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：必修

实验题目：逻辑回归

学号：1190200523

姓名：石翔宇

1 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。

2 实验要求及实验环境

2.1 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。

验证方法

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。

2.2 实验环境

Windows 11 + Python 3.7.8

3 设计思想

3.1 逻辑回归

逻辑回归又名对数几率回归，虽然名字叫"回归"，但实际上是一种分类学习方法。

考虑广义线性模型

$$y = g(w^T x + b) \tag{1}$$

，除了可以做回归问题，还可以做分类任务，只需找到一个单调可微函数将分类任务的真是标记 y 与线性回归模型的预测值联系起来。

考虑连续可微的对数几率函数

$$y = \frac{1}{1 + e^{-z}} \tag{2}$$

将其代入式1得到

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \tag{3}$$

将其取对数，则可以变化为

$$\ln \frac{y}{1 - y} = w^T x + b \tag{4}$$

若将 y 视为样本 x 作为正例的可能性，则 $1 - y$ 是其反例可能性，则称 $\frac{y}{1-y}$ 为几率，$\ln \frac{y}{1-y}$ 为对数几率。实际上，就是在用线性回归模型的预测结果去逼近真实标记的对数几率。

现在我们来确定式3中的 w 和 b。

我们将式3中的 y 视为类后验概率估计 $p(y=1|x)$ ，则式4可重写为

$$\ln \frac{p(y=1|x)}{p(y=0|x)} = w^T x + b \tag{5}$$

由式3可得

$$p(y=1|x) = \frac{e^{w^T x+b}}{1+e^{w^T x+b}} p(y=0|x) = \frac{1}{1+e^{w^T x+b}} \tag{6}$$

于是，我们可以用极大似然法估计 w 和 b。给定数据集 $\{(x_i, y_i\}_{i=1}^m$，为便于讨论，令 $\beta=(w;b), \hat{x}=(x;1)$，则 $w^T x + b = \beta^T \hat{x}$。则对数几率回归模型最大化对数似然为

$$l(w,b) = \sum_{i=1}^{m} \ln p(y_i|x_i;w,b) \tag{7}$$

要将 $l(w,b)$ 最大化，也即最小化

$$\begin{aligned}
J(\beta) &= -\frac{1}{m}\sum_{i=1}^{m} \ln(y_i \frac{e^{\beta^T \hat{x}_i}}{1+e^{\beta^T \hat{x}_i}} + (1-y_i)\frac{1}{1+e^{\beta^T \hat{x}_i}}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} \ln \frac{y_i e^{\beta^T \hat{x}_i} + 1 - y_i}{1+e^{\beta^T \hat{x}_i}} \\
&= -\frac{1}{m}\sum_{i=1}^{m} (\ln(y_i e^{\beta^T \hat{x}_i} + 1 - y_i) - \ln(1+e^{\beta^T \hat{x}_i})) \\
&= \frac{1}{m}\sum_{i=1}^{m} (-y_i \beta^T \hat{x}_i + \ln(1+e^{\beta^T \hat{x}_i}))
\end{aligned} \tag{8}$$

则我们的任务目标就变为

$$\min_{\beta} J(\beta) \tag{9}$$

若我们加入正则项，则有

$$J'(\beta) = J(\beta) + \frac{\lambda}{2}\|\beta\|_2^2 \tag{10}$$

相应地，加入正则项后的任务目标变为

$$\min_{\beta} J'(\beta) \tag{11}$$

3.2 梯度下降法（无正则项）

$J(\beta)$ 是关于 β 的高阶可导连续凸函数，则可用梯度下降法求导其最优解。

我们将 $J(\beta)$ 对 β 求偏导可得

$$\frac{\partial J(\beta)}{\partial \beta} = \frac{1}{m}\sum_{i=1}^{m} \hat{x}_i(-y_i + \frac{e^{\beta^T \hat{x}_i}}{1+e^{\beta^T \hat{x}_i}}) = \nabla J(\beta) \tag{12}$$

则按照梯度下降法，每一步我们都按照

$$\beta_{i+1} = \beta_i - \alpha \nabla J(\beta) \tag{13}$$

来更新 β ，其中 α 被称作为学习率或者步长，满足 $\alpha > 0$ 。

3.3 梯度下降法（有正则项）

相应地，加入正则项后，我们将 $J'(\beta)$ 对 β 求偏导可得

$$\frac{\partial J'(\beta)}{\partial \beta} = \frac{1}{m}\sum_{i=1}^{m}\hat{x}_i(-y_i + \frac{e^{\beta^T\hat{x}_i}}{1+e^{\beta^T\hat{x}_i}}) + \lambda\beta = \nabla J'(\beta) \tag{14}$$

