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：

姓名：

目录

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

2.1 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。

2.2 实验环境

Windows10+python3.7+PyCharm

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 概率公式推导

简单来说，逻辑回归就是使用线性分类器进行的一个分类任务，其与线性回归任务十分相似。分类器的分类任务事实上就是根据一个样本的已知特征预测样本的标签。也就是得到这样预测结果：$P(Y=y_k|X=<x_1\ldots\ldots,x_n>)$。而在逻辑回归中假设满足朴素贝叶斯的条件，将其转换为类条件概率和类概率的乘积。

假设进行的是 0/1 二分类任务，接下来进行任务的简单假设：每一个样本有 n 个特征表示，也就是每一个样本可以表示为：$X=<x_1,\ldots\ldots,x_n>$，假设在训练集中一共有 L 个样本，也就是说训练集可以表示为$\{(X^1,Y^1),\ldots,(X^L,Y^L)\}$，其中 Y 的取值为 0 或 1，在下述推导中使用y_k表示 Y 的取值，则有推导如下：

$$\begin{aligned}P(Y=1|X) &= \frac{P(Y=1)P(X|Y=1)}{P(X)}\\ &= \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1)+P(Y=0)P(X|Y=0)}\\ &= \frac{P(Y=1)P(X|Y=1)}{1+exp(ln\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})}\end{aligned}$$

令$\pi=P(Y=1)$,$1-\pi=P(Y=0)$

则有：

$$P(Y=1|X)=\frac{1}{1+exp(ln\frac{1-\pi}{\pi})+\sum_i ln(\frac{P(X_i|Y=0)}{P(X_i|Y=1)})}$$

假设满足朴素贝叶斯条件，也就是类分布独立，且假设类条件分布满足正态分布，也就是：$P(x|y_k)=\frac{1}{\sigma_i\sqrt{2\pi}}e^{\frac{-(x-\mu_k)^2}{2\sigma_i^2}}$。那么就有：

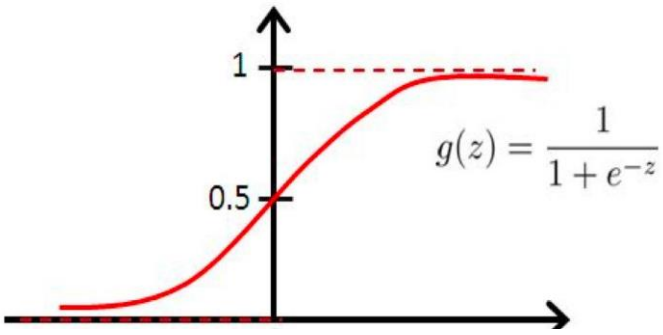
$$P(Y=1|X) = \frac{1}{1+exp(ln\frac{1-\pi}{\pi}+\sum_i ln\frac{\frac{1}{\sigma_i\sqrt{2\pi}}e^{\frac{-(x-\mu_{i_1})^2}{2\sigma_i^2}}}{\frac{1}{\sigma_i\sqrt{2\pi}}e^{\frac{-(x-\mu_{i0})^2}{2\sigma_i^2}}})}$$

$$= \frac{1}{1+exp(ln\frac{1-\pi}{\pi}+\sum_i \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}x_i+\frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2})}$$

令$w_0 = ln\frac{1-\pi}{\pi}+\sum_i \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}, w_i = \frac{\mu_{i0}-\mu i1}{\sigma_i^2}$

则原式可以化为：$P(Y=1|X) = \frac{1}{1+exp(w_0+\sum_i w_i x_i)}$

如果使用向量形式表示，那么可以将原式表示为：$P(Y=1|X) = \frac{1}{1+exp(-W^T X)}$，这也就是 sigmoid 函数的基本形式，也就是说在我们的假设条件下逻辑回归任务可以使用 sigmoid 函数实现。sigmoid 函数的图像如下：



3.2 损失函数

上述已经推出逻辑回归事实上可以使用 sigmoid 函数表示。在推导损失函数的过程中我们发现可以使用最大似然函数的方式来对参数进行优化从而进行学习，因此我们的优化目标是：

$$W_{MLE} = max_w l(W)$$
$$= max_w \prod_l [P(Y^l=1|X^l,W)]^{Y^l} + [1-P(Y^l=1|X^l,W)]^{1-Y^l}$$

