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：1190202002

姓名： 李艺峰

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。

1. 实验要求及实验环境

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。

环境：

Windows 10 20H2

Python 3.6.6

Visual Studio Code 1.60.2

1. 设计思想（本程序中的用到的主要算法及数据结构）
2. 实验原理：从训练数据集学习到一个预测新样本所属类别的分类器。其中X是样本，假设进行0/1二分类,Y是所属类别为0或1。其中关于Y条件独立

因此

=

=



则 =

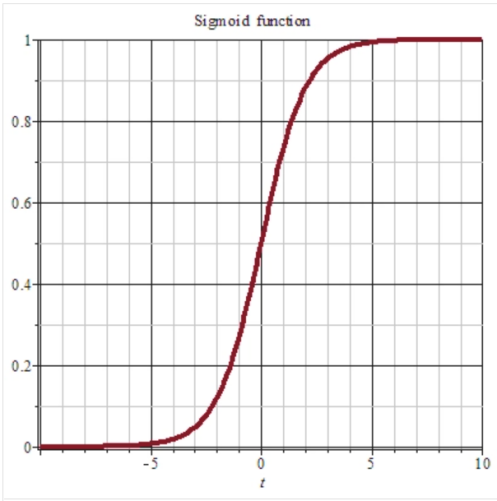
其中

向量化



则

