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：1190202401

姓名：陈豪

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。

实验环境

- OS:Windows10
- Python3.9.2

三、算法设计和原理分析

<一>算法原理

本次实验中使用的是逻辑回归，并利用梯度下降法和牛顿法对其损失含函数进行参数估计。牛顿法相比梯度下降法，迭代次数少，速度快。

求解损失函数

考虑二分类问题，$f:X\rightarrow Y$，其中X为实数向量, $X=<X_1,X_2,...,X_n>$，$Y\in\{0,1\}$，并且假设所有的X_i在给定Y的前提下均条件独立，并且有$P(X_i|Y=y_k)\sim N(\mu_{ik},\sigma_i)$，$P(Y)\sim B(\pi)$成立，令$P(Y=1)=\pi$，则$P(Y=0)=1-\pi$

因此利用贝叶斯公式和朴素贝叶斯假设可得

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

又因为$P(X_i|Y=y_k) \sim N(\mu_{ik}, \sigma_i)$，代入上式可得

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

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

$$P(Y=0|X) = \frac{1}{1+\exp\left(w_0 + \sum_{i=1}^n w_i X_i\right)} = \frac{1}{1+exp(X\mathbf{w})}$$

则

$$P(Y=1|X) = \frac{exp(X\mathbf{w})}{1+exp(X\mathbf{w})} = \frac{1}{1+exp(X\mathbf{w})}$$

其中，$\mathbf{w}_0 = \sum_i^n\left(\frac{\mu_{i0}^2-\mu_{i1}^2}{2\sigma_i^2}\right) + ln(\frac{\pi}{1-\pi})$，$w_i = \frac{\mu_{i1}-\mu_{i0}}{\sigma_i^2}$，$i > 0$，
$X =< X_1, X_2, \ldots, X_n >$

得到上面的假设后，利用最大似然估计可得

$$L_1(W) = \prod_l P(Y^l|X^l, \mathbf{w})$$

其中$1 \le l \le N$,其中N是样本数量，为方便计算取对数可得

$$\begin{aligned}
L_2(W) &= \sum_l \ln P(Y^l|X^l, \mathbf{w}) \\
&= \sum_l Y^l \ln\, P(Y^l=1|X^l, W) + (1-Y^l)\ln\, P(Y^l=0|X^l, W) \\
&= \sum_l Y^l \ln\, \frac{P(Y^l=1|X^l, W)}{P(Y^l=0|X^l, W)} - \ln\, P(Y^l=0|X^l, W) \\
&= \sum_l Y^l(w_0 + \sum_{i=1}^n w_i X_i^l) - \ln\left(1 + exp(w_0 + \sum_{i=1}^n w_i X_i^l\right)
\end{aligned}$$

在取相反数后，得到最终的损失函数

$$\begin{aligned}
L(W) &= \sum_l -Y^l(w_0 + \sum_{i=1}^n w_i X_i^l) + \ln\left(1 + exp(w_0 + \sum_{i=1}^n w_i X_i^l\right) \\
&= \sum_l \left(-Y^l X^l \mathbf{w} + ln(1 + exp(X^l\mathbf{w}))\right)
\end{aligned}$$

梯度法原理

$$\frac{\partial\, L(W)}{\partial\, w_i} = \sum_l\; -X_i^l(Y^l - \frac{1}{1+\exp\left(w_0 + \sum_{i=1}^n w_i X_i^l\right)})$$

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

向量形式

$$\boldsymbol{W} = \boldsymbol{W} + \eta\sum_l \boldsymbol{(}X^l)^T(\boldsymbol{Y}^l - \frac{1}{1+\exp\left(\boldsymbol{X^l W}\right)})$$

为了避免过拟合，因此添加了正则项的梯度,它的向量形式是

$$\boldsymbol{W} = \boldsymbol{W} - \eta\lambda\boldsymbol{W} + \eta\sum_l (X^l)^T(\boldsymbol{Y}^l - \frac{1}{1+\exp\left(\boldsymbol{X^l W}\right)})$$

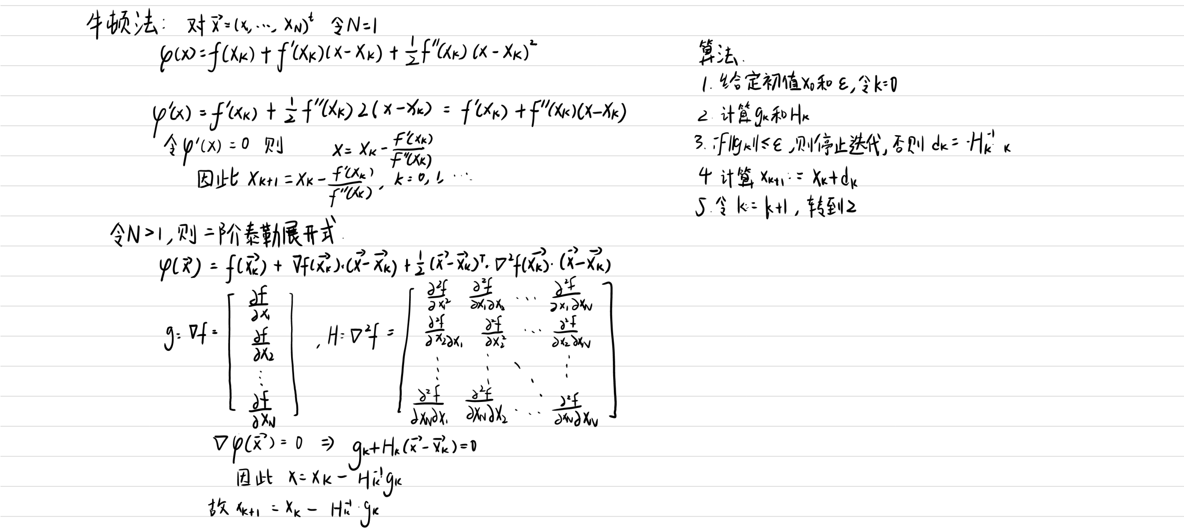
为了防止溢出，添加了归一项因子$\frac{1}{N}$，如下所示：

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

$$\boldsymbol{W} = \boldsymbol{W} + \frac{\eta}{N}(-\eta\lambda\boldsymbol{W} + \sum_l (X^l)^T(\boldsymbol{Y}^l - \frac{1}{1+\exp\left(\boldsymbol{X^l W}\right)}))$$

牛顿法原理

牛顿法的基本原理是利用泰勒展开式在一特定的点进行二阶展开，然后对自变量求一阶导数和二阶导数。算法如下：



上面是牛顿法的基本原理，在本次实验中，梯度g已经在上面的梯度下降法中求出。因此现在只需求得海森矩阵即可。

而对于带正则项的海森矩阵结果如下

$$H_k = \frac{\partial L(w)}{\partial w \partial w^T} = \frac{1}{N}(-\sum_l X^l (X^l)^T p_1(1-p_1) + \lambda I)$$