而根据我们上述推出的结果，有：

$$P(Y=1|X,W) = \frac{exp(-(w_0+\sum_i w_i X_i))}{1+exp(-(w_0+\sum_i w_i X_i))}$$

$$P(Y=0|X,W) = \frac{1}{1+exp(-(w_0+\sum_i w_i X_i))}$$

为了下述推导方便，不失一般性的，我们将式子转化为：

$$P(Y=1|X,W) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

因此进一步推出待优化式子：

$$\begin{aligned} max_w l(W) &= \sum_l Y^l \, lnP(Y^l = 1|X^l, W) + (1 - Y^l) lnP(Y^l = 0|X^l, W) \\ &= \sum_l Y^l \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} + lnP(Y^l = 0|X^l, W) \\ &= \sum_l Y^l \, (w_0 + \sum_i w_i \, X_i) - ln(1 + exp(w_0 + \sum_i w_i \, X_i)) \end{aligned}$$

由于我们使用的主要方式是梯度下降法，因此我们希望优化的式子还是一个求最小值的形式，尽量保持和之前的梯度下降算法的一致性，因此我们定义损失函数：

$$cost(W) = -l(W)$$

从而将最大化似然函数的优化问题转化为最小化损失函数的问题，从而有：

$$\begin{aligned} cost(W) = -l(W) &= -[\sum_l Y^l \, lnP(Y^l = 1|X^l, W) + (1 - Y^l) lnP(Y^l = 0|X^l, W)] \\ &= -[\sum_l Y^l \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} + lnP(Y^l = 0|X^l, W)] \\ &= -[\sum_l Y^l \, (w_0 + \sum_i w_i \, X_i) - ln(1 + exp(w_0 + \sum_i w_i \, X_i))] \end{aligned}$$

$$\frac{\partial cost(W)}{\partial w_i} = -\sum_l X_i^l \, (Y^l - P(Y = 1|X, W))$$

$$w_i = w_i + \alpha \sum_l X_i^l \, (Y^l - P(Y = 1|X, W))$$

上述式子是不带正则项的更新方式，如果在损失函数之后加上正则项的话则有：

$$\begin{aligned} cost(W) = -l(W) &= -[\sum_l Y^l \, lnP(Y^l = 1|X^l, W) + (1 - Y^l) lnP(Y^l = 0|X^l, W)] + \frac{\lambda}{2} ||W||^2 \\ &= -[\sum_l Y^l \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} + lnP(Y^l = 0|X^l, W)] + \frac{\lambda}{2} ||W||^2 \\ &= -[\sum_l Y^l \, (w_0 + \sum_i w_i \, X_i) - ln(1 + exp(w_0 + \sum_i w_i \, X_i))] + \frac{\lambda}{2} ||W||^2 \end{aligned}$$

则梯度为：

$$\frac{\partial cost(W)}{\partial w_i} = -\sum_l X_i^l \, (Y^l - P(Y = 1|X, W)) + \lambda w_i$$

W 的更新方式为：

$$w_i = w_i + \alpha \sum_l X_i^l \, (Y^l - P(Y = 1|X, W)) - \alpha \lambda w_i$$

3.3 牛顿迭代法

牛顿法的思路就是在现有的极小值点的附近对需要优化的函数做二阶泰勒展开，进而找到极小值点的下一个估计值，由上述推导得出，需要优化的函数是$cost(W)$，假设第 k 步迭代的时候最小值的估计值为w_k，则在这一点展开，有：

$$f(w) = cost(w_k) + cost'(w_k)(w - w_k) + \frac{1}{2}cost''(w_k)(w - w_k)^2$$

令$f'(w) = 0$，则有：$w^{k+1} = w^k - \frac{cost'(w_k)}{cost''(w_k)}$，因此有迭代式：

$$w^{k+1} = w^k - H_k^{-1}cost'(w)$$

其中H_k为黑塞矩阵，其中第 m 行第 n 列元素可以表示为：

$$H_{mn} = \sum_i \frac{\partial cost(W)}{\partial w_m w_n} = \sum_i P\,(Y = 1|X, W)(1 - P(Y = 1|X, W))x_m^i x_n^i$$