通过以上变换，将概率形式变换为sigmoid函数，将一个实数映射到(0,1)的区间，sigmoid函数平滑、易于求导。。

因此 ，，当时认为X属于Y=0类，当时认为X属于Y=1类

1. 确定损失值loss函数

2.1用最大似然估计确定无正则项loss

思想：使得参数W下，概率最大。只需计算P（Y|X，W）即可。



2.2用贝叶斯估计确定带正则项loss

思想：给随机变量W添加先验P（W），使得P(W)P(Y|X,W)最大。

设



1. 求loss函数中参数W(梯度下降法）

思想：沿着函数loss当前点W的梯度反方向就是loss当前的最速下降方向，每次W沿着梯度走一定的距离，直至loss不再下降。

为了防止溢出，将grad归一化：





这样逐渐逼近loss最低点，直至loss(W)-loss（W’）<要求精度。

1. 生成数据

使用二维数据，若满足朴素贝叶斯，则X的各维度数据关于Y条件独立，则协方差矩阵为，是的方差，是的方差。

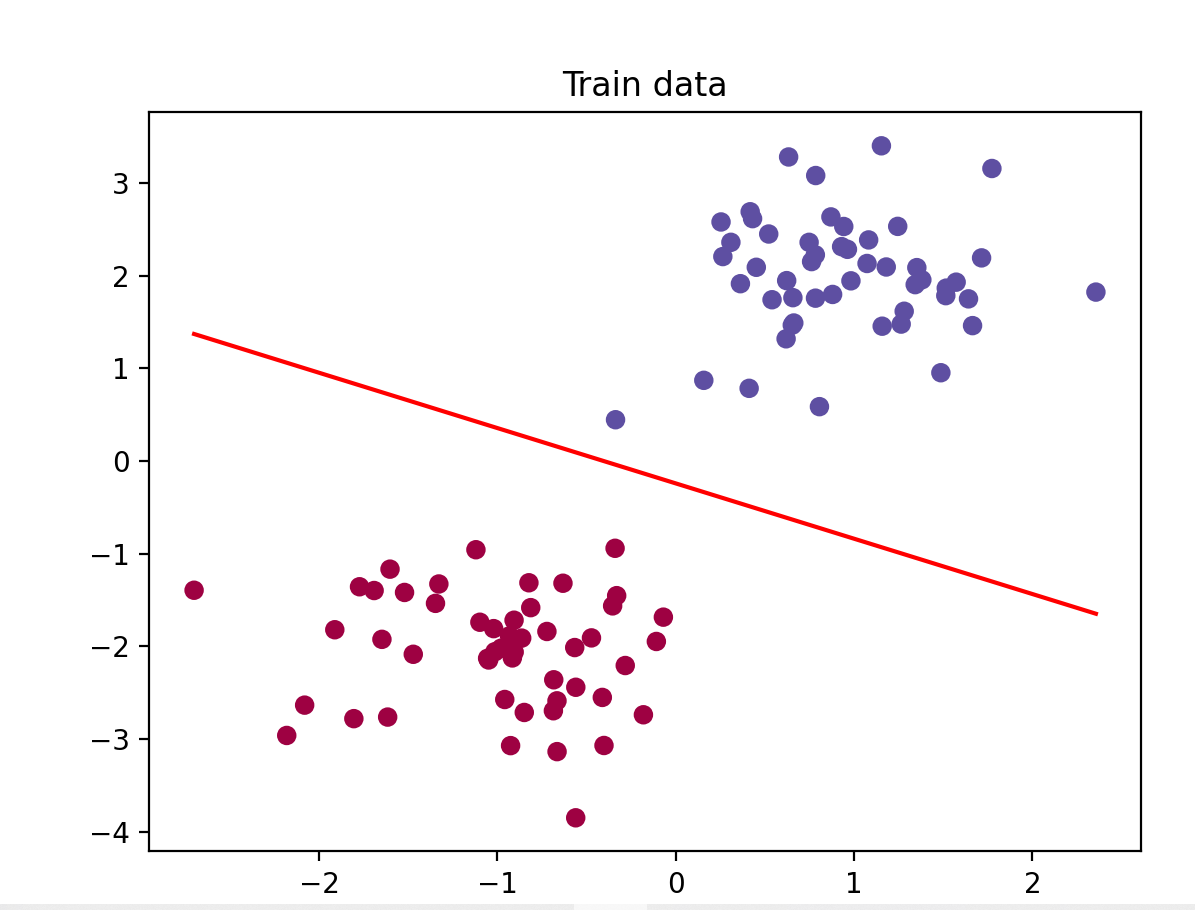
若不满足朴素贝叶斯，则

1. 实验结果与分析

1.正例均值[1,2],反例均值[-1,-2],满足朴素贝叶斯协方差矩阵，不满足朴素贝叶斯，协方差矩阵

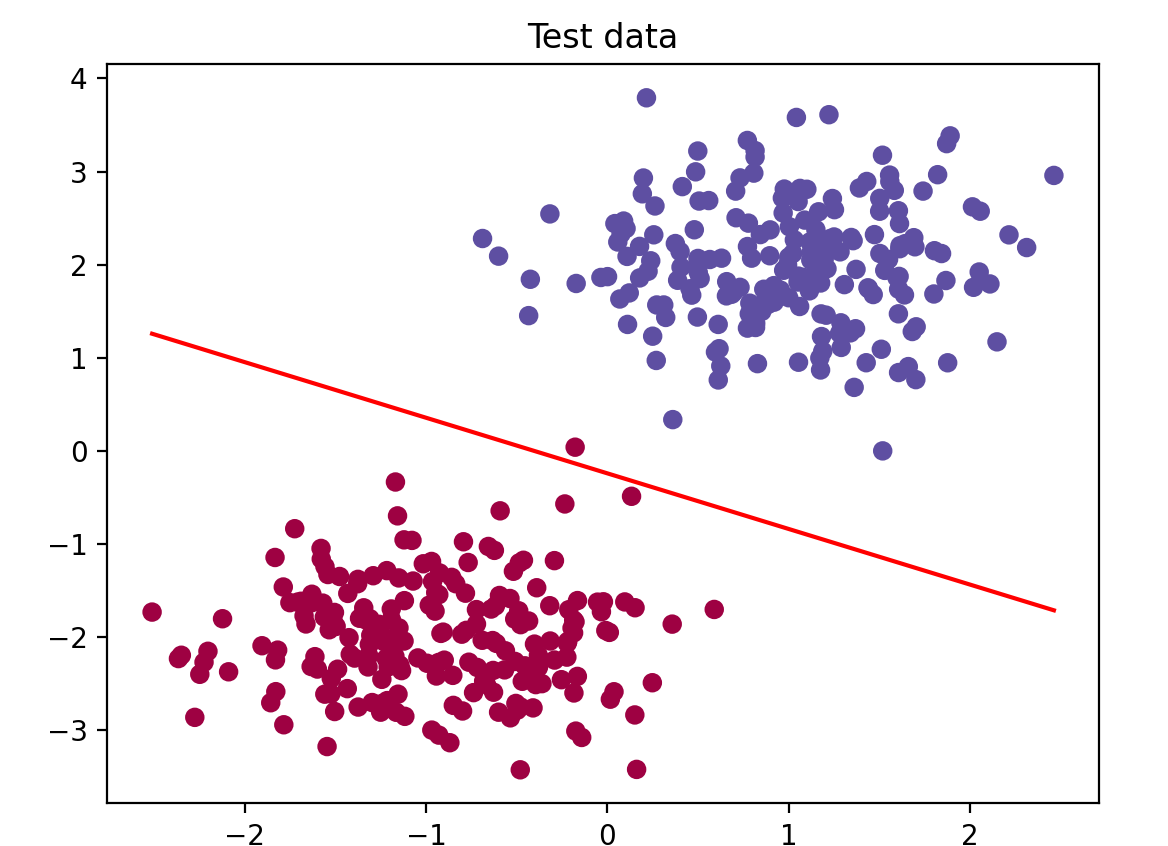
（1）满足朴素贝叶斯，无惩罚项：

迭代次数：11575



训练集正确率：1.0

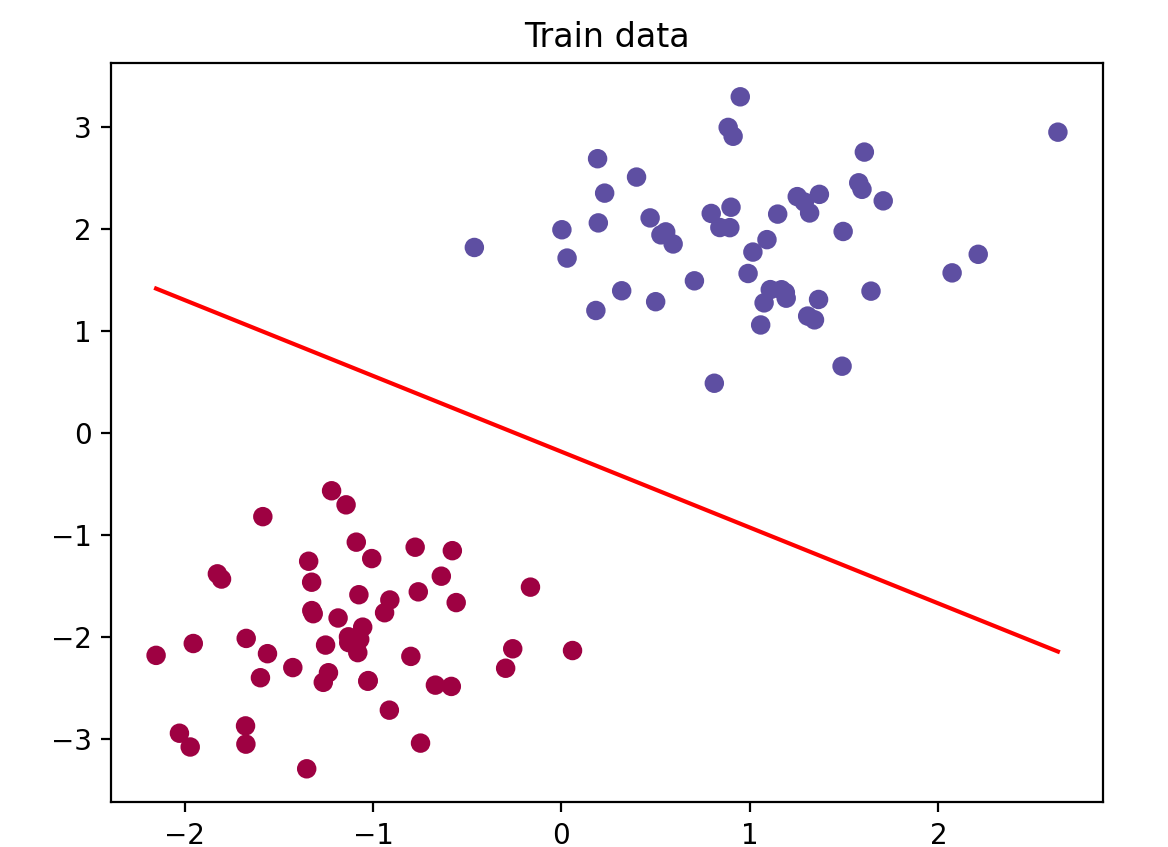
分类函数 y =-0.5972 x - 0.2415



测试集正确率: 0.9975

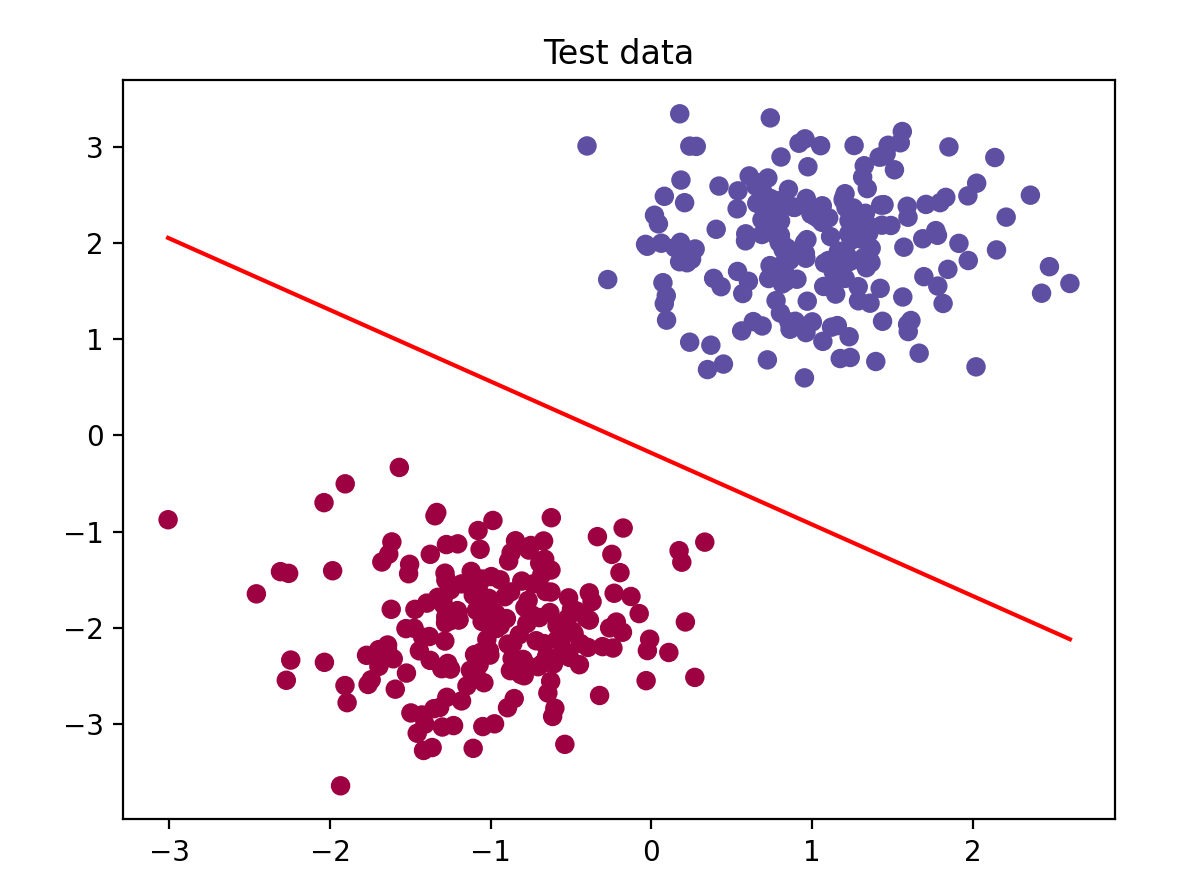
（2）满足朴素贝叶斯，有惩罚项

迭代次数：10531



训练数据集正确率: 1.0

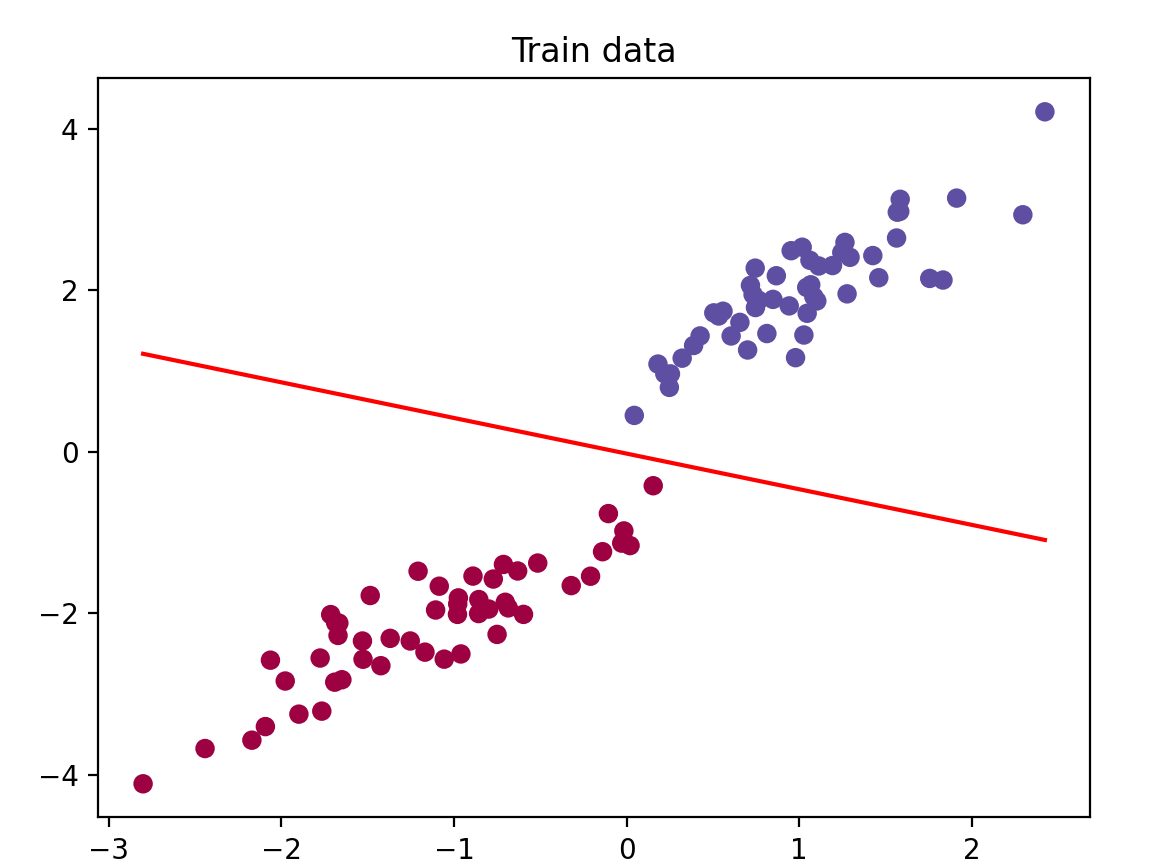
Discriminant function: y =-0.7432 x - 0.1831