每一步我们按照

$$\beta_{i+1} = \beta_i - \alpha\nabla J'(\beta) \tag{15}$$

来更新 β 。

3.4 牛顿法（无正则项）

$J(\beta)$ 是关于 β 的高阶可导连续凸函数，则也可用牛顿法求导其最优解。

假设 β 的第 k 次迭代值为 $\beta^{(k)}$，则可将 $J(\beta)$ 在 $\beta^{(k)}$ 附近进行二阶泰勒展开

$$J(\beta) = J(\beta^{(k)}) + g_k^T(\beta-\beta^{(k)}) + \frac{1}{2}(\beta-\beta^{(k)})^T H(\beta^{(k)})(\beta-\beta^{(k)}) \tag{16}$$

其中 $g_k = g(\beta^{(k)}) = \nabla J(\beta^{(k)})$ 是 $J(\beta)$ 的梯度向量在点 $\beta^{(k)}$ 的值，$H(\beta^{(k)})$ 是 $J(\beta)$ 的黑塞矩阵

$$\begin{aligned} H(\beta) &= \left[\frac{\partial^2 J}{\partial\beta_i\partial\beta_j}\right]_{n\times n} \\ &= \frac{\partial^2 J}{\partial\beta\partial\beta^T} \\ &= \frac{1}{m}\sum_{i=1}^{m}\hat{x}_i\hat{x}_i^T\frac{e^{\beta^T\hat{x}_i}}{(1+e^{\beta^T\hat{x}_i})^2} \end{aligned} \tag{17}$$

在点 $\beta^{(k)}$ 的值。函数 $J(\beta)$ 有极值的必要条件是在极值点处梯度为 0，即

$$\nabla J(\beta) = 0 \tag{18}$$

假设第 $k+1$ 次迭代时有 $\nabla J(\beta^{(k+1)}) = 0$，则由式16可得

$$\nabla J(\beta) = g_k + H_k(\beta^{(k+1)} - \beta^{(k)}) = 0 \tag{19}$$

则迭代式为

$$\beta^{(k+1)} = \beta^{(k)} - H_k^{-1}g_k \tag{20}$$

3.5 牛顿法（有正则项）

相应地，加入正则项后，$J'(\beta)$ 的黑塞矩阵为

$$\begin{aligned} H'(\beta) &= \left[\frac{\partial^2 J'}{\partial\beta_i\partial\beta_j}\right]_{n\times n} \\ &= \frac{\partial^2 J'}{\partial\beta\partial\beta^T} \\ &= \frac{1}{m}\sum_{i=1}^{m}\hat{x}_i\hat{x}_i^T\frac{e^{\beta^T\hat{x}_i}}{(1+e^{\beta^T\hat{x}_i})^2} + \lambda \end{aligned} \tag{21}$$

则迭代式为

$$\beta^{(k+1)} = \beta^{(k)} - H_k'^{-1}g_k' \tag{22}$$

4 实验结果分析

4.1 手工数据集生成

我们将正例和负例的均值分别设置为 $(1,3)$ 和 $(2,5)$，标准差都设置为 0.2，相关系数为 0.1 或 0（前者为不满足朴素贝叶斯假设时，后者为满足时）。训练集和测试集的个数分别设置为 200 和 300（以下所有测试的数据皆是如此）。图 1分别展示了两个集合的分布。

