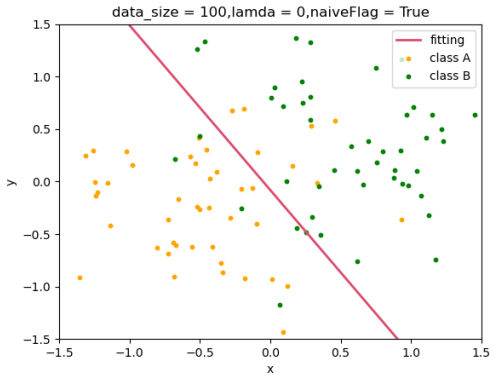
四、实验结果与分析

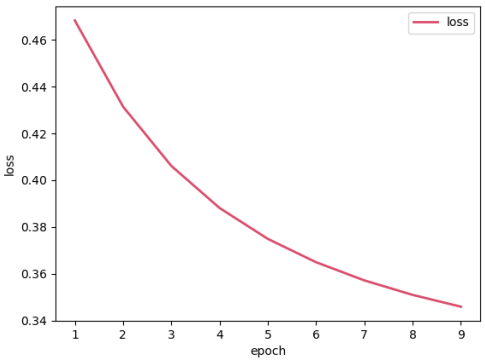
1. 根据自己生成的数据，研究正则项的影响

设置生成的数据方差为 0.3，两类的均值分别为[-0.7, -0.3]与[0.7, 0.3]，学习率为 1e-2,超参数 λ 为 1e-3，训练次数 10。设置 naiveFlag = True，即满足贝叶斯假设。

