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：1190201018
姓名：　李昆泽

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

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。

实验环境

OS：Win 10
Python 3.8

三、设计思想

1. 逻辑回归基本原理

Logistic 回归的基本思想就是利用朴素贝叶斯的假设取计算$P(Y|X)$。即利用 $P(Y)$，$P(X|Y)$ 以及各个维度之间的条件独立假设来计算$P(Y|X)$。

本实验是一个二分类问题，我们需要从数据集中学习到一个分类器$f: X \rightarrow Y$，其中X是一个向量，$X =< X_1, X_2, \dots, X_n >$，$Y \in \{0,1\}$。我们要求解的其实是$P(Y|X)$。假设$X$的各维度在$Y$的条件下是独立分布的，并且$P(X_i|Y = y_k) \sim N(\mu_{ik}, \sigma_i)$，$P(Y) \sim B(\pi)$。下面对$P(Y|X)$进行转化。

$$
\begin{aligned}
P\left(Y=0 \mid X\right) &= \frac{P\left(Y=0\right)P\left(X \mid Y=0\right)}{P\left(X\right)} \\
&= \frac{P\left(Y=0\right)P\left(X \mid Y=0\right)}{P\left(Y=0\right)P\left(X \mid Y=0\right)+P\left(Y=1\right)P\left(X \mid Y=1\right)} \\
&= \frac{1}{1+\frac{P\left(Y=1\right)P\left(X \mid Y=1\right)}{P\left(Y=0\right)P\left(X \mid Y=0\right)}} \\
&= \frac{1}{1+\exp(\ln\frac{P\left(Y=1\right)P\left(X \mid Y=1\right)}{P\left(Y=0\right)P\left(X \mid Y=0\right)})}
\end{aligned}
$$

因为Y符合伯努利分布，可令$\pi=P(Y=1)$，带入得

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

又由于各个维度的条件概率均服从高斯分布，因此

$$
P\left(Y=0 \mid X\right) = \frac{1}{1+\exp(\ln\frac{\pi}{1-\pi}+\sum_i (\frac{\mu_{i1}-\mu_{i0}}{\sigma_i^2}X_i+\frac{\mu_{i0}^2-\mu_{i1}^2}{2\sigma_i^2}))}
$$

令$w_0=\ln\frac{\pi}{1-\pi}+\sum_i(\frac{\mu_{i0}^2-\mu_{i1}^2}{2\sigma_i^2})$， $w_i=\frac{\mu_{i1}-\mu_{i0}}{\sigma_i^2}$， 则有

$$
P\left(Y=0 \mid X,W\right) = \frac{1}{1+\exp(w_0+\sum_i w_i X_i)}
$$

进而有

$$
P\left(Y=1 \mid X,W\right) = \frac{\exp(w_0+\sum_i w_i X_i)}{1+\exp(w_0+\sum_i w_i X_i)}
$$

使用极大条件似然估计来计算损失函数$l(W)$，有

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

我们要求$argmax\ l(W)$，也就是求$argmin - l(W)$，用梯度下降法求解。为了消除负号，我们令$L(W) = -l(W)$，$L(W)$为真正的损失函数。

这时损失函数$L(W)$可能存在上溢出的问题，即当数据量较大时，$L(W)$可能出现上溢出的问题。对此，我们对$L(W)$进行归一化处理。记归一化后的损失函数为$\hat{L}(W)$。

$$\hat{L}(W) = -\frac{1}{l} \sum_l Y^l (w_0 + \sum_{i=1}^n w_i X_i) - \ln(1 + \exp(w_0 + \sum_{i=1}^n w_i X_i))$$

2. 梯度下降法

梯度下降法的基本思想和实验一差不多，我们需要先求出损失函数关于W的梯度，然后结合学习率η的大小沿着负梯度的方向对W进行迭代。下面分别是损失函数关于W的梯度和每次迭代的公式。

$$\frac{\partial \hat{L}(W)}{\partial w_i} = -\frac{1}{l} \sum_l X_i^l (Y^l - sigmoid(w_0 + \sum_{i=1}^n w_i X_i))$$

$$w_i = w_i + \frac{\eta}{l} \sum_l X_i^l (Y^l - sigmoid(w_0 + \sum_{i=1}^n w_i X_i))$$

