
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

# 2020 年春季学期

# 计算学部《机器学习》课程

# Lab 1 实验报告



| | |
|---|---|
| 姓名 | 赵仁杰 |
| 学号 | 1180300113 |
| 班号 | 1803001 |
| 电子邮件 | 1579974122@qq.com |
| 手机号码 | 15122925619 |

# 目录

# 1 实验目的

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、环境

## 2.1 实验要求

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。--这里使用的使用的是梯度下降。

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。

## 2.2 实验环境

x86-64,Win 10
Pycharm 2019.1
python 3.7

# 3 设计思想

## 3.1 逻辑回归介绍

Logistic Regression 虽然被称为回归，但其实际上是分类模型，并常用于二分类。Logistic Regression 因其简单、可并行化、可解释强深受工业界喜爱。Logistic 回归的本质是：假设数据服从这个分布，然后使用极大似然估计做参数的估计。

### 3.1.1 逻辑分布

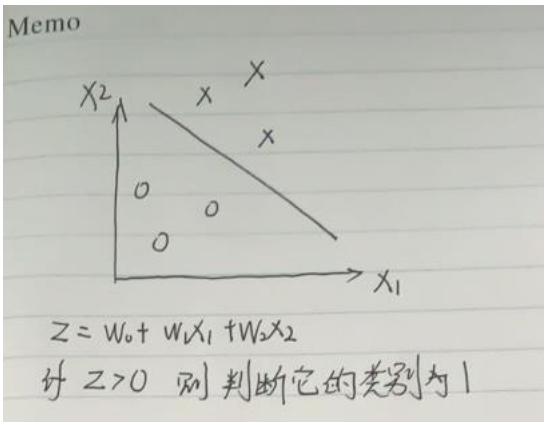
Logistic 分布是一种连续型的概率分布，其分布函数和密度函数分别为：

$$F(x) = P(X \leq x) = \frac{1}{1 + e^{-(x-\mu)/\gamma}}$$

$$f(x) = F'(X \leq x) = \frac{e^{-(x-\mu)/\gamma}}{\gamma(1 + e^{-(x-\mu)/\gamma})^2}$$

## 3.1.2 逻辑回归

之前说到 Logistic 回归主要用于分类问题，我们以二分类为例，对于所给数据集假设存在这样的一条直线可以将数据完成线性可分如下图所示：




我们所求的便是决策边界，决策边界可以表示 w1x1+w2x2+b=0,假设某个样本点 h(x)=w1w1+w2w2+b>0 那么可以判断它的类别为 1,这个过程其实是感知机。

Logistic 回归还需要加一层,它要找到分类概率 P(Y=1)与输入向量 X 的直接关系,然后通过比较概率值来判断类别。

考虑二分类问题,给定数据集：

$$D = (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N), x_i \subseteq R^n, y_i \in 0, 1, i = 1, 2, \cdots, N$$

我们要考虑(w.t)x+b 的取值是连续的，所以不可以拟合离散变量，他的取值空间为 R，不符合概率的理论模型，因此我们要考虑使用广义线性模型，比如说：单位阶跃函数：

$$p(y = 1|x) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases}, \quad z = w^T x + b$$

但是这个不可微，对数几率函数可以将其替代：$y = \frac{1}{1 + e^{-(w^T x + b)}}$

我们把 y 看作类后验概率估计，可以做的公式的重写，如下：

$$w^T x + b = ln\frac{P(Y=1|x)}{1-P(Y=1|x)}$$
$$P(Y=1|x) = \frac{1}{1+e^{-(w^T x+b)}}$$

也就是说，输出 Y=1 的对数几率是由输入 x 的线性函数表示的模型，这就是逻辑回归模型。当（w.t）x+b 的值越接近正无穷， [公式] 概率值也就越接近 1。因此逻辑回归的思路是，先拟合决策边界(不局限于线性，还可以是多项式)，再建立这个边界与分类的概率联系，从而得到了二分类情况下的概率。

按照上面的推导，直接将分类的模型建模，不用实现假设数据分布，同样可以得到预测的类型以及该预测的概率。

## 3.1.3 代价函数

逻辑回归模型的数学形式确定后，剩下就是如何去求解模型中的参数。在统计学中，常常使用极大似然估计法来求解，即找到一组参数，使得在这组参数下，我们的数据的似然度（概率）最大。令如下等式并且达到似然函数:

$$P(Y=1|x) = p(x)$$
$$P(Y=0|x) = 1-p(x)$$
$$L(w) = \prod [p(x_i)]^{y_i}[1-p(x_i)]^{1-y_i}$$

