

**2020年春季学期  
计算学部《机器学习》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 余涛 |
| 学号 | 1180300829 |
| 班号 | 1803202 |
| 电子邮件 | 1063695334@qq.com |
| 手机号码 | 15586430583 |

**目录**

[1 实验内容 3](#_Toc54519786)

[1.1 实验目的 3](#_Toc54519787)

[1.2实验要求 3](#_Toc54519788)

[1.3实验环境 3](#_Toc54519789)

[2 实验设计思想 3](#_Toc54519790)

[2.1 算法原理 3](#_Toc54519791)

[2.1.1 手工生成0/1数据集（是否满足贝叶斯假设） 3](#_Toc54519792)

[2.1.2 梯度下降法实现逻辑回归（无惩罚项和有惩罚项） 3](#_Toc54519793)

[2.1.3 UCI数据集样本太多时取部分数据进行试验 6](#_Toc54519794)

[2.1.4 判断测试集的准确率 6](#_Toc54519795)

[2.1.5 数据太大时防止exp溢出 6](#_Toc54519796)

[2.2 算法实现 7](#_Toc54519797)

[2.2.1手工生成0/1数据集（是否满足贝叶斯假设） 7](#_Toc54519798)

[2.2.2 梯度下降法实现逻辑回归（无惩罚项和有惩罚项） 8](#_Toc54519799)

[2.2.3 UCI数据集样本太多时取部分数据进行试验 9](#_Toc54519800)

[2.2.4 判断测试集的准确率 10](#_Toc54519801)

[2.2.5 数据太大时防止exp溢出 11](#_Toc54519802)

[2.2.6 绘制逻辑回归图像 12](#_Toc54519803)

[2.2.7 绘制迭代次数与损失函数值关系图像 13](#_Toc54519804)

[3 实验结果分析 13](#_Toc54519805)

[3.1 生成满足朴素贝叶斯假设的数据 13](#_Toc54519806)

[3.2 生成不满足朴素贝叶斯假设的数据 14](#_Toc54519807)

[3.3 使用UCI上的Skin\_NonSkin.csv进行测试（三维） 16](#_Toc54519808)

[3.4 使用UCI上的blood.csv进行测试 16](#_Toc54519809)

[3.5 使用UCI上的heart.csv进行测试 18](#_Toc54519810)

[3.6 使用UCI上的data\_banknote\_authentication.csv进行测试 19](#_Toc54519811)

[4 结论 20](#_Toc54519812)

[5 源代码（含详细注释） 20](#_Toc54519813)

# 实验内容

## 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。

## 1.2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。

## 1.3实验环境

Windows 10 专业版；python 3.8.6；PyCharm Community Edition 2020.2.2 x64

# 实验设计思想

## 2.1 算法原理

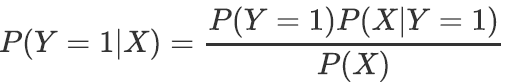
### 2.1.1 手工生成0/1数据集（是否满足贝叶斯假设）

首先设置随机变量方差，然后设置两个维度的协方差，设置类别1的两个均值和类别2的两个均值，然后创建一个二维的点集和点集的分类集。对于二维点集，根据是否满足贝叶斯假设，对前一半创建第一类数据集，对后一半创建第二类数据集并且都加上一个N(0,1)的高斯噪声，最后给分类集赋值0或者1。

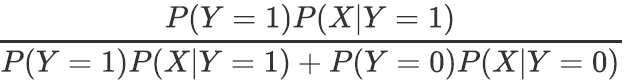
### 2.1.2 梯度下降法实现逻辑回归（无惩罚项和有惩罚项）

分类器做分类问题的实质，就是预测一个已知样本的位置标签，即P(Y = 1|X = <X1,…,Xn>)，根据朴素贝叶斯的方法，可以用贝叶斯概率公式，将上式转化为条件概率和类概率的乘积，即利用P(Y)，P(X|Y)和各个维度之间的计算条件独立的假设来计算P(Y|X)。本次实验却不需要这么复杂，而是直接求解这个概率。

对于0/1分类来说，可以有以下推导：

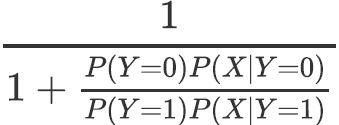


对分母P(X)进行变换可得：  
右边=



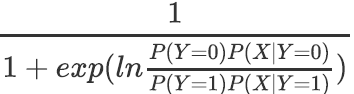
上下同除P(Y=1)P(X|Y=1)得：

右边=



继续变换可得：

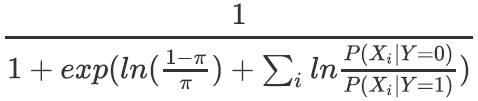
右边=



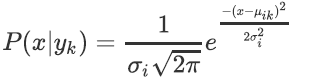
令

，

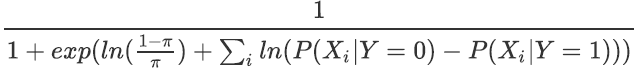
这时右边=



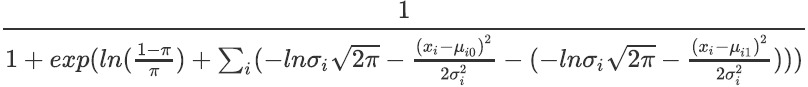
假设类条件分布服从正态分布并且方差不依赖于k，则有



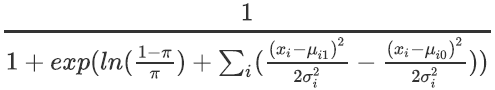
此时右边=



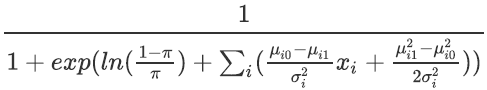
=



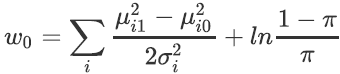
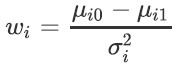
=



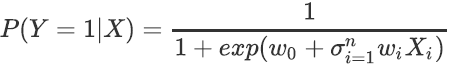
=



令

，

则有：



sigmoid函数：这个函数将线性模型预测的连续值映射到0到1的概率上，从而得到0/1的离散值：



其作用是将实数值映射到0到1之间的某个值，在x=0处函数值取0.5，并且以x=0为界，函数值分别向1和0逼近。

此时的

由于P( Y = 0 | X )=1-P( Y=1 | X ),则有：

两者相除=1时即可以得到分类决策面(线)，即有：  
=1，此时



即是分类决策面(线)，所以需要求解参数w。

我们使用梯度下降法来获得逻辑回归的参数w，参数为wi，0≤i≤n，n是样本集的维度。

需要计算P(< X,Y>|w),可以转化为极大条件似然估计(MCLE)，只需要计算P(X|Y，w), 有MCLE的推导式：

l(w)=



=



=



需要求得MCLE最大时对应的w值，即求解的w值，对l(w)取反。此时就可以将-l(w)作为梯度下降法的损失函数，由于数据特别多的情况下，可能导致溢出，所以需要对-l(w)添加一个系数1/L（L为样本的总数），就可以得到此时的损失函数:

l(w)=\*



然后对损失函数求偏导数，有：



可以得到梯度下降的迭代公式为：



为了避免出现过拟合的现象，可以增加惩罚项，增加惩罚项后的梯度下降的迭代公式为：



然后就可以通过迭代来完成对参数w的估计，进而得到分界决策面(线)，完成分类。

### 2.1.3 UCI数据集样本太多时取部分数据进行试验

在进行Skin\_NonSkin.csv进行逻辑回归试验时，由于该数据集含有几十万条数据，数据集的样本太大，需要将其先打乱顺序，然后按照比较大的步长取部分数据作为数据集。

### 2.1.4 判断测试集的准确率

在求解出系数w后，对于测试集中每个数据点X，只需要将的结果与0进行判断，就能得出X预测的分类。在本实验中，若＞0，则X属于分类1。若＜0，则X属于分类0。然后将测试集预测的分类正确数除实际的数据总数，就可以得到测试集的准确率。

### 2.1.5 数据太大时防止exp溢出

由于在试验中，经常调用exp(-x)，很容易由于数据过大导致溢出（在本人的试验中，blood.csv和heart.csv都出现了exp溢出），为了防止exp溢出，可以对于样本集的所有维度的数据，计算其最大值和最小值的差作为极差，然后用最大值减当前数据，然后用得到的数据除极差作为新的数据，就可以将所有的数据缩减为[0,1]区间的数据，这样就能防止exp溢出。

## 2.2 算法实现

### 2.2.1手工生成0/1数据集（是否满足贝叶斯假设）

作用：

'''  
自己生成数据  
根据是否满足贝叶斯假设，在范围[0,1]上生成sample\_number个二维点集，并将其均匀分为两类  
并添加一个N(0,1)的高斯噪声  
'''

具体实现：

首先设置随机变量方差为0.2，然后设置两个维度的协方差为0.01，设置类别1的两个均值为-0.7和-0.3，设置类别2的两个均值为0.7和0.3，然后创建一个二维的点集train\_point\_X和点集的分类集classification\_Y。对于二维点集train\_point\_X，根据是否满足贝叶斯假设，对前一半创建第一类数据集0，对后一半创建第二类数据集1并且都加上一个N(0,1)的高斯噪声，最后给分类集赋值0或者1。以注释形式给出。

def create\_datas(sample\_number, satisfy\_naive):  
 half = np.ceil(sample\_number / 2).astype(np.int32) # sample\_number的一半  
 variance = 0.2 # 随机变量方差  
 covariance\_xy = 0.01 # 两个维度的协方差  
 point\_mean1 = [-0.7, -0.3] # 类别1的均值  
 point\_mean2 = [0.7, 0.3] # 类别2的均值  
 train\_point\_X = np.zeros((sample\_number, 2)) # 二维的点集  
 classification\_Y = np.zeros(sample\_number) # 点集的分类  
 if satisfy\_naive: # 满足朴素贝叶斯假设  
 train\_point\_X[:half, :] = np.random.multivariate\_normal(point\_mean1, [[variance, 0], [0, variance]],  
 size=half) # 生成half个类别1的1\*2数组，每个数组含有两个维度  
 train\_point\_X[half:, :] = np.random.multivariate\_normal(point\_mean2, [[variance, 0], [0, variance]],  
 size=sample\_number - half) # 生成half个类别2的1\*2数组，每个数组含有两个维度  
 classification\_Y[:half] = 0 # 将前half个数组标记为类别1  
 classification\_Y[half:] = 1 # 将后half个数组标记为类别2  
 else: # 不满足朴素贝叶斯假设  
 train\_point\_X[:half, :] = np.random.multivariate\_normal(point\_mean1,  
 [[variance, covariance\_xy], [covariance\_xy, variance]],  
 size=half) # 生成half个类别1的1\*2数组，每个数组含有两个维度  
 train\_point\_X[half:, :] = np.random.multivariate\_normal(point\_mean2,  
 [[variance, covariance\_xy], [covariance\_xy, variance]],  
 size=sample\_number - half) # 生成half个类别2的1\*2数组，每个数组含有两个维度  
 classification\_Y[:half] = 0 # 将前half个数组标记为类别1  
 classification\_Y[half:] = 1 # 将后half个数组标记为类别2  
 return train\_point\_X, classification\_Y # 返回生成的所有点及类别