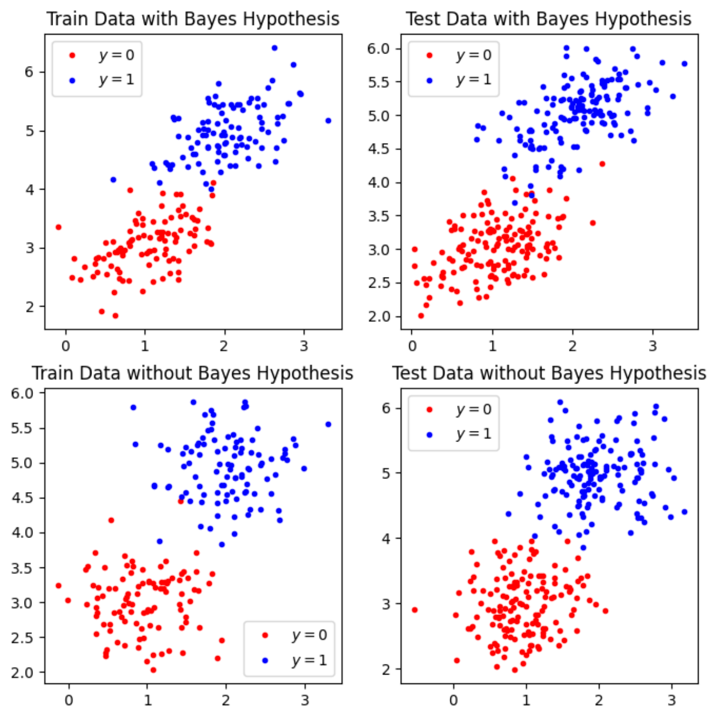

图 1

4.2 梯度下降法在不满足朴素贝叶斯假设的手工数据集上的测试

在不满足朴素贝叶斯假设时，正例和负例的相关系数为 0.1。我们将最大迭代次数设置为 70000，学习率设置为 10^{-1}，早停策略的参数值设置为 10^{-7}，正则项系数设置为 10^{-3}（若有的话）。无正则项和有正则项时的测试结果如图 2。

可以看到，正则项对分类结果影响并不大，但是对收敛速度影响很大。

4.3 梯度下降法在满足朴素贝叶斯假设的手工数据集上的测试

在满足朴素贝叶斯假设时，正例和负例的相关系数为 0。我们将最大迭代次数设置为 70000，学习率设置为 10^{-1}，早停策略的参数值设置为 10^{-7}，正则项系数设置为 10^{-3}（若有的话）。无正则项和有正则项时的测试结果如图 3。

可以看到，正则项对分类结果影响并不大，但是对收敛速度影响较大。

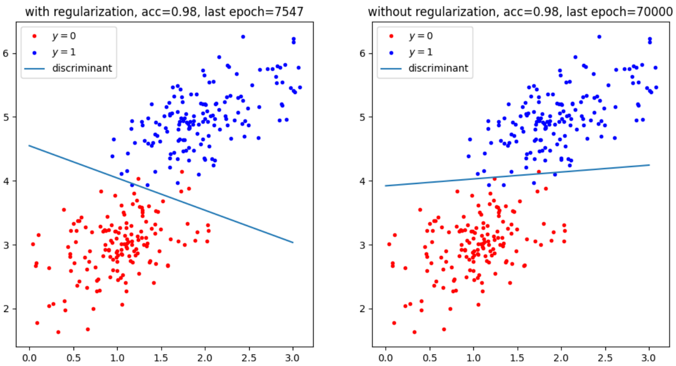

图 2

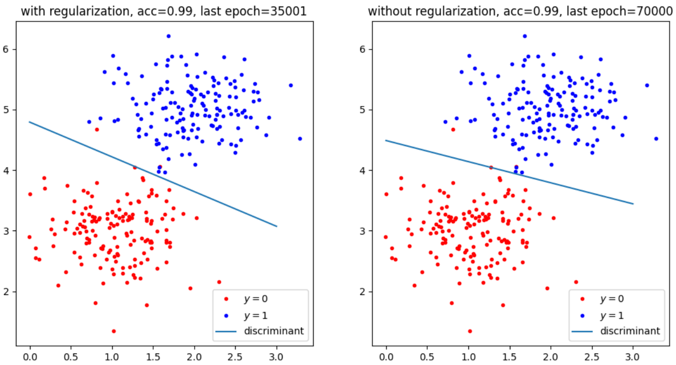

图 3

4.4 牛顿法在不满足朴素贝叶斯假设的手工数据集上的测试

在满足朴素贝叶斯假设时，正例和负例的相关系数为 0。我们将最大迭代次数设置为 70000，早停策略的参数值设置为 10^{-7}，正则项系数设置为 10^{-3}（若有的话）。无正则项和有正则项时的测试结果如图 4。

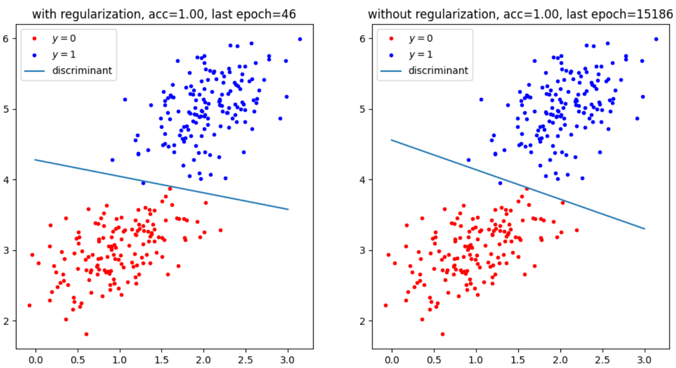

图 4

可以看到，正则项对分类结果影响并不大，但是对收敛速度影响很大。

4.5 牛顿法在满足朴素贝叶斯假设的手工数据集上的测试

在满足朴素贝叶斯假设时，正例和负例的相关系数为 0。我们将最大迭代次数设置为 70000，早停策略的参数值设置为 10^{-7}，正则项系数设置为 10^{-3}（若有的话）。无正则项和有正则项时的测试结果如图 **5**。