四．实验结果与分析

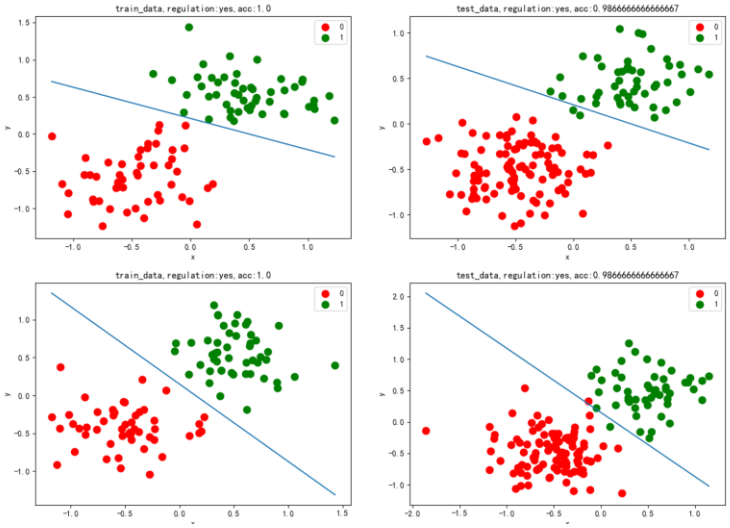
4.1 手工生成数据测试

以下测试过程中当数据符合朴素贝叶斯条件时，训练数据生成方式如下：数据均满足二元正态，分布标签为 0 的数据集在两个维度上均值为[0.5, 0.5]，方差为[0.1, 0.1], 数据集大小为 50，分布标签为 1 的数据集在两个维度上均值为[-0.5, -0.5]，方差为[0.1, 0.1], 数据集大小为 5。测试数据集只将两个数据集大小改为 100，其余参数一致。不讨论学习率影响时在训练过程中学习率=0.01，精度=10^{-6}，最大训练代数为 100000，如果取正则项则正则项为10^{-9}。

如果不满足朴素贝叶斯条件时，取两个维度数据的协方差为 0.5。

4.1.1 梯度下降和牛顿迭代对比

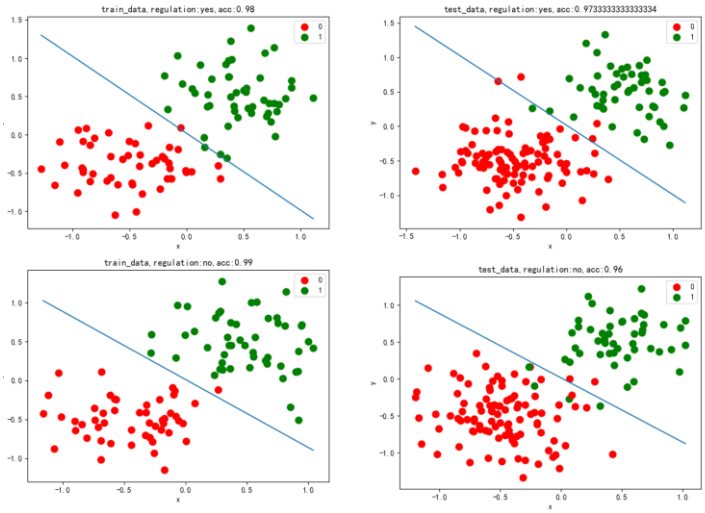
根据上述生成测试数据的方式，设置梯度下降的最大训练次数为 10000，牛顿迭代法迭代次数为 100，测试结果如下：



上图中上方两图是由牛顿迭代法生成的结果，下两图是由梯度下降法生成的结果，可以发现使用牛顿迭代法和梯度下降法最后生成的分类器的效果是基本一致的，但是牛顿迭代法的迭代法只使用了 100 次迭代，而梯度下降法的迭代次数超过 2000 次，因此牛顿迭代法的迭代次数相较于梯度下降法是更小的。但是需要注意的是牛顿迭代法在实际操作中需要多次计算二阶导数，且需要黑塞矩阵非奇异，因此单次迭代计算量可能相较于梯度下降法略大。