注：这里的X^l是一个列向量，而上面我使用的都是行向量，所以很多地方进行了转置当列向量使用。

<二>算法实现

梯度下降

实现时为了利用python提供的矩阵运算方法，对于梯度是用向量的形式进行的计算。判断收敛是通过判断损失值收敛和梯度收敛。在迭代的过程中将会打印梯度的值，方便debug。

```python
class GradientDescent(object):
    """
    使用梯度下降方法求解逻辑回归
    """

    def __init__(self,w0,X,Y,alpha=0.03,lambda_penalty=0,epsilon=1e-3) -> None:
        """
        w0: 初始参数 是d*1列向量
```

```
        X：特征集  是n*d一个矩阵
        Y：标签集  是n*1的列向量
        lambda_penalty：惩罚项系数
        alpha：学习系数
        epsilon：收敛判断条件
        """
        assert len(X) == len(Y)
        self.w = w0.reshape(w0.shape[0],1)
        self.X = X.reshape(X.shape[0],X.shape[1])
        self.Y = Y.reshape(Y.shape[0],1)
        self.alpha = alpha
        self.lambda_penalty = lambda_penalty
        self.epsilon = epsilon
        self.size = self.X.shape[0]

    def __sigmoid(self,z):
        return 1 / (1 + np.exp(-z))

    def loss(self,X,Y):
        """
        计算损失函数的值
        X：特征集  是n*d一个矩阵
        Y：标签集  是n*1的列向量
        """
        loss = 0
        for i in range(len(X)):
            loss = loss +  Y[i]*( X[i] @ self.w) - np.log( 1+np.exp(X[i] @
self.w) )
        return -loss/self.size #进行归一化操作

    def __gradient(self):
        """
        函数功能计算梯度
        """
        gradient = self.X.T @ (self.Y - self.__sigmoid(self.X @ self.w))
        return -gradient/self.size #进行归一化操作

    def train(self):
        """
        使用梯度下降法求解w
        """
        gradient = self.__gradient()
        new_loss = self.loss(self.X,self.Y)
        old_loss = new_loss
        while True:
            old_loss = new_loss
            self.w = self.w - self.alpha * (self.lambda_penalty / self.size *
self.w  +  gradient)
            gradient = self.__gradient()
            print(gradient.T @ gradient)
            new_loss = self.loss(self.X,self.Y)
            if old_loss <= new_loss: #损失增大说明学习率过大，因此减小学习率
                self.alpha = self.alpha/2
                continue
            elif np.absolute(old_loss-new_loss) < self.epsilon  and gradient.T @
gradient < self.epsilon:
                break
        return self.w.reshape(self.w.shape[0],1)
```

牛顿法

分别实现了梯度和海森矩阵的计算

```
class NewTon(object):

    def __init__(self,w0,X,Y,lambda_penalty=0,epsilon=1e-5) -> None:
        """
        w0: 初始参数 是d*1列向量
        X: 特征集 是n*d一个矩阵
        Y: 标签集 是n*1的列向量
        lambda_penalty: 惩罚项系数
        alpha: 学习系数
        epsilon: 收敛判断条件
        """
        assert len(X) == len(Y)
        self.w = w0.reshape(w0.shape[0],1)
        self.X = X.reshape(X.shape[0],X.shape[1])
        self.Y = Y.reshape(Y.shape[0],1)
        self.lambda_penalty = lambda_penalty
        self.epsilon = epsilon
        self.size = len(self.X)
        self.dim = len(self.X[0])

    def __sigmoid(self,z):
        return 1 / (1 + np.exp(-z))

    def loss(self,X,Y):
        """
        计算损失函数的值
        X: 特征集 是n*d一个矩阵
        Y: 标签集 是n*1的列向量
        """
        loss = 0
        for i in range(len(X)):
            loss = loss +  Y[i]*( X[i] @ self.w) - np.log( 1+np.exp(X[i] @
self.w) )
        return -loss/self.size

    def __hessianMatrix(self):
        """
        计算海森矩阵
        """
        hessian = self.lambda_penalty * np.eye(self.dim)
        for i in range(self.size):
            temp = self.__sigmoid(self.X[i] @ self.w)
            hessian += self.X[i] * np.transpose([self.X[i]]) * temp * (1 - temp)
        return hessian/self.size

    def __gradient(self):
        """
        函数功能:计算梯度
        """
        gradient = self.X.T @ (self.Y - self.__sigmoid(self.X @ self.w))
        return (-gradient + self.lambda_penalty*self.w)/self.size
```

```
    def train(self):
        hessian = self.__hessianMatrix()
        gradient = self.__gradient()
        old_loss = self.loss(self.X,self.Y)
        new_loss = old_loss
        while True:
            old_loss = new_loss
            self.w = self.w - np.linalg.pinv(hessian) @ gradient
            hessian = self.__hessianMatrix()
            print(gradient.T @ gradient)
            gradient = self.__gradient()
            new_loss = self.loss(self.X,self.Y)
            if np.absolute(old_loss-new_loss) < self.epsilon and gradient.T @ 
gradient < self.epsilon:
                break
        return self.w
```

四、实验结果分析

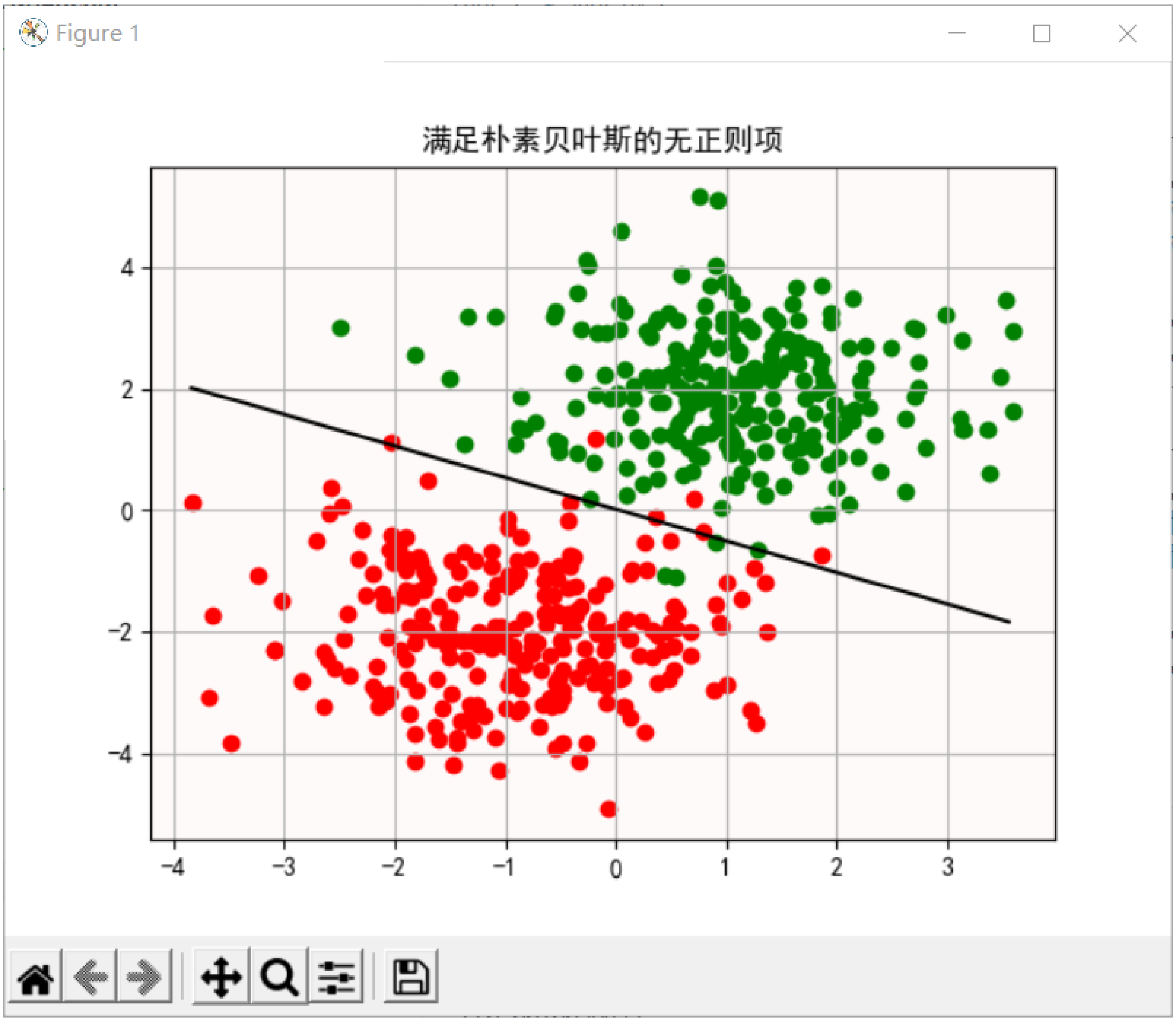
<一>自己生成数据

在本次实验中利用二元高斯分布生成数据，生成的正例数据均值是[1,2],反例数据均值[-1,-2],两类数据的方差均为1，正例和反例训练数据数分别是50，50。测试数据数是250，250。当不满足朴素贝叶斯时，设置协方差为0.3。

