

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

$$y = g(w^T x + b) \quad (1)$$

，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

$$y = \frac{1}{1 + e^{-z}} \quad (2)$$

将其代入式1得到

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

将其取对数，则可以变化为

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

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

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

由式3可得

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

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

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

要将 $l(w, b)$ 最大化, 也即最小化

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

则我们的任务目标就变为

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

若我们加入正则项, 则有

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

相应地, 加入正则项后的任务目标变为

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

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

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

则按照梯度下降法, 每一步我们都按照

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

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

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

每一步我们按照

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

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

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

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

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

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

$$\nabla J(\beta) = 0 \quad (18)$$

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

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

则迭代式为

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

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

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

则迭代式为

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

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。

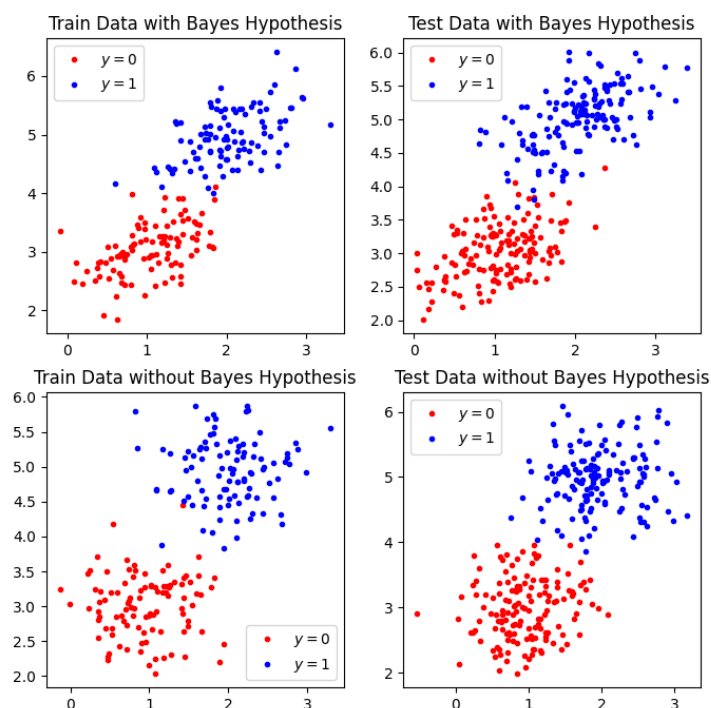


图 1

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。

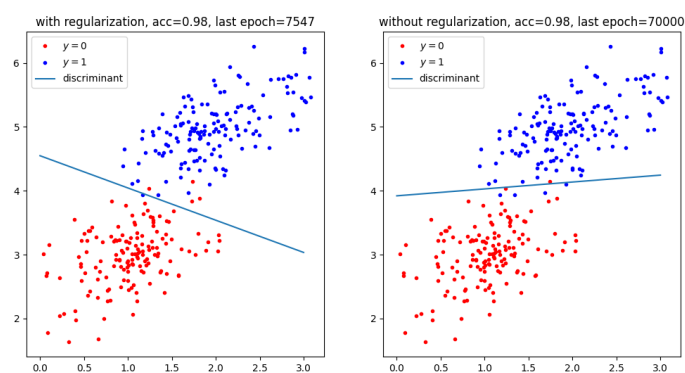


图 2

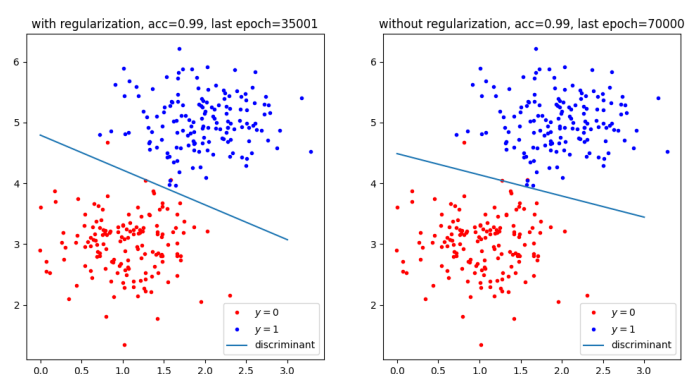


图 3

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。

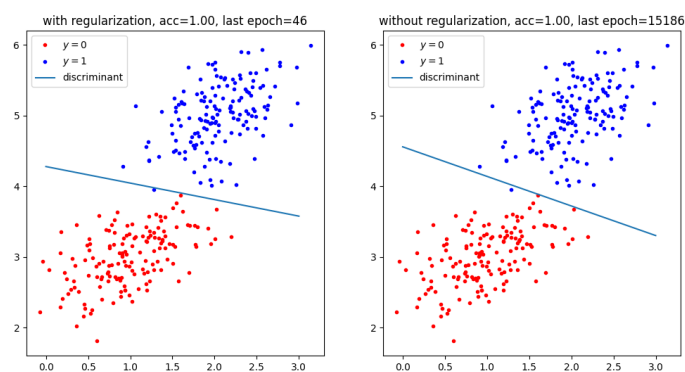


图 4

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。

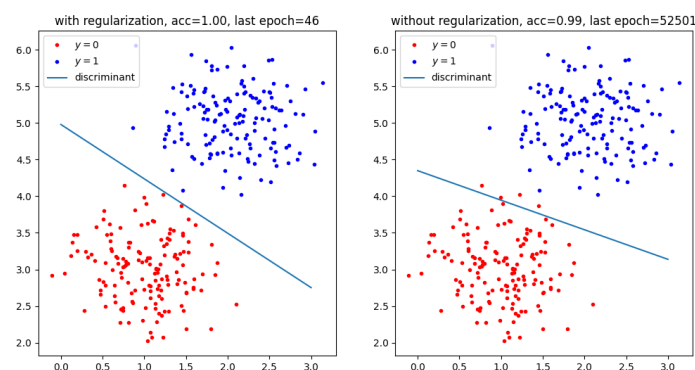


图 5

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。

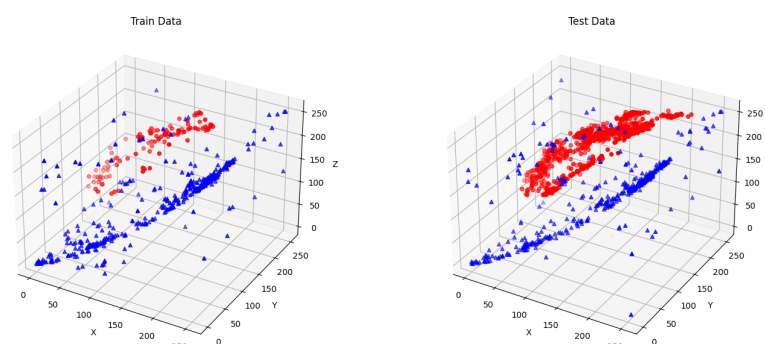


图 6

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。

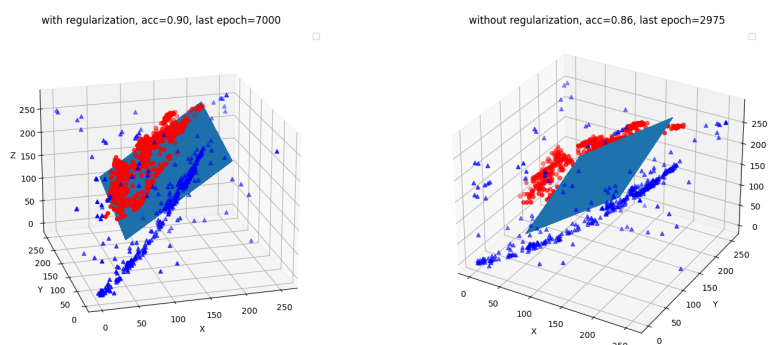


图 7

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。

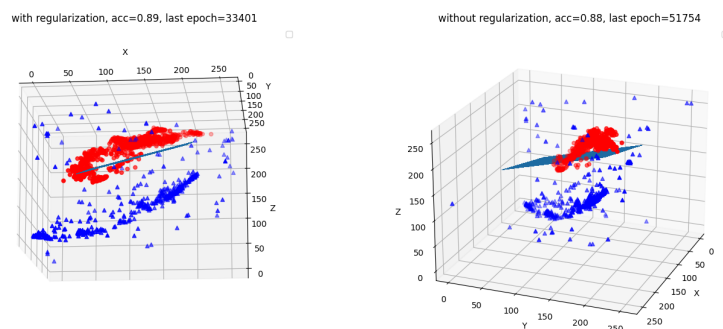


图 8

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

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from utils import *
4
5
6 def data_generator(size=(100, 150), loc=np.array([[1, 3],[2, 5]]), cov=[[0.4, 0.2], [0.2,
7     0.4]]):
8     """
9     生成数据
10
11     Args:
12         size (tuple, optional): 训练集和测试集的数据大小. Defaults to (100, 150).
13         loc (list, optional): 均值. Defaults to np.array([[1, 3],[2, 5]]).
14         cov (list, optional): 协方差. Defaults to [[0.4, 0.2], [0.2, 0.4]].
15     """
16     def generate(siz):
17         X = np.zeros((siz * 2, 2))
18         Y = np.zeros((siz * 2, 1))
19         X[:siz, :] = np.random.multivariate_normal(loc[0], cov, size=siz)
20         X[siz:, :] = np.random.multivariate_normal(loc[1], cov, size=siz)
21         Y[siz:] = 1
22         return X, Y
23
24     train_data, test_data = generate(size[0]), generate(size[1])
25     return train_data, test_data
26
27 # 画图
28 def show(dataset):
29     plt.figure(figsize=(8, 8))
30     messages = ['with Bayes Hypothesis', 'without Bayes Hypothesis']
31     titles = ['Train Data', 'Test Data']
32     for i in range(2):
33         for j in range(2):
34             X_0, X_1 = get_Xs(dataset[i][j])