向量形式

$$W = W + \frac{\eta}{l} \sum_l X^l (Y^l - sigmoid(W^T X^l))$$

加入正则项后

$$w_i = w_i - \eta \lambda W + \frac{\eta}{l} \sum_l X_i^l (Y^l - sigmoid(w_0 + \sum_{i=1}^n w_i X_i))$$

$$W = W - \eta \lambda W + \frac{\eta}{l} \sum_l X^l (Y^l - sigmoid(W^T X^l))$$

其中

$$sigmoid(t)=\frac{1}{1+e^{-t}}$$

3. 牛顿法

牛顿法的思路是使用二阶泰勒展开去估计曲线，然后用二阶泰勒展开的函数的极值点去估计曲线的极值点，重复迭代直到找到极值点。

对于无约束最优化问题

$$\min_x f(x)$$

设$f(x)$ 有二阶连续偏导数，若第k次迭代值为x^k，则可以将$f(x)$在x^k附近进行二阶泰勒展开。

$$f(x)=f(x^k)+g_k^T(x-x^k)+\frac{1}{2}(x-x^k)^T H(x^k)(x-x^k)$$

其中$g_k=\nabla f(x)$是梯度向量，$H(x)$是海森矩阵

$$H(x)=\left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{n\times n}$$

当$f(x)$取到极值时，$g_k=0$。若此时$H(x)$是正定的，则极值为极小值。我们对$f(x)$关于 x求导。

$$\frac{df(x)}{dx}=g_k+H_k(x-x^k)$$

在极值点处$\frac{df(x)}{dx}=0$，即

$$x^{k+1}=x^k-H_k^{-1}g_k$$

这便是我们的迭代公式。将其应用到我们的损失函数中，可得

$$W^{k+1}=W^k-\left(\frac{\partial^2\hat{L}(W)}{\partial W\partial W^T}\right)^{-1}\frac{\partial\hat{L}(W)}{\partial W}$$

其中

$$\frac{\partial^2\hat{L}(W)}{\partial W\partial W^T}=\frac{1}{l}\sum_l (XX^T sigmoid(W^T X) sigmoid(-W^T X))+\lambda I$$

$$\frac{\partial\hat{L}(W)}{\partial w_i}=-\frac{1}{l}\sum_l X_i^l (Y^l-sigmoid(w_0+\sum_{i=1}^n w_i X_i))$$

4. 数据生成

在正式使用逻辑回归训练一个分类器之前，首先需要对数据进行

生成。为方便数据的可视化，我们生成一组二维的数据。我们利用高斯分布生成数据，正反例数据的均值分别为[-1, -1]和[1, 1]，方差均为0.4，每种类别各生成 50 个数据。这 100 个数据作为我们的训练集。我们划分训练集和测试集的比例为 2:8，即训练集共有 400 个数据，在训练集和测试集中样本的分布相同，且正反例数量相同。下面为生成训练集数据的示意图。