梯度下降法

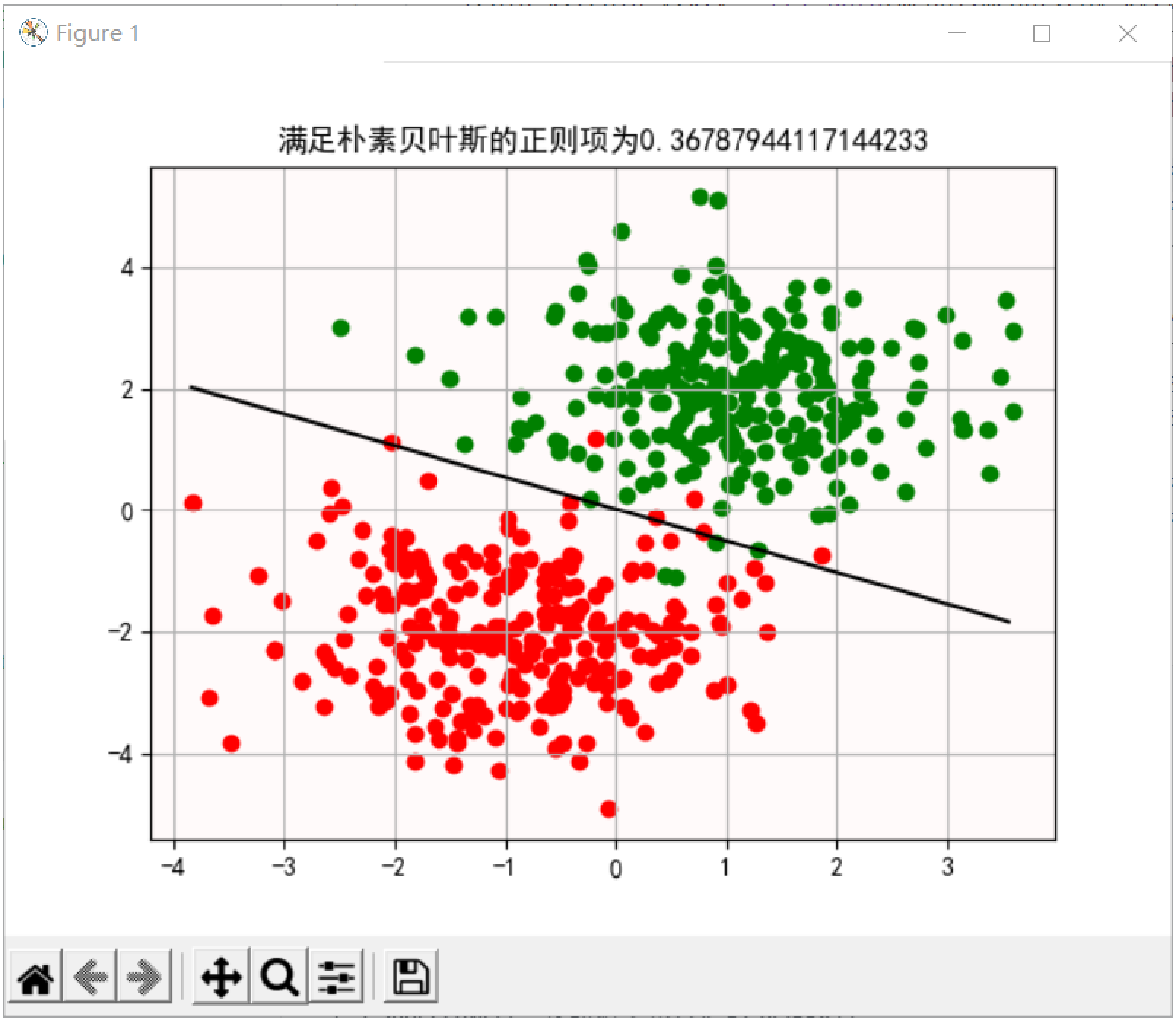
学习率设置为0.03，惩罚项设置为e^{-1}，收敛的精度是10^{-3}

满足朴素贝叶斯的不带正则项



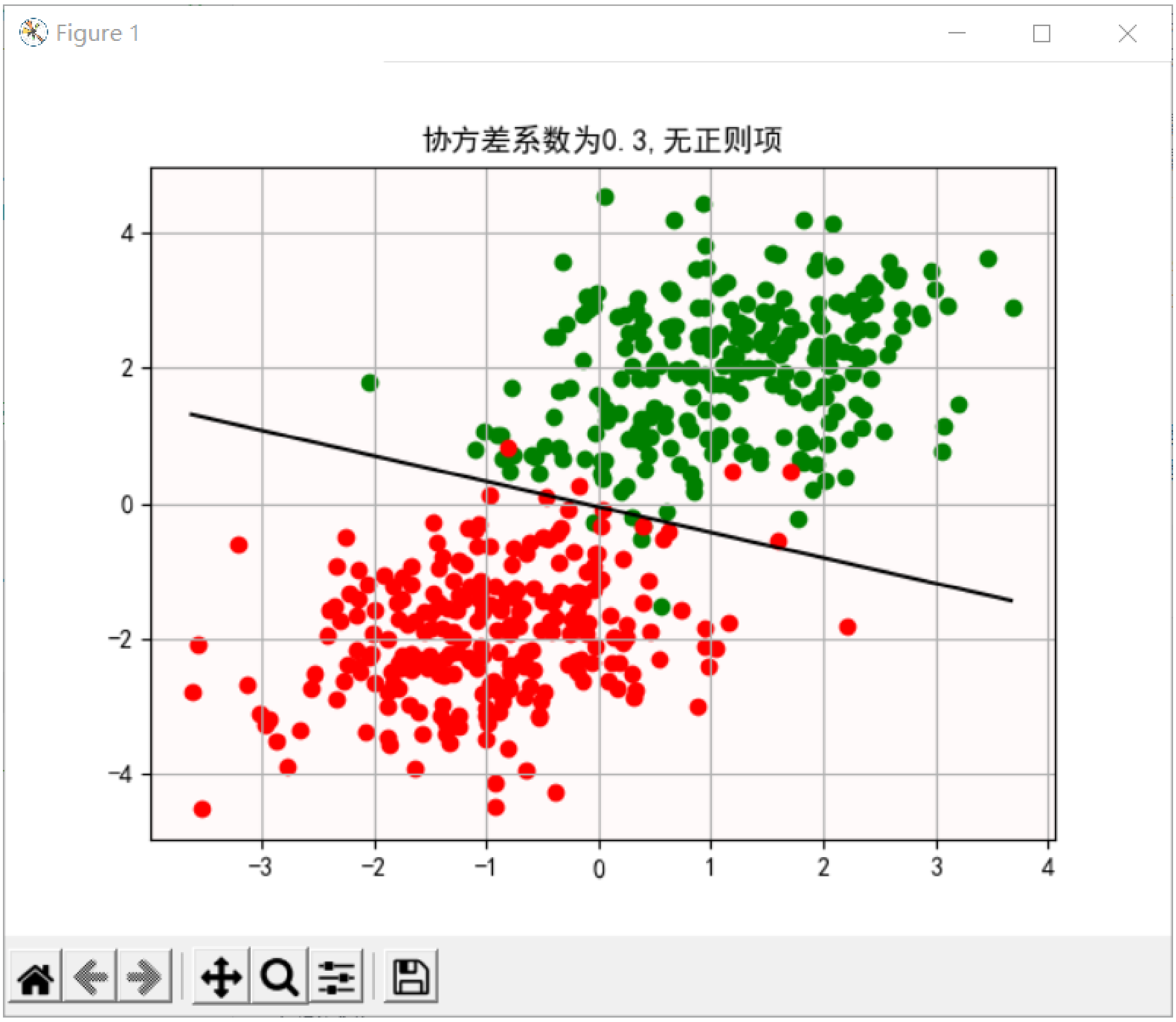

```
w=[[ 0.03010966]
 [-0.97919009]
 [-1.88511715]]，它的模大小[[4.51338651]]
在训练集loss[0.03192825]
在测试集loss[0.30340613]
在训练集准确率1.0
在测试集准确率0.98
```