1） 不带正则项，小样本

设置 lamda = 0，样本点个数为 100，训练次数 10，结果如下：

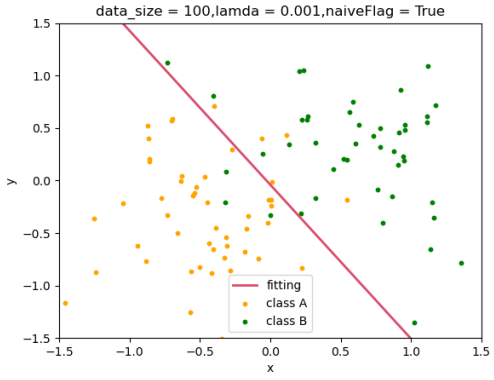


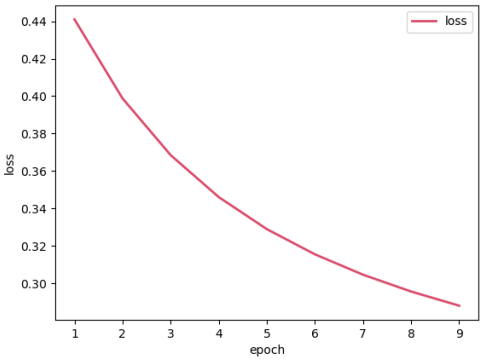



```
acc:0.88
```

2） 带正则项，小样本

设置 lamda = 1e-3，样本点个数为 100，训练次数 10，结果如下：

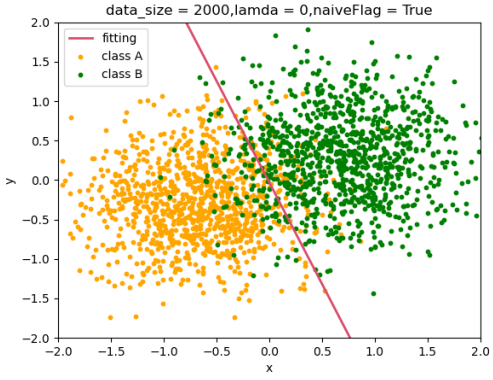


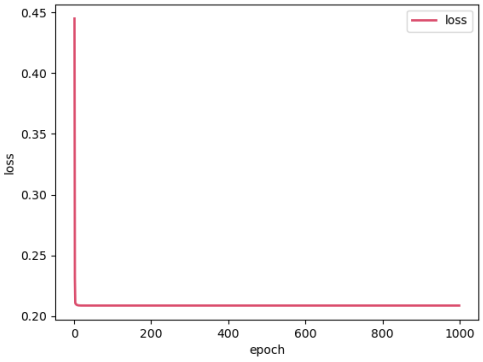



```
acc:0.95
```

3） 不带正则项，大样本

设置 lamda = 0，样本点个数为 2000，训练次数 1000，结果如下：

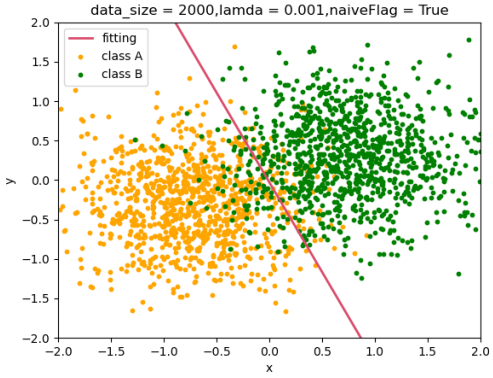


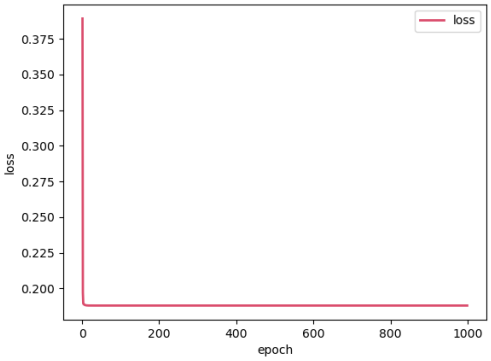


acc:0.9105

4) 带正则项，大样本

设置 lamda = 1e-3，样本点个数为 2000，训练次数 1000，结果如下：





acc:0.9195

综上所述，对于我们自己生成的数据，控制变量研究正则项的影响，我们可以得出如下结论：

① 在大样本和小样本的情况下，加上正则项计算出的 loss 值都比不加正则项要低，因此加上正则项可以得到更为准确的结果；

② 在小样本的情况下，不加正则项容易产生过拟合的问题，从而使得准确率较低，加上正则项可以有效缓解这个问题；

③ 逻辑回归可以很好地解决线性分类问题，而且收敛速度较快，在自己生成的数据下往往能很快便使 loss 降低到一个可以接受的值，从而早早迭代得到结果。

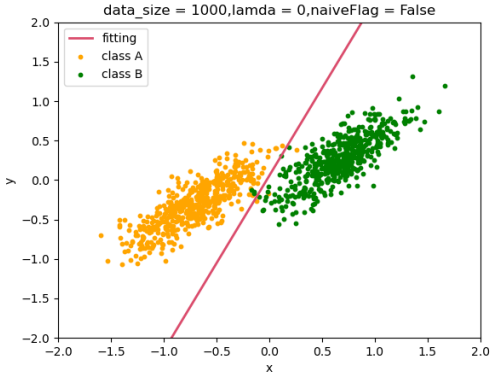
④ 在样本数量较多的情况下，加不加惩罚项都能得到比较好的分类结果，即在数据量较大时，正则项对结果的影响不大

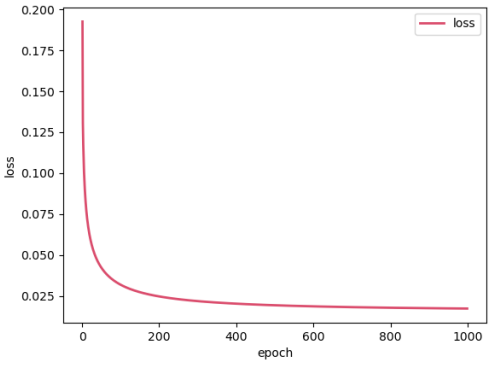
2. 根据自己生成的数据，研究是否满足朴素贝叶斯假设的影响

设置生成的数据方差为 0.1，两类的均值分别为[-0.7, -0.3]与[0.7, 0.3]，学习率为 1e-2,超参数 λ 为 1e-3，样本数 1000，训练次数 1000。若不满足朴素贝叶斯假设，则协方差矩阵的值为 0.08。