Test data accuracy: 1.0

（3）不满足朴素贝叶斯，无惩罚项

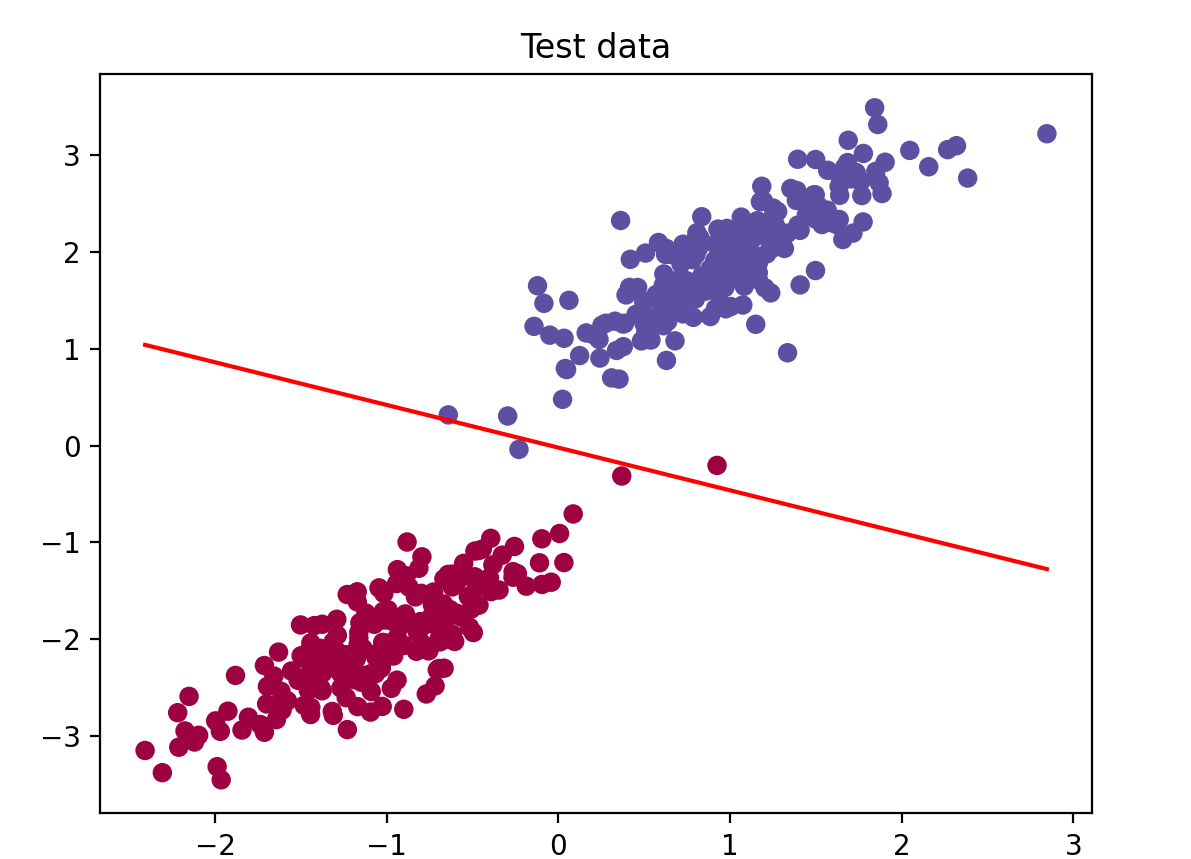
迭代次数：15943



Train data accuracy: 1.0

Discriminant function: y =

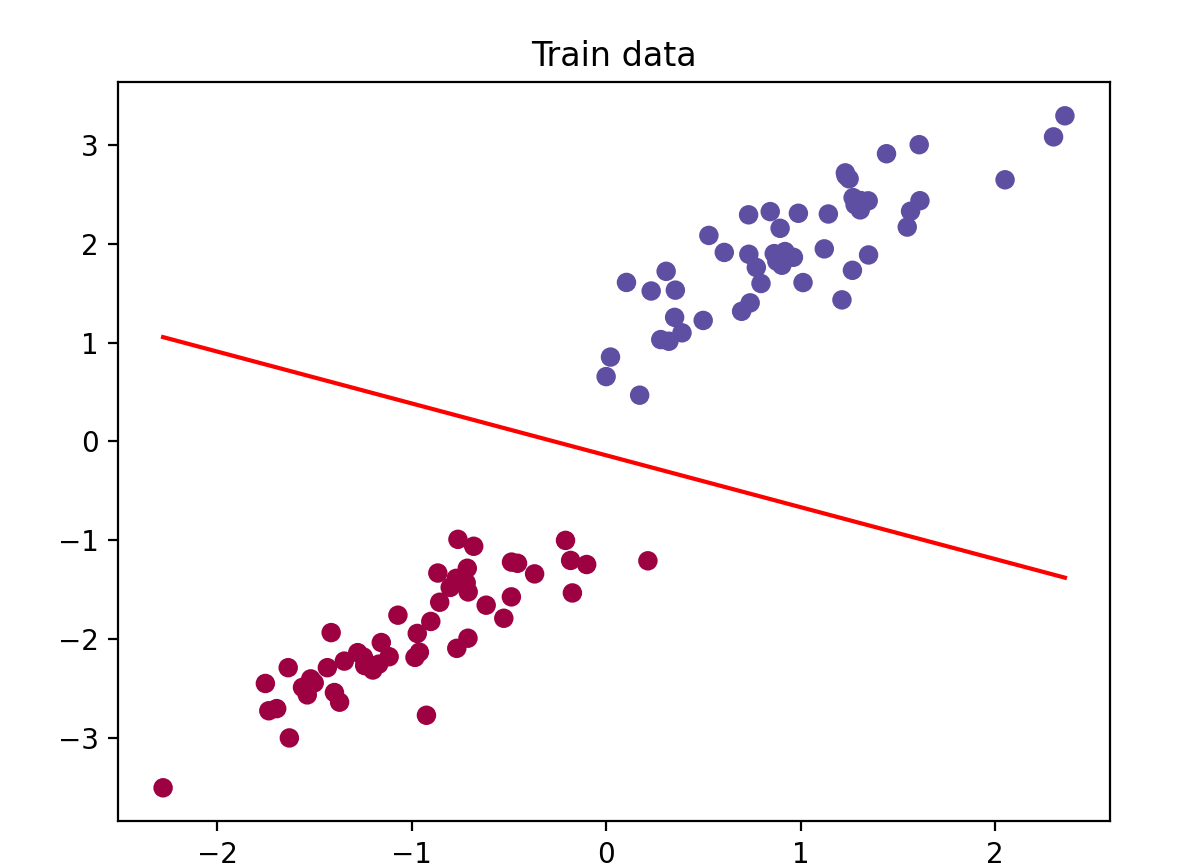
-0.4413 x - 0.02093



Test data accuracy: 0.995

（4）不满足朴素贝叶斯，有惩罚项

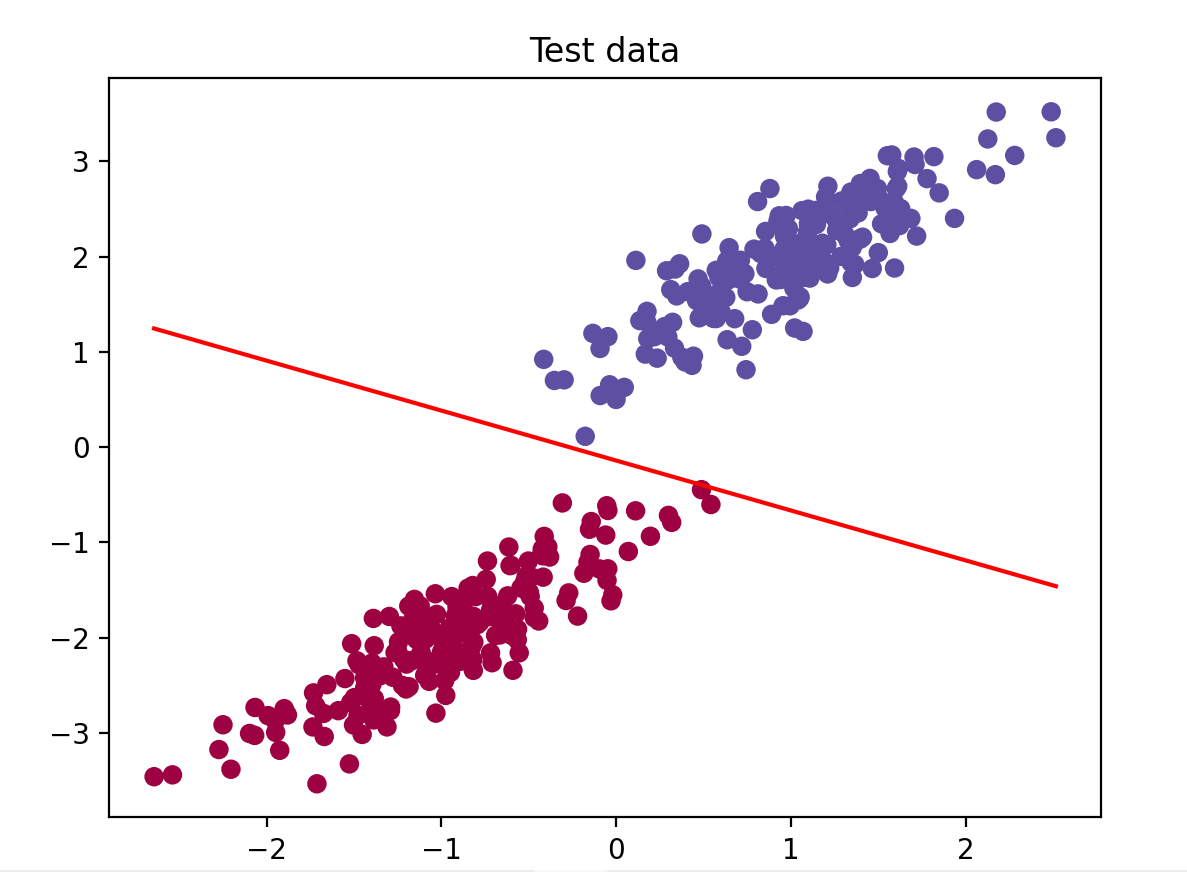
迭代次数：13523



Train data accuracy: 1.0

Discriminant function: y =

-0.5245 x - 0.1404



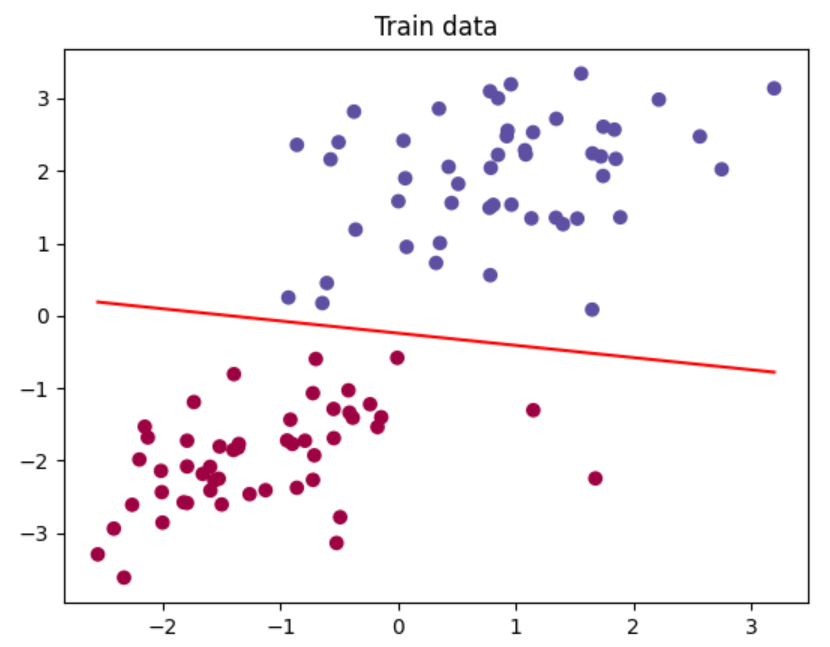
Test data accuracy: 1.0

可见，在满足朴素贝叶斯分布时分类正确率几乎100%，在不满足朴素贝叶斯分布时点的分布出现相关性，此图分布较为狭长，但分类效果也与满足时相差并不大。

1. 增大属性相关性，改变协方差矩阵

（1）不带惩罚项

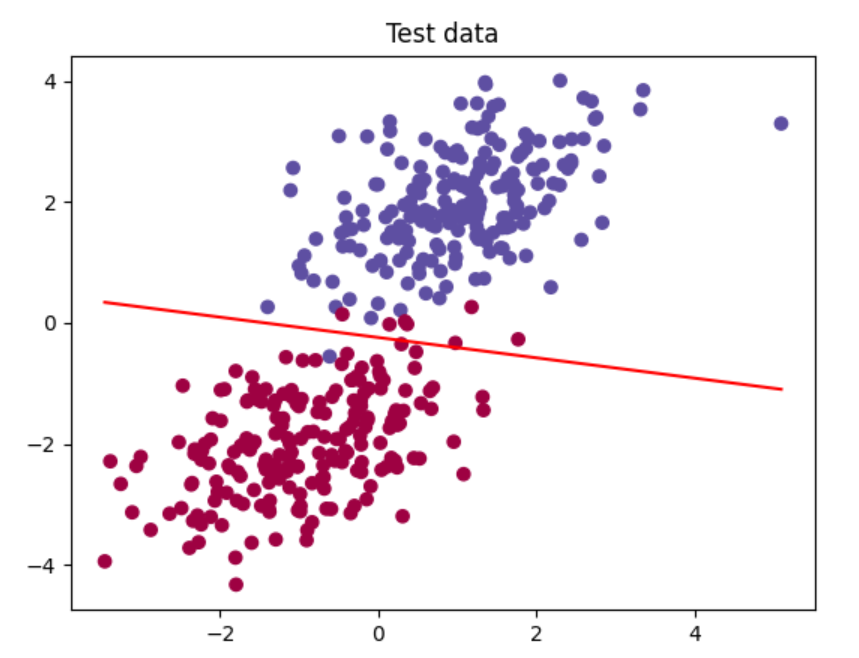
迭代次数：22818



Train data accuracy: 1.0