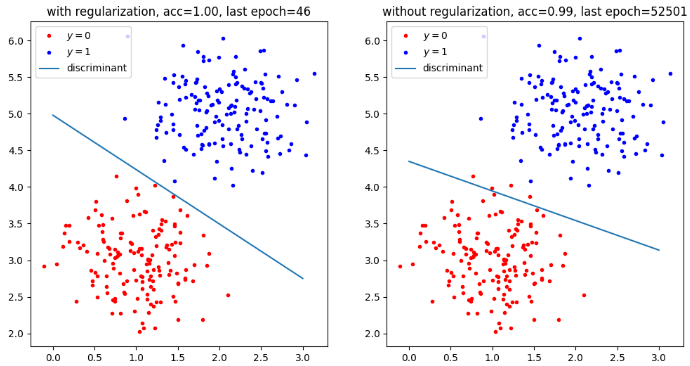

图 5

可以看到，正则项略微提升了分类性能，对收敛速度影响很大。

4.6 手工数据集上的测试对比

通过上面的测试我们可以看出，满足朴素贝叶斯假设与否，加入正则项与否，都对分类性能的影响不太大。与梯度下降法相比，牛顿法收敛速度更快；并且可以看出，在牛顿法中加入正则项对收敛速度的提升较大。

4.7 UCI 数据上的测试

我们使用 UCI 网站上的数据集Skin Segmentation Data Set。该数据集大小为 245057，维度为 4。我们随机选择 2000 个数据（1000 个正例，1000 个负例），将其按照 30%70% 的比例划分为训练集和测试集，训练集中有 600 个数据，测试集中有 1400 个数据。数据分布如图 **6**所示。

