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

# 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。

# 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

# 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 。即利用 ， 以及各个维度之间的条件独立假设来计算。

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，其中是一个向量，，。我们要求解的其实是。假设的各维度在的条件下是独立分布的，并且，。下面对进行转化。



因为符合伯努利分布，可令，带入得



又由于各个维度的条件概率均服从高斯分布，因此



令，，则有



进而有



使用极大条件似然估计来计算损失函数，有



我们要求，也就是求，用梯度下降法求解。为了消除负号，我们令，为真正的损失函数。

这时损失函数可能存在上溢出的问题，即当数据量较大时，可能出现上溢出的问题。对此，我们对进行归一化处理。记归一化后的损失函数为。



1. 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的梯度，然后结合学习率的大小沿着负梯度的方向对进行迭代。下面分别是损失函数关于的梯度和每次迭代的公式。





向量形式



加入正则项后





其中



1. 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



设 有二阶连续偏导数，若第次迭代值为，则可以将在附近进行二阶泰勒展开。



其中是梯度向量，是海森矩阵



当取到极值时，。若此时是正定的，则极值为极小值。我们对关于求导。



在极值点处，即



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



其中





1. 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。

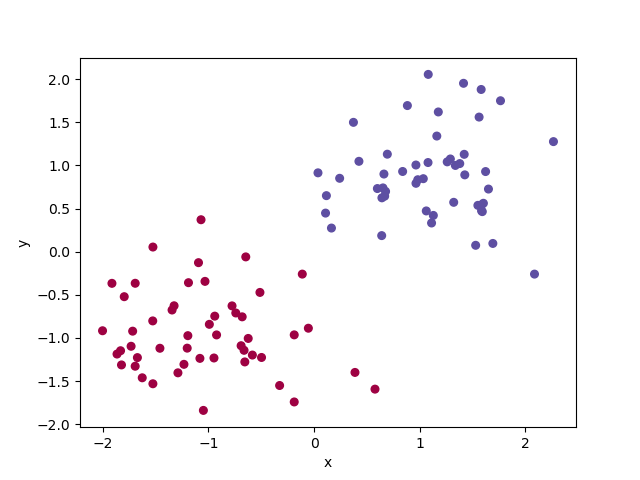


图 1 训练集数据示意图

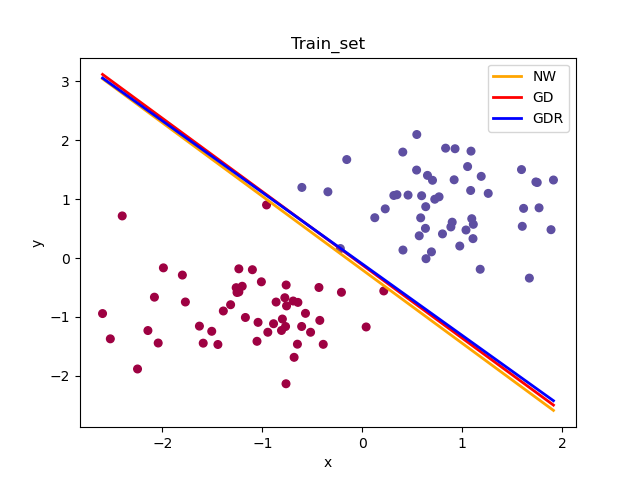
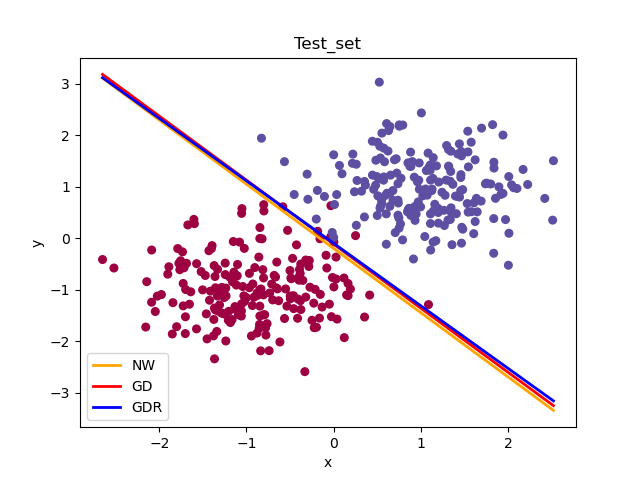
1. 计算分类准确率

在训练完成后，会得到对应的参数向量，可以利用这个参数对数据进行分类，即计算和，相关的计算公式在上文已经给出。若且在样本中或者且在样本中，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。