35             plt.subplot(2, 2, i * 2 + j + 1)
36             plt.plot(X_0[:, 0], X_0[:, 1], '.', color='r', label="$y=0$")
37             plt.plot(X_1[:, 0], X_1[:, 1], '.', color='b', label="$y=1$")
38             plt.title('{} {}'.format(titles[j], messages[i]))
39             plt.legend()
40     plt.show()
41
42 def get_data_option(loc1=(1, 3), loc2=(2, 5), scale=0.2, conv=0.1):
43     """
```

```

42     生成与数据生成函数相匹配的参数
43
44     Args:
45         loc1 (tuple, optional): 反例的均值. Defaults to (1, 3).
46         loc2 (tuple, optional): 正例的均值. Defaults to (2, 5).
47         scale (float, optional): 离散程度. Defaults to 0.2.
48         conv (float, optional): 相关系数. Defaults to 0.1.
49
50     Returns:
51         tuple: 与数据生成函数相匹配的参数
52     """
53     loc = np.array([[loc1[0], loc1[1]], [loc2[0], loc2[1]]])
54     cov = [[scale, conv], [conv, scale]]
55     return loc, cov
56
57 if __name__ == '__main__':
58     size = (100, 150)
59     loc, cov = get_data_option()
60     dataset0 = data_generator(size, loc, cov)
61     loc, cov = get_data_option(conv=0)
62     dataset1 = data_generator(size, loc, cov)
63     show((dataset0, dataset1))

```

A.2 梯度下降法

Listing 2: gradient_descent.py

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from data import *
4  from uci_data import *
5  from utils import *
6
7  class gradient_descent:
8      def __init__(self, dataset, lr=1e-1, epochs=70000, delta=1e-7, lambda_=1e-3):
9          """
10             梯度下降法初始化
11
12             Args:
13                 dataset (tuple): 训练样本
14                 lr (float, optional): 学习率. Defaults to 1e-1.
15                 epochs (int, optional): 最大迭代次数. Defaults to 70000.
16                 delta (float, optional): 早停策略的参数值. Defaults to 1e-7.
17                 lambda_ (float, optional): 正则项系数. Defaults to 1e-3.
18             """
19             self.lambda_ = lambda_
20             self.dataset = dataset