Discriminant function: y =

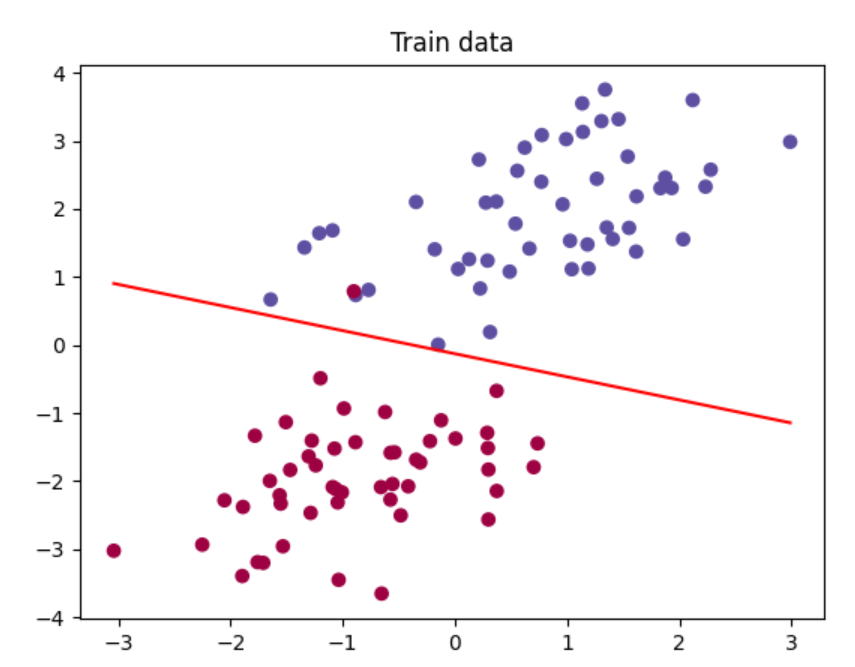
-0.1684 x - 0.2403



Test data accuracy: 0.98

（2）带惩罚项

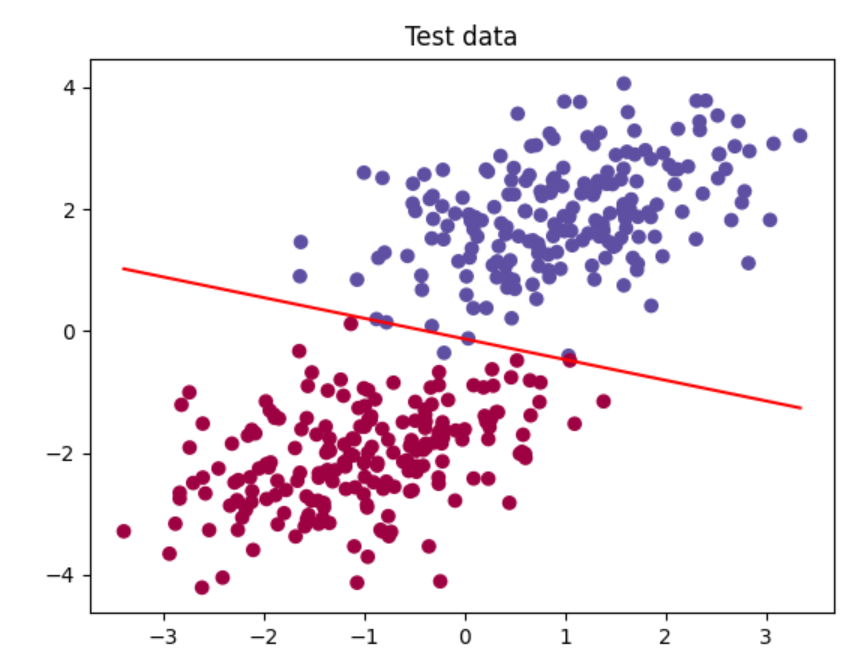
迭代次数：10560



Train data accuracy: 0.99

Discriminant function: y =

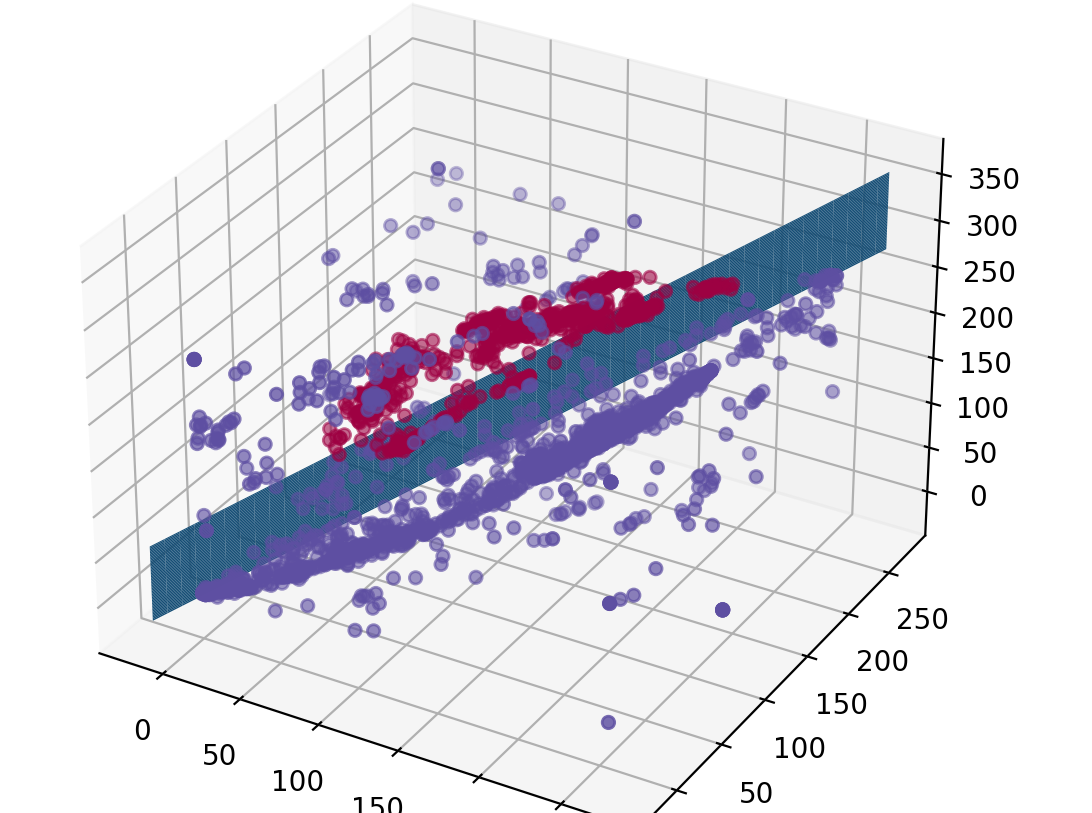
-0.3393 x - 0.1283



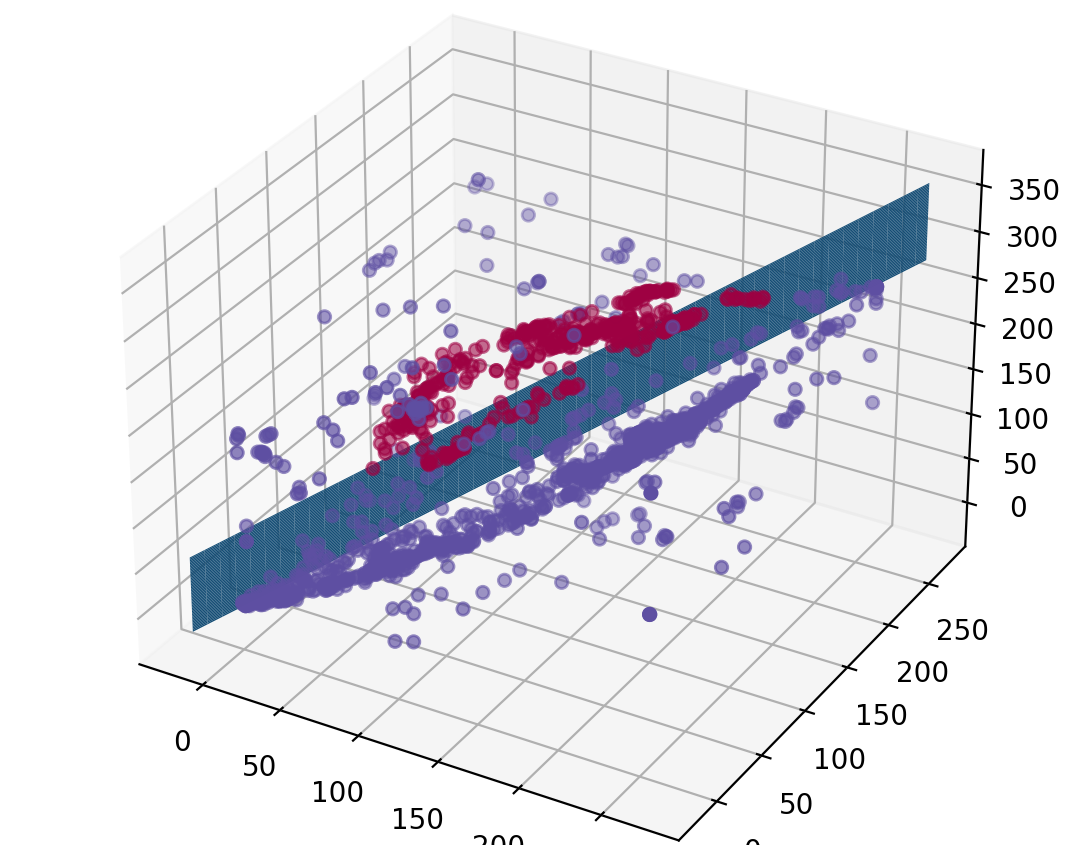
Test data accuracy: 0.995

可见，随着属性相关性增大，正确率略有下降，且有惩罚项的比没有惩罚项的迭代次数少，且正确率更高。但正确率并没有下降很多，并且是否有惩罚项对其影响也不大。推断应该是只有二维，所以影响不大。但当减小训练集时，所得到的判别函数确实存在过拟合现象，加入正则项可以预防此现象的发生。

3.uci数据集



Train data accuracy: 0.9266666666666666



Test data accuracy: 0.938