4.1.2 正则项影响

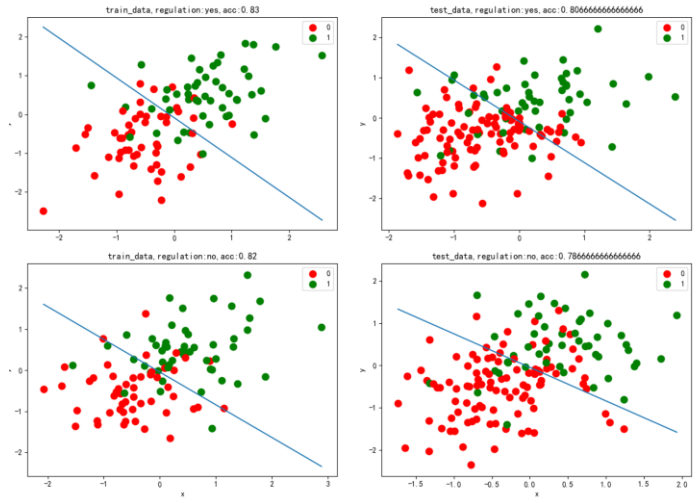
生成测试集的参数以及测试过程中的参数设置如上所述，使用梯度下降法，测试结果如下：



上图中上两张图是加上了正则项的结果，而下两张图是没有正则项的结果，可以发现当数据量比较大的时候由于分类平面只是直线，因此正则项并没有起较大作用，这一点也是比较好解释的，由于使用的是线性分类器，而且次数仅为 1，学习能力并不强，因此在数据集上表现出过拟合的可能性较小。而可以发现测试集上的结果都比训练集略差，这也是符合一般认识的。

4.1.3 是否满足贝叶斯条件影响

当数据不符合朴素贝叶斯条件时，为了显示朴素贝叶斯条件的作用，不妨假设两个维度数据的协方差为 0.5，其余参数与上述一致则结果如下：



与符合朴素贝叶斯的结果相比可以发现，当分类数据不符合朴素贝叶斯的条件，也就是类条件独立的时候分类的结果明显下降，但是同时可以发现的是虽然精度下降，但是仍然保持在一个较高的水平，也就是说如果类条件之间的协方差并不是很大的时候，将其近似为类条件独立进行分类仍然可以得到较好的结果。

4.2 UCI 数据测试

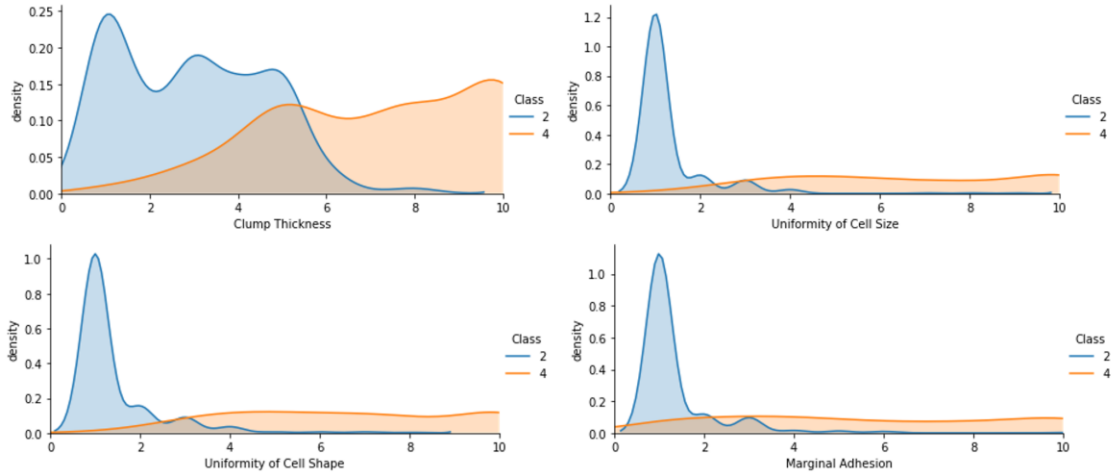
使用的数据集是 UCI 上的乳腺癌良恶性的数据集，由于这个数据集处理的也是二分类的问题，因此使用在逻辑回归任务中比较合适。接下来简单介绍一下这个数据集的数据组成。