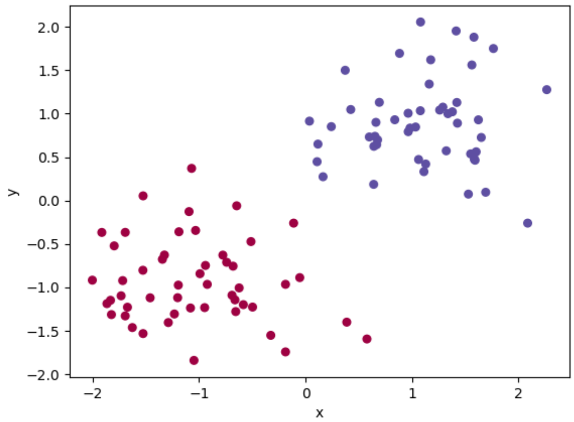


图 1 训练集数据示意图

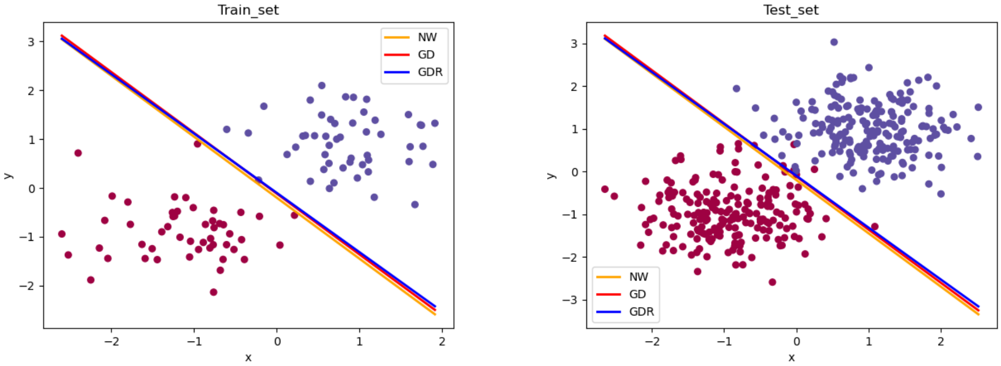
5. 计算分类准确率

在训练完成后，会得到对应的参数向量W，可以利用这个参数对数据进行分类，即计算$P(Y=0|X)$和$P(Y=1|X)$，相关的计算公式在上文已经给出。若 $P(Y=1|X^l)>0.5$ 且在样本中 $Y^l=1$ 或者 $P(Y=0|X^l)<0.5$且在样本中$Y^l=0$，就认为分类正确，反之则认为分类错误。据此可以计算出分类的准确率。

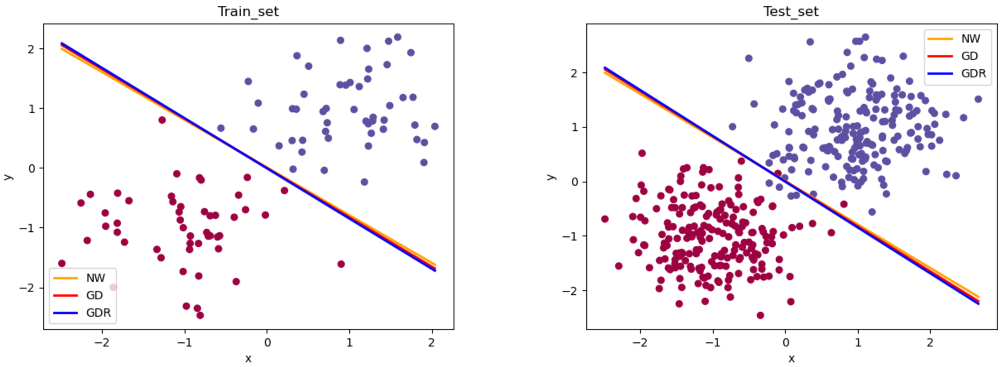
四、实验结果与分析

1. 满足朴素贝叶斯假设

我们使用三种方法训练的分类器在同一训练集和测试集上进行测试，两次的测试结果如下。



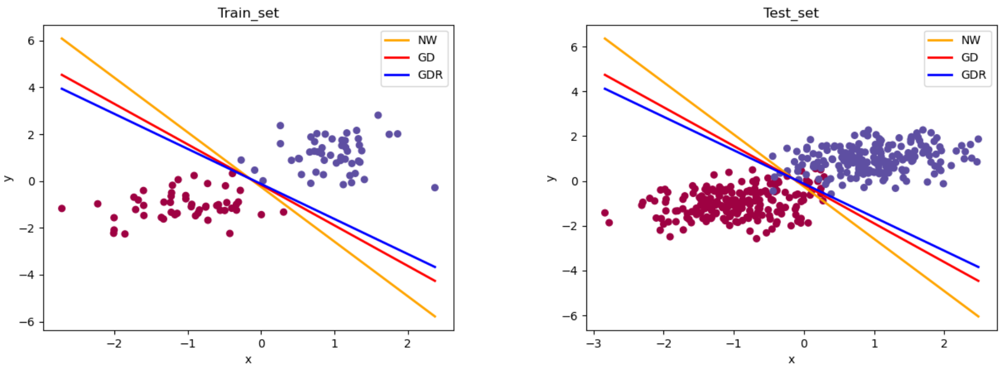
在第一次测试中，牛顿法、梯度下降法（不带正则项）和梯度下降法（带正则项）在训练集上的准确率分别为 1.0，1.0，0.99，在测试集上的准确率分别为 0.9825，0.985，0.9825。