经过转变（两边取对数），写成对数似然:

$$\begin{aligned} L(w) &= \sum [y_i ln p(x_i) + (1-y_i) ln(1-p(x_i))] \\ &= \sum [y_i ln \frac{p(x_i)}{1-p(x_i)} + ln(1-p(x_i))] \\ &= \sum [y_i (w \cdot x_i) - ln(1+e^{w \cdot x_i})] \end{aligned}$$

这里利用卢函数的概念，衡量的是模型预测错误的程度，最大化似然函数和最小化损失函数是等价的。

## 3.1.4 求解

我们只需要对逻辑回归的损失函数进行求解即可，优化的主要目的是找到找到一个方向，参数朝着这个方向移动令损失函数的值能够减少，特点是：方向往往是由一阶偏导或者二阶偏导组合而成。

损失函数为:

$$J(w) = -\frac{1}{n}(\sum_{i=1}^{n}(y_i ln p(x_i) + (1-y_i) ln(1-p(x_i))))$$

# 3.2： 优化方式及正则化

这里使用的是梯度下降法，由于上一个实验已经实现过梯度下降法，这里仅仅是提供

简单的数学表示即可

### 3.2.1 梯度下降

梯度下降是通过 J(w) 对 w 的一阶导数来找下降方向，并且以迭代的方式来更新参数，更新方式为 ：

$$g_i = \frac{\partial J(w)}{\partial w_i} = (p(x_i) - y_i)x_i$$

$$w_i^{k+1} = w_i^k - \alpha g_i$$

这里的 k 是迭代次数，我们每次更新参数之后可以比较 w（k+1）-w（k）是否满足阈值或者到达最大迭代次数来停止迭代。

### 3.2.2 正则化

正则化是一个通用的算法和思想，所以会产生过拟合现象的算法都可以使用正则化来避免过拟合。

在经验风险最小化的基础上（也就是训练误差最小化），尽可能采用简单的模型，可以有效提高泛化预测精度。如果模型过于复杂，变量值稍微有点变动，就会引起预测精度问题。正则化之所以有效，就是因为其降低了特征的权重，使得模型更为简单。

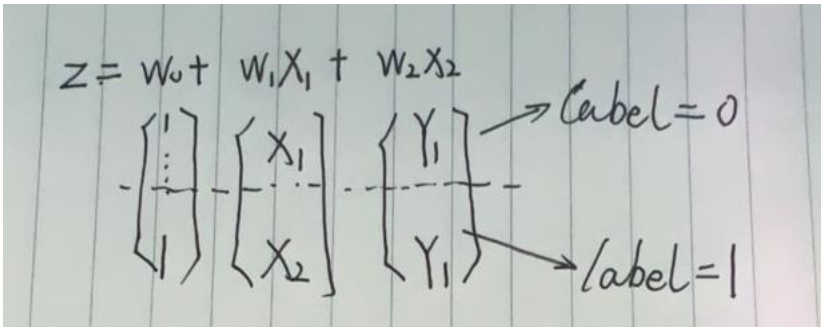
正则化一般会采用 L1 范式或者 L2 范式。给 loss function 加上正则化项，能使新得到的优化目标函数 [公式] ，需要在 f 和 ||w|| 中做一个权衡，如果还像原来只优化 f 的情况下，那可能得到一组解比较复杂，使得正则项 ||w|| 比较大，那么 h 就不是最优的，因此可以看出加正则项能让解更加简单，符合奥卡姆剃刀理论，同时也比较符合在偏差和方差（方差表示模型的复杂度）分析中，通过降低模型复杂度，得到更小的泛化误差，降低过拟合程度。正则化是结构风险最小化的一种策略实现。

# 4：算法实现

有关参数： data ： 数据量 alpha：步长 max：设置的最大迭代次数

## 4.1 生成数据:

先是生成满足朴素贝叶斯的数据，生成的逻辑如下:




代码如下：

```
# data 个高斯噪声 data*1
X1 = np.random.normal(0,2,data).reshape(data,1)
# data 个高斯噪声 data*1
Y1 = np.random.normal(0,2,data).reshape(data,1)
# data 个高斯噪声 data*1
X2 = np.random.normal(5,2,data).reshape(data,1)
# data 个高斯噪声 data*1
Y2 = np.random.normal(5,2,data).reshape(data,1)
```