### 2.2.2 梯度下降法实现逻辑回归（无惩罚项和有惩罚项）

作用：

'''  
加惩罚项的梯度下降法  
对于train\_point\_X和classification\_Y对参数w做梯度下降，对损失函数使用梯度下降法，当误差函数收敛到期望的最小值时，得到此时的w并返回w  
参数中：  
迭代最大次数为cycle\_times  
下降步长descending\_step\_size  
迭代误差iteration\_error  
数据点集维度dimension  
惩罚项参数lamda  
'''

具体实现：在上面原理中已经给出损失函数和迭代公式，通过对迭代公式



不断迭代，当此时的迭代误差小于规定的迭代误差时终止迭代，如果迭代后的新的迭代误差大于原迭代误差，说明此时的迭代值在解析解左右摇摆，此时需要将步长变为原来的一半。

对于不需要惩罚项的迭代，只需要将惩罚项的系数传为0即可。以注释形式给出。

def descent\_gradient\_add\_errorfunction(train\_point\_X, classification\_Y, cycle\_times, descending\_step\_size,  
 iteration\_error, dimension, lamda):  
 total\_points = np.size(train\_point\_X, axis=0) # 得到train\_point\_X的行数，即点集个数  
 w = np.ones((1, dimension + 1)) # 生成系数矩阵w，一个列数为dimension + 1，行数为1的矩阵，元素值全为1  
 cycle\_times\_list = np.zeros(cycle\_times) # 迭代次数统计  
 loss\_list = np.zeros(cycle\_times) # 迭代次数对应的损失函数值统计  
 for i in range(cycle\_times):  
 old\_loss = - 1 / total\_points \* maximum\_conditional\_likelihood(train\_point\_X, classification\_Y, w) # 原先损失函数的值  
 t = np.zeros((total\_points, 1))  
 for j in range(total\_points):  
 t[j] = w.dot(train\_point\_X[j].T) # 极大条件似然中的exp中的部分，即求和wi \* Xl  
 gradient = - 1 / total\_points \* (classification\_Y - sigmoid(t.T)).dot(train\_point\_X)  
 w = w - descending\_step\_size \* lamda \* w - descending\_step\_size \* gradient # 梯度下降加惩罚项的迭代公式  
 new\_loss = - 1 / total\_points \* maximum\_conditional\_likelihood(train\_point\_X, classification\_Y, w) # 新的损失函数的值  
 cycle\_times\_list[i] = i # 储存迭代次数  
 loss\_list[i] = new\_loss # 储存每次迭代对应的损失函数值  
 if abs(new\_loss - old\_loss) < iteration\_error: # 如果新的误差函数值与旧的误差函数值的差小于迭代误差则终止迭代  
 cycle\_times\_list = cycle\_times\_list[:i + 1]  
 loss\_list = loss\_list[:i + 1]  
 print('迭代次数=', i, ', 对应的损失函数值=', new\_loss, ', 对应的系数w=', w, '对应的梯度=', gradient)  
 break  
 if new\_loss > old\_loss: # 当新的误差函数值大于旧的误差函数值时，将步长变为原来的一半  
 descending\_step\_size \*= 0.5  
 return w, cycle\_times\_list, loss\_list

### 2.2.3 UCI数据集样本太多时取部分数据进行试验

作用：

'''  
读入Skin\_NonSkin.csv文件，对数据进行拆分，拆分成训练集合测试集  
由于原文件中数据量巨大，所以对数据集以50步长取部分数据作为数据集  
'''

具体实现：该数据集含有几十万条数据，数据集的样本太大，需要将其先打乱顺序，然后按照50的步长取部分数据作为数据集。然后将数据集分为训练集和测试集。测试集所占比例为20%。以注释形式给出。

def Skin\_NonSkin\_getdata():  
 all\_data = np.loadtxt(open('./Skin\_NonSkin.csv'), delimiter=",", skiprows=0) # 读取文件中的所有数据  
 np.random.shuffle(all\_data) # 将原数据集打乱，分成训练集和测试集  
 test\_rate = 0.2 # 测试集所占比例  
 all\_data\_size = np.size(all\_data, axis=0) # 总数据集数据数量  
 train\_data\_X = all\_data[:int(test\_rate \* all\_data\_size), :] # 训练集数据  
 test\_data\_x = all\_data[int(test\_rate \* all\_data\_size):, :] # 测试集数据  
 dimension = np.size(all\_data, axis=1) - 1 # 训练集样本维度  
 step = 50 # 由于Skin\_NonSkin的数据集太大，所以采用步长为50的方式取数据  
 train\_point\_X = train\_data\_X[:, 0:dimension] # 将所有数据集赋给train\_point\_X  
 train\_point\_X = train\_point\_X[::step] # 以step为间隔取数据  
 train\_point\_X = train\_point\_X - 100 # 对样本点进行坐标平移，方便在3D图中显示  
 train\_classification\_Y = train\_data\_X[:, dimension:dimension + 1] - 1 # 因为数据集的分类是1/2,需要减1变成0/1  
 train\_classification\_Y = train\_classification\_Y[::step] # 以step为间隔取数据  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集数据总数  
 train\_classification\_Y = train\_classification\_Y.reshape(train\_size) # 将矩阵转化为行向量  
 test\_point\_X = test\_data\_x[:, 0:dimension] # 将所有数据集赋给test\_point\_X  
 test\_point\_X = test\_point\_X[::step] - 100 # 对样本点进行坐标平移，方便在3D图中显示  
 test\_classification\_Y = test\_data\_x[:, dimension:dimension + 1] - 1 # 因为数据集的分类是1/2,需要减1变成0/1  
 test\_classification\_Y = test\_classification\_Y[::step] # 以step为间隔取数据  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集数据总数  
 test\_classification\_Y = test\_classification\_Y.reshape(test\_size) # 将矩阵转化为行向量  
 return train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y

### 2.2.4 判断测试集的准确率

作用：

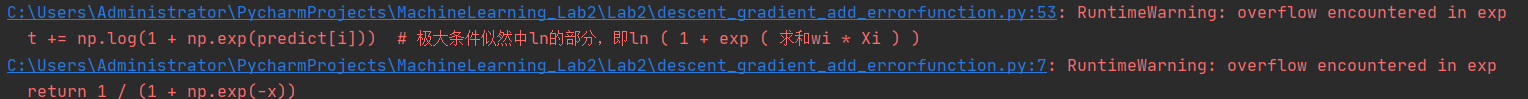
得到测试集命中的准确率。

具体实现：在求解出系数w后，对于测试集中每个数据点X，只需要将的结果与0进行判断，就能得出X预测的分类。在本实验中，若＞0，则X属于分类1。若＜0，则X属于分类0。然后将测试集预测的分类正确数除实际的数据总数，就可以得到测试集的准确率。以注释形式给出。

for i in range(test\_size): # 对每种预测给label进行赋值  
 if w.dot(test\_all[i].T) >= 0:  
 label[i] = 1  
 else:  
 label[i] = 0  
for i in range(test\_size):  
 if label[i] == test\_classification\_Y[i]: # 如果预测的结果与真实结果相同计数加一  
 hit\_count += 1  
hit\_rate = hit\_count / test\_size  
print('数据的测试集的准确率为：', hit\_rate)

### 2.2.5 数据太大时防止exp溢出

作用：

防止调用exp(-x)发生exp溢出。exp溢出如下：  


具体实现：可以对于样本集的所有维度的数据，计算其最大值和最小值的差max(train\_data\_X[:, i]) - min(train\_data\_X[:, i])作为极差d\_length，然后用最大值max减当前数据，然后用得到的数据除极差(max(train\_data\_X[:, i]) - train\_data\_X[j, i]) / d\_length作为新的数据，就可以将所有的数据缩减为[0,1]区间的数据，这样就能防止exp溢出。以注释形式给出。

# 消除exp溢出，防止数据太大导致exp溢出  
for i in range(dimension): # 对于样本的所有维度  
 d\_length = max(train\_data\_X[:, i]) - min(train\_data\_X[:, i]) # 计算最大值和最小值之间的极差  
 for j in range(np.size(train\_data\_X, axis=0)): # 对于每一维度的所有数  
 train\_data\_X[j, i] = (max(train\_data\_X[:, i]) - train\_data\_X[j, i]) / d\_length # 将其化为[0,1]之间的值，防止exp溢出

### 2.2.6 绘制逻辑回归图像

作用：

'''  
对于train\_point\_X中，含有二维数据，分别对应横坐标x和纵坐标y  
对于classification\_Y,为train\_point\_X中两种数据的点对应的类别  
根据train\_point\_X和classification\_Y画出二维坐标下的点图，然后画出分界判别函数boundary\_check\_function  
'''

'''  
Skin\_NonSkin数据集是三个维度的，所以可以画出三维的图像。  
'''

具体实现：对于自己手工生成的二维数据，可以生成二维图像。对于Skin\_NonSkin.csv，由于是三维数据，可以绘制三维图像。其他的UCI数据集由于是多维的，所以并没有绘制图像以注释形式给出。

def draw\_picture(train\_point\_X, classification\_Y, boundary\_check\_function):  
 if boundary\_check\_function: # 绘制分界判定函数boundary\_check\_function  
 d\_length = max(train\_point\_X[:, 0]) - min(train\_point\_X[:, 0]) # 找到横坐标x的极差  
 real\_x = np.linspace(min(train\_point\_X[:, 0]), min(train\_point\_X[:, 0]) + d\_length, 50) # 在x的范围内均匀产生50个点  
 real\_y = boundary\_check\_function(real\_x) # 对real\_x每个点调用boundary\_check\_function求解对应的real\_y  
 plt.plot(real\_x, real\_y, 'r', label='boundary\_check\_function') # 绘制图像  
 plot = plt.scatter(train\_point\_X[:, 0], train\_point\_X[:, 1], s=30, c=classification\_Y, marker='o',  
 cmap=plt.cm.Spectral) # 绘制两种点集  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend(loc=1)  
 plt.title('The regression curve')  
 plt.show()

def Skin\_NonSkin\_draw\_picture(train\_point\_X, classification\_Y, function\_coefficient):  
 fig = plt.figure()  
 ax = Axes3D(fig)  
 ax.set\_title('3D of the regression curve')  
 ax.scatter(train\_point\_X[:, 0], train\_point\_X[:, 1], train\_point\_X[:, 2], c=classification\_Y, cmap=plt.cm.Spectral)  
 real\_x = np.arange(np.min(train\_point\_X[:, 0]), np.max(train\_point\_X[:, 0]), 1)  
 real\_y = np.arange(np.min(train\_point\_X[:, 1]), np.max(train\_point\_X[:, 1]), 1)  
 real\_X, real\_Y = np.meshgrid(real\_x, real\_y)  
 real\_z = function\_coefficient[0] + function\_coefficient[1] \* real\_X + function\_coefficient[2] \* real\_Y  
 ax.plot\_surface(real\_x, real\_y, real\_z, rstride=1, cstride=1)  
 ax.set\_zlim(np.min(real\_z) - 10, np.max(real\_z) + 10)  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('y')  
 ax.set\_zlabel('z')  
 plt.show()