满足朴素贝叶斯的带正则项



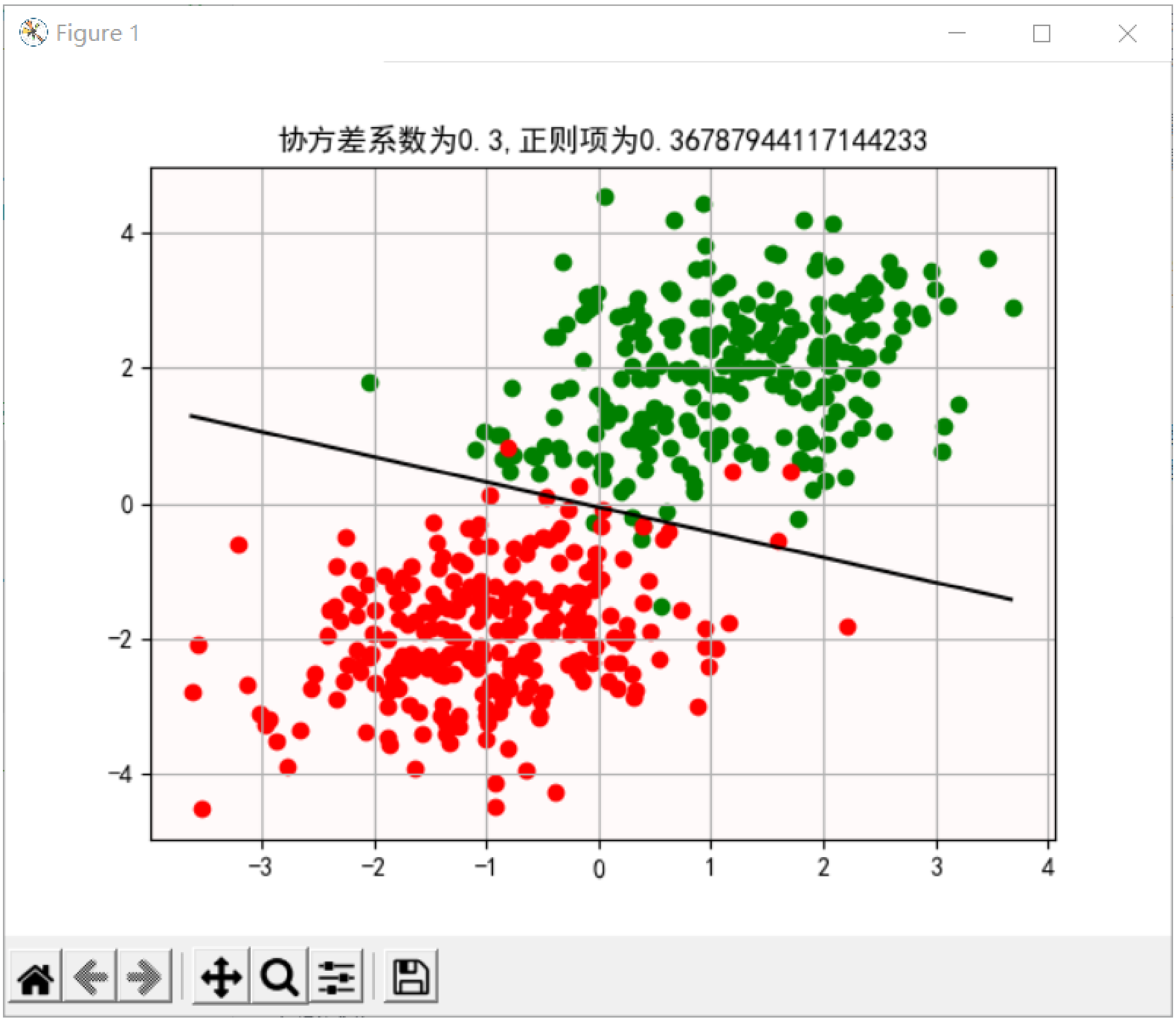

```
w=[[ 0.03597122]
 [-0.98021844]
 [-1.88433104]], 它的模大小[[4.51282558]]
在训练集loss[0.03191535]
在测试集loss[0.30338587]
在训练集准确率1.0
在测试集准确率0.98
```

不满足朴素贝叶斯且不带正则项




```
w=[[-0.1051402 ]
 [-0.73735039]
 [-1.95274692]], 它的模大小[[4.36796058]]
在训练集loss[0.07552564]
在测试集loss[0.39056309]
在训练集准确率0.97
在测试集准确率0.98
```

不满足朴素贝叶斯但带正则项

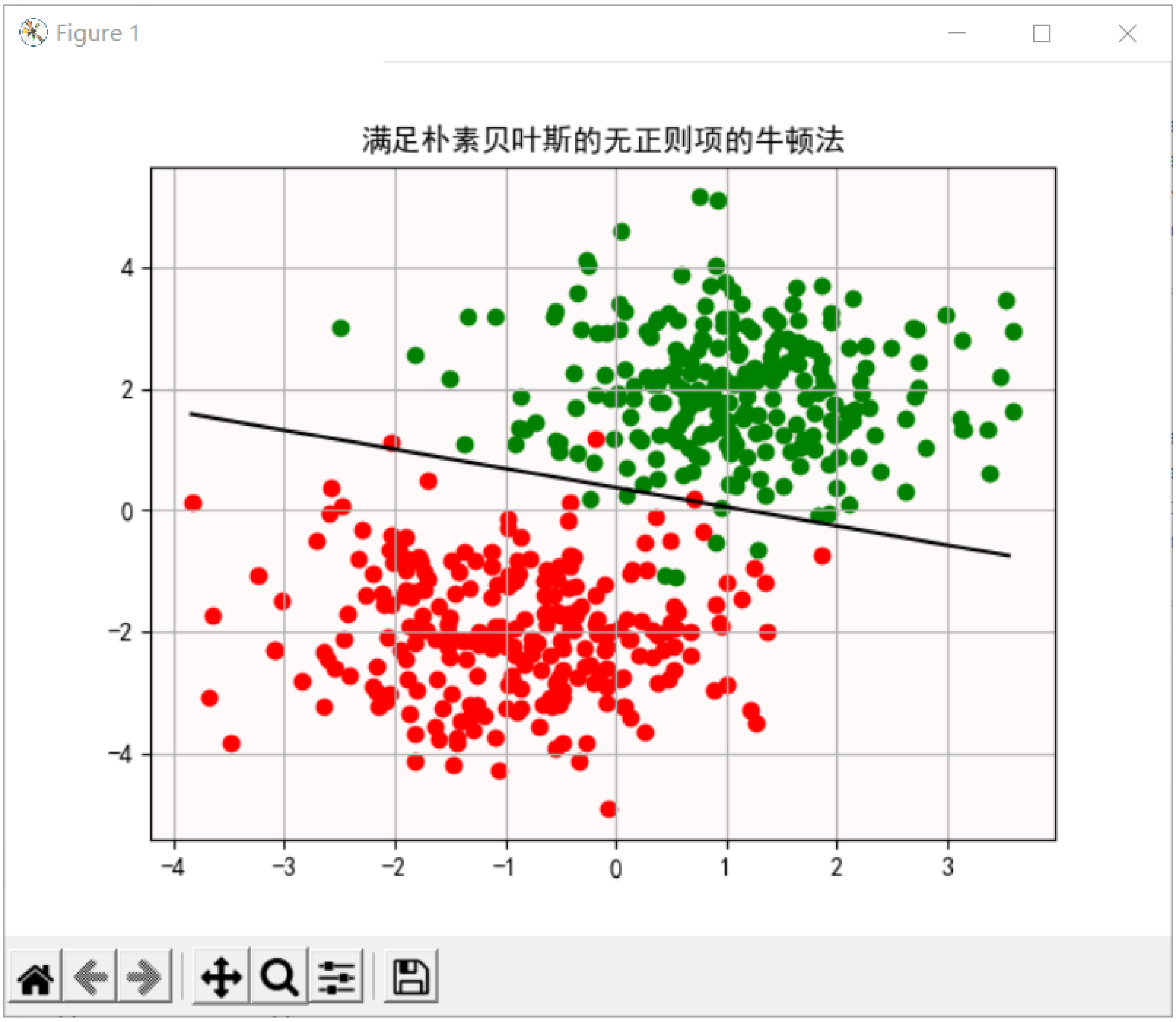



```
w=[[-0.11274483]
 [-0.72749549]
 [-1.95509398]], 它的模大小[[4.36435355]]
在训练集loss[0.07548572]
在测试集loss[0.39085907]
在训练集准确率0.97
在测试集准确率0.98
```