之后 `def data_normolize(x1,y1,x2,y2):`

将生成的数据转化成（3*data,1）的数据。

不满足朴素贝叶斯的数据和上面的生成方法类似，只是让协方差不等于 0 的时候，使两个参数相关即可。

## 4.2 读取数据

这个使读取网上的数据，因为这个实验我只是实现了二元逻辑回归，所以读取数据指数读取前两列以及最后的 label 值。

函数名为：`def load_data(filename):`

思路：将数据按照结构读取，之后切片转化

## 4.3 梯度下降：

我们已经设置了最大迭代次数，具体实现代码如下：

```
#梯度下降
def grad(x1,y1,alpha,max):
    #随机生成系数
    W = np.mat(np.random.randn(3,1))
    for i in range(0,max):
        H = 1/(1+np.exp(-1*x1*W))
        # 3,1，损失函数
        dw = x1.T*(H-y1)
        W = W - alpha*dw
    return W
```

至于加入正则项，只是在上面做些许改动即可。

## 4.4 计算正确率：

```
def test_correct(tx1,ty1,tx2,ty2,tlabel,w0,w1,w2,data):
    x = 0
    y = 0
    for i in range(0,data):
        if  (w0+w1*tx1[i]+w2*ty1[i]>=0.0):
            x = x+1
        if  (w0+w1*tx2[i]+w2*ty2[i]<=0.0):
            y = y+1
    return 1-(x+y)/(2*data)
```

我们只需要比较（w.t）x+b 与边界的比较即可。

# 5 实验结果与分析

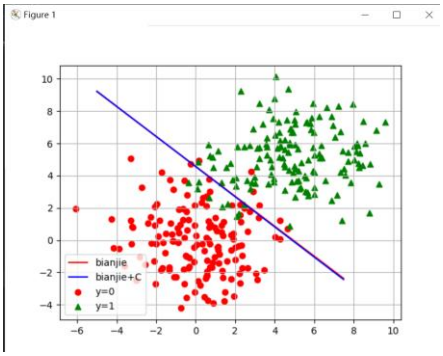
如上述生成数据，x1 与 y1 是由高斯函数生成的数据其对应 z=w0+w1x1+w2x2 里面的标签为 0 的（x1，x2）.同理 x2 与 y2 相同，只不过标签为 1.

## 5.1 验证算法正确性

这里使用的使：满足朴素贝叶斯的数据，其余参数设置如下：

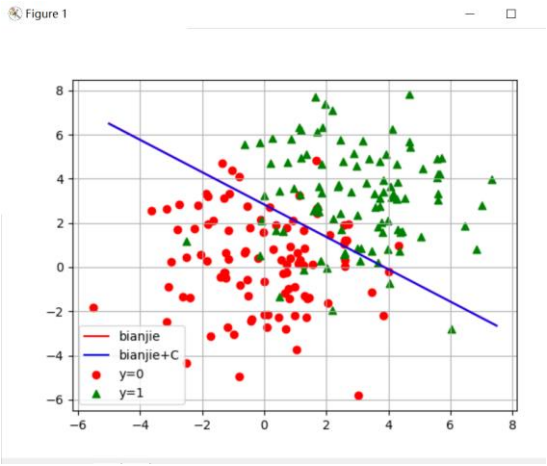
```
#数据量
data = 150
#步长
alpha = 0.001
#设置迭代次数
max = 10000
```

结果如下图所示（附加正确率）：




```
Accuracy: 95.0 %
Accuracy+C: 95.0 %
```

上面是满足朴素贝叶斯，下面是不满足朴素贝叶斯的结果与正确率：




```
Accuracy: 86.66666666666667 %
Accuracy+C: 86.66666666666667 %
```