```

```

21     self.X = transform(dataset[0])
22     self.Y = dataset[1]
23     self.lr = lr
24     self.epochs = epochs
25     self.delta = delta
26
27     def gradient_descent(self):
28         """
29         梯度下降法
30
31         Returns:
32             tuple: (beta, 总迭代次数)
33         """
34         beta = np.random.normal(size=(self.X.shape[1], 1))
35         lr_scheduler = lr_scheduler.MultiStep(self.epochs, self.lr)
36         loss_last = loss = calc_loss(self.X, beta, self.Y, self.lambda_)
37         last_epoch = self.epochs
38         for epoch in range(self.epochs):
39             beta = beta - self.lr * calc_gradient(self.X, beta, self.Y, self.lambda_)
40             loss = calc_loss(self.X, beta, self.Y, self.lambda_)
41             if early_stop(loss, loss_last, self.delta):
42                 last_epoch = epoch
43                 break
44             else: loss_last = loss
45             self.lr = lr_scheduler.step(epoch)
46         return beta, last_epoch
47
48     # 画图
49     def draw(train_data, test_data):
50         def draw_(dataset, beta, last_epoch, message):
51             acc = calc_accuracy(dataset, beta)
52             X_0, X_1 = get_Xs(dataset)
53             plt.plot(X_0[:, 0], X_0[:, 1], '.', color='r', label="$y=0$")
54             plt.plot(X_1[:, 0], X_1[:, 1], '.', color='b', label="$y=1$")
55             plot(beta, 0, 3)
56             plt.title('{}, acc={:.2f}, last epoch={}'.format(message, acc, last_epoch))
57             plt.legend()
58
59         plt.figure(figsize=(12, 6))
60         plt.subplot(1, 2, 1)
61         beta, last_epoch = gradient_descent(train_data).gradient_descent()
62         draw_(test_data, beta, last_epoch, 'with regularization')
63         plt.subplot(1, 2, 2)
64         beta, last_epoch = gradient_descent(train_data, lambda_=None).gradient_descent()
65         draw_(test_data, beta, last_epoch, 'without regularization')
66
67         plt.show()
68

```

```

69 # 画图
70 def draw_3d(train_data, test_data):
71     def draw_(dataset, beta, last_epoch, message):
72         acc = calc_accuracy(dataset, beta)
73         fig = plt.figure(figsize=(7, 7))
74         X_0, X_1 = get_Xs(dataset)
75         ax = fig.add_subplot(111, projection='3d')
76         ax.scatter(X_0[:, 0], X_0[:, 1], X_0[:, 2], c='r', marker='o')
77         ax.scatter(X_1[:, 0], X_1[:, 1], X_1[:, 2], c='b', marker='^')
78         plot_3d(beta, 50, 200, ax)
79         ax.set_xlabel('X')
80         ax.set_ylabel('Y')
81         ax.set_zlabel('Z')
82         plt.title('{}, acc={:.2f}, last epoch={}'.format(message, acc, last_epoch))
83         plt.legend()
84         plt.show()
85
86     beta, last_epoch = gradient_descent(train_data, lr=1e-4, epochs=7000).gradient_descent()
87     draw_(test_data, beta, last_epoch, 'with regularization')
88
89     beta, last_epoch = gradient_descent(train_data, lr=1e-4, epochs=7000, lambda_=None).
90         gradient_descent()
91     draw_(test_data, beta, last_epoch, 'without regularization')
92
93 if __name__ == "__main__":
94     size = (100, 150)
95     loc, cov = get_data_option()
96     train_data, test_data = data_generator(size, loc, cov)
97     draw(train_data, test_data)
98
99     loc, cov = get_data_option(conv=0)
100    train_data, test_data = data_generator(size, loc, cov)
101    draw(train_data, test_data)
102
103    train_data, test_data = get_uci_data_Skin_NonSkin()
104    draw_3d(train_data, test_data)