在第二次测试中，牛顿法、梯度下降法（不带正则项）和梯度下降法（带正则项）在训练集上的准确率均为 1.0，在测试集上的准确率均为 0.985。

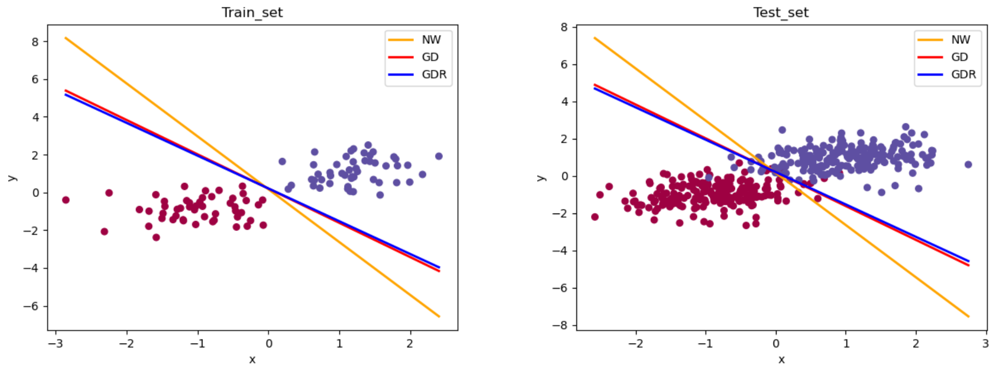
2. 不满足朴素贝叶斯假设

如果样本不满足朴素贝叶斯假设，则样本各个维度之间不是条件独立的，我们假设它们之间的协方差为 0.1。我们使用三种方法训练的分类器在同一训练集和测试集上进行测试，两次的测试结果如下。



在第一次测试中，牛顿法、梯度下降法（不带正则项）和梯度下

降法（带正则项）在训练集上的准确率均为 1.0，在测试集上的准确率分别为 0.965，0.97，0.9725。



在第二次测试中，牛顿法、梯度下降法（不带正则项）和梯度下降法（带正则项）在训练集上的准确率均为 1.0，在测试集上的准确率均为 0.96。

另外，由于三种方法都需要进行迭代，我们也统计了三种情况下的迭代次数，并进行对比，具体结果如下表所示（正反例数据的均值分别为[-1, -1]和[1, 1]，方差均为 0.4，每种类别各生成 50 个数据）。

表 1 迭代次数对比

序号	牛顿法	梯度下降法（不带正则项）	梯度下降法（带正则项）
1	8	40728	29164
2	7	19560	16768
3	7	13393	12173
4	16	52010	30567
5	7	23113	19113
6	16	81218	40843
7	14	50970	31192
8	6	10225	9676
9	6	16382	14606
10	9	35357	24967
平均	9.6	34295.6	22906.9

通过上表，可以发现牛顿法的迭代次数较少，大约在 10 次左右，而梯度下降法的迭代次数较多。而就梯度下降法而言，是否带正则项对迭代次数有一定影响，带正则项的迭代次数较少，大约在 23000 次

左右，而不带正则项的梯度下降法迭代次数大约在 34000 次左右。

结合上面的实验结果，我们发现：

（1）在数据分布不满足朴素贝叶斯假设时，采用三种不同方法训练的分类器表现会有些许下降，但是总体分类准确率依旧维持在一个较高的水平上。

（2）在同样的数据分布条件下，梯度下降法的准确性和牛顿法相比要更好，但差距不是很大。

（3）采用牛顿法进行逻辑回归的迭代次数较少，而使用梯度下降法所需的迭代次数较多。

3. 使用 UCI 数据集进行测试

我使用的数据集是 Haberman's Survival Data Set，其中包含的主要是一些病人的数据。数据是三维的，每一维度分别表示手术时的年龄、手术年份和腋下淋巴结个数，标签表示存活状态（1 表示该病人手术后活了 5 年或更长时间，2 表示病人在手术后的 5 年内死亡）。显然，这是一个二分类的问题，这与我们的分类器相符，我们在我们的分类器上对该数据进行测试。训练集和测试集的比例为 3:7，测试结果如下。