如上面可以分析得出，该实验由手工生产的数据，算法实现成功，符合预想结果。

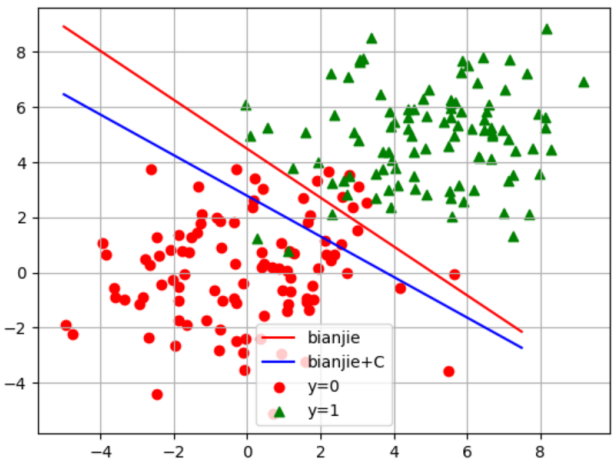
## 5.2： 观察加入正则项

这里只针对朴素贝叶斯贝叶斯进行分析：（补偿和迭代次数都不修改）

```
#步长
alpha = 0.001
#设置迭代次数
max = 1000
```

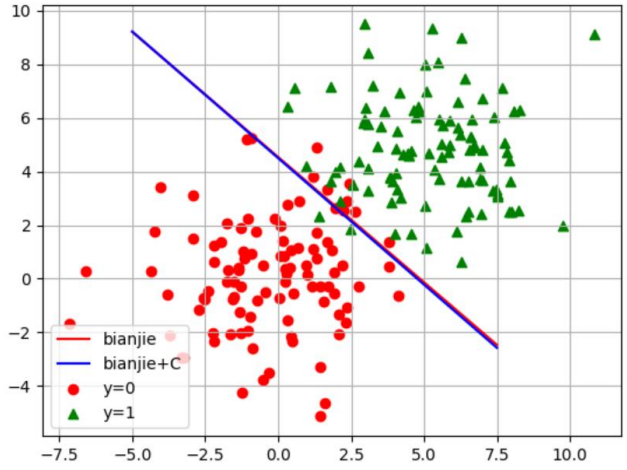
多次修改数据量以及系数观察加入正则项对其影响

```
#数据量
data = 100
# 正则项
W = W*(1-1e-2)
```




```
Accuracy: 94.5 %
Accuracy+C: 91.0 %
```

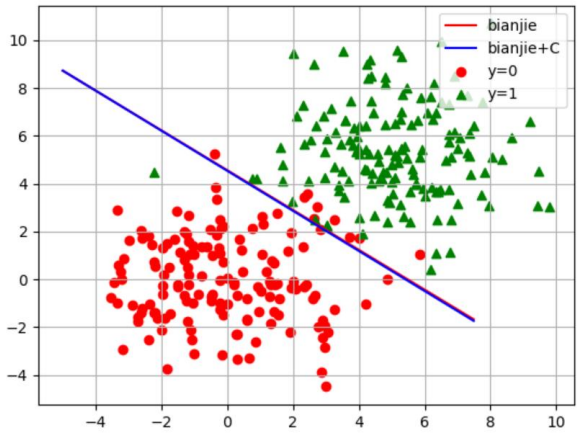
```
#数据量
data = 100
# 正则项
W = W*(1-1e-4)
```




```
Accuracy: 95.0 %
Accuracy+C: 95.0 %
```

这里可以看到进行到一定的迭代次数，两者的表现都十分优秀。

```
#数据量
data = 150
# 正则项
W = W*(1-1e-4)
```




```
Accuracy: 96.66666666666667 %
Accuracy+C: 96.66666666666667 %
```

由照片可以得知，加不加正则项，其实对于逻辑回归的正确率影响不是很大。

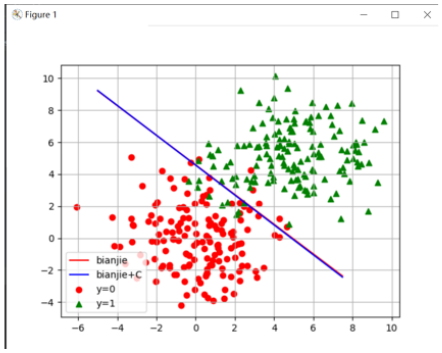
## 5.3： 观察是否满足贝叶斯分布

```
#数据量
data = 150
#步长
alpha = 0.001
#设置迭代次数
max = 10000
```