1) 满足朴素贝叶斯假设

设置 naiveFlag = True，训练结果如下：

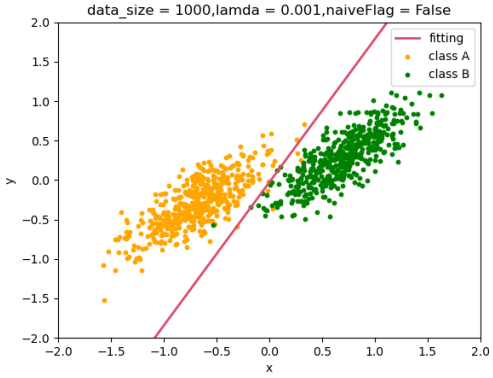


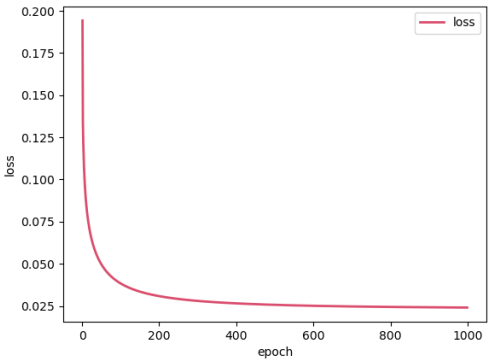



```
acc:0.993
```

2） 不满足朴素贝叶斯假设

设置 naiveFlag = False，训练结果如下：






```
acc:0.991
```

我们可以得出如下结论：

① 可以明显发现：不满足朴素贝叶斯假设时，数据点呈现“长条”状，这表明数据的 2 个维度之间存在线性相关关系，这符合我们的预期。

② 虽然数据分布有所不同，但我们设计的逻辑回归模型仍然可以很好地对其进行线性分类，但收敛速度相较满足朴素贝叶斯假设时有所下降。

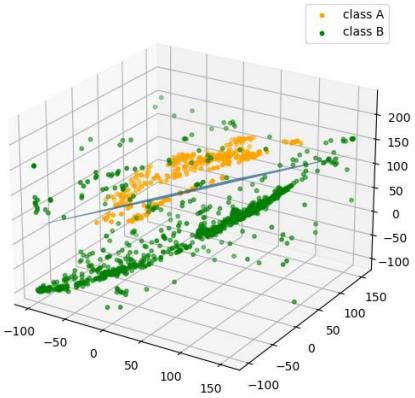
3. 采用 UCI 数据集

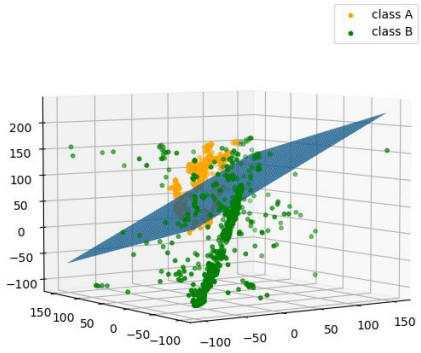
我们选取 UCI 的 Skin Segmentation Data Set 进行分类测试。