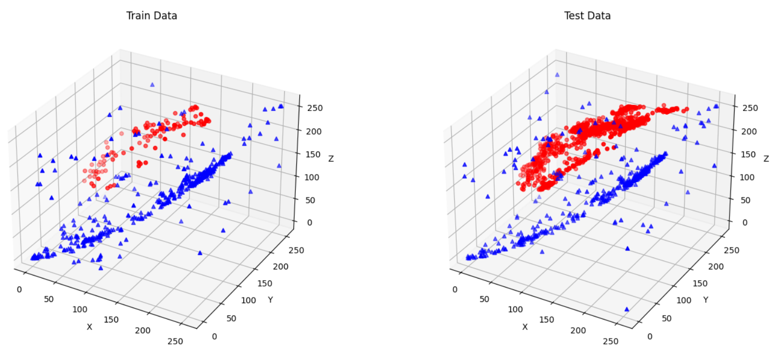

图 6

我们利用梯度下降法测试，最大迭代次数设置为 7000，学习率设置为 10^{-4}，测试结果如图 7所示。

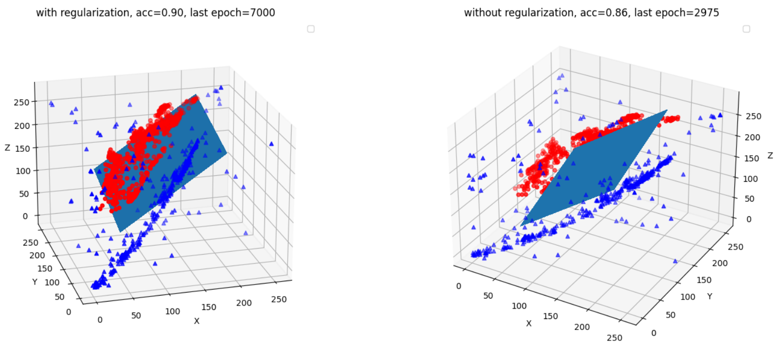

图 7

我们利用牛顿法测试，最大迭代次数设置为 70000，测试结果如图 8所示。

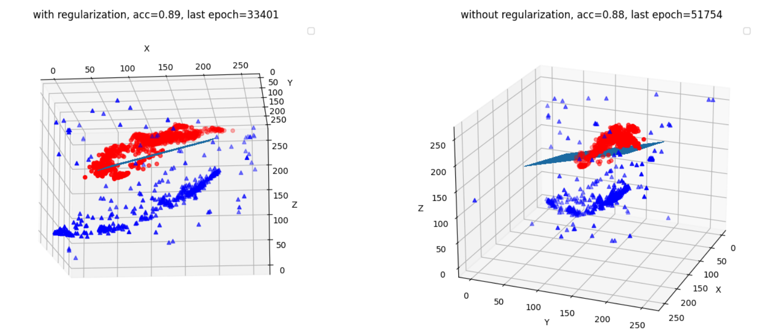

图 8

通过上面的测试我们可以看出，分类器在真实数据集上也能够表现良好。

5 结论

在求解逻辑回归问题中，我们用到了两种方法，分别是梯度下降法和牛顿法。梯度下降法每次迭代较快，但是所需迭代次数较多；牛顿法每次迭代较慢，但所需迭代次数较少。总体来说，牛顿法较快，但是适用范围较小，当黑塞矩阵奇异时，牛顿法不再适用。

对于数据是否满足朴素贝叶斯假设对性能的影响，从上述实验可以看出，是否满足假设影响不大。

A 源代码（带注释）

A.1 手工数据集生成

Listing 1: data.py

```python
import numpy as np
import matplotlib.pyplot as plt
from utils import *


def data_generator(size=(100, 150), loc=np.array([[1, 3],[2, 5]]), cov=[[0.4, 0.2], [0.2,
    0.4]]):
    """
    生成数据

    Args:
        size (tuple, optional): 训练集和测试集的数据大小. Defaults to (100, 150).
        loc (list, optional): 均值. Defaults to np.array([[1, 3],[2, 5]]).
        cov (list, optional): 协方差. Defaults to [[0.4, 0.2], [0.2, 0.4]].
    """
    def generate(siz):
        X = np.zeros((siz * 2, 2))
        Y = np.zeros((siz * 2, 1))
        X[:siz, :] = np.random.multivariate_normal(loc[0], cov, size=siz)
        X[siz:, :] = np.random.multivariate_normal(loc[1], cov, size=siz)
        Y[siz:] = 1
        return X, Y
    train_data, test_data = generate(size[0]), generate(size[1])
    return train_data, test_data

# 画图
def show(dataset):
    plt.figure(figsize=(8, 8))
    messages = ['with Bayes Hypothesis', 'without Bayes Hypothesis']
    titles = ['Train Data', 'Test Data']
    for i in  range(2):
        for j in  range(2):
            X_0, X_1 = get_Xs(dataset[i][j])
            plt.subplot(2, 2, i * 2 + j + 1)
            plt.plot(X_0[:, 0], X_0[:, 1], '.', color='r', label="$y=0$")
            plt.plot(X_1[:, 0], X_1[:, 1], '.', color='b', label="$y=1$")
            plt.title('{} {}'. format(titles[j], messages[i]))
            plt.legend()
    plt.show()

def get_data_option(loc1=(1, 3), loc2=(2, 5), scale=0.2, conv=0.1):
    """
```

```
    生成与数据生成函数相匹配的参数

    Args:
        loc1 (tuple, optional): 反例的均值. Defaults to (1, 3).
        loc2 (tuple, optional): 正例的均值. Defaults to (2, 5).
        scale (float, optional): 离散程度. Defaults to 0.2.
        conv (float, optional): 相关系数. Defaults to 0.1.

    Returns:
        tuple: 与数据生成函数相匹配的参数
    """
    loc = np.array([[loc1[0], loc1[1]],[loc2[0], loc2[1]]])
    cov = [[scale, conv], [conv, scale]]
    return loc, cov

if __name__ == '__main__':
    size = (100, 150)
    loc, cov = get_data_option()
    dataset0 = data_generator(size, loc, cov)
    loc, cov = get_data_option(conv=0)
    dataset1 = data_generator(size, loc, cov)
    show((dataset0, dataset1))
```