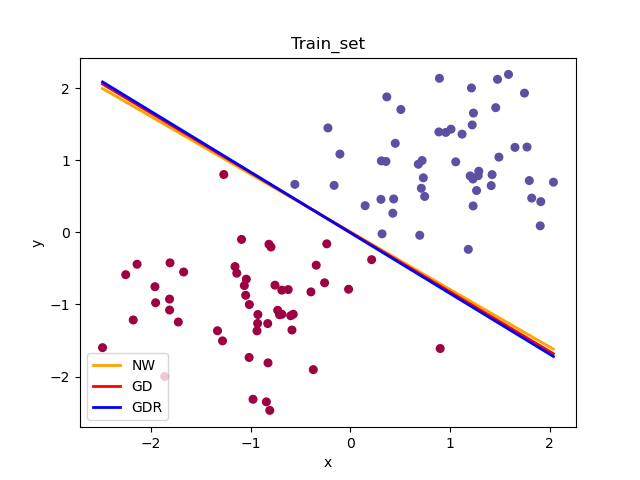
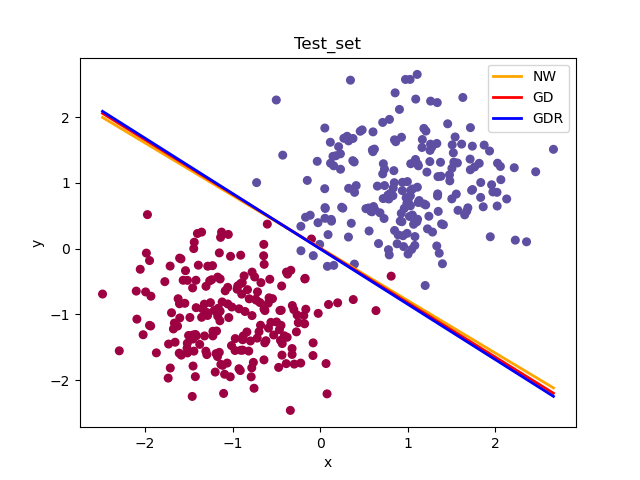
# 实验结果与分析

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。

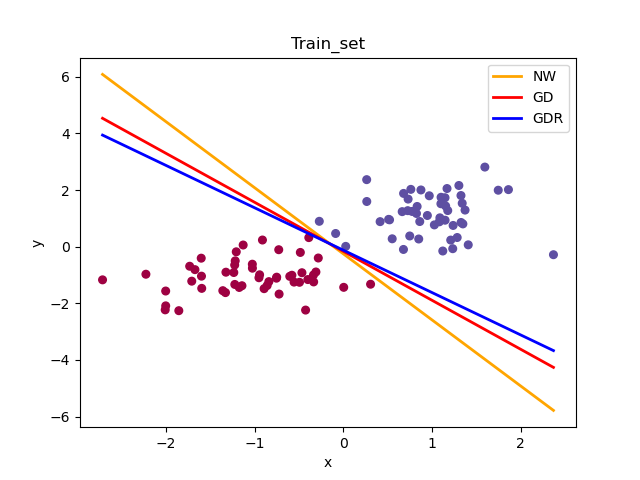
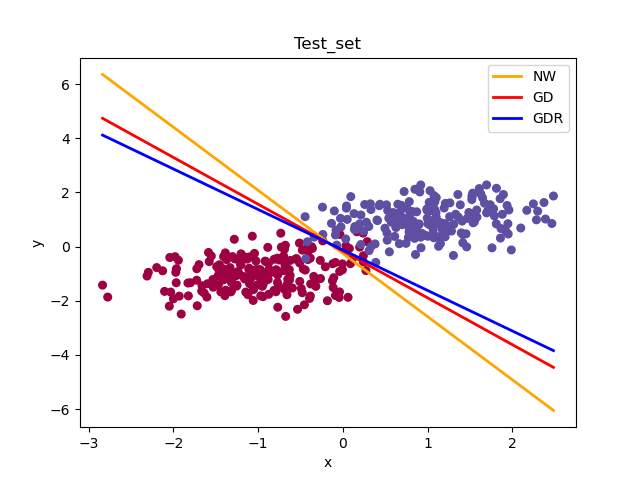
在第一次测试中，牛顿法、梯度下降法（不带正则项）和梯度下降法（带正则项）在训练集上的准确率分别为1.0，1.0，0.99，在测试集上的准确率分别为0.9825，0.985，0.9825。