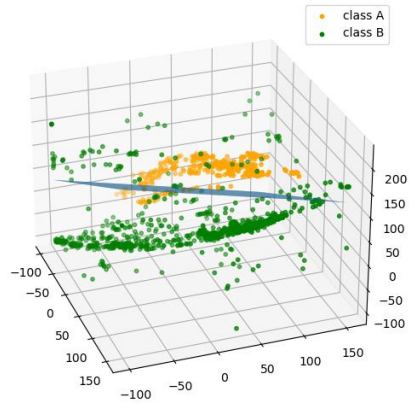
设置训练参数如下：

学习率为 1e-1，超参数 λ 1e-3，训练次数 10000。

训练结果如下：








```
acc:0.9233476132190942
```

可以看出，在经过足够多次迭代之后，我们训练出的模型在测试集上得到了 92%左右的准确率。

五、结论

我们以梯度下降法完成了 Logistic 回归实验。在实验中，我们通过自己生成数据，研究了正则项与数据集是否满足朴素贝叶斯分布这两个因素对分类结果的影响。在样本数量较小时，加入正则项可以有效避免过拟合的情况，对分类结果准确率有一定的提升，在样本数量较大时，是否加入正则项对分类准确率的影响较小。而是否满足朴素贝叶斯分布则决定了样本数据的分布，若不满足朴素贝叶斯分布，则样本数据间存在着某种线性关系，但这对分类结果的影响不大。另外，我们通过 UCI 的 Skin Segmentation Data Set 进行分类测试，验证我们的模型。在足够多轮的训练之后，我们的模型在 UCI 数据集上表现出了 92%左右的准确率，表现结果较好。

六、参考文献

[1] 博客园.机器学习(1)：Logistic 回归原理及其实现[EB/OL].2017[2021-10-10].https://www.cnblogs.com/cv-pr/p/7081861.html.
[2] 知乎.如何理解 Logistic 回归分析原理？[EB/OL].2018[2021-10-10].https://zhuanlan.zhihu.com/p/49481076.
[3] CSDN.Logistic 回归与梯度下降法[EB/OL].2015[2021-10-11].https://blog.csdn.net/acdreamers/article/details/44657979.
[4] UCI Machine Learning Repository.Skin Segmentation Data

Set[EB/OL].2012[2021-10-13].https://archive.ics.uci.edu/ml/datasets/skin+segmentation.

七、附录：源代码（带注释）

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# 全局变量
sigma = 0.1  # 生成数据的标准差
n = 1000  # 生成的样本点数量
naiveFlag = True  # if true,生成的数据满足朴素贝叶斯；否则不满足
epsilon = 1e-50  # 允许的误差值
eta = 1e-1  # 学习率
lamda = 1e-3  # 超参数 λ
epoch = 10000  # 训练次数


def genData():
    """在范围[0,1]内生成二维样本点数据"""
    a = n // 2  # 类别A 的样本点个数
    b = n - a  # 类别B 的样本点个数
    cov_xy = 0.08  # 两个维度的协方差（若不满足朴素贝叶斯假设）
    x_mean1 = [-0.7, -0.3]  # 类别1 的均值
    x_mean2 = [0.7, 0.3]  # 类别2 的均值
    train_x = np.zeros((n, 2))  # train_x 中保存着点的坐标(data)
    train_y = np.zeros(n)  # train_y 中保存着点的类别(label)
    if naiveFlag:  # 满足朴素贝叶斯的假设
        train_x[:a, :] = np.random.multivariate_normal(
            x_mean1, [[sigma, 0], [0, sigma]], size=a)
        train_x[a:, :] = np.random.multivariate_normal(
            x_mean2, [[sigma, 0], [0, sigma]], size=b)
        train_y[:a] = 0
        train_y[a:] = 1
    else:  # 不满足朴素贝叶斯的假设
        train_x[:a, :] = np.random.multivariate_normal(
            x_mean1, [[sigma, cov_xy], [cov_xy, sigma]], size=a)
        train_x[a:, :] = np.random.multivariate_normal(
            x_mean2, [[sigma, cov_xy], [cov_xy, sigma]], size=b)
        train_y[:a] = 0
        train_y[a:] = 1
    return train_x, train_y
```