牛顿法

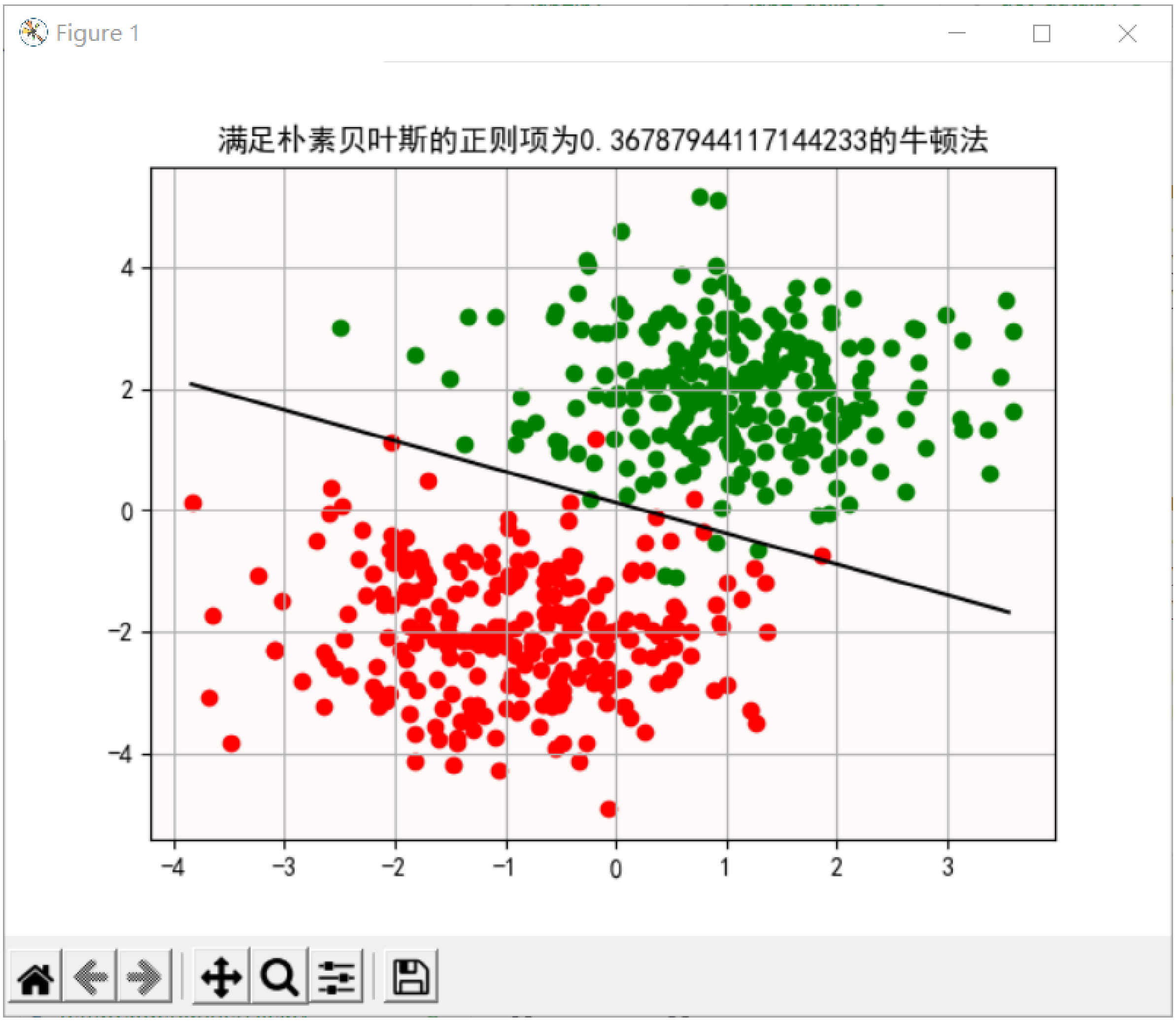
正则项参数同梯度下降法，收敛精度为10^{-5}

满足朴素贝叶斯的不带正则项



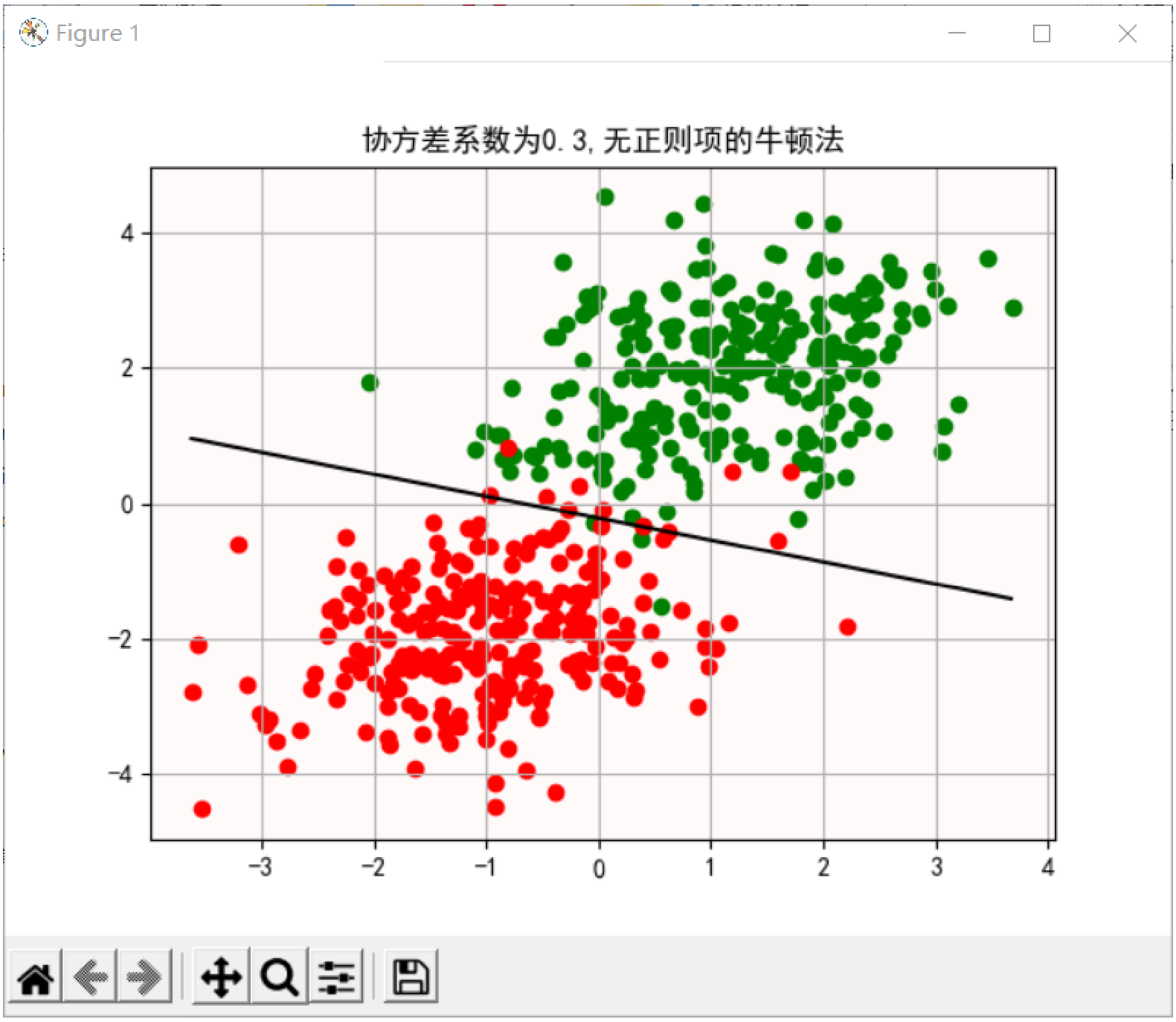

```
w=[[  6.28733126]
 [ -5.33876453]
 [-16.95990069]], 它的模大小[[355.67117242]]
在训练集loss[2.27081385e-06]
在测试集loss[0.88250187]
在训练集准确率1.0
在测试集准确率0.98
```