```

A.3 牛顿法

Listing 3: newton.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from data import *
4 from uci_data import *
5 from utils import *

```

```

6
7 class newton:
8     def __init__(self, dataset, epochs=70000, delta=1e-7, lambda_=1e-3):
9         """
10         牛顿法初始化
11
12         Args:
13             dataset (tuple): 训练样本
14             epochs (int, optional): 最大迭代次数. Defaults to 70000.
15             delta (float, optional): 早停策略的参数值. Defaults to 1e-7.
16             lambda_ (float, optional): 正则项系数. Defaults to 1e-3.
17         """
18         self.lambda_ = lambda_
19         self.dataset = dataset
20         self.X = transform(dataset[0])
21         self.Y = dataset[1]
22         self.epochs = epochs
23         self.delta = delta
24
25     def hessian(self, beta):
26         """
27         生成黑塞矩阵
28
29         Args:
30             beta (array): beta
31         Returns:
32             array: 黑塞矩阵
33         """
34         exp = sigmoid(self.X @ beta)
35         H = -self.X.T @ self.X * (exp.T @ (exp - 1)) / self.X.shape[0]
36         return H
37
38     def newton(self):
39         """
40         牛顿法
41
42         Returns:
43             tuple: (beta, 总迭代次数)
44         """
45         beta = np.random.normal(size=(self.X.shape[1], 1))
46         loss_last = loss = calc_loss(self.X, beta, self.Y, self.lambda_)
47         last_epoch = self.epochs
48         for epoch in range(self.epochs):
49             beta = beta - np.linalg.pinv(self.hessian(beta)) @ calc_gradient(self.X, beta, self
                    .Y, self.lambda_)
50             loss = calc_loss(self.X, beta, self.Y, self.lambda_)
51             if early_stop(loss, loss_last, self.delta):
52                 last_epoch = epoch

```

```

53         break
54     else: loss_last = loss
55     return beta, last_epoch
56
57 # 画图
58 def draw(train_data, test_data):
59     def draw_(dataset, beta, last_epoch, message):
60         acc = calc_accuracy(dataset, beta)
61         X_0, X_1 = get_Xs(dataset)
62         plt.plot(X_0[:, 0], X_0[:, 1], '.', color='r', label="$y=0$")
63         plt.plot(X_1[:, 0], X_1[:, 1], '.', color='b', label="$y=1$")
64         plot(beta, 0, 3)
65         plt.title('{} , acc={:.2f}, last epoch={}'.format(message, acc, last_epoch))
66         plt.legend()
67
68     plt.figure(figsize=(12, 6))
69     plt.subplot(1, 2, 1)
70     beta, last_epoch = newton(train_data).newton()
71     draw_(test_data, beta, last_epoch, 'with regularization')
72     plt.subplot(1, 2, 2)
73     beta, last_epoch = newton(train_data, lambda=None).newton()
74     draw_(test_data, beta, last_epoch, 'without regularization')
75
76     plt.show()
77
78 # 画图
79 def draw_3d(train_data, test_data):
80     def draw_(dataset, beta, last_epoch, message):
81         acc = calc_accuracy(dataset, beta)
82         fig = plt.figure(figsize=(7, 7))
83         X_0, X_1 = get_Xs(dataset)
84         ax = fig.add_subplot(111, projection='3d')
85         ax.scatter(X_0[:, 0], X_0[:, 1], X_0[:, 2], c='r', marker='o')
86         ax.scatter(X_1[:, 0], X_1[:, 1], X_1[:, 2], c='b', marker='^')
87         plot_3d(beta, 50, 200, ax)
88         ax.set_xlabel('X')
89         ax.set_ylabel('Y')
90         ax.set_zlabel('Z')
91         plt.title('{} , acc={:.2f}, last epoch={}'.format(message, acc, last_epoch))
92         plt.legend()
93         plt.show()
94
95     beta, last_epoch = newton(train_data).newton()
96     draw_(test_data, beta, last_epoch, 'with regularization')
97
98     beta, last_epoch = newton(train_data, lambda=None).newton()
99     draw_(test_data, beta, last_epoch, 'without regularization')
100

```