### 2.2.7 绘制迭代次数与损失函数值关系图像

作用：

'''  
根据迭代次数cycle\_times\_list和对应的误差loss\_list画出损失函数图像  
参数中，cycle\_times\_list为迭代次数表，loss\_list为对应的误差表  
'''

具体实现：以注释形式给出。

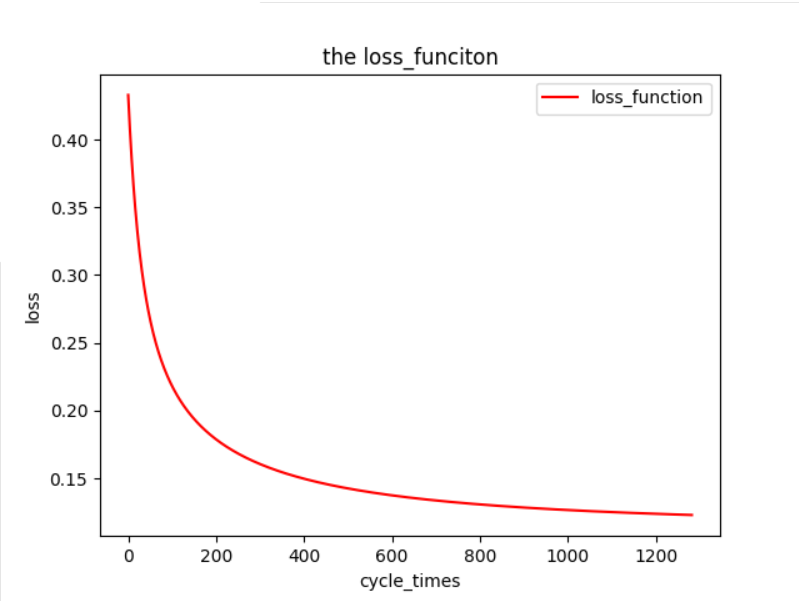
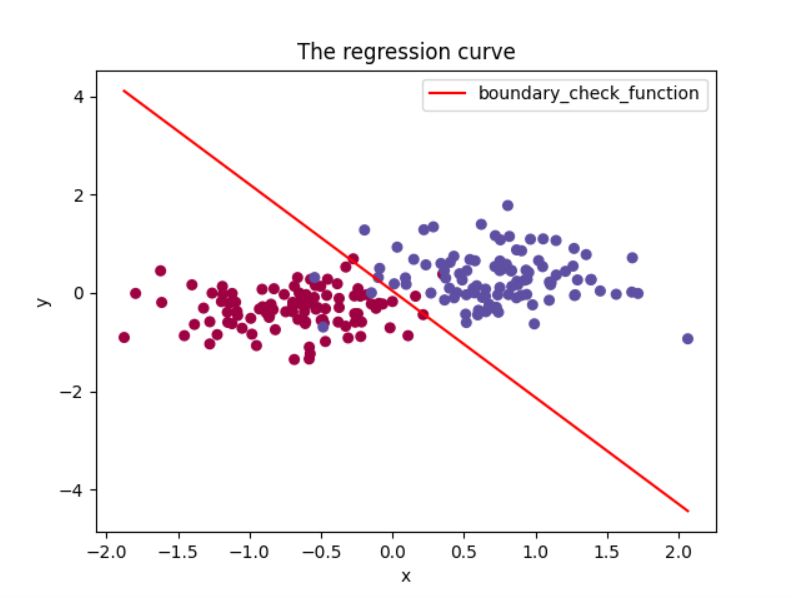
def draw\_picture\_loss(cycle\_times\_list, loss\_list):  
 plt.plot(cycle\_times\_list, loss\_list, 'r', label='loss\_function')  
 plt.xlabel('cycle\_times')  
 plt.ylabel('loss')  
 plt.legend(loc=1)  
 plt.title('the loss\_funciton')  
 plt.show()

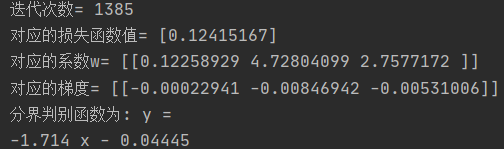
# 实验结果分析

## 生成满足朴素贝叶斯假设的数据

（1）不带惩罚项

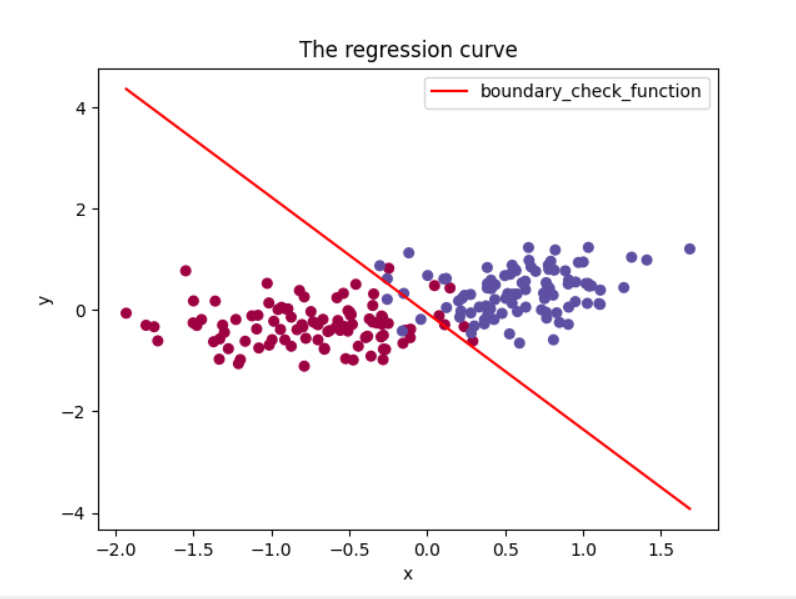
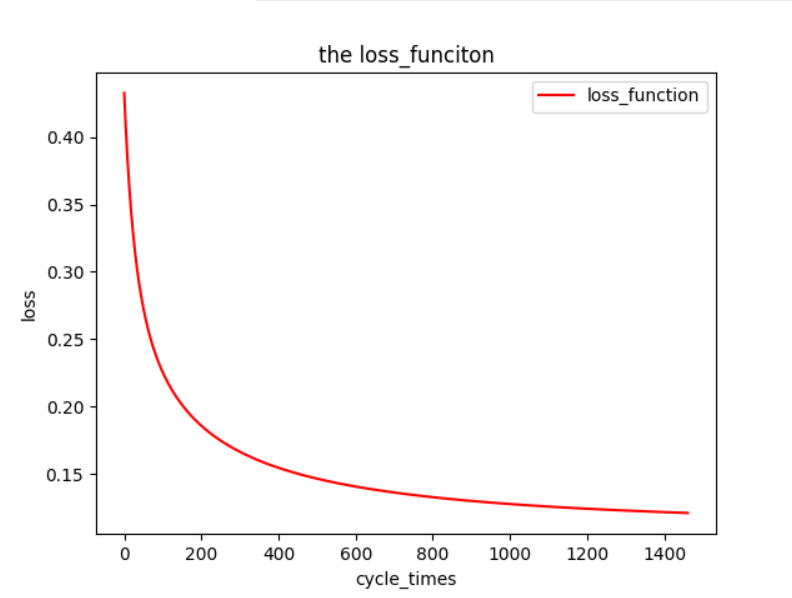
生成的0/1数据比例为1:1，生成200个测试用例，迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5

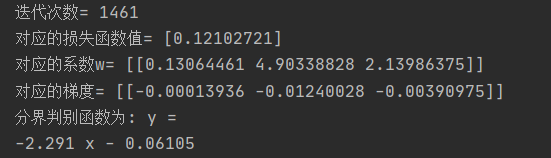




（2）带惩罚项

生成的0/1数据比例为1:1，生成200个测试用例，惩罚项系数为0.001迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5

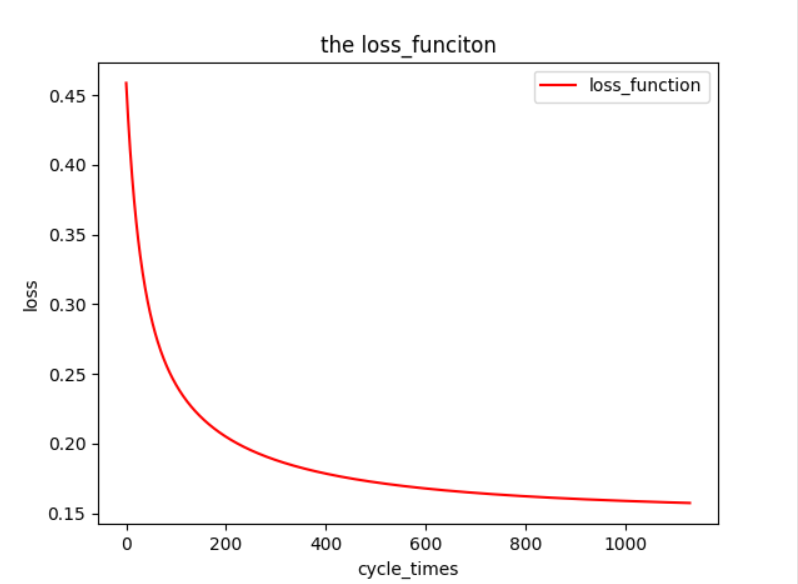
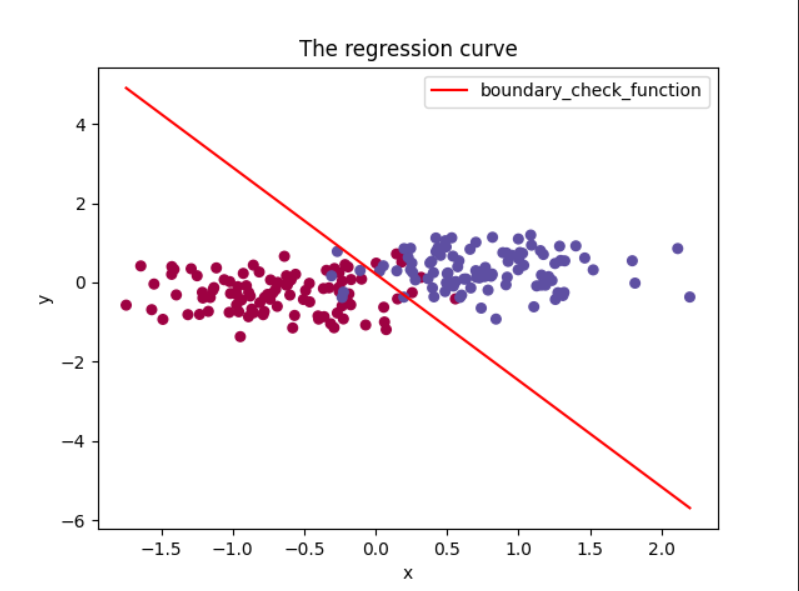