```python
def genX(train_x):
    """生成 X 矩阵"""
    raw_X = np.ones((train_x.shape[0], 1))
    for i in range(0, train_x.shape[1]):
        temp = train_x[:, i]
        temp = temp.reshape((train_x.shape[0], 1))
        raw_X = np.hstack((raw_X, temp))
    X = raw_X.T
    assert X.shape[0] == train_x.shape[1] + 1
    assert X.shape[1] == train_x.shape[0]
    return X  # (3,500) (4, 49011)


def sigmoid(inx):
    """sigmoid 函数，为减缓溢出进行了优化操作"""
    if inx >= 0:  # 对 sigmoid 函数的优化，避免了出现极大的数据溢出
        return 1.0 / (1 + np.exp(-inx))
    else:
        return np.exp(inx) / (1 + np.exp(inx))


def loss(X, train_y, w):
    """损失函数"""
    loss = 0
    size = float(train_y.shape[0])
    for i in range(0, X.shape[1]):
        wXT = w @ X[:, i]
        loss += (train_y[i] * wXT / size - np.log(1 + np.exp(wXT)) /
size)
    return loss


def gradient(X, train_y, w):
    """计算梯度"""
    grad = np.zeros((1, X.shape[0]))
    for i in range(0, X.shape[1]):
        wXT = w @ X[:, i]
        grad += X[:, i].T * (train_y[i] - sigmoid(wXT))
    return grad


def gradient_descent(train_x, train_y):
    """梯度下降法"""
    eta_temp = eta
    w = np.ones((1, train_x.shape[1] + 1))  # (1,3)
```

```
    X = genX(train_x)
    grad = gradient(X, train_y, w)
    loss_list = []
    loss0 = 0
    loss1 = loss(X, train_y, w)
    count = 0
    while count < epoch:  # 误差值，小于该值时停止迭代
        count += 1
        w = w - eta_temp * lamda * w + eta_temp * grad
        loss0 = loss1
        loss1 = loss(X, train_y, w)
        if(np.abs(loss1) > np.abs(loss0)):  # loss 不降反增，则减半学习率
            eta_temp *= 0.5
        grad = gradient(X, train_y, w)
        loss_list.append(np.abs(loss0))
        print(count)
    return w, count, loss_list


def lossShow(loss_list):
    """展示 loss 函数的变化情况"""
    epoch_list = [x for x in range(1, epoch, 1)]
    loss_list.remove(loss_list[0])
    plt.plot(epoch_list, loss_list, c='#DB4D6D', linewidth=2,
             label='loss')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend()
    plt.show()
    return


def accuracy(X, train_y, w):
    """计算分类准确率"""
    wrong = 0
    total = train_y.shape[0]
    for i in range(0, train_y.shape[0]):
        flag = train_y[i]
        value = w @ X[:, i]
        if((flag == 1 and value < 0) or (flag != 1 and value > 0)):
            wrong += 1
    temp = float(wrong) / total
    acc = temp if temp > 0.5 else 1 - temp
    return acc
```

```
def classShow(train_x, train_y, w, count):
    """绘制分类效果图"""
    xa = []
    xb = []
    ya = []
    yb = []

    for i in range(0, train_x.shape[0]):
        if train_y[i] == 0:
            xa.append(train_x[i][0])
            ya.append(train_x[i][1])
        elif train_y[i] == 1:
            xb.append(train_x[i][0])
            yb.append(train_x[i][1])

    plt.scatter(xa, ya, c='orange', s=10, label='class A')
    plt.scatter(xb, yb, c='green', s=10, label='class B')  # 绘制散点图,
按不同标签赋予颜色

    x_real = np.arange(-2, 2, 0.01)
    w0 = w[0][0]
    w1 = w[0][1]
    w2 = w[0][2]
    y_real = -(w1 / w2) * x_real - w0 / w2