该原始数据共有 699 个样本，每个样本有 11 列不同的数值，第一列是 ID 值（在分类问题中去除），中间 9 列是与肿瘤相关的医学特征，以及一列表征肿瘤类型的数值。所有 9 列用于表示肿瘤医学特质的数值均被量化为 1-10 之间的数字，而肿瘤的类型也用数字 2 和数字 4 分别指代良性与恶性。由于上述我们讨论的都是 0/1 分类问题，因此将肿瘤类型中的良性表示为 0, 恶性表示为 1。可以观察数据集前五行结果如下：

Out[2]:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

接着，不妨查看前四个特征在肿瘤的不同类型下的分布情况，下图中 class=2 表示良性肿瘤，class=4 表示恶性肿瘤，则有如下结果：



可以发现对于前四个特征来说在肿瘤的情况不同的情况下分布的差异是比较大的，因此猜测数据是能够较好分类的。因此接着可以对数据集进行预测，将数据集分为训练集和测试集，由于维数过高，无法画图展示，因此只展示最终的预测结果：

```
test_acc:0.9602510460251046
train_acc:0.9951219512195122
```

可以发现预测精度在训练集和测试集上都相当高，说明在乳腺癌分类问题上使用逻辑回归学习到的分类器性能和泛化能力都不错。

五．结论

1.逻辑回归的公式是基于数据的类条件分布独立，也就是满足朴素贝叶斯的条件推出的，因此针对不满足这一条件的数据的分类效果将会下降。

2.逻辑回归对于线性分类问题能够给出一个相对好的分类器，能较好解决问题
3.在解决二分类问题的时候，也就是使用直线进行分类的时候，正则项的效果并不是很大，这是因为在给出的数据集的数据量大小下不会出现明显的过拟合现象。
4.在使用 sigmoid 函数的时候如果不进行一定的处理可能出现数值的上溢或者下溢，因此需要适当的防止溢出操作。

六．参考文献

[1]周志华,《机器学习》,清华大学出版社,2016

七、附录：源代码（带注释）

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
##保证绘图中输出中文正常
plt.rcParams['font.sans-serif'] = [u'SimHei']
plt.rcParams['axes.unicode_minus'] = False
def generateData(mean_0,var_0,size_0,mean_1,var_1,size_1,cov=0):
    '''''
    使用多维正态分布生成数据
    :param mean_0: 第 0 类均值，形如 1*dimension
    :param var_0: 第 0 类方差，形如 1*dimension
    :param size_0: 第 0 类数据及大小，标量
    :param mean_1: 第 1 类均值，形如 1*dimension
    :param var_1:第 2 类方差，形如 1*dimension
    :param size_1:第 1 类数据及大小，标量
    :param cov: 协方差
    :return: 训练特征和标签，分别形如(size0+size1)*dimension,(size0+size1)*1
    '''
    dimension=mean_0.shape[0]
    train_x=np.zeros((size_0+size_1,dimension))
    train_label=np.zeros(size_0+size_1)
    cov_0=cov*np.ones((dimension,dimension))
    cov_1 = cov * np.ones((dimension, dimension))
    for i in range(dimension):
        cov_0[i][i]=var_0[i]
        cov_1[i][i]=var_1[i]
    train_x[:size_0,:]=np.random.multivariate_normal(mean_0,cov_0,size=size_0)
    train_x[size_0:,:]=np.random.multivariate_normal(mean_1,cov_1,size=size_1)
    train_label[size_0:]=1
    train_data=np.column_stack((train_x, train_label))
    np.random.shuffle(train_data)
    return train_data[:,:-1],train_data[:,-1:]

def sigmoid(x):
    size=x.shape[0]
    res=[]
    for i in range(size):
        if x[i]>=0:
            res.append(1/(1+np.exp(-x[i])))
        else:
            res.append(np.exp(x[i])/(1+np.exp(x[i])))
    return np.array(res).reshape([size,-1])

```