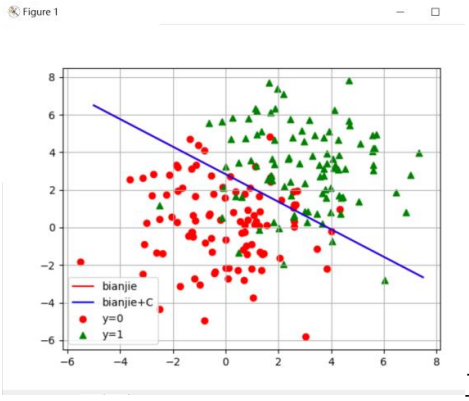
第一张图与对应的正确率是满足朴素贝叶斯的数据

第二张图与对应的正确率是不满足朴素贝叶斯的数据

结果如下图所示（附加正确率）：




```
Accuracy: 95.0 %
Accuracy+C: 95.0 %
```




正确率为：90

在数据量=150 的时候：

正确率分别是 96.5 和 93

其他的 data 量有关的正确率有一点误差影响，但是如图所示，数据是否满足朴素贝叶斯对其正确率的印象很小，逻辑回归是在线性回归的基础上套上了逻辑函数，对于数据是否满足朴素贝叶斯并没有硬性要求。

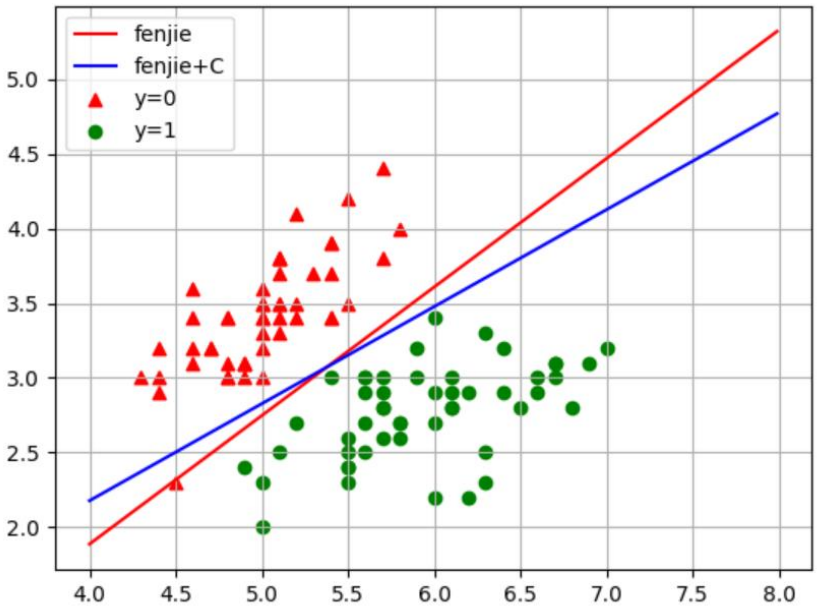
## 5.4： 对于 UCI 数据集测试

自己写的是二元逻辑回归，因此准确率有一点偏差，实现如下：

参数为：

```
#数据量
data = 150
#步长
alpha = 0.001
#设置迭代次数
max = 10000
#从文件中读取数据
def load_data(filename):
    file = open(filename)
    x = []
    #y 是这里该数据的标签
    y = []
    for line in file.readlines():
        line = line.strip().split()
```

```
        x.append([1,float(line[0]),float(line[1])])
        y.append(float(line[-1]))
    #转化成相对应的矩阵
    x1 = np.mat(x)
    y1 = np.mat(y).T
    file.close
    return x1,y1
```




对应的正确率如下：

```
data1 100
Accuracy: 99.0 %
data1 100
Accuracy+C: 99.0 %
```

可以看出：该模型在这个数据集上的表现十分优秀，可能与所取的数据量过小有一定的关系。

# 6 总结：

1：有关数据是否满足朴素贝叶斯：从结果中可以看出，类条件分布在满足朴素贝叶斯假设时的分类表现略好于不满足朴素贝叶斯假设时。

2：逻辑回归可以很好地解决线性分类问题，而且收敛速度较快，其实几百次迭代就可以得到非常理想的结果（我的模型里迭代次数过高而且没有使用阈值造成则相应的浪费）

3：关于惩罚项：由于上述原因， 对于逻辑回归而言，带正则项和不带正则项的差别没有多项式拟合函数那么大。尤其是当使用随机梯度下降法时，由于随机梯度下降法选择样本的不确定性，在相同迭代次数和相同参数条件下，基本无法看出显著差异。