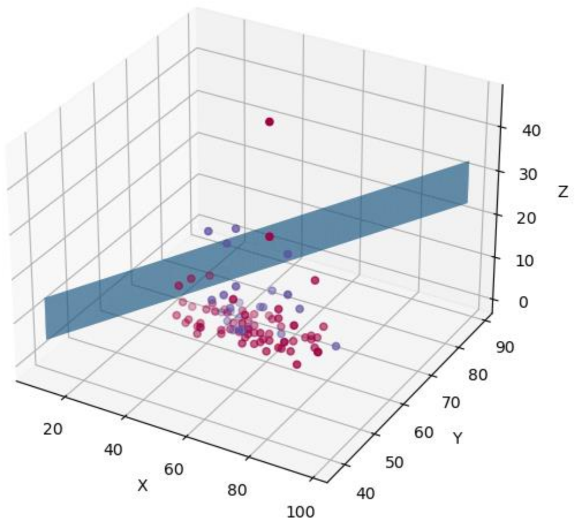


图 2 训练集

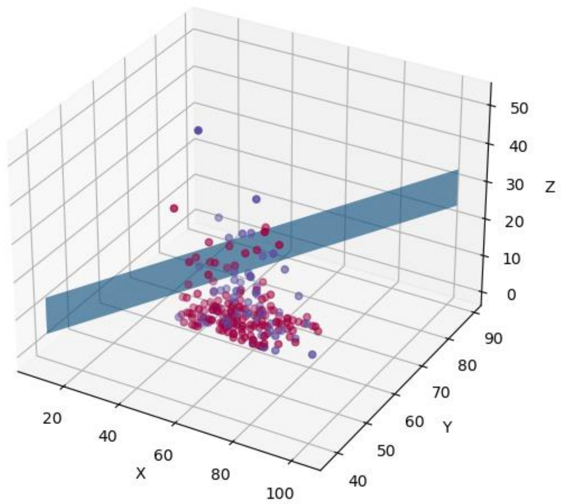


图 3 测试集

我使用了梯度下降法（带正则项）来进行实验，发现迭代次数为383995，可以看出迭代次数相比之前大大增加。另外，我们的逻辑回归分类器在训练集和测试集上的准确率分别为 0.747 和 0.744，这样的准确率是不太理想的。通过上图可以看出，不同类别数据的分类面本就不太明显，而我们的逻辑回归模型只能学习到平面的分类器，无法进行更复杂分类面的学习，准确率自然会有所降低。

五、结论

（1）在数据分布不满足朴素贝叶斯假设时，逻辑回归的分类器表现会有些许下降，但是总体分类准确率依旧维持在一个较高的水平上。
（2）在同样的数据分布条件下，梯度下降法的准确性和牛顿法相比要更好，但差距不是很大。
（3）采用牛顿法进行逻辑回归的迭代次数较少，而使用梯度下降法所需的迭代次数较多。

六、参考文献

[1] 周志华 著. 机器学习, 北京: 清华大学出版社, 2016.1
[2] 李航 著. 统计学习方法, 北京: 清华大学出版社, 2019.5
[3] Haberman, S. J. (1976). Generalized Residuals for Log-Linear Models, Proceedings of the 9th International Biometrics Conference, Boston, pp. 104-122.