对于本数据集正确率可达到90%以上，不过并不是每个数据集都能达到如此高的正确率，主要看属性之间相关性强弱，相关性越强，正确率越低。

1. 结论
2. 逻辑回归是对已有的数据进行求解出一个能将数据进行较好分类的超平面，再利用这个超平面对测试数据集进行分类。
3. 对于线性分类且满足朴素贝叶斯的问题分类问题效果较好，速度较快。
4. 但若维度较高，且属性之间相关性较大，则分类效果不好。正则项在数据量较小时，可以有效解决过拟合问题。正则项对于非朴素贝叶斯分布正确率有一定提高。
5. 参考文献

[1]周志华.机器学习.清华大学出版社

1. 附录：源代码（带注释）

import numpy as np

from numpy import matlib

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

pos\_num = 50    #正例个数

pos\_mean = [1, 2]    #正例均值，二维

pos\_cov = neg\_cov = np.mat([[0.3, 0], [0, 0.4]])      #满足朴素贝叶斯正反例协方差

fpos\_cov = fneg\_cov = np.mat([[0.3, 0.7], [0.9, 0.4]])    #不满足朴素贝叶斯正反例协方差

neg\_num = 50         #反例个数

neg\_mean = [-1, -2]    #反例均值，二维

lam = np.exp(-10)      #惩罚项

cur = 1e-5            #精度

learn\_rate = 0.01       #学习率

# 生成训练集x，根据均值与方差随机生成正例反例的坐标，在y中进行标记

def generate\_f(pos\_num, pos\_mean, pos\_cov, neg\_num, neg\_mean, neg\_cov):

    x = np.zeros((pos\_num+neg\_num, 2))

    y = np.zeros(pos\_num+neg\_num)

    x[:pos\_num, :] = np.random.multivariate\_normal(

        pos\_mean, pos\_cov, size=pos\_num)          #前面放由均值和协方差生成的正例

    x[pos\_num:, :] = np.random.multivariate\_normal(

        neg\_mean, neg\_cov, size=neg\_num)           #后面放由均值和协方差生成的反例

    y[:pos\_num] = 1          #前面正例为1，后面反例为0

    return x, y

def sig(x):

    return 1/(1+np.exp(-x))