满足朴素贝叶斯的带正则项



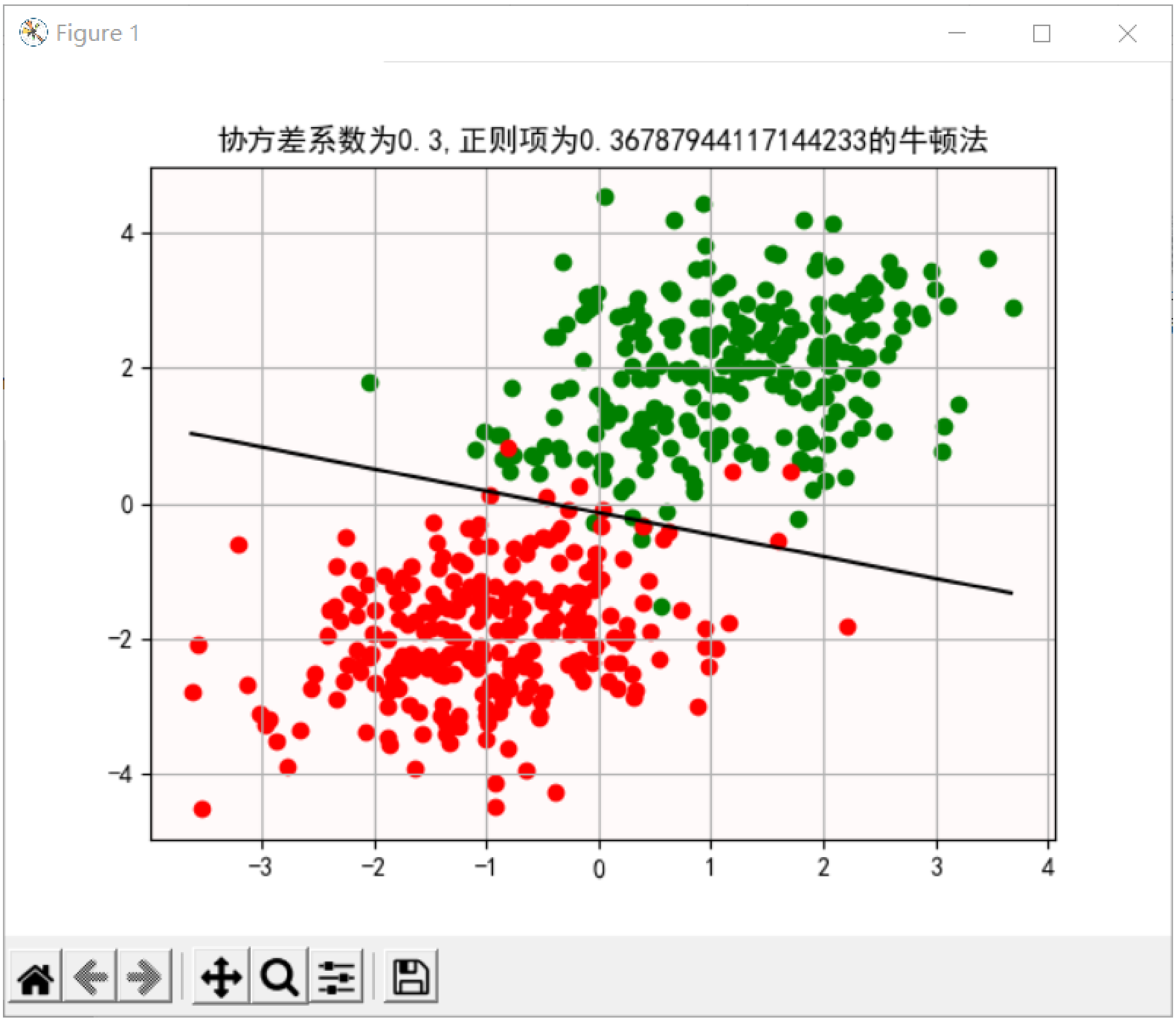

```
w=[[ 0.34169529]
 [-1.36408257]
 [-2.69134252]], 它的模大小[[9.22080151]]
在训练集loss[0.01427556]
在测试集loss[0.23669168]
在训练集准确率1.0
在测试集准确率0.984
```

不满足朴素贝叶斯且不带正则项




```
w=[[-0.97033109]
 [-1.42412837]
 [-4.38802609]], 它的模大小[[22.22445697]]
在训练集loss[0.05100839]
在测试集loss[0.34768672]
在训练集准确率0.98
在测试集准确率0.974
```

不满足朴素贝叶斯但带正则项




```
w=[[-0.3939883 ]
 [-0.90718741]
 [-2.80064478]],它的模大小[[8.82182695]]
在训练集loss[0.05819316]
在测试集loss[0.32708452]
在训练集准确率0.97
在测试集准确率0.978
```

通过上面测试数据可以观察到，在满足朴素贝叶斯的时候，准确率比较高，而在不满足朴素贝叶斯的时候，准确率稍有下降。同时观察到正则项在梯度下降法时对模型的模值影响不大，这可能是因为牛顿法设定的收敛精度更高导致的。同时，经过测试很容易发现牛顿法的运行速度比梯度下降法要快很多。

<二>uci数据

经过在uci上进行数据查找，找到了下面这两个数据样例。因为牛顿法运行比较快，因此接下来的实验都是使用牛顿法来运行的。

CNAE-9

这个数据集的特征有857维，样本量是1080，分为9个类别。这里将数据集按比例划分30%作为训练集，剩下的70%作为测试集。并且选择类别1与其他进行类别进行分类。由于维度过大，因此这里不在展示模型参数。

不带惩罚项

```
它的模大小[[4533.64724486]]
train_loss[5.01594186e-06]
在训练集准确率1.0
test_loss[0.85647319]
在测试集准确率0.9775132275132276
```

带惩罚项

```
它的模大小[[46.95304516]]
train_loss[0.01867492]
在训练集准确率1.0
test_loss[0.15571845]
在测试集准确率0.9735449735449735
```

haberman

这个数据集特征有4维，样本量306，分为两个类别。这里将数据集按比例划分30%作为训练集，剩下的70%作为测试集。

不带惩罚项

```
它的模大小[[0.02979401]]
train_loss[5.01594186e-06]
在训练集准确率1.0
test_loss[1115.35367903]
在测试集准确率0.7344262295081967
```