七、附录：源代码（带注释）

```
import numpy as np
```

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

# 超参数
mean_1 = [-1., -1.]
mean_2 = [1., 1.]
var = 0.4
cov = 0.1
train_size = 50
test_size = 200

loss_history = []


# 随机生成数据
def generate_data(mean_1, var_1, size_1, mean_2, var_2, size_2, cov=0.0):
    train_x = np.zeros((size_1 + size_2, 2))
    train_y = np.zeros(size_1 + size_2)
    train_x[:size_1, :] = np.random.multivariate_normal(
        mean=mean_1, cov=[[var_1, cov], [cov, var_1]], size=size_1)
    train_x[size_1:, :] = np.random.multivariate_normal(
        mean=mean_2, cov=[[var_2, cov], [cov, var_2]], size=size_2)
    train_y[size_1:] = 1
    return train_x.T, train_y.reshape(1, -1)


# sigmoid 函数
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# 损失函数
def calculate_loss(X, Y, W, lambd=0):
    size = Y.shape[1]
    loss = (np.sum(Y * W.T.dot(X)) - np.sum(np.log(1 + np.exp(W.T.dot(X))))) / size - 0.5 * lambd
* W.T.dot(W)
    return -float(loss)


# 梯度下降
def gradient_descent(X, Y, lambd=0, lr=0.05, epsilon=1e-6):
    # 初始化
    iter = 0
    size = X.shape[1]
```

```
    dimension = X.shape[0]
    ones = np.ones((1, size))
    X = np.row_stack((ones, X))   # 构造 X 的增广矩阵，增加一个全 1 的行
    W = np.ones((dimension + 1, 1))

    last_loss = calculate_loss(X, Y, W, lambd=lambd)

    while True:
        loss_history.append(last_loss)

        # 计算梯度
        dW = - np.sum(X * (Y - sigmoid(W.T.dot(X))), axis=1).reshape(-1, 1) / size + lambd * W

        # 梯度下降
        W -= lr * dW

        # 计算新的损失
        loss = calculate_loss(X, Y, W, lambd=lambd)
        print(loss)

        if np.abs(loss - last_loss) < epsilon and np.dot(dW.T, dW) < epsilon:
            break
        else:
            # 更新迭代次数，更新损失
            iter += 1
            last_loss = loss

    coefficient = - W[:dimension, 0] / W[dimension]

    print(iter)

    return coefficient, W


# 海森阵
def Hessian(X, W, lambd=0):
    size = X.shape[1]
    dimension = W.shape[0]
    return (sigmoid(W.T.dot(X)) * sigmoid(-W.T.dot(X)) * X).dot(X.T) / size + lambd *
np.identity(dimension)


# 牛顿法
def newton(X, Y, lambd=0, epsilon=1e-6):
```

```
    # 初始化
    iter = 0
    size = X.shape[1]
    dimension = X.shape[0]
    ones = np.ones((1, size))
    X = np.row_stack((ones, X))   # 构造 X 的增广矩阵，增加一个全 1 的行
    W = np.ones((dimension + 1, 1))

    last_loss = calculate_loss(X, Y, W, lambd=lambd)

    while True:
        loss_history.append(last_loss)

        # 计算梯度
        dW = - np.sum(X * (Y - sigmoid(W.T.dot(X))), axis=1).reshape(-1, 1) / size + lambd * W

        # 计算海森阵
        H = Hessian(X, W, lambd=lambd)

        # 迭代
        W = W - np.linalg.inv(H).dot(dW)

        # 计算新的损失
        loss = calculate_loss(X, Y, W, lambd=lambd)
        # print(loss)

        if np.abs(loss - last_loss) < epsilon and np.dot(dW.T, dW) < epsilon:
            break
        else:
            # 更新迭代次数，更新损失
            iter += 1
            last_loss = loss

    coefficient = - W[:dimension, 0] / W[dimension]

    return coefficient, W


# 计算分类准确率
def accuracy(X, Y, W):
    total = X.shape[1]
    correct_num = 0

    ones = np.ones((1, total))
```

```
    X = np.row_stack((ones, X))

    for i in range(total):
        if sigmoid(W.T.dot(X[:, i])) > 0.5 and Y[0, i] == 1 or sigmoid(W.T.dot(X[:, i])) < 0.5 and
Y[0, i] == 0:
            correct_num += 1

    return float(correct_num) / total


# 作图
def show_data(X, Y, coefficient1, coefficient2, coefficient3, title):
    X = X.T
    Y = Y.T
    plt.scatter(X[:, 0], X[:, 1], c=Y, s=30, marker='o', cmap=plt.cm.Spectral)

    bottom = np.min(X[:, 0])
    top = np.max(X[:, 0])
    X = np.linspace(bottom, top, 100).reshape(-1, 1)
    Y = coefficient1[0] + coefficient1[1] * X
    plt.plot(X, Y, 'orange', linewidth=2, label='NW')