# 损失函数

def loss(X, y, w, lam):

    YWX = 0

    ln = 0

    for i in range(X.shape[0]):

        YWX += y[i]\*w@X[i].T

        ln += np.log(1 + np.exp(w@X[i].T))

    loss = -YWX+ln+(lam\*w@w.T)/2

    return loss/(X.shape[1])

# 梯度下降法求参数

def diff(x, y, learn\_rate, lam):

    X = np.concatenate((np.ones((x.shape[0], 1)), x), 1)  # 扩增x一个维度为X，第一维均为1

    w1 = np.ones((1, X.shape[1]))  # 扩增w与X维度相同

    loss1 = loss(X, y, w1, lam)

    for i in range(100000):

        loss0 = loss1

        t = np.zeros((X.shape[0], 1))

        t = X@w1.T

        grad = - (y - sig(t.T)) @ X / X.shape[0]  # 计算下降梯度

        w0 = w1

        w1 = w1 - learn\_rate \* lam \* w1 - learn\_rate \* grad    # 根据下降梯度找到沿此梯度方向的下个值

        loss1 = loss(X, y, w1, lam)

        if abs(loss0) < abs(loss1):  # 如果损失函数增大了，降低学习率重新找

            w1 = w0

            learn\_rate /= 2

            loss1 = loss(X, y, w1, lam)

        if abs(loss0-loss1) < cur:  # 如果几乎不再下降，则停止

            break

    print(i)

    w1 = w1.reshape(X.shape[1])

    coef = -(w1 / w1[X.shape[1]-1])[0:X.shape[1]-1]  # 对w做归一化，得到方程系数

    print(w1)

    print(coef)

    return coef, w1

#画出二维散点图和分类函数线

def plt2d(x, y, discriminant, title):

    print('Discriminant function: y = ', discriminant)

    plt.scatter(x[:, 0], x[:, 1], c=y, marker='o', cmap=plt.cm.Spectral)

    res\_x = np.linspace(min(x[:, 0]), max(x[:, 0]), x.shape[0])

    res\_y = discriminant(res\_x)

    plt.plot(res\_x, res\_y, 'r', label='discriminant')

    plt.title(title)

    plt.show()

#画出三维散点图和分类函数面

def plt3d(x, y, coef, title):

    d3 = Axes3D(plt.figure())

    d3.scatter(x[:, 0], x[:, 1], x[:, 2], c=y, cmap=plt.cm.Spectral)

    res\_x = np.linspace(np.min(x[:, 0])-20, np.max(x[:, 0])+20, 200)

    res\_y = np.linspace(np.min(x[:, 1])-20, np.max(x[:, 1])+20, 200)

    res\_X, res\_Y = np.meshgrid(res\_x, res\_y)

    res\_z = coef[0] + coef[1] \* res\_X + coef[2] \* res\_Y

    d3.plot\_surface(res\_x, res\_y, res\_z)

    d3.set\_title(title)

    plt.show()

#进行二维测试

def test(lam, pos\_cov, neg\_cov):

    x, y = generate\_f(pos\_num, pos\_mean, pos\_cov, neg\_num,neg\_mean, neg\_cov)  # 生成训练数据集

    coef, w = diff(x, y, learn\_rate, lam)           #计算出方程系数

    discriminant = np.poly1d(coef[::-1])

    print('Train data accuracy:', fit(x, y, w))

    plt2d(x, y, discriminant, 'Train data')      #画出训练集二维散点图和分类函数线

    test\_x, test\_y = generate\_f(pos\_num\*4, pos\_mean, pos\_cov, neg\_num\*4, neg\_mean, neg\_cov)  # 生成测试数据集

    print('Test data accuracy:', fit(test\_x, test\_y, w))

    plt2d(test\_x, test\_y, discriminant, 'Test data')     #画出测试集二维散点图和分类函数线

# 从文件读取uci数据，分离训练数据集和测试数据集

def uciread(path):

    data = np.loadtxt(path, dtype=np.int32)

    np.random.shuffle(data)  # 随机打乱数据，便于选取数据

    dim = data.shape[1]

    if data.shape[0] < 5000:

        train\_size = int(0.6 \* data.shape[0])  # 按照6：4的比例分配训练集和测试集

        test\_size = int(0.4 \* data.shape[0])

    else:

        train\_size = 3000

        test\_size = 2000

    x = data[:train\_size, 0:dim-1]

    y = data[:train\_size, dim-1] - 1

    test\_x = data[train\_size:train\_size+test\_size, 0:dim-1]

    test\_y = data[train\_size:train\_size+test\_size, dim-1] - 1

    return x, y, test\_x, test\_y

#进行三维uci数据集测试

def uci\_test(path):

    x, y, test\_x, test\_y = uciread(path)          # 生成训练数据集和测试数据集

    coef, w = diff(x, y, learn\_rate, lam)          #计算出方程系数

    print('Train data accuracy:', fit(x, y, w))

    plt3d(x, y, coef, 'Train data set')            #画出训练集三维散点图和分类函数面

    print('Test data accuracy:', fit(test\_x, test\_y, w))

    plt3d(test\_x, test\_y, coef, 'Test data set')       #画出测试集三维散点图和分类函数面

#计算训练集和数据集进行相应划分后的正确率

def fit(x, y, w):

    count = 0

    X = np.concatenate((np.ones((x.shape[0], 1)), x), 1)   # 扩增x一个维度为X，第一维均为1

    for i in range(X.shape[0]):

        if w@X[i].T >= 0 and y[i] == 1:

            count += 1

        if w@X[i].T < 0 and y[i] == 0:

            count += 1

    return count / X.shape[0]

def main():

    test(0, pos\_cov, neg\_cov)  # 满足朴素贝叶斯，无惩罚项

    test(lam,pos\_cov,neg\_cov)    #满足朴素贝叶斯，有惩罚项

    test(0,pos\_cov=fpos\_cov,neg\_cov=fneg\_cov)      #不满足朴素贝叶斯，无惩罚项

    test(lam,pos\_cov=fpos\_cov,neg\_cov=fneg\_cov)        #不满足朴素贝叶斯，有惩罚项

    uci\_test("Skin\_NonSkin.data")

main()