```

101
102 if __name__ == "__main__":
103     size = (100, 150)
104     loc, cov = get_data_option()
105     train_data, test_data = data_generator(size, loc, cov)
106     draw(train_data, test_data)
107
108     loc, cov = get_data_option(conv=0)
109     train_data, test_data = data_generator(size, loc, cov)
110     draw(train_data, test_data)
111
112     train_data, test_data = get_uci_data_Skin_NonSkin()
113     draw_3d(train_data, test_data)

```

A.4 导入 UCI 数据

Listing 4: uci data.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from utils import *
4 from mpl_toolkits.mplot3d import Axes3D
5
6 def get_uci_data_Skin_NonSkin():
7     """
8     导入Uci数据
9
10    Returns:
11        tuple: 数据
12    """
13    file_path = './uci_data/Skin_NonSkin.txt'
14    with open(file_path, 'r') as f:
15        lines = [l[:-1].split('\t') for l in f.readlines()]
16        np.random.shuffle(lines)
17    X = np.array([[int(l[i]) for i in range(len(l) - 1)] for l in lines])
18    Y = np.array([[int(l[i]) - 1 for i in range(len(l) - 1, len(l))] for l in lines])
19    select_0, select_1, select = 0, 0, []
20    for i in range(len(lines)):
21        if select_0 < 1000 and Y[i][0] == 0:
22            select.append(i)
23            select_0 += 1
24        elif select_1 < 1000 and Y[i][0] == 1:
25            select.append(i)
26            select_1 += 1
27
28    X = X[select]
29    Y = Y[select]

```

```

30     train_num = int(X.shape[0] * 0.3)
31     train_data = X[:train_num], Y[:train_num]
32     test_data = X[train_num:], Y[train_num:]
33     return train_data, test_data
34
35 # 画图
36 def show(dataset):
37     titles = ['Train Data', 'Test Data']
38     for i in range(2):
39         fig = plt.figure(figsize=(7, 7))
40         X_0, X_1 = get_Xs(dataset[i])
41         ax = fig.add_subplot(111, projection='3d')
42         ax.scatter(X_0[:, 0], X_0[:, 1], X_0[:, 2], c='r', marker='o')
43         ax.scatter(X_1[:, 0], X_1[:, 1], X_1[:, 2], c='b', marker='^')
44         plt.title('{}'.format(titles[i]))
45         ax.set_xlabel('X')
46         ax.set_ylabel('Y')
47         ax.set_zlabel('Z')
48         plt.show()
49
50 if __name__ == '__main__':
51     dataset = get_uci_data_Skin_NonSkin()
52     show(dataset)

```

A.5 辅助代码

Listing 5: utils.py

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from data import *
4
5 def transform(X):
6     """
7     将X加一维，以便与beta对应
8
9     Args:
10         X (list): X
11
12     Returns:
13         list: 转换后的X
14     """
15     X_transformed = np.zeros((X.shape[0], X.shape[1] + 1))
16     for i in range(X.shape[0]):
17         for j in range(X.shape[1]):
18             X_transformed[i][j] = X[i][j]
19             X_transformed[i][X.shape[1]] = 1

```