带惩罚项

```
它的模大小[[0.01125791]]
train_loss[0.0005523]
在训练集准确率1.0
test_loss[685.69021602]
在测试集准确率0.7344262295081967
```

从上可以看出对于不同的数据集，它们测试出来的效果是不一样的。在有的数据集上逻辑回归表现得很好，但是在有些数据集上逻辑回归表现一般。

五、结论

- 逻辑回归不同的数据集表现存在差异，这可能是因为数据集不是线性可分的，而我们设定的逻辑回归是线性可分的。
- 牛顿法相比梯度下降方法的迭代次数小，运行时间短，并且效果也不错。
- 逻辑回归的正则项可以使得所得的模型参数的模值降低，有利于得到简单的模型。
- 逻辑回归对于数据集是否满足朴素贝叶斯假设所受到的影响不大，这说明朴素贝叶斯假设合理，可以用来简化模型，得到不错的结果。

六、代码

get_data.py

```
import numpy as np
from display import displayResult
from display import displayMoreData


def generate_data(mean,cov_xy,var,tag,size=30):
    """
    生成指定的多元正态分布的数据
    mean：均值点
    cov_xy：协方差矩阵
    var：独立同分布的每个随机变量的方差
    size：数据量，是一个int类型数据
    tag：数据标签
    """
    cov = [[var,cov_xy],[cov_xy,var]]
    x = np.random.multivariate_normal(mean, cov, size)
    y = np.zeros(size)
    if tag == 1:
        y = np.ones(size)
    return x.reshape(size,2),y.reshape(size,1)

def get_data(mean1,mean2,cov_xy,var,size_pos,size_neg):
    """
    生成两组多元正态分布数据
    返回训练数据集：特征集和标签集
    同时返回：可以供打印的点(x1,x2)
    """
    k = []
    k.append(generate_data(mean1,cov_xy,var,0,size_pos))
    k.append(generate_data(mean2,cov_xy,var,1,size_neg))

    train_x = np.zeros((size_pos+size_neg,3))
    train_x[:,:1] = np.ones((size_neg+size_pos,1))
    train_x[:size_pos,1:] = k[0][0]
    train_x[size_pos:,1:] = k[1][0]
    train_y = np.zeros((size_neg+size_pos,1))
    train_y[:size_pos,:] = k[0][1]
    train_y[size_pos::] = k[1][1]

    x = []
    y = []
    x.append(k[0][0][:,0])
    y.append(k[0][0][:,1])
    x.append(k[1][0][:,0])
    y.append(k[1][0][:,1])
    return train_x,train_y,x,y
```

display.py

```
import matplotlib.pyplot as plt
import numpy as np



plt.rcParams['axes.facecolor']='snow'
```

```
plt.rcParams['font.sans-serif'] = ['SimHei']  #显示中文
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

def displayData(x,y,color):
    """
    函数功能：展示数据
    """
    plt.scatter(x,y,c=color)

def displayMoreData(x,y,title,colors):
    """
    函数功能：展示多组数据
    """
    for i in range(len(x)):
        displayData(x[i],y[i],color=colors[i])
    plt.title(title)
    plt.grid()
    plt.show()

def displayResult(w,x,y,title,colors,):
    """
    函数功能：展示两组数据以及数据之间的分界线
    """
    minx = min(x[0])
    maxx = max(x[0])
    for i in range(len(x)):
        displayData(x[i],y[i],color=colors[i])
        minx = min(minx,min(x[i]))
        maxx = max(maxx,max(x[i]))
    xlim = np.arange(minx,maxx,0.1)
    plt.plot(xlim,-w[1]/w[2]*xlim-w[0]/w[2],c='black')
    plt.title(title)
    plt.grid()
    plt.show()
```

gradient_descent.py

```
import matplotlib.pyplot as plt
import numpy as np



plt.rcParams['axes.facecolor']='snow'
plt.rcParams['font.sans-serif'] = ['SimHei']  #显示中文
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

def displayData(x,y,color):
    """
    函数功能：展示数据
    """
    plt.scatter(x,y,c=color)

def displayMoreData(x,y,title,colors):
    """
    函数功能：展示多组数据
    """
```

```
    for i in range(len(x)):
        displayData(x[i],y[i],color=colors[i])
    plt.title(title)
    plt.grid()
    plt.show()

def displayResult(w,x,y,title,colors,):
    """
    函数功能：展示两组数据以及数据之间的分界线
    """
    minx = min(x[0])
    maxx = max(x[0])
    for i in range(len(x)):
        displayData(x[i],y[i],color=colors[i])
        minx = min(minx,min(x[i]))
        maxx = max(maxx,max(x[i]))
    xlim = np.arange(minx,maxx,0.1)
    plt.plot(xlim,-w[1]/w[2]*xlim-w[0]/w[2],c='black')
    plt.title(title)
    plt.grid()
    plt.show()
```

new_ton.py

```
import numpy as np

class NewTon(object):

    def __init__(self,w0,X,Y,lambda_penalty=0,epsilon=1e-5) -> None:
        """
        w0: 初始参数 是d*1列向量
        X：特征集 是n*d一个矩阵
        Y：标签集 是n*1的列向量
        lambda_penalty：惩罚项系数
        alpha：学习系数
        epsilon：收敛判断条件
        """
        assert len(X) == len(Y)
        self.w = w0.reshape(w0.shape[0],1)
        self.X = X.reshape(X.shape[0],X.shape[1])
        self.Y = Y.reshape(Y.shape[0],1)
        self.lambda_penalty = lambda_penalty
        self.epsilon = epsilon
        self.size = len(self.X)
        self.dim = len(self.X[0])

    def __sigmoid(self,z):
        return 1 / (1 + np.exp(-z))

    def loss(self,X,Y):
        """
        计算损失函数的值
        X：特征集 是n*d一个矩阵
        Y：标签集 是n*1的列向量
        """
        loss = 0
```

```
        for i in range(len(X)):
            loss = loss +  Y[i]*( X[i] @ self.w) - np.log( 1+np.exp(X[i] @ 
self.w) )
        return -loss/self.size

    def __hessianMatrix(self):
        """
        计算海森矩阵
        """
        hessian = self.lambda_penalty * np.eye(self.dim)
        for i in range(self.size):
            temp = self.__sigmoid(self.X[i] @ self.w)
            hessian += self.X[i] * np.transpose([self.X[i]]) * temp * (1 - temp)
        return hessian/self.size

    def __gradient(self):
        """
        函数功能:计算梯度
        """
        gradient = self.X.T @ (self.Y - self.__sigmoid(self.X @ self.w))
        return (-gradient + self.lambda_penalty*self.w)/self.size

    def train(self):
        hessian = self.__hessianMatrix()
        gradient = self.__gradient()
        old_loss = self.loss(self.X,self.Y)
        new_loss = old_loss
        while True:
            old_loss = new_loss
            self.w = self.w - np.linalg.pinv(hessian) @ gradient
            hessian = self.__hessianMatrix()
            print(gradient.T @ gradient)
            gradient = self.__gradient()
            new_loss = self.loss(self.X,self.Y)
            if np.absolute(old_loss-new_loss) < self.epsilon and gradient.T @ 
gradient < self.epsilon:
                break
        return self.w
```