    Y = coefficient2[0] + coefficient2[1] * X
    plt.plot(X, Y, 'r', linewidth=2, label='GD')

    Y = coefficient3[0] + coefficient3[1] * X
    plt.plot(X, Y, 'b', linewidth=2, label='GDR')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(loc="best", fontsize=10)
    plt.title(title)
    plt.show()


# 损失曲线
def loss_curve(loss_history):
    plt.plot(np.linspace(1, len(loss_history) + 1, len(loss_history)), loss_history)
    plt.show()


# 读取 UCI 数据
def uci_data(path):
    data = np.loadtxt(path, dtype=np.int32)
```

```
    np.random.shuffle(data) # 随机打乱数据，便于选取数据
    dimension = data.shape[1]
    train_size = int(0.3 * data.shape[0]) # 按照 3：7 的比例分配训练集和测试集

    # 划分训练集和测试集
    train_data = data[:train_size, :]
    test_data = data[train_size:, :]

    train_x = train_data[:, 0:dimension-1]
    train_y = train_data[:, dimension-1] - 1
    test_x = test_data[:, 0:dimension-1]
    test_y = test_data[:, dimension-1] - 1

    return train_x.T, train_y.reshape(1, -1), test_x.T, test_y.reshape(1, -1)


# 绘制三维图像
def show_3D(X, Y, coefficient, title):
    X = X.T
    Y = Y.T
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=Y, cmap=plt.cm.Spectral)
    real_x = np.linspace(np.min(X[:, 0]) - 20, np.max(X[:, 0]) + 20, 255)
    real_y = np.linspace(np.min(X[:, 1]) - 20, np.max(X[:, 1]) + 20, 255)
    real_X, real_Y = np.meshgrid(real_x, real_y)
    real_z = coefficient[0] + coefficient[1] * real_X + coefficient[2] * real_Y
    ax.plot_surface(real_x, real_y, real_z, rstride=1, cstride=1)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title(title)
    plt.show()

# 生成训练样本
train_x, train_y = generate_data(mean_1, var, train_size, mean_2, var, train_size, cov)

# 牛顿法
coefficient1, W1 = newton(train_x, train_y)

# 梯度下降法
coefficient2, W2 = gradient_descent(train_x, train_y, lambd=0)
coefficient3, W3 = gradient_descent(train_x, train_y, lambd=1e-4)
```

```
# 计算训练集分类准确率
print('newton_train', accuracy(train_x, train_y, W1))
print('gd_train', accuracy(train_x, train_y, W2))
print('gdr_train', accuracy(train_x, train_y, W3))

# 训练集分类结果
show_data(train_x, train_y, coefficient1, coefficient2, coefficient3, 'Train_set')


# 损失函数曲线
# loss_curve(loss_history)

# 生成测试样本
test_x, test_y = generate_data(mean_1, var, test_size, mean_2, var, test_size, cov)

# 计算测试集分类准确率
print('newton_test', accuracy(test_x, test_y, W1))
print('gd_test', accuracy(test_x, test_y, W2))
print('gdr_test', accuracy(test_x, test_y, W3))

# 测试集分类结果
show_data(test_x, test_y, coefficient1, coefficient2, coefficient3, 'Test_set')

# UCI 数据集测试
train_x, train_y, test_x, test_y = uci_data('haberman.txt')
coefficient, W = gradient_descent(train_x, train_y, lr=0.0005, lambd=1e-3, epsilon=1e-5)
print('uci_train', accuracy(train_x, train_y, W))
show_3D(train_x, train_y, coefficient, 'Train_set')

print('uci_test', accuracy(test_x, test_y, W))
show_3D(test_x, test_y, coefficient, 'Test_set')
```