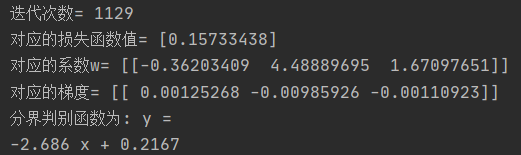
通过对图像进行分析可以得到：增加惩罚项几乎对准确率没有什么影响，但会增加一点准确率，更好的进行分类。

## 3.2 生成不满足朴素贝叶斯假设的数据

（1）不带惩罚项

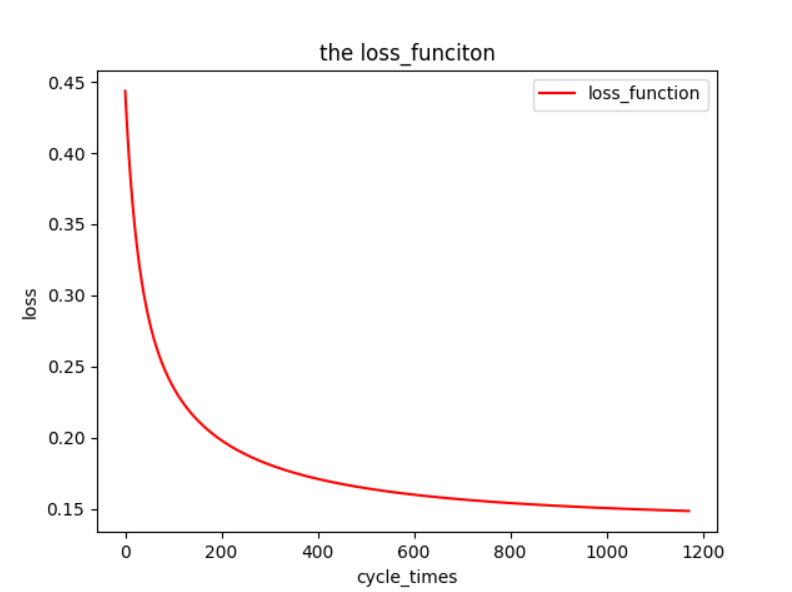
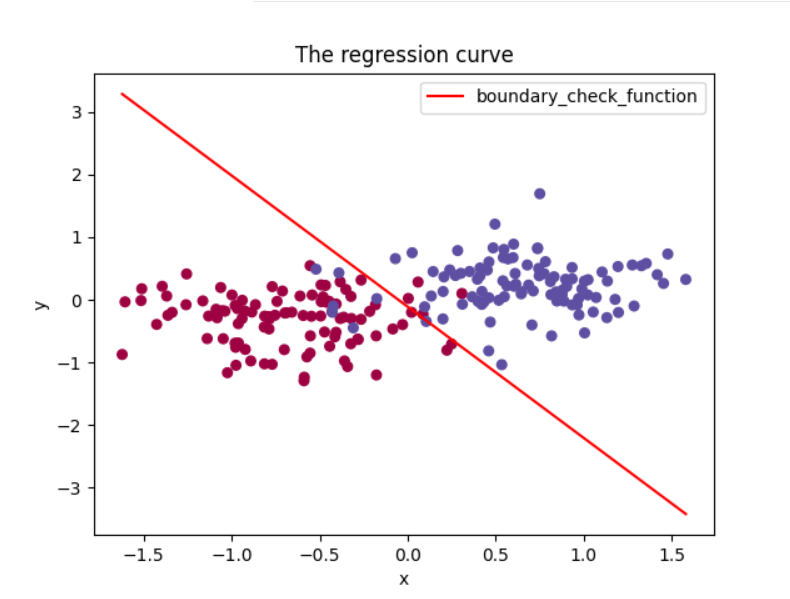
生成的0/1数据比例为1:1，生成200个测试用例，迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5

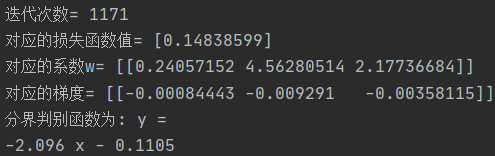




（2）带惩罚项

生成的0/1数据比例为1:1，生成200个测试用例，惩罚项系数为0.001迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5



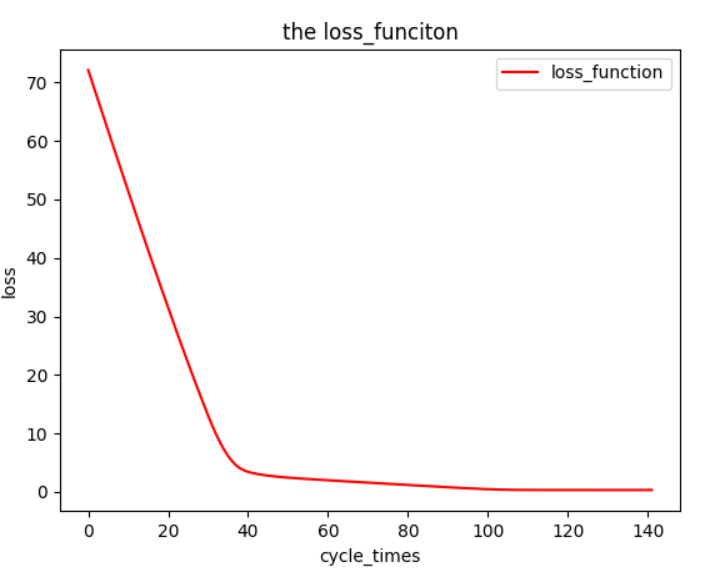
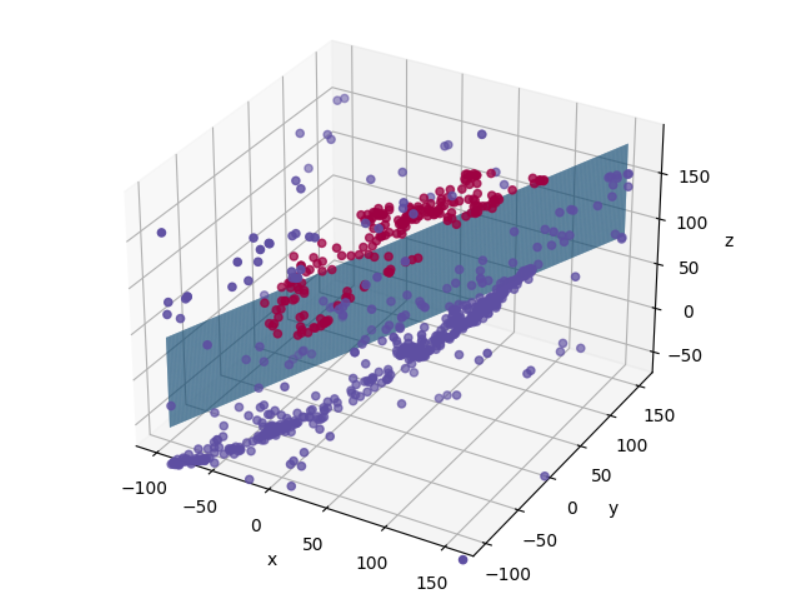


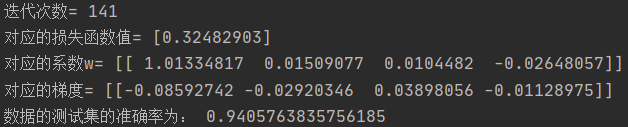
通过对图像进行分析可以得到：若数据不满足朴素贝叶斯假设，损失函数值会变大，即准确率会降低。

## 3.3 使用UCI上的Skin\_NonSkin.csv进行测试（三维）

（1）不带惩罚项

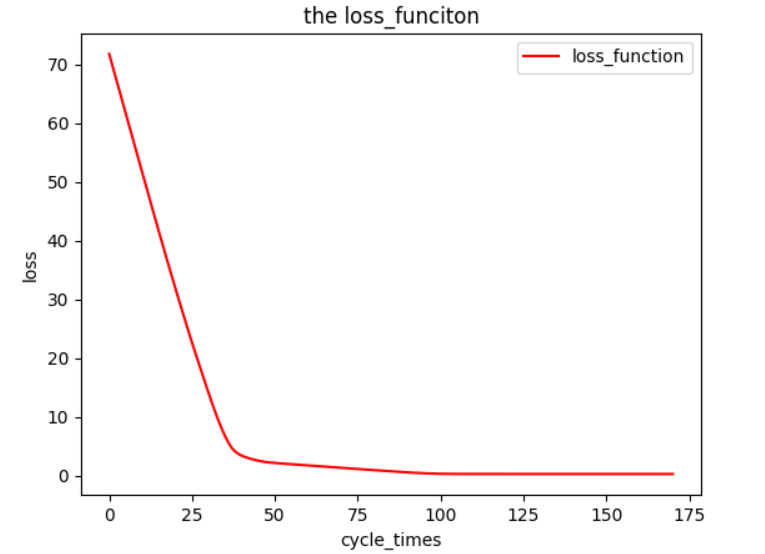
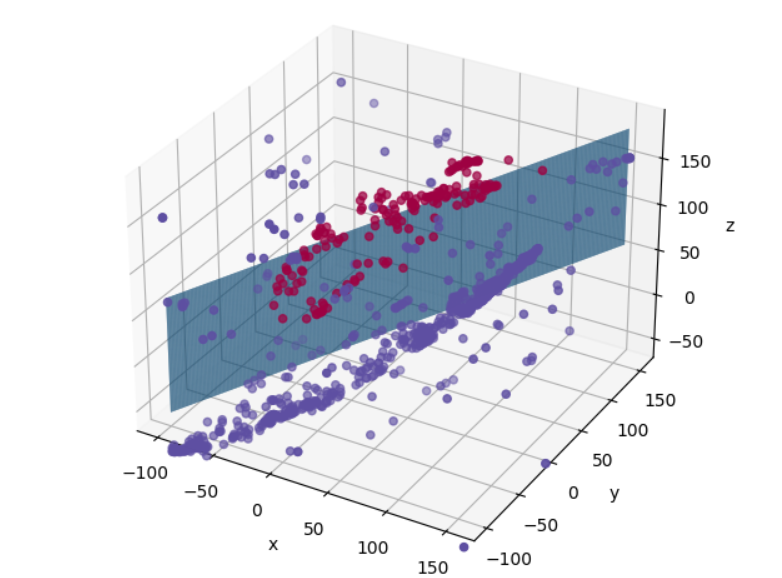
迭代最多次数为1000000，初始步长为0.001，迭代误差为1e-5