A.2 梯度下降法

Listing 2: gradient descent.py

```
import numpy as np
import matplotlib.pyplot as plt
from data import *
from uci_data import *
from utils import *

class gradient_descent:
    def __init__(self, dataset, lr=1e-1, epochs=70000, delta=1e-7, lambda_=1e-3):
        """
        梯度下降法初始化

        Args:
            dataset (tuple): 训练样本
            lr (float), optional): 学习率. Defaults to 1e-1.
            epochs (int, optional): 最大迭代次数. Defaults to 70000.
            delta (float), optional): 早停策略的参数值. Defaults to 1e-7.
            lambda_ (float), optional): 正则项系数. Defaults to 1e-3.
        """
        self.lambda_ = lambda_
        self.dataset = dataset
```

```python
        self.X = transform(dataset[0])
        self.Y = dataset[1]
        self.lr = lr
        self.epochs = epochs
        self.delta = delta

    def gradient_descent(self):
        """
        梯度下降法

        Returns:
            tuple: (beta, 总迭代次数)
        """
        beta = np.random.normal(size=(self.X.shape[1], 1))
        lr_scheduler = lr_scheduler_MultiStep(self.epochs, self.lr)
        loss_last = loss = calc_loss(self.X, beta, self.Y, self.lambda_)
        last_epoch = self.epochs
        for epoch in  range(self.epochs):
            beta = beta - self.lr * calc_gradient(self.X, beta, self.Y, self.lambda_)
            loss = calc_loss(self.X, beta, self.Y, self.lambda_)
            if early_stop(loss, loss_last, self.delta):
                last_epoch = epoch
                break
            else: loss_last = loss
            self.lr = lr_scheduler.step(epoch)
        return beta, last_epoch

# 画图
def draw(train_data, test_data):
    def draw_(dataset, beta, last_epoch, message):
        acc = calc_accuracy(dataset, beta)
        X_0, X_1 = get_Xs(dataset)
        plt.plot(X_0[:, 0], X_0[:, 1], '.', color='r', label="$y=0$")
        plt.plot(X_1[:, 0], X_1[:, 1], '.', color='b', label="$y=1$")
        plot(beta, 0, 3)
        plt.title('{}, acc={:.2f}, last epoch={}'. format(message, acc, last_epoch))
        plt.legend()

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    beta, last_epoch = gradient_descent(train_data).gradient_descent()
    draw_(test_data, beta, last_epoch, 'with regularization')
    plt.subplot(1, 2, 2)
    beta, last_epoch = gradient_descent(train_data, lambda_=None).gradient_descent()
    draw_(test_data, beta, last_epoch, 'without regularization')

    plt.show()

```

```python
# 画图
def draw_3d(train_data, test_data):
    def draw_(dataset, beta, last_epoch, message):
        acc = calc_accuracy(dataset, beta)
        fig = plt.figure(figsize=(7, 7))
        X_0, X_1 = get_Xs(dataset)
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(X_0[:, 0], X_0[:, 1], X_0[:, 2], c='r', marker='o')
        ax.scatter(X_1[:, 0], X_1[:, 1], X_1[:, 2], c='b', marker='^')
        plot_3d(beta, 50, 200, ax)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.title('{}, acc={:.2f}, last epoch={}'. format(message, acc, last_epoch))
        plt.legend()
        plt.show()

    beta, last_epoch = gradient_descent(train_data, lr=1e-4, epochs=7000).gradient_descent()
    draw_(test_data, beta, last_epoch, 'with regularization')

    beta, last_epoch = gradient_descent(train_data, lr=1e-4, epochs=7000, lambda_=None).gradient_descent()
    draw_(test_data, beta, last_epoch, 'without regularization')


if __name__ == "__main__":
    size = (100, 150)
    loc, cov = get_data_option()
    train_data, test_data = data_generator(size, loc, cov)
    draw(train_data, test_data)

    loc, cov = get_data_option(conv=0)
    train_data, test_data = data_generator(size, loc, cov)
    draw(train_data, test_data)

    train_data, test_data = get_uci_data_Skin_NonSkin()
    draw_3d(train_data, test_data)
```