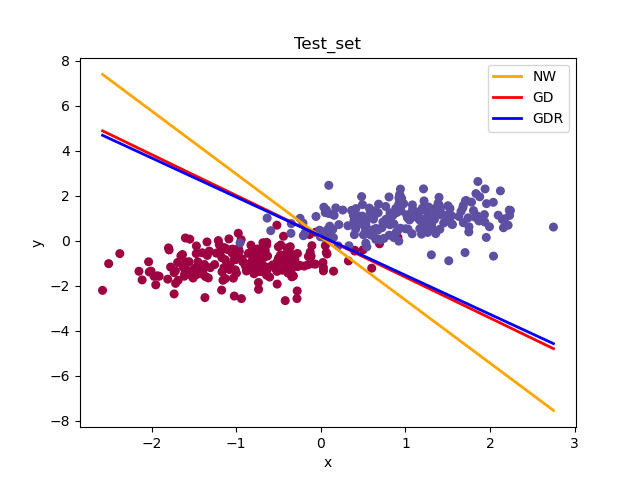
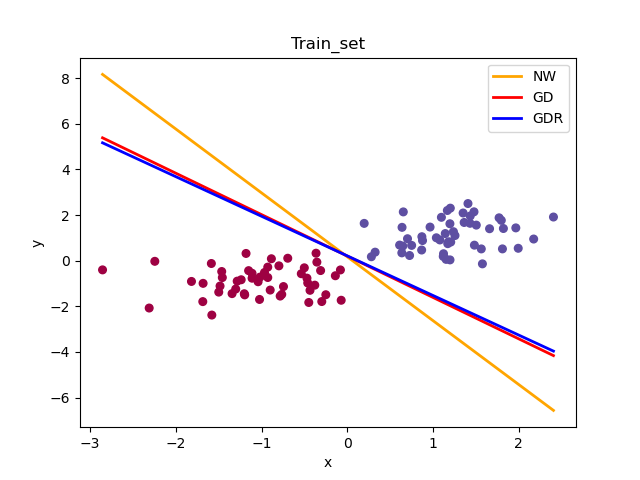
在第二次测试中，牛顿法、梯度下降法（不带正则项）和梯度下降法（带正则项）在训练集上的准确率均为1.0，在测试集上的准确率均为0.985。

1. 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。

1. 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。

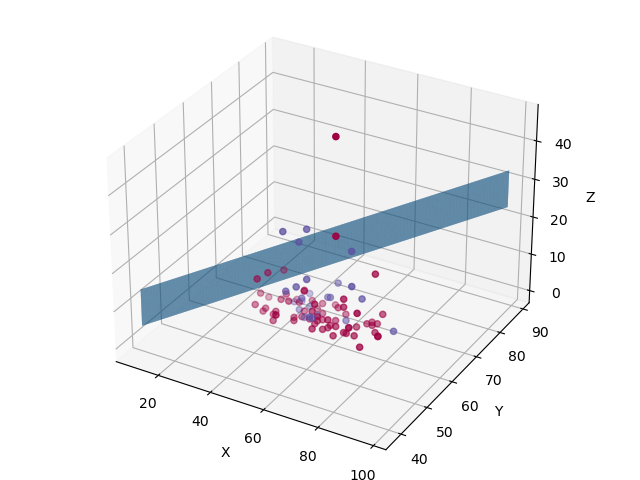


图 2 训练集

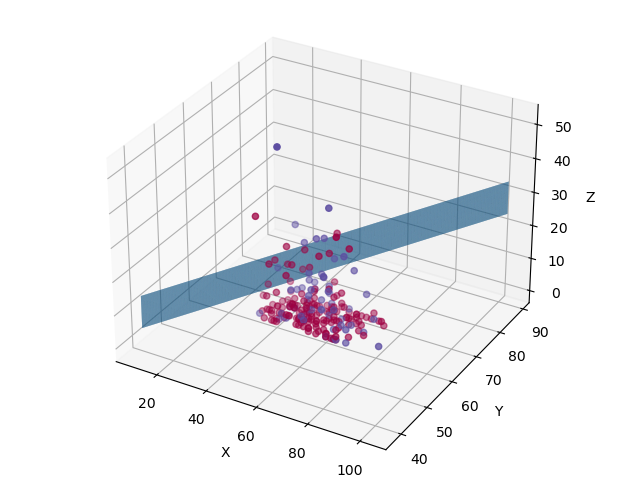


图 3 测试集

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。

# 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。

# 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.

# 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')