    plt.plot(x_real, y_real, c='#DB4D6D', linewidth=2,
             label='fitting')

    plt.xlim(-2, 2)
    plt.ylim(-2, 2)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('data_size = ' + str(n) + ',' + 'lamda = ' +
              str(lamda) + ',' + 'naiveFlag = ' + str(naiveFlag))  # 梯
度下降
    plt.show()
    return 0


def GD_show():
    """对梯度下降法结果进行展示"""
```

```
    train_x, train_y = genData()
    # print(train_x, train_y)
    # print(train_x.shape, train_y.shape)
    X = genX(train_x)
    w, count, loss_list = gradient_descent(train_x, train_y)
    acc = accuracy(X, train_y, w)
    print(w)
    print("acc:" + str(acc))
    classShow(train_x, train_y, w, count)
    lossShow(loss_list)
    return 0


def skin_read():
    """读取 UCI Skin 数据集"""
    skin_data = np.loadtxt('./Skin_NonSkin.txt', dtype=np.int32)
    np.random.shuffle(skin_data)  # 打乱读入的数据，以便分成训练集和测试集
    test_rate = 0.2  # 测试集比例
    step = 30  # 抽取部分样本，step 为步长
    # print(skin_data.shape) # (245057, 4)
    # print(skin_data)
    new_data = skin_data[::step]
    n = new_data.shape[0]  # 样本数量
    raw_train_data = new_data[:int(test_rate * n), :]  # 训练集
    raw_test_data = new_data[int(test_rate * n):, :]  # 测试集
    train_x = raw_train_data[:, :-1] - 100
    train_y = raw_train_data[:, -1:] - 1
    test_x = raw_test_data[:, :-1] - 100
    test_y = raw_test_data[:, -1:] - 1
    # print(train_x.shape) # (980, 3)
    # print(train_y.shape) # (980, 1)
    return train_x, train_y, test_x, test_y


def skin_plot_show(train_x, train_y, w):
    """对 skin 数据集的运行结果进行 3D 展示"""
    xa = []
    xb = []
    ya = []
    yb = []
    za = []
    zb = []
    w0 = w[0][0]
    w1 = w[0][1]
```

```
    w2 = w[0][2]
    w3 = w[0][3]

    for i in range(0, train_x.shape[0]):
        if train_y[i] == 0:
            xa.append(train_x[i][0])
            ya.append(train_x[i][1])
            za.append(train_x[i][2])
        elif train_y[i] == 1:
            xb.append(train_x[i][0])
            yb.append(train_x[i][1])
            zb.append(train_x[i][2])

    fig = plt.figure()
    ax = Axes3D(fig)

    ax.scatter(xa, ya, za, c='orange', s=10, label='class A')
    ax.scatter(xb, yb, zb, c='green', s=10, label='class B')  # 绘制散点
图，按不同标签赋予颜色

    real_x = np.arange(np.min(train_x[:,0]), np.max(train_x[:,0]), 1)
    real_y = np.arange(np.min(train_x[:,1]), np.max(train_x[:,1]), 1)
    real_X, real_Y = np.meshgrid(real_x, real_y)
    real_z = - w0 / w3 - (w1 / w3) * real_X - (w2 / w3) * real_Y
    ax.plot_surface(real_X, real_Y, real_z)

    ax.legend(loc='best')
    plt.title("Skin Dataset Classification")
    plt.show()
    return 0


def skin_show():
    """对 skin 数据集的训练结果进行展示"""
    train_x, train_y, test_x, test_y = skin_read()
    X = genX(train_x)
    w, count, loss_list = gradient_descent(train_x, train_y)
    test_X = genX(test_x)
    acc = accuracy(test_X, test_y, w)
    print(w)
    print("acc:" + str(acc))
    skin_plot_show(train_x, train_y, w)
    return
```

```python
# 主函数部分
method = 2
if method == 1:
    GD_show()
elif method == 2:
    skin_show()
```