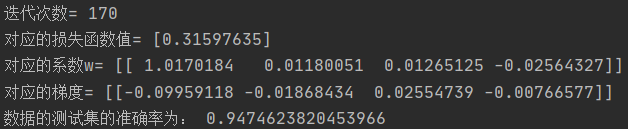




（2）带惩罚项

惩罚项系数为0.01，迭代最多次数为1000000，初始步长为0.001，迭代误差为1e-5

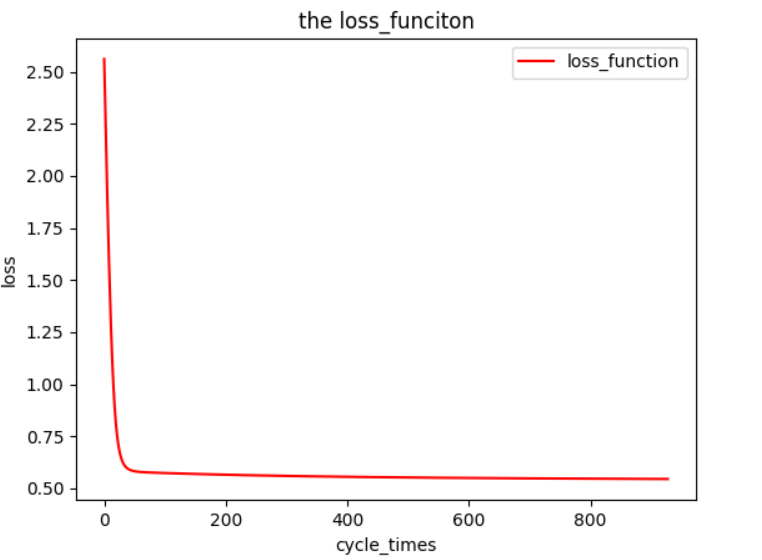


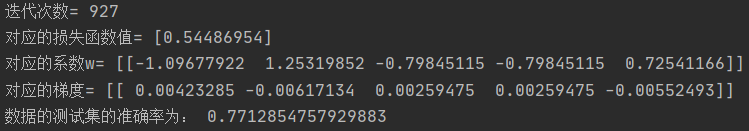


## 3.4 使用UCI上的blood.csv进行测试

（1）不带惩罚项

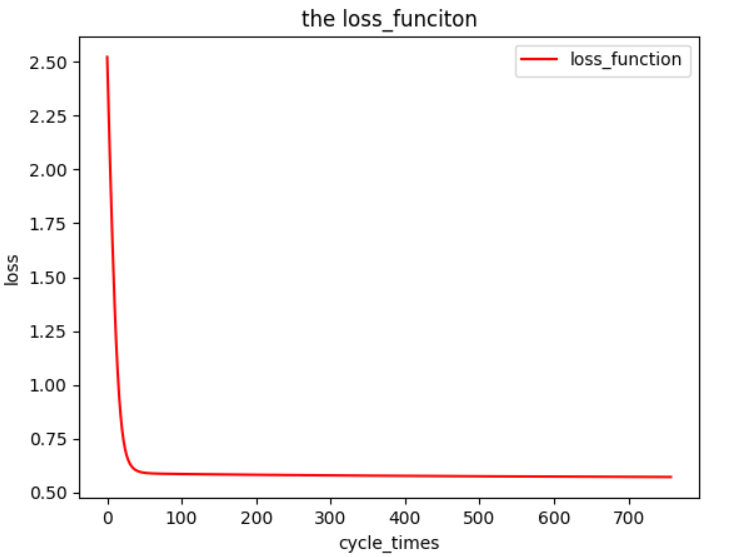
迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5

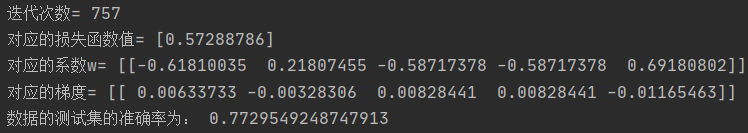




（2）带惩罚项

惩罚项系数为0.01，迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5

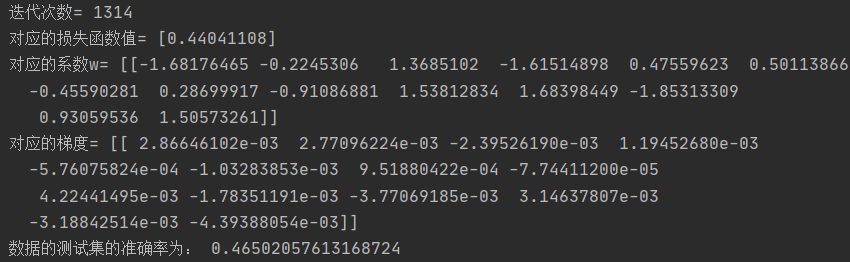
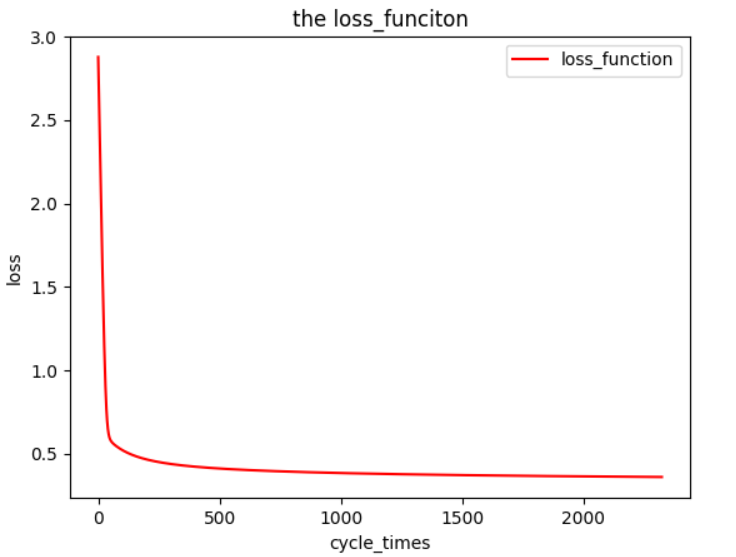




## 3.5 使用UCI上的heart.csv进行测试

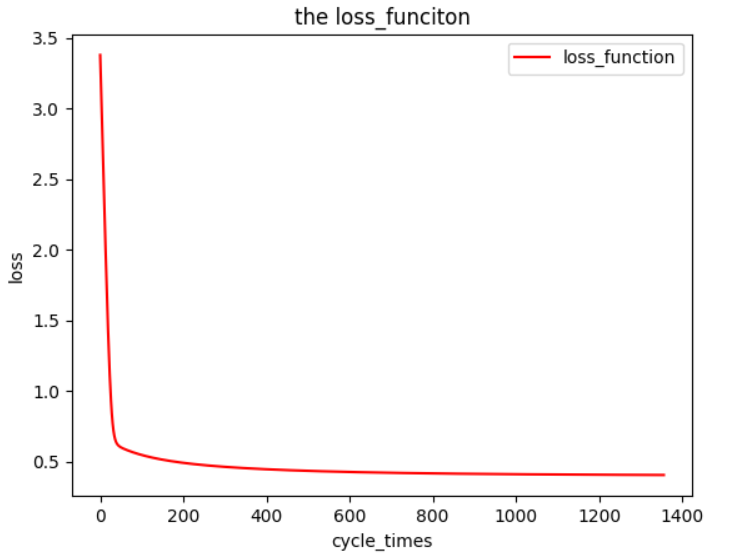
（1）不带惩罚项

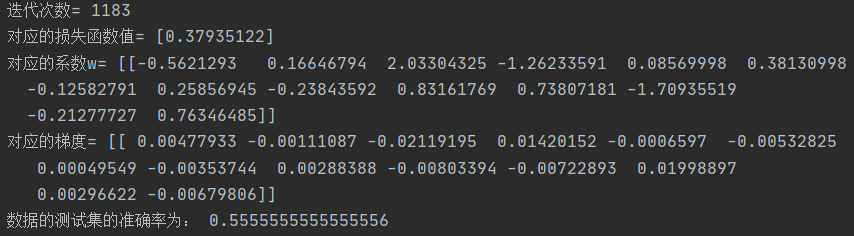
迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5



（2）带惩罚项

惩罚项系数为0.01，迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5

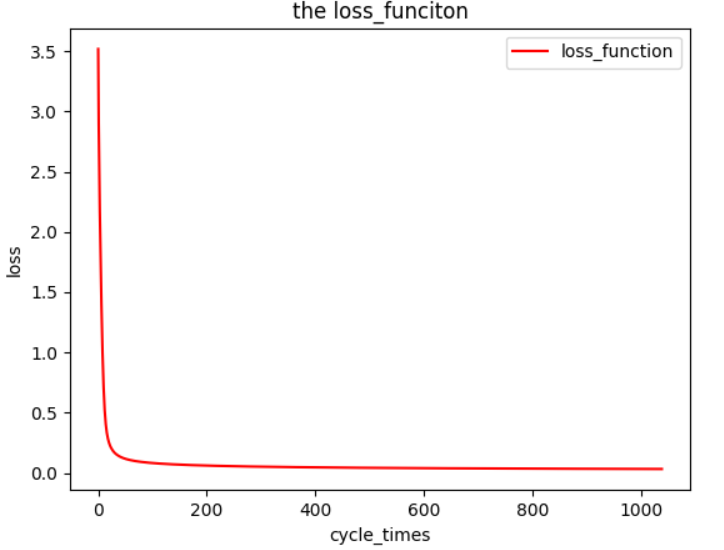


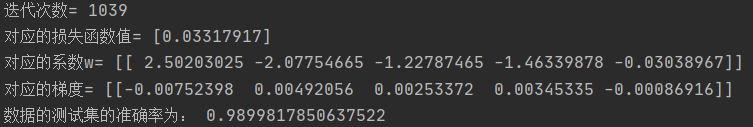


## 3.6 使用UCI上的data\_banknote\_authentication.csv进行测试

（1）不带惩罚项

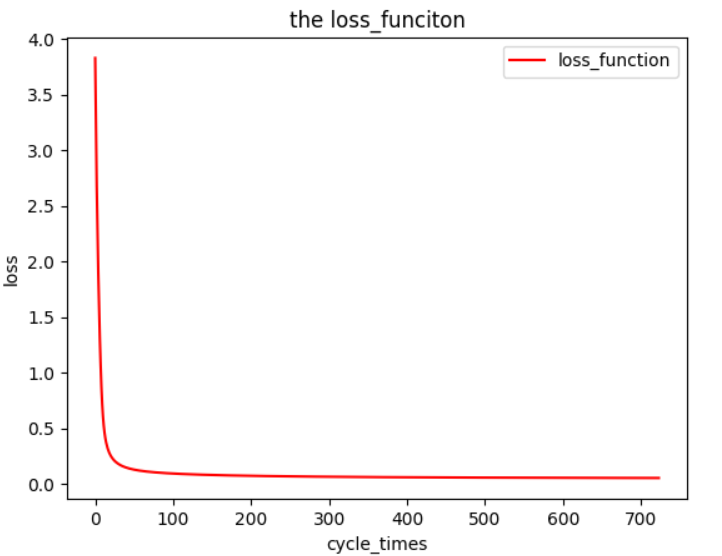
迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5