```

20     return X_transformed
21
22
23 def calc_loss(X, beta, Y, lambda_):
24     """
25     计算loss
26
27     Args:
28         X (list): X
29         beta (list): beta
30         Y (list): Y
31         lambda_ (float | None): 正则项系数
32
33     Returns:
34         float: loss
35     """
36     loss = np.mean(-Y.T @ X @ beta - np.log(1 - sigmoid(X @ beta)))
37     if lambda_ != None:
38         loss += lambda_ / 2 * np.mean(beta.T @ beta)
39     return loss
40
41 def calc_gradient(X, beta, Y, lambda_):
42     """
43     计算梯度
44
45     Args:
46         X (list): X
47         beta (list): beta
48         Y (list): Y
49         lambda_ (float | None): 正则项系数
50
51     Returns:
52         list: 梯度
53     """
54     gradient = (X.T @ model(X, beta) - X.T @ Y) / X.shape[0]
55
56     if lambda_ != None:
57         gradient += lambda_ * beta
58     return gradient
59
60 def sigmoid(x):
61     """
62     sigmoid函数
63
64     Args:
65         x (list): 输入
66
67     Returns:

```

```

68         list: 输出
69     """
70     x_ravel = x.ravel()
71     length = len(x_ravel)
72     y = []
73     for index in range(length):
74         if x_ravel[index] >= 0:
75             y.append(1.0 / (1 + np.exp(-x_ravel[index])))
76         else:
77             y.append(np.exp(x_ravel[index]) / (np.exp(x_ravel[index]) + 1))
78     return np.array(y).reshape(x.shape)
79
80 def model(X, beta):
81     """
82     形式化为模型
83
84     Args:
85         X (list): X
86         beta (list): beta
87
88     Returns:
89         list: 结果
90     """
91     output = sigmoid(X @ beta)
92     return output
93
94 def calc_accuracy(dataset, beta):
95     """
96     计算准确率
97
98     Args:
99         dataset (tuple): 数据集
100         beta (list): beta
101
102     Returns:
103         float: 准确率
104     """
105     X, Y = transform(dataset[0]), dataset[1].flatten()
106     output = model(X, beta)
107     output = np.array([1 if x >= 0.5 else 0 for x in output])
108     output = np.logical_xor(output, Y)
109     accuracy = (len(Y) - np.sum(output, axis=0)) / len(Y)
110     return accuracy
111
112 def early_stop(loss, loss_last, delta):
113     """
114     早停策略
115

```

```

116     Args:
117         loss (float): 当前的loss
118         loss_last (float): 上一个loss
119         delta (float): 早停策略参数
120
121     Returns:
122         bool: 是否停止
123     """
124     return abs(loss - loss_last) < delta
125
126
127 class lr_scheduler_MultiStep:
128     def __init__(self, epochs, lr):
129         """
130         学习率下降策略
131
132         Args:
133             epochs (int): 总迭代轮数
134             lr (float): 初始学习率
135         """
136         self.epochs = epochs
137         self.lr = lr
138         self.gamma = 0.1
139         self.milestones = [int(epochs * 0.5), int(epochs * 0.75)]
140     def step(self, epoch):
141         """
142         更新学习率
143
144         Args:
145             epoch (int): 当前迭代轮数
146
147         Returns:
148             float: 新的学习率
149         """
150         if epoch in self.milestones:
151             self.lr *= self.gamma
152         return self.lr
153
154     def get_Xs(dataset):
155         """
156         将X按照不同标签分开
157
158         Args:
159             dataset (tuple): 数据集
160
161         Returns:
162             tuple: 分开后的X
163         """

```

```

164     X, Y = dataset
165     X_0, X_1 = [], []
166     for index in range(Y.shape[0]):
167         if Y[index][0] == 0: X_0.append(X[index])
168         else: X_1.append(X[index])
169     return np.array(X_0), np.array(X_1)
170
171 # 画图支持函数
172 def plot(beta, start, end):
173     x0 = np.linspace(start, end, 1000)
174     x1 = -(beta[0] * x0 + beta[2]) / beta[1]
175     plt.plot(x0, x1, label="discriminant")
176
177 # 画图支持函数
178 def plot_3d(beta, start, end, ax):
179     x0 = np.linspace(start, end, 1000)
180     x1 = np.linspace(start, end, 1000)
181     x0, x1 = np.meshgrid(x0, x1)
182     x2 = -(beta[0] * x0 + beta[1] * x1 + beta[3]) / beta[2]
183     ax.plot_surface(x0, x1, x2, linewidth=0, antialiased=False)

```