A.3 牛顿法

Listing 3: newton.py

```python
import numpy as np
import matplotlib.pyplot as plt
from data import *
from uci_data import *
from utils import *
```

```python

class newton:
    def __init__(self, dataset, epochs=70000, delta=1e-7, lambda_=1e-3):
        """
        牛顿法初始化

        Args:
            dataset (tuple): 训练样本
            epochs (int, optional): 最大迭代次数. Defaults to 70000.
            delta (float), optional): 早停策略的参数值. Defaults to 1e-7.
            lambda_ (float), optional): 正则项系数. Defaults to 1e-3.
        """
        self.lambda_ = lambda_
        self.dataset = dataset
        self.X = transform(dataset[0])
        self.Y = dataset[1]
        self.epochs = epochs
        self.delta = delta

    def hessian(self, beta):
        """
        生成黑塞矩阵

        Args:
            beta (array): beta
        Returns:
            array: 黑塞矩阵
        """
        exp = sigmoid(self.X @ beta)
        H = -self.X.T @ self.X * (exp.T @ (exp - 1)) / self.X.shape[0]
        return H

    def newton(self):
        """
        牛顿法

        Returns:
            tuple: (beta, 总迭代次数)
        """
        beta = np.random.normal(size=(self.X.shape[1], 1))
        loss_last = loss = calc_loss(self.X, beta, self.Y, self.lambda_)
        last_epoch = self.epochs
        for epoch in range(self.epochs):
            beta = beta - np.linalg.pinv(self.hessian(beta)) @ calc_gradient(self.X, beta, self
                .Y, self.lambda_)
            loss = calc_loss(self.X, beta, self.Y, self.lambda_)
            if early_stop(loss, loss_last, self.delta):
                last_epoch = epoch
```

```python
                break
            else: loss_last = loss
        return beta, last_epoch

# 画图
def draw(train_data, test_data):
    def draw_(dataset, beta, last_epoch, message):
        acc = calc_accuracy(dataset, beta)
        X_0, X_1 = get_Xs(dataset)
        plt.plot(X_0[:, 0], X_0[:, 1], '.', color='r', label="$y=0$")
        plt.plot(X_1[:, 0], X_1[:, 1], '.', color='b', label="$y=1$")
        plot(beta, 0, 3)
        plt.title('{}, acc={:.2f}, last epoch={}'. format(message, acc, last_epoch))
        plt.legend()

    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    beta, last_epoch = newton(train_data).newton()
    draw_(test_data, beta, last_epoch, 'with regularization')
    plt.subplot(1, 2, 2)
    beta, last_epoch = newton(train_data, lambda_=None).newton()
    draw_(test_data, beta, last_epoch, 'without regularization')

    plt.show()

# 画图
def draw_3d(train_data, test_data):
    def draw_(dataset, beta, last_epoch, message):
        acc = calc_accuracy(dataset, beta)
        fig = plt.figure(figsize=(7, 7))
        X_0, X_1 = get_Xs(dataset)
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(X_0[:, 0], X_0[:, 1], X_0[:, 2], c='r', marker='o')
        ax.scatter(X_1[:, 0], X_1[:, 1], X_1[:, 2], c='b', marker='^')
        plot_3d(beta, 50, 200, ax)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.title('{}, acc={:.2f}, last epoch={}'. format(message, acc, last_epoch))
        plt.legend()
        plt.show()

    beta, last_epoch = newton(train_data).newton()
    draw_(test_data, beta, last_epoch, 'with regularization')

    beta, last_epoch = newton(train_data, lambda_=None).newton()
    draw_(test_data, beta, last_epoch, 'without regularization')

```

```python

if __name__ == "__main__":
    size = (100, 150)
    loc, cov = get_data_option()
    train_data, test_data = data_generator(size, loc, cov)
    draw(train_data, test_data)

    loc, cov = get_data_option(conv=0)
    train_data, test_data = data_generator(size, loc, cov)
    draw(train_data, test_data)

    train_data, test_data = get_uci_data_Skin_NonSkin()
    draw_3d(train_data, test_data)
```

A.4 导入 UCI 数据

Listing 4: uci data.py

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

def get_uci_data_Skin_NonSkin():
    """
    导入Uci数据

    Returns:
        tuple: 数据
    """
    file_path = './uci_data/Skin_NonSkin.txt'
    with  open(file_path, 'r') as f:
        lines = [l[:-1].split('\t') for l in f.readlines()]
        np.random.shuffle(lines)
    X = np.array([[ int(l[i]) for i in  range( len(l) - 1)] for l in lines])
    Y = np.array([[ int(l[i]) - 1 for i in  range( len(l) - 1,  len(l))] for l in lines])
    select_0, select_1, select = 0, 0, []
    for i in  range( len(lines)):
        if select_0 < 1000 and Y[i][0] == 0:
            select.append(i)
            select_0 += 1
        elif select_1 < 1000 and Y[i][0] == 1:
            select.append(i)
            select_1 += 1

    X = X[select]
    Y = Y[select]
```

```
    train_num =  int(X.shape[0] * 0.3)
    train_data = X[:train_num], Y[:train_num]
    test_data = X[train_num:], Y[train_num:]
    return train_data, test_data

# 画图
def show(dataset):
    titles = ['Train Data', 'Test Data']
    for i in  range(2):
        fig = plt.figure(figsize=(7, 7))
        X_0, X_1 = get_Xs(dataset[i])
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(X_0[:, 0], X_0[:, 1], X_0[:, 2], c='r', marker='o')
        ax.scatter(X_1[:, 0], X_1[:, 1], X_1[:, 2], c='b', marker='^')
        plt.title('{}'. format(titles[i]))
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.show()

if __name__ == '__main__':
    dataset = get_uci_data_Skin_NonSkin()
    show(dataset)
```

A.5 辅助代码

Listing 5: utils.py