4：正确率除了模型的好坏之外，还受到数据的影响，比如说数据的大小、数据的特征等等。

代码如下：

```
'''
    @author:1180300113
    导入数据，观察自己写的梯度下降实现的逻辑回归正确率
    其中包括不带正确项以及带正确项
'''
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
import random

#数据量
data = 150
#步长
alpha = 0.001
#设置迭代次数
max = 100000

#将生成的数据进行规范换
def data_normolize(x1,y1,x2,y2):
    #data*1 的全 0 矩阵
    x0 = np.ones((data,1))*0
    # 合并 x1,y1
    x1y1 = np.hstack((x1,y1))
    # data*1 的全 1 矩阵
    list1 = np.ones((data,1))*1
    # 合并 x2,y2
    x2y2 = np.hstack((x2,y2))
    # 生成 2*data*1 的全 1 矩阵
    list2 = np.ones((2*data,1))*1
    # 合并 x1y1 和 x2y2
    data1 = np.vstack((x1y1,x2y2))
    # 生成[1,x1y1,x2y2]矩阵
    data1 = np.hstack((list2,data1))

    # 生成特征矩阵 2data*1 的
    label = np.vstack((x0,list1))
    return data1,label


#从文件中读取数据
def load_data(filename):
    file = open(filename)
    x = []
```

```
    #y 是这里该数据的标签
    y = []
    for line in file.readlines():
        line = line.strip().split()
        x.append([1,float(line[0]),float(line[1])])
        y.append(float(line[-1]))
    #转化成相对应的矩阵
    x1 = np.mat(x)
    y1 = np.mat(y).T
    file.close
    return x1,y1
 #梯度下降
def grad(x1,y1,alpha,max):
    #随机生成对应的系数
    W = np.mat(np.random.randn(3,1))
    for i in range(0,max):
        H = 1/(1+np.exp(-1*x1*W))
        #根据数据生成对应的损失函数
        dw = x1.T*(H-y1)#3,1
        W = W - alpha*dw
    return W

#梯度下降+正则项
def grad_c(x1,y1,alpha,max):
    W = np.mat(np.random.randn(3,1))
    for i in range(0,max):
        #加入正则项
        W = W*(1-1e-4)
        H = 1/(1+np.exp(-1*x1*W))
        #损失函数
        dw = x1.T*(H-y1)#3,1
        W = W - alpha*dw
    return W

def test_correct(data,label,w0,w1,w2):
    data1 = np.size(label,0)
    print('data1',data1)
    x = 0.0
    y = 0.0
    for i in range(0,data1):
        if((w0+w1*data[:,1][i]+w2*data[:,2][i]>=0.0) and
(label[i]==0)):
            x=x+1
        if((w0+w1*data[:,1][i]+w2*data[:,2][i]<=0.0) and
```

```
(label[i]==1)):
            y =y+1
    return 1-(x+y)/data1
#读入文件，生成对应数据
data,label = load_data('data.txt')
#梯度下降生成对应参数
W = grad(data,label,alpha,max)
#梯度下降带正则项生成对应参数
W_C = grad_c(data,label,alpha,max)
print('W',W)
print('WC',W_C)


W0 = W[0,0]
W1 = W[1,0]
W2 = W[2,0]
WC0 = W_C[0,0]
WC1 = W_C[1,0]
WC2 = W_C[2,0]

# data 文件夹里的数据的正确率
per = test_correct(data,label,W0,W1,W2)
#正确率
print('Accuracy:',per*100,'%')
# data 文件夹里的数据的正确率
per_c = test_correct(data,label,WC0,WC1,WC2)
# 正确率
print('Accuracy+C:',per_c*100,'%')

plotx1 = np.arange(4,8,0.01)
plotx2 = -W0/W2 - W1/W2*plotx1
plotxWC2 = -WC0/WC2 - WC1/WC2*plotx1
plt.plot(plotx1,plotx2,c='r',label='fenjie')
plt.plot(plotx1,plotxWC2,c='b',label='fenjie+C')

# 文件中读取的数 y=0
plt.scatter(data[:,1][label==0].A,data[:,2][label==0].A,marker='^',c
= 'r',label='y=0')
#  y=1
plt.scatter(data[:,1][label==1].A,data[:,2][label==1].A,c =