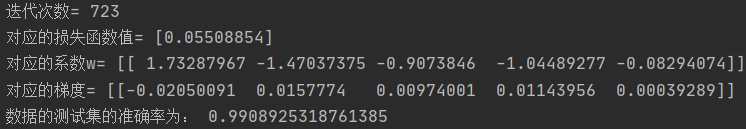




（2）带惩罚项

惩罚项系数为0.01，迭代最多次数为1000000，初始步长为0.1，迭代误差为1e-5





# 4 结论

1. 在数据量特别大时，使用梯度下降法，使用惩罚项对准确率提升影响很小，几乎可以忽略。可能原因是过拟合现象几乎可以忽略。

2. 数据量小时使用惩罚项可以提高准确率。

3. 类条件分布如果满足朴素贝叶斯假设时比不满足的情况准确率高。

4. 数据集的数据分布会影响数据测试集的准确率，blood.csv和heart.csv的准确率不是很高，而Skin\_NonSkin.csv和data\_banknote\_authentication.csv的准确率很高，一方面的原因可能来自数据集的分布规律，另外一个原因可能是因为blood.csv和heart.csv的数据太大，为了防止exp溢出进行了归一化。

5. 逻辑回归可以解决简单的数据集分类问题，梯度下降法的迭代次数也不是很大。

# 5 源代码（含详细注释）

Lab2\_1180300829.py

from Lab2.Skin\_NonSkin import Skin\_NonSkin\_experiment  
from Lab2.blood import blood\_exp  
from Lab2.data\_banknote\_authentication import data\_banknote\_authentication\_exp  
from Lab2.design\_experiment import design\_experiment  
from Lab2.heart import heart\_exp  
  
cycle\_times = 1000000  
descending\_step\_size = 0.1  
iteration\_error = 1e-5  
design\_experiment(200, 0, True, cycle\_times, descending\_step\_size, iteration\_error)  
design\_experiment(200, 0.001, True, cycle\_times, descending\_step\_size, iteration\_error)  
design\_experiment(200, 0, False, cycle\_times, descending\_step\_size, iteration\_error)  
design\_experiment(200, 0.001, False, cycle\_times, descending\_step\_size, iteration\_error)  
  
cycle\_times1 = 1000000  
descending\_step\_size1 = 0.001  
iteration\_error1 = 1e-5  
Skin\_NonSkin\_experiment(0, cycle\_times1, descending\_step\_size1, iteration\_error1)  
Skin\_NonSkin\_experiment(0.01, cycle\_times1, descending\_step\_size1, iteration\_error1)  
  
cycle\_times3 = 1000000  
descending\_step\_size3 = 0.1  
iteration\_error3 = 1e-5  
blood\_exp(0, cycle\_times3, descending\_step\_size3, iteration\_error3)  
blood\_exp(0.01, cycle\_times3, descending\_step\_size3, iteration\_error3)  
  
  
cycle\_times4 = 1000000  
descending\_step\_size4 = 0.1  
iteration\_error4 = 1e-5  
heart\_exp(0, cycle\_times4, descending\_step\_size4, iteration\_error4)  
heart\_exp(0.01, cycle\_times4, descending\_step\_size4, iteration\_error4)  
  
cycle\_times2 = 1000000  
descending\_step\_size2 = 0.1  
iteration\_error2 = 1e-5  
data\_banknote\_authentication\_exp(0, cycle\_times2, descending\_step\_size2, iteration\_error2)  
data\_banknote\_authentication\_exp(0.01, cycle\_times2, descending\_step\_size2, iteration\_error2)

blood.py

import numpy as np  
  
from Lab2.descent\_gradient\_add\_errorfunction import descent\_gradient\_add\_errorfunction  
from Lab2.draw import draw\_picture\_loss  
  
'''  
读入blood.csv文件，对数据进行拆分，拆分成训练集合测试集  
'''  
def blood\_getdata():  
 all\_data = np.loadtxt(open('./blood.csv'), delimiter=",", skiprows=0) # 读取文件中的所有数据  
 np.random.shuffle(all\_data) # 将原数据集打乱，分成训练集和测试集  
 test\_rate = 0.2 # 测试集所占比例  
 all\_data\_size = np.size(all\_data, axis=0) # 总数据集数据数量  
 train\_data\_X = all\_data[:int(test\_rate \* all\_data\_size), :] # 训练集数据  
 test\_data\_x = all\_data[int(test\_rate \* all\_data\_size):, :] # 测试集数据  
 dimension = np.size(all\_data, axis=1) - 1 # 数据集样本维度  
 # 消除exp溢出，防止数据太大导致exp溢出  
 for i in range(dimension): # 对于样本的所有维度  
 d\_length = max(train\_data\_X[:, i]) - min(train\_data\_X[:, i]) # 计算最大值和最小值之间的极差  
 for j in range(np.size(train\_data\_X, axis=0)): # 对于每一维度的所有数  
 train\_data\_X[j, i] = (max(train\_data\_X[:, i]) - train\_data\_X[j, i]) / d\_length # 将其化为[0,1]之间的值，防止exp溢出  
 train\_point\_X = train\_data\_X[:, 0:dimension] # 将所有数据集赋给train\_point\_X  
 train\_classification\_Y = train\_data\_X[:, dimension:dimension + 1] # 为train\_classification\_Y赋值为0/1  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集数据总数  
 train\_classification\_Y = train\_classification\_Y.reshape(train\_size) # 将矩阵转化为行向量  
 test\_point\_X = test\_data\_x[:, 0:dimension] # 将所有数据集赋给test\_point\_X  
 test\_classification\_Y = test\_data\_x[:, dimension:dimension + 1] # 为test\_classification\_Y赋值为0/1  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集数据总数  
 test\_classification\_Y = test\_classification\_Y.reshape(test\_size) # 将矩阵转化为行向量  
 return train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y  
  
  
'''  
使用blood.csv上的数据进行试验  
参数中lamda为惩罚项系数，cycle\_times为梯度下降迭代最大次数,descending\_step\_size为梯度下降下降步长,iteration\_error为梯度下降迭代误差  
'''  
def blood\_exp(lamda, cycle\_times, descending\_step\_size, iteration\_error):  
 train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y = blood\_getdata()  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集样本数量  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集样本数量  
 dimension = np.size(train\_point\_X, axis=1) # 样本维度  
 # 构造训练集样本矩阵  
 train\_all = np.ones((train\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵train\_all  
 for i in range(dimension): # 依次将训练集样本的每一个维度放入train\_all的下一个维度  
 train\_all[:, i + 1] = train\_point\_X[:, i]  
 w, cycle\_times\_list, loss\_list = descent\_gradient\_add\_errorfunction(train\_all, train\_classification\_Y, cycle\_times,  
 descending\_step\_size, iteration\_error,  
 dimension, lamda)  
 w = w.reshape(-1) # 得到的w是一个一行dimension + 1列的矩阵,需要先将w改成行向量  
 function\_coefficient = - (w / w[dimension])[0:dimension] # w整体除y的系数然后移项得到决策面系数  
 draw\_picture\_loss(cycle\_times\_list, loss\_list)  
 # 计算测试集准确率  
 label = np.ones(test\_size)  
 hit\_count = 0  
 test\_all = np.ones((test\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵test\_all  
 for i in range(dimension): # 依次将测试集样本的每一个维度放入train\_all的下一个维度  
 test\_all[:, i + 1] = test\_point\_X[:, i]  
 for i in range(test\_size): # 对每种预测给label进行赋值  
 if np.dot(w, test\_all[i].T) >= 0:  
 label[i] = 1  
 else:  
 label[i] = 0  
 for i in range(test\_size):  
 if label[i] == test\_classification\_Y[i]: # 如果预测的结果与真实结果相同计数加一  
 hit\_count += 1  
 hit\_rate = hit\_count / test\_size  
 print('数据的测试集的准确率为：', hit\_rate)

data\_banknote\_authentication.py

import numpy as np  
  
from Lab2.descent\_gradient\_add\_errorfunction import descent\_gradient\_add\_errorfunction  
from Lab2.draw import draw\_picture\_loss  
  
'''  
读入data\_banknote\_authentication.csv文件，对数据进行拆分，拆分成训练集合测试集  
'''  
def data\_banknote\_authentication\_getdata():  
 all\_data = np.loadtxt(open('./data\_banknote\_authentication.csv'), delimiter=",", skiprows=0) # 读取文件中的所有数据  
 np.random.shuffle(all\_data) # 将原数据集打乱，分成训练集和测试集  
 test\_rate = 0.2 # 测试集所占比例  
 all\_data\_size = np.size(all\_data, axis=0) # 总数据集数据数量  
 train\_data\_X = all\_data[:int(test\_rate \* all\_data\_size), :] # 训练集数据  
 test\_data\_x = all\_data[int(test\_rate \* all\_data\_size):, :] # 测试集数据  
 dimension = np.size(all\_data, axis=1) - 1 # 训练集样本维度  
 train\_point\_X = train\_data\_X[:, 0:dimension] # 将所有数据集赋给train\_point\_X  
 train\_classification\_Y = train\_data\_X[:, dimension:dimension + 1] # 为train\_classification\_Y赋值为0/1  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集数据总数  
 train\_classification\_Y = train\_classification\_Y.reshape(train\_size) # 将矩阵转化为行向量  
 test\_point\_X = test\_data\_x[:, 0:dimension] # 将所有数据集赋给test\_point\_X  
 test\_classification\_Y = test\_data\_x[:, dimension:dimension + 1] # 为test\_classification\_Y赋值为0/1  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集数据总数  
 test\_classification\_Y = test\_classification\_Y.reshape(test\_size) # 将矩阵转化为行向量  
 return train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y  
  