la2.py

```
import numpy as np
from new_ton import NewTon
from get_data import get_data
from display import displayResult
from gradient_descent import GradientDescent

def accuracy(w,X,Y):
    """
    计算正确率
    P(Y=0|X,W)=1/(1+exp(w0+sum(wiXi)))
    """
    predict_Y = np.zeros((X.shape[0],1))
    for i in range(len(X)):
        if X[i] @ w > 0 :
            predict_Y[i] = 1
        else:
```

```
            predict_Y[i] = 0
    boolY = predict_Y == Y
    return np.sum(boolY)/boolY.shape[0]

def
test(test_x,test_y,train_x,train_y,x,y,title,colors,alpha,lambda_penalty,display
=True,w = np.zeros(3)):
    """
    对模型测试并打印
    将会输出模型，模型的模值，损失值，准确率，以及展示图
    """
    model =
GradientDescent(w,train_x,train_y,alpha,lambda_penalty=lambda_penalty)
    w = model.train()
    print('w={0}，它的模大小{1}'.format(w,w.T @ w))
    print('在训练集loss{0}'.format(model.loss(train_x,train_y)))
    print('在测试集loss{0}'.format(model.loss(test_x,test_y)))
    print('在训练集准确率{0}'.format(accuracy(w,train_x,train_y)))
    print('在测试集准确率{0}'.format(accuracy(w,test_x,test_y)))
    if display:
        displayResult(w,x,y,title,colors)


def
newtonTest(test_x,test_y,train_x,train_y,x,y,title,colors,lambda_penalty,display
=True,w = np.zeros(3)):
    """
    对模型测试并打印
    将会输出模型，模型的模值，损失值，准确率，以及展示图
    """
    model = NewTon(w,train_x,train_y,lambda_penalty=lambda_penalty)
    w = model.train()
    print('w={0}，它的模大小{1}'.format(w,w.T @ w))
    print('在训练集loss{0}'.format(model.loss(train_x,train_y)))
    print('在测试集loss{0}'.format(model.loss(test_x,test_y)))
    print('在训练集准确率{0}'.format(accuracy(w,train_x,train_y)))
    print('在测试集准确率{0}'.format(accuracy(w,test_x,test_y)))
    if display:
        displayResult(w,x,y,title,colors)

def newtonUCITest(train_x,train_y,x,y,title,colors,lambda_penalty,display
=True,w = np.zeros(3),test = False,test_x=None,test_y=None):
    """
    对模型测试并且，如果数据是二维的就打印,否则无法打印
    将会输出模型，模型的模值，损失值，准确率，以及展示图
    """
    model = NewTon(w,train_x,train_y,lambda_penalty=lambda_penalty)
    w = model.train()
    print('它的模大小{}'.format(w.T @ w))
    print('train_loss{0}'.format(model.loss(train_x,train_y)))
    print('在训练集准确率{0}'.format(accuracy(w,train_x,train_y)))
    if display:
        displayResult(w,x,y,title,colors)
    if test:
        print('test_loss{0}'.format(model.loss(test_x,test_y)))
        print('在测试集准确率{0}'.format(accuracy(w,test_x,test_y)))

if __name__ == '__main__':
```

```
    mean1 = np.array([1,2]) #分类一的均值
    mean2 = np.array([-1,-2]) #分类二的均值
    cov_xy = 0 #协方差
    var = 1 #方差
    order = 3
    size_pos = 50 #反例数量
    size_neg = 50 #正例数量
    colors=['green','red']#正反例颜色
    penalty = np.exp(-1)

    train_x,train_y,x,y = get_data(mean1,mean2,cov_xy,var,size_pos,size_neg)
    test_x,test_y,x,y = get_data(mean1,mean2,cov_xy,var,size_pos*5,size_neg*5)
    test(test_x,test_y,train_x,train_y,x,y,'满足朴素贝叶斯的无正则项',colors=
['green','red'],alpha=0.03,lambda_penalty=0)
    test(test_x,test_y,train_x,train_y,x,y,'满足朴素贝叶斯的正则项为
{0}'.format(penalty),colors=['green','red'],alpha=0.03,lambda_penalty=penalty)
    # #---------牛顿法-------------
    newtonTest(test_x,test_y,train_x,train_y,x,y,'满足朴素贝叶斯的无正则项的牛顿
法',colors=['green','red'],lambda_penalty=0)
    newtonTest(test_x,test_y,train_x,train_y,x,y,'满足朴素贝叶斯的正则项为{0}的牛顿
法'.format(penalty),colors=['green','red'],lambda_penalty=penalty)
    #---------不满足朴素贝叶斯------------------------------
    cov_xy = 0.3
    train_x,train_y,x,y = get_data(mean1,mean2,cov_xy,var,size_pos,size_neg)
    test_x,test_y,x,y = get_data(mean1,mean2,cov_xy,var,size_pos*5,size_neg*5)
    test(test_x,test_y,train_x,train_y,x,y,'协方差系数为{},无正则
项'.format(cov_xy),colors=['green','red'],alpha=0.03,lambda_penalty=0)
    test(test_x,test_y,train_x,train_y,x,y,'协方差系数为{},正则项为
{}'.format(cov_xy,penalty),colors=
['green','red'],alpha=0.03,lambda_penalty=penalty)
    #---------牛顿法-------------
    newtonTest(test_x,test_y,train_x,train_y,x,y,'协方差系数为{},无正则项的牛顿
法'.format(cov_xy),colors=['green','red'],lambda_penalty=0)
    newtonTest(test_x,test_y,train_x,train_y,x,y,'协方差系数为{},正则项为{}的牛顿
法'.format(cov_xy,penalty),colors=['green','red'],lambda_penalty=penalty)
```

lab2_uci.py

```
import numpy as np
from lab2 import newtonUCITest
def read_data(path):
    """
    读取数据
    """
    return np.loadtxt(path,str)

def processCNAE_Data(data):
    """
    处理数据，原本数据中有9个类别，这里将类别1和其他类别分开
    """
    dim = len(data[1].split(','))
    feature_data = np.ones((data.shape[0],dim))
    tag_data = np.zeros((data.shape[0],1))
    for i in range(len(data)):
        strlist = data[i].split(',')
        feature_data[i,1:] = strlist[1:]
```

```
        tag_data[i] = 1 if int(data[i].split(',')[0]) == 1 else 0
    return feature_data,tag_data,dim

def processHaberman_Data(data):
    dim = len(data[1].split(','))
    feature_data = np.ones((data.shape[0],dim))
    tag_data = np.zeros((data.shape[0],1))
    for i in range(len(data)):
        strlist = data[i].split(',')
        feature_data[i,1:] = strlist[:3]
        tag_data[i] = 1 if int(data[i].split(',')[3]) == 1 else 0
    return feature_data,tag_data,dim

def divide_data(feature_data,tag_data,ratio=0.3):
    trainsize = int(len(feature_data)*ratio)
    x_train_data = feature_data[:trainsize]
    x_test_data = feature_data[trainsize:]
    y_train_data = tag_data[:trainsize]
    y_test_data = tag_data[trainsize:]
    return x_train_data,x_test_data,y_train_data,y_test_data

if __name__ == '__main__':

    feature_data,tag_data,dim = processCNAE_Data(read_data('Lab2\CNAE-9.data'))
    x_train_data,x_test_data,y_train_data,y_test_data =
divide_data(feature_data,tag_data,ratio=0.1)
    #不带惩罚项
    newtonUCITest(x_train_data,y_train_data,None,None,None,None,0,display =
False,w = np.zeros((dim,1)),test = True,test_x=x_test_data,test_y=y_test_data)
    #带惩罚项

 newtonUCITest(x_train_data,y_train_data,None,None,None,None,np.exp(-1),display
= False,w = np.zeros((dim,1)),test = True,test_x=x_test_data,test_y=y_test_data)

    feature_data,tag_data,dim =
processHaberman_Data(read_data('Lab2\haberman.txt'))
    x_train_data,x_test_data,y_train_data,y_test_data =
divide_data(feature_data,tag_data,ratio=2/360)
    #不带惩罚项
    newtonUCITest(x_train_data,y_train_data,None,None,None,None,0,display =
False,w = np.zeros((dim,1)),test = True,test_x=x_test_data,test_y=y_test_data)
    #带惩罚项

 newtonUCITest(x_train_data,y_train_data,None,None,None,None,np.exp(-1),display
= False,w = np.zeros((dim,1)),test = True,test_x=x_test_data,test_y=y_test_data)
```