```
import numpy as np
import matplotlib.pyplot as plt
from data import *

def transform(X):
    """
    将X加一维，以便与beta对应

    Args:
        X (list): X

    Returns:
        list: 转换后的X
    """
    X_transformed = np.zeros((X.shape[0], X.shape[1] + 1))
    for i in  range(X.shape[0]):
        for j in  range(X.shape[1]):
            X_transformed[i][j] = X[i][j]
        X_transformed[i][X.shape[1]] = 1
```

```python
    return X_transformed


def calc_loss(X, beta, Y, lambda_):
    """
    计算loss

    Args:
        X (list): X
        beta (list): beta
        Y (list): Y
        lambda_ (float | None): 正则项系数

    Returns:
        float: loss
    """
    loss = np.mean(-Y.T @ X @ beta - np.log(1 - sigmoid(X @ beta)))
    if lambda_ != None:
        loss += lambda_ / 2 * np.mean(beta.T @ beta)
    return loss

def calc_gradient(X, beta, Y, lambda_):
    """
    计算梯度

    Args:
        X (list): X
        beta (list): beta
        Y (list): Y
        lambda_ (float | None): 正则项系数

    Returns:
        list: 梯度
    """
    gradient = (X.T @ model(X, beta) - X.T @ Y) / X.shape[0]

    if lambda_ != None:
        gradient += lambda_ * beta
    return gradient

def sigmoid(x):
    """
    sigmoid函数

    Args:
        x (list): 输入

    Returns:
```

```
        list: 输出
    """
    x_ravel = x.ravel()
    length =  len(x_ravel)
    y = []
    for index in  range(length):
        if x_ravel[index] >= 0:
            y.append(1.0 / (1 + np.exp(-x_ravel[index])))
        else:
            y.append(np.exp(x_ravel[index]) / (np.exp(x_ravel[index]) + 1))
    return np.array(y).reshape(x.shape)

def model(X, beta):
    """
    形式化为模型

    Args:
        X (list): X
        beta (list): beta

    Returns:
        list: 结果
    """
    output = sigmoid(X @ beta)
    return output

def calc_accuracy(dataset, beta):
    """
    计算准确率

    Args:
        dataset (tuple): 数据集
        beta (list): beta

    Returns:
        float: 准确率
    """
    X, Y = transform(dataset[0]), dataset[1].flatten()
    output = model(X, beta)
    output = np.array([1 if x >= 0.5 else 0 for x in output])
    output = np.logical_xor(output, Y)
    accuracy = ( len(Y) - np. sum(output, axis=0)) /  len(Y)
    return accuracy

def early_stop(loss, loss_last, delta):
    """
    早停策略

```

```
    Args:
        loss (float): 当前的loss
        loss_last (float): 上一个loss
        delta (float): 早停策略参数

    Returns:
        bool: 是否停止
    """
    return  abs(loss - loss_last) < delta


class lr_scheduler_MultiStep:
    def __init__(self, epochs, lr):
        """
        学习率下降策略

        Args:
            epochs (int): 总迭代轮数
            lr (float): 初始学习率
        """
        self.epochs = epochs
        self.lr = lr
        self.gamma = 0.1
        self.milestones = [ int(epochs * 0.5),  int(epochs * 0.75)]
    def step(self, epoch):
        """
        更新学习率

        Args:
            epoch (int): 当前迭代轮数

        Returns:
            float: 新的学习率
        """
        if epoch in self.milestones:
            self.lr *= self.gamma
        return self.lr

def get_Xs(dataset):
    """
    将X按照不同标签分开

    Args:
        dataset (tuple): 数据集

    Returns:
        tuple: 分开后的X
    """
```

```python
    X, Y = dataset
    X_0, X_1 = [], []
    for index in  range(Y.shape[0]):
        if Y[index][0] == 0: X_0.append(X[index])
        else: X_1.append(X[index])
    return np.array(X_0), np.array(X_1)

# 画图支持函数
def plot(beta, start, end):
    x0 = np.linspace(start, end, 1000)
    x1 = -(beta[0] * x0 + beta[2]) / beta[1]
    plt.plot(x0, x1, label="discriminant")

# 画图支持函数
def plot_3d(beta, start, end, ax):
    x0 = np.linspace(start, end, 1000)
    x1 = np.linspace(start, end, 1000)
    x0, x1 = np.meshgrid(x0, x1)
    x2 = -(beta[0] * x0 + beta[1] * x1 + beta[3]) / beta[2]
    ax.plot_surface(x0, x1, x2, linewidth=0, antialiased=False)
```