  
'''  
使用data\_banknote\_authentication.csv上的数据进行试验  
参数中lamda为惩罚项系数，cycle\_times为梯度下降迭代最大次数,descending\_step\_size为梯度下降下降步长,iteration\_error为梯度下降迭代误差  
'''  
def data\_banknote\_authentication\_exp(lamda, cycle\_times, descending\_step\_size, iteration\_error):  
 train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y = data\_banknote\_authentication\_getdata()  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集样本数量  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集样本数量  
 dimension = np.size(train\_point\_X, axis=1) # 样本维度  
 # 构造训练集样本矩阵  
 train\_all = np.ones((train\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵train\_all  
 for i in range(dimension): # 依次将训练集样本的每一个维度放入train\_all的下一个维度  
 train\_all[:, i + 1] = train\_point\_X[:, i]  
 w, cycle\_times\_list, loss\_list = descent\_gradient\_add\_errorfunction(train\_all, train\_classification\_Y, cycle\_times,  
 descending\_step\_size, iteration\_error,  
 dimension, lamda)  
 w = w.reshape(-1) # 得到的w是一个一行dimension + 1列的矩阵,需要先将w改成行向量  
 function\_coefficient = - (w / w[dimension])[0:dimension] # w整体除y的系数然后移项得到决策面系数  
 draw\_picture\_loss(cycle\_times\_list, loss\_list)  
 # 计算测试集准确率  
 label = np.ones(test\_size)  
 hit\_count = 0  
 test\_all = np.ones((test\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵test\_all  
 for i in range(dimension): # 依次将测试集样本的每一个维度放入train\_all的下一个维度  
 test\_all[:, i + 1] = test\_point\_X[:, i]  
 for i in range(test\_size): # 对每种预测给label进行赋值  
 if np.dot(w, test\_all[i].T) >= 0:  
 label[i] = 1  
 else:  
 label[i] = 0  
 for i in range(test\_size):  
 if label[i] == test\_classification\_Y[i]: # 如果预测的结果与真实结果相同计数加一  
 hit\_count += 1  
 hit\_rate = hit\_count / test\_size  
 print('数据的测试集的准确率为：', hit\_rate)

descent\_gradient\_add\_errorfunction.py

import numpy as np  
  
'''  
sigmoid函数a=1/(1+exp(-b)  
'''  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
  
'''  
自己生成数据  
根据是否满足贝叶斯假设，在范围[0,1]上生成sample\_number个二维点集，并将其均匀分为两类  
并添加一个N(0,1)的高斯噪声  
'''  
def create\_datas(sample\_number, satisfy\_naive):  
 half = np.ceil(sample\_number / 2).astype(np.int32) # sample\_number的一半  
 variance = 0.2 # 随机变量方差  
 covariance\_xy = 0.01 # 两个维度的协方差  
 point\_mean1 = [-0.7, -0.3] # 类别1的均值  
 point\_mean2 = [0.7, 0.3] # 类别2的均值  
 train\_point\_X = np.zeros((sample\_number, 2)) # 二维的点集  
 classification\_Y = np.zeros(sample\_number) # 点集的分类  
 if satisfy\_naive: # 满足朴素贝叶斯假设  
 train\_point\_X[:half, :] = np.random.multivariate\_normal(point\_mean1, [[variance, 0], [0, variance]],  
 size=half) # 生成half个类别1的1\*2数组，每个数组含有两个维度  
 train\_point\_X[half:, :] = np.random.multivariate\_normal(point\_mean2, [[variance, 0], [0, variance]],  
 size=sample\_number - half) # 生成half个类别2的1\*2数组，每个数组含有两个维度  
 classification\_Y[:half] = 0 # 将前half个数组标记为类别1  
 classification\_Y[half:] = 1 # 将后half个数组标记为类别2  
 else: # 不满足朴素贝叶斯假设  
 train\_point\_X[:half, :] = np.random.multivariate\_normal(point\_mean1,  
 [[variance, covariance\_xy], [covariance\_xy, variance]],  
 size=half) # 生成half个类别1的1\*2数组，每个数组含有两个维度  
 train\_point\_X[half:, :] = np.random.multivariate\_normal(point\_mean2,  
 [[variance, covariance\_xy], [covariance\_xy, variance]],  
 size=sample\_number - half) # 生成half个类别2的1\*2数组，每个数组含有两个维度  
 classification\_Y[:half] = 0 # 将前half个数组标记为类别1  
 classification\_Y[half:] = 1 # 将后half个数组标记为类别2  
 return train\_point\_X, classification\_Y # 返回生成的所有点及类别  
  
  
  
'''  
根据公式得到点集的极大条件似然得到极大条件似然l(W)  
'''  
def maximum\_conditional\_likelihood(train\_point\_X, classification\_Y, w):  
 total\_points = np.size(train\_point\_X, axis=0) # 得到train\_point\_X的行数，即点集个数  
 predict = np.zeros((total\_points, 1))  
 for i in range(total\_points):  
 predict[i] = w.dot(train\_point\_X[i].T) # 极大条件似然中的exp中的部分，即求和wi \* Xl  
 t = 0  
 for i in range(total\_points):  
 t += np.log(1 + np.exp(predict[i])) # 极大条件似然中ln的部分，即ln ( 1 + exp ( 求和wi \* Xi ) )  
 MCLE = classification\_Y.dot(predict) - t # 得到极大条件似然l(w)  
 return MCLE  
  
  
'''  
加惩罚项的梯度下降法  
对于train\_point\_X和classification\_Y对参数w做梯度下降，对损失函数使用梯度下降法，当误差函数收敛到期望的最小值时，得到此时的w并返回w  
参数中：  
迭代最大次数为cycle\_times  
下降步长descending\_step\_size  
迭代误差iteration\_error  
数据点集维度dimension  
惩罚项参数lamda  
'''  
def descent\_gradient\_add\_errorfunction(train\_point\_X, classification\_Y, cycle\_times, descending\_step\_size,  
 iteration\_error, dimension, lamda):  
 total\_points = np.size(train\_point\_X, axis=0) # 得到train\_point\_X的行数，即点集个数  
 w = np.ones((1, dimension + 1)) # 生成系数矩阵w，一个列数为dimension + 1，行数为1的矩阵，元素值全为1  
 cycle\_times\_list = np.zeros(cycle\_times) # 迭代次数统计  
 loss\_list = np.zeros(cycle\_times) # 迭代次数对应的损失函数值统计  
 for i in range(cycle\_times):  
 old\_loss = - 1 / total\_points \* maximum\_conditional\_likelihood(train\_point\_X, classification\_Y, w) # 原先损失函数的值  
 t = np.zeros((total\_points, 1))  
 for j in range(total\_points):  
 t[j] = w.dot(train\_point\_X[j].T) # 极大条件似然中的exp中的部分，即求和wi \* Xl  
 gradient = - 1 / total\_points \* (classification\_Y - sigmoid(t.T)).dot(train\_point\_X)  
 w = w - descending\_step\_size \* lamda \* w - descending\_step\_size \* gradient # 梯度下降加惩罚项的迭代公式  
 new\_loss = - 1 / total\_points \* maximum\_conditional\_likelihood(train\_point\_X, classification\_Y, w) # 新的损失函数的值  
 cycle\_times\_list[i] = i # 储存迭代次数  
 loss\_list[i] = new\_loss # 储存每次迭代对应的损失函数值  
 if abs(new\_loss - old\_loss) < iteration\_error: # 如果新的误差函数值与旧的误差函数值的差小于迭代误差则终止迭代  
 cycle\_times\_list = cycle\_times\_list[:i + 1]  
 loss\_list = loss\_list[:i + 1]  
 print('迭代次数=', i, '\n对应的损失函数值=', new\_loss, '\n对应的系数w=', w, '\n对应的梯度=', gradient)  
 break  
 if new\_loss > old\_loss: # 当新的误差函数值大于旧的误差函数值时，将步长变为原来的一半  
 descending\_step\_size \*= 0.5  
 return w, cycle\_times\_list, loss\_list

design\_experiment.py

import numpy as np  
  
from Lab2.descent\_gradient\_add\_errorfunction import create\_datas, descent\_gradient\_add\_errorfunction  
from Lab2.draw import draw\_picture, draw\_picture\_loss  
  
'''  
自定义二维点集进行试验并绘制图像  
参数中：sample\_number为点集中点的个数，lamda为惩罚项系数（可以为0，此时没有惩罚项），satisfy\_naive为是否满足朴素贝叶斯假设  
cycle\_times为梯度下降迭代最大次数,descending\_step\_size为梯度下降下降步长,iteration\_error为梯度下降迭代误差  
'''  
def design\_experiment(sample\_number, lamda, satisfy\_naive, cycle\_times, descending\_step\_size, iteration\_error):  
 train\_point\_X, classification\_Y = create\_datas(sample\_number, satisfy\_naive) # 生成sample\_number个二维点集数据  
 train\_all = np.ones((sample\_number, 3)) # 创建行为sample\_number，列为3的矩阵train\_all  
 train\_all[:, 1] = train\_point\_X[:, 0] # 将生成点集的第一个维度放入train\_all的第二列  
 train\_all[:, 2] = train\_point\_X[:, 1] # 将生成点集的第二个维度放入train\_all的第三列  
 dimension = np.size(train\_point\_X, axis=1)  
 w, cycle\_times\_list, loss\_list = descent\_gradient\_add\_errorfunction(train\_all, classification\_Y, cycle\_times,  
 descending\_step\_size, iteration\_error,  
 dimension, lamda)  
 w = w.reshape(-1) # 得到的w是一个一行三列的矩阵,需要先将w改成行向量  
 function\_coefficient = -(w / w[2])[0:2] # w整体除y的系数然后移项得到决策面系数  
 boundary\_check\_function = np.poly1d(function\_coefficient[::-1]) # 将function\_coefficient从后往前倒序然后调用poly1d得到多项式函数  
 print('分界判别函数为: y = ', boundary\_check\_function)  
 draw\_picture(train\_point\_X, classification\_Y, boundary\_check\_function)  
 draw\_picture\_loss(cycle\_times\_list, loss\_list)

draw.py

import numpy as np  
import matplotlib.pyplot as plt  
  
'''  
对于train\_point\_X中，含有二维数据，分别对应横坐标x和纵坐标y  
对于classification\_Y,为train\_point\_X中两种数据的点对应的类别  
根据train\_point\_X和classification\_Y画出二维坐标下的点图，然后画出分界判别函数boundary\_check\_function  
'''  
def draw\_picture(train\_point\_X, classification\_Y, boundary\_check\_function):  
 if boundary\_check\_function: # 绘制分界判定函数boundary\_check\_function  
 d\_length = max(train\_point\_X[:, 0]) - min(train\_point\_X[:, 0]) # 找到横坐标x的极差  
 real\_x = np.linspace(min(train\_point\_X[:, 0]), min(train\_point\_X[:, 0]) + d\_length, 50) # 在x的范围内均匀产生50个点  
 real\_y = boundary\_check\_function(real\_x) # 对real\_x每个点调用boundary\_check\_function求解对应的real\_y  
 plt.plot(real\_x, real\_y, 'r', label='boundary\_check\_function') # 绘制图像  
 plot = plt.scatter(train\_point\_X[:, 0], train\_point\_X[:, 1], s=30, c=classification\_Y, marker='o',  
 cmap=plt.cm.Spectral) # 绘制两种点集  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.legend(loc=1)  
 plt.title('The regression curve')  
 plt.show()  
  