'g',label='y=1')

plt.grid()
```

```
plt.legend()
plt.show()
'''
    @author:1180300113
    自己产生数据
    之后利用梯度下降实现逻辑回归
'''
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
import random

#数据量
data = 90
#步长
alpha = 0.001
#设置迭代次数
max = 10000

def gauss():
    # data 个高斯噪声 data*1
    X1 = np.random.normal(0,2,data).reshape(data,1)
    # data 个高斯噪声 data*1
    Y1 = np.random.normal(0,2,data).reshape(data,1)
    # data 个高斯噪声 data*1
    X2 = np.random.normal(5,2,data).reshape(data,1)
    # data 个高斯噪声 data*1
    Y2 = np.random.normal(5,2,data).reshape(data,1)
    return X1,Y1,X2,Y2

#将生成的数据进行规范换
def data_normolize(x1,y1,x2,y2):
    #data*1 的全 0 矩阵
    x0 = np.ones((data,1))*0
    # 合并 x1,y1
    x1y1 = np.hstack((x1,y1))
    # data*1 的全 1 矩阵
    list1 = np.ones((data,1))*1
    # 合并 x2,y2
    x2y2 = np.hstack((x2,y2))
    # 生成 2*data*1 的全 1 矩阵
    list2 = np.ones((2*data,1))*1
    # 合并 x1y1 和 x2y2
```

```
    data1 = np.vstack((x1y1,x2y2))
    # 生成[1,x1y1,x2y2]矩阵
    data1 = np.hstack((list2,data1))

    # 生成特征矩阵 2data*1 的
    label = np.vstack((x0,list1))
    return data1,label

#梯度下降
def grad(x1,y1,alpha,max):
    #随机生成系数
    W = np.mat(np.random.randn(3,1))
    for i in range(0,max):
        H = 1/(1+np.exp(-1*x1*W))
        # 3,1，损失函数
        dw = x1.T*(H-y1)
        W = W - alpha*dw
    return W

#梯度下降+正则项
def grad_C(x1,y1,alpha,max):
    # 随机生成系数
    W = np.mat(np.random.randn(3,1))
    for i in range(0,max):
        # 正则项
        W = W*(1-1e-4)
        H = 1/(1+np.exp(-1*x1*W))
        # 3,1，损失函数
        dw = x1.T*(H-y1)
        W = W - alpha*dw
    return W

def test_correct(tx1,ty1,tx2,ty2,tlabel,w0,w1,w2,data):
    x = 0
    y = 0
    for i in range(0,data):
        if  (w0+w1*tx1[i]+w2*ty1[i]>=0.0):
            x = x+1
        if  (w0+w1*tx2[i]+w2*ty2[i]<=0.0):
            y = y+1
    return 1-(x+y)/(2*data)
```

```
 #训练集
x1,y1,x2,y2 = gauss()
#根据训练集生成对应矩阵以及标签
data1,label = data_normolize(x1,y1,x2,y2)
# 测试集
tx1,ty1,tx2,ty2 = gauss()
#根据测试集生成矩阵以及标签'''
tdata1,tlabel = data_normolize(tx1,ty1,tx2,ty2)



W = grad(data1,label,alpha,max)
#加入惩罚项的
W_C = grad_C(data1,label,alpha,max)
print('W',W)
print('WC',W_C)


W0 = W[0,0]
W1 = W[1,0]
W2 = W[2,0]
WC0 = W_C[0,0]
WC1 = W_C[1,0]
WC2 = W_C[2,0]

# 测试集正确率
per = test_correct(tx1,ty1,tx2,ty2,tlabel,W0,W1,W2,data)
print('Accuracy:',per*100,'%')
# 测试集+惩罚项正确率
per_c = test_correct(tx1,ty1,tx2,ty2,tlabel,WC0,WC1,WC2,data)
print('Accuracy+C:',per_c*100,'%')

plotx1 = np.arange(-5,7.5,0.01)
plotx2 = -W0/W2 - W1/W2*plotx1
plotxWC2 = -WC0/WC2 - WC1/WC2*plotx1
plt.plot(plotx1,plotx2,c='r',label='bianjie')
plt.plot(plotx1,plotxWC2,c='b',label='bianjie+C')

#测试集的点  y=0
plt.scatter(tx1,ty1,c = 'r',label='y=0')
# 测试集的点  y=1
plt.scatter(tx2,ty2,c = 'g',marker='^',label='y=1')
```

```
plt.grid()
plt.legend()
plt.show()
```