  
'''  
根据迭代次数cycle\_times\_list和对应的误差loss\_list画出损失函数图像  
参数中，cycle\_times\_list为迭代次数表，loss\_list为对应的误差表  
'''  
def draw\_picture\_loss(cycle\_times\_list, loss\_list):  
 plt.plot(cycle\_times\_list, loss\_list, 'r', label='loss\_function')  
 plt.xlabel('cycle\_times')  
 plt.ylabel('loss')  
 plt.legend(loc=1)  
 plt.title('the loss\_funciton')  
 plt.show()

heart.py

import numpy as np  
  
from Lab2.descent\_gradient\_add\_errorfunction import descent\_gradient\_add\_errorfunction  
from Lab2.draw import draw\_picture\_loss  
  
'''  
读入heart.csv文件，对数据进行拆分，拆分成训练集合测试集  
'''  
def heart\_getdata():  
 all\_data = np.loadtxt(open('./heart.csv'), delimiter=",", skiprows=0) # 读取文件中的所有数据  
 np.random.shuffle(all\_data) # 将原数据集打乱，分成训练集和测试集  
 test\_rate = 0.2 # 测试集所占比例  
 all\_data\_size = np.size(all\_data, axis=0) # 总数据集数据数量  
 train\_data\_X = all\_data[:int(test\_rate \* all\_data\_size), :] # 训练集数据  
 test\_data\_x = all\_data[int(test\_rate \* all\_data\_size):, :] # 测试集数据  
 dimension = np.size(all\_data, axis=1) - 1 # 数据集样本维度  
 # 消除exp溢出，防止数据太大导致exp溢出  
 for i in range(dimension): # 对于样本的所有维度  
 d\_length = max(train\_data\_X[:, i]) - min(train\_data\_X[:, i]) # 计算最大值和最小值之间的极差  
 for j in range(np.size(train\_data\_X, axis=0)): # 对于每一维度的所有数  
 train\_data\_X[j, i] = (max(train\_data\_X[:, i]) - train\_data\_X[j, i]) / d\_length # 将其化为[0,1]之间的值，防止exp溢出  
 train\_point\_X = train\_data\_X[:, 0:dimension] # 将所有数据集赋给train\_point\_X  
 train\_classification\_Y = train\_data\_X[:, dimension:dimension + 1] # 为train\_classification\_Y赋值为0/1  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集数据总数  
 train\_classification\_Y = train\_classification\_Y.reshape(train\_size) # 将矩阵转化为行向量  
 test\_point\_X = test\_data\_x[:, 0:dimension] # 将所有数据集赋给test\_point\_X  
 test\_classification\_Y = test\_data\_x[:, dimension:dimension + 1] # 为test\_classification\_Y赋值为0/1  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集数据总数  
 test\_classification\_Y = test\_classification\_Y.reshape(test\_size) # 将矩阵转化为行向量  
 return train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y  
  
  
'''  
使用heart.csv上的数据进行试验  
参数中lamda为惩罚项系数，cycle\_times为梯度下降迭代最大次数,descending\_step\_size为梯度下降下降步长,iteration\_error为梯度下降迭代误差  
'''  
def heart\_exp(lamda, cycle\_times, descending\_step\_size, iteration\_error):  
 train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y = heart\_getdata()  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集样本数量  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集样本数量  
 dimension = np.size(train\_point\_X, axis=1) # 样本维度  
 # 构造训练集样本矩阵  
 train\_all = np.ones((train\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵train\_all  
 for i in range(dimension): # 依次将训练集样本的每一个维度放入train\_all的下一个维度  
 train\_all[:, i + 1] = train\_point\_X[:, i]  
 w, cycle\_times\_list, loss\_list = descent\_gradient\_add\_errorfunction(train\_all, train\_classification\_Y, cycle\_times,  
 descending\_step\_size, iteration\_error,  
 dimension, lamda)  
 w = w.reshape(-1) # 得到的w是一个一行dimension + 1列的矩阵,需要先将w改成行向量  
 function\_coefficient = - (w / w[dimension])[0:dimension] # w整体除y的系数然后移项得到决策面系数  
 draw\_picture\_loss(cycle\_times\_list, loss\_list)  
 # 计算测试集准确率  
 label = np.ones(test\_size)  
 hit\_count = 0  
 test\_all = np.ones((test\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵test\_all  
 for i in range(dimension): # 依次将测试集样本的每一个维度放入train\_all的下一个维度  
 test\_all[:, i + 1] = test\_point\_X[:, i]  
 for i in range(test\_size): # 对每种预测给label进行赋值  
 if np.dot(w, test\_all[i].T) >= 0:  
 label[i] = 1  
 else:  
 label[i] = 0  
 for i in range(test\_size):  
 if label[i] == test\_classification\_Y[i]: # 如果预测的结果与真实结果相同计数加一  
 hit\_count += 1  
 hit\_rate = hit\_count / test\_size  
 print('数据的测试集的准确率为：', hit\_rate)

Skin\_NonSkin.py

import numpy as np  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
  
from Lab2.descent\_gradient\_add\_errorfunction import descent\_gradient\_add\_errorfunction  
from Lab2.draw import draw\_picture\_loss  
  
'''  
Skin\_NonSkin数据集是三个维度的，所以可以画出三维的图像。  
'''  
def Skin\_NonSkin\_draw\_picture(train\_point\_X, classification\_Y, function\_coefficient):  
 fig = plt.figure()  
 ax = Axes3D(fig)  
 ax.set\_title('3D of the regression curve')  
 ax.scatter(train\_point\_X[:, 0], train\_point\_X[:, 1], train\_point\_X[:, 2], c=classification\_Y, cmap=plt.cm.Spectral)  
 real\_x = np.arange(np.min(train\_point\_X[:, 0]), np.max(train\_point\_X[:, 0]), 1)  
 real\_y = np.arange(np.min(train\_point\_X[:, 1]), np.max(train\_point\_X[:, 1]), 1)  
 real\_X, real\_Y = np.meshgrid(real\_x, real\_y)  
 real\_z = function\_coefficient[0] + function\_coefficient[1] \* real\_X + function\_coefficient[2] \* real\_Y  
 ax.plot\_surface(real\_x, real\_y, real\_z, rstride=1, cstride=1)  
 ax.set\_zlim(np.min(real\_z) - 10, np.max(real\_z) + 10)  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('y')  
 ax.set\_zlabel('z')  
 plt.show()  
  
  
'''  
读入Skin\_NonSkin.csv文件，对数据进行拆分，拆分成训练集合测试集  
由于原文件中数据量巨大，所以对数据集以50步长取部分数据作为数据集  
'''  
def Skin\_NonSkin\_getdata():  
 all\_data = np.loadtxt(open('./Skin\_NonSkin.csv'), delimiter=",", skiprows=0) # 读取文件中的所有数据  
 np.random.shuffle(all\_data) # 将原数据集打乱，分成训练集和测试集  
 test\_rate = 0.2 # 测试集所占比例  
 all\_data\_size = np.size(all\_data, axis=0) # 总数据集数据数量  
 train\_data\_X = all\_data[:int(test\_rate \* all\_data\_size), :] # 训练集数据  
 test\_data\_x = all\_data[int(test\_rate \* all\_data\_size):, :] # 测试集数据  
 dimension = np.size(all\_data, axis=1) - 1 # 训练集样本维度  
 step = 50 # 由于Skin\_NonSkin的数据集太大，所以采用步长为50的方式取数据  
 train\_point\_X = train\_data\_X[:, 0:dimension] # 将所有数据集赋给train\_point\_X  
 train\_point\_X = train\_point\_X[::step] # 以step为间隔取数据  
 train\_point\_X = train\_point\_X - 100 # 对样本点进行坐标平移，方便在3D图中显示  
 train\_classification\_Y = train\_data\_X[:, dimension:dimension + 1] - 1 # 因为数据集的分类是1/2,需要减1变成0/1  
 train\_classification\_Y = train\_classification\_Y[::step] # 以step为间隔取数据  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集数据总数  
 train\_classification\_Y = train\_classification\_Y.reshape(train\_size) # 将矩阵转化为行向量  
 test\_point\_X = test\_data\_x[:, 0:dimension] # 将所有数据集赋给test\_point\_X  
 test\_point\_X = test\_point\_X[::step] - 100 # 对样本点进行坐标平移，方便在3D图中显示  
 test\_classification\_Y = test\_data\_x[:, dimension:dimension + 1] - 1 # 因为数据集的分类是1/2,需要减1变成0/1  
 test\_classification\_Y = test\_classification\_Y[::step] # 以step为间隔取数据  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集数据总数  
 test\_classification\_Y = test\_classification\_Y.reshape(test\_size) # 将矩阵转化为行向量  
 return train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y  
  
  
'''  
使用Skin\_NonSkin.csv上的数据进行试验  
参数中lamda为惩罚项系数，cycle\_times为梯度下降迭代最大次数,descending\_step\_size为梯度下降下降步长,iteration\_error为梯度下降迭代误差  
'''  
def Skin\_NonSkin\_experiment(lamda, cycle\_times, descending\_step\_size, iteration\_error):  
 train\_point\_X, train\_classification\_Y, test\_point\_X, test\_classification\_Y = Skin\_NonSkin\_getdata() # 得到Skin\_NonSkin.csv上的训练集样本和测试集样本  
 train\_size = np.size(train\_point\_X, axis=0) # 训练集样本数量  
 test\_size = np.size(test\_point\_X, axis=0) # 测试集样本数量  
 dimension = np.size(train\_point\_X, axis=1) # 样本维度  
 train\_all = np.ones((train\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵train\_all  
 for i in range(dimension): # 依次将训练集样本的每一个维度放入train\_all的下一个维度  
 train\_all[:, i + 1] = train\_point\_X[:, i]  
 w, cycle\_times\_list, loss\_list = descent\_gradient\_add\_errorfunction(train\_all, train\_classification\_Y, cycle\_times,  
 descending\_step\_size, iteration\_error,  
 dimension, lamda)  
 w = w.reshape(-1) # 得到的w是一个一行dimension + 1列的矩阵,需要先将w改成行向量  
 function\_coefficient = - (w / w[dimension])[0:dimension] # w整体除y的系数然后移项得到决策面系数  
 Skin\_NonSkin\_draw\_picture(train\_point\_X, train\_classification\_Y, function\_coefficient)  
 draw\_picture\_loss(cycle\_times\_list, loss\_list)  
 # 计算测试集的准确率  
 label = np.ones(test\_size)  
 hit\_count = 0  
 test\_all = np.ones((test\_size, dimension + 1)) # 创建行为train\_size，列为样本维度+1的矩阵test\_all  
 for i in range(dimension): # 依次将测试集样本的每一个维度放入train\_all的下一个维度  
 test\_all[:, i + 1] = test\_point\_X[:, i]  
 for i in range(test\_size): # 对每种预测给label进行赋值  
 if w.dot(test\_all[i].T) >= 0:  
 label[i] = 1  
 else:  
 label[i] = 0  
 for i in range(test\_size):  
 if label[i] == test\_classification\_Y[i]: # 如果预测的结果与真实结果相同计数加一  
 hit\_count += 1  
 hit\_rate = hit\_count / test\_size  
 print('数据的测试集的准确率为：', hit\_rate)