```
def loss(train_x,train_y,w,lamda):
    '''''
    计算需要优化的式子，归一化
    :param train_x: 训练集特征集合，形如(size0+size1)*(dimension+1)，第一列全 1
    :param train_y:训练集标签，形如(size0+size1)*1
    :param w:参数 形如(dimension+1)*1
    :param lamda:正则项大小
    :return:
    '''
    size=train_x.shape[0]
    w_x=np.zeros((size,1))
    ln_x=0.0
    for i in range(size):
        w_x[i]=np.matmul(train_x[i],w)
        ln_x+=np.log(1+np.exp(w_x[i]))
    loss=np.matmul(train_y.T,w_x)-ln_x+lamda*np.matmul(w.T,w)
    return -loss
def gradient_optim(lamda, train_x, train_y, alpha, epsilon, train_num):
    '''''
    :param lamda: 表示正则项系数，==0 的时候表示无正则项
    :param train_x: 训练数据集 size*dimension
    :param train_y: 训练数据集 size*1
    :param alpha: 学习率
    :param epsilon: 精度
    :param train_num:训练最大次数
    :return: 优化后的参数取值
    '''
    size=train_x.shape[0]
    dimension=train_x.shape[1]
    ##为了加入 w0，将 X 第一列全部置为 1
    X=np.ones((size,dimension+1))
    X[:,1:]=train_x
    w=np.zeros((dimension+1,1))
    new_loss=loss(X,train_y,w,lamda)
    temp=0
    for i in range(train_num):
        old_loss=new_loss
        part_gradient=np.zeros((size,1))
        for j in range(size):
            part_gradient[j]=np.matmul(X[j],w)
        gradient=np.matmul(X.T,train_y-sigmoid(part_gradient))#梯度
        temp_w=w
        w=np.add(w,alpha*gradient-alpha*lamda*w)#迭代式
        new_loss=loss(X,train_y,w,lamda)
        temp=i
        if old_loss<new_loss:#迭代过程中步幅太大
            w=temp_w
            alpha/=2
            continue
        if old_loss-new_loss<epsilon:
            break
    print('梯度下降迭代次数：'+str(temp))
    w=w.reshape(dimension+1)
    fit = -(w / w[dimension])[0:dimension]  # 归一化得到系数
    return np.poly1d(fit[::-1]), w
       def Hessian(X,w):
            '''''
            计算当下的黑塞矩阵
            :param X: 训练数据集 size*(dimension+1)
```

```
104.            :param w: (dimension+1)*1
105.            :return: 黑塞矩阵 (dimension+1)*(dimension+1)
106.            '''
107.            size=X.shape[0]
108.            dimenson_1=X.shape[1]
109.            hessian=np.zeros((dimenson_1,dimenson_1))
110.            for i in range(size):
111.                w_x=np.matmul(X[i],w)
112.                for j in range(dimenson_1):
113.                    for k in range(dimenson_1):
114.                        p_1=sigmoid(w_x)
115.                        hessian[j][k]+=X[i][j]*X[i][k]*p_1*(1-p_1)
116.            return hessian
117.        def newton_optim(train_x, train_y, train_num):
118.            '''''
119.            :param train_x: 训练数据集 size*dimension
120.            :param train_y: 训练数据集 size*1
121.            :param train_num:训练最大次数
122.            :return: 优化后的参数取值
123.            '''
124.            size = train_x.shape[0]
125.            dimension = train_x.shape[1]
126.            ##为了加入 w0，将 X 第一列全部置为 1
127.            X = np.ones((size, dimension + 1))
128.            X[:, 1:] = train_x
129.            w = np.ones((dimension + 1, 1))
130.            for i in range(train_num):
131.                part_gradient = np.zeros((size, 1))
132.                for j in range(size):
133.                    part_gradient[j] = np.matmul(X[j], w)
134.                gradient = -np.matmul(X.T, train_y - sigmoid(part_gradient))
135.                hessian=Hessian(X,w)#输出黑塞矩阵
136.                w-=np.matmul(np.linalg.pinv(hessian),gradient)
137.            w = w.reshape(dimension + 1)
138.            fit = -(w / w[dimension])[0:dimension]  # 归一化得到系数
139.            return np.poly1d(fit[::-1]), w
140.        def acc(train_x,train_y,w):
141.            '''''
142.            计算 w 预测正确率
143.            :param train_x:测试集 size*dimension
144.            :param train_y:标签 size*1
145.            :param w:w 1*(dimension+1)
146.            :return:正确率
147.            '''
148.            size=train_x.shape[0]
149.            dimension=train_x.shape[1]
150.            acc = 0
151.            X=np.ones((size,dimension+1))
152.            X[:,1:]=train_x
153.            for i in range(size):
154.                label=0
155.                if np.matmul(w,X[i].T)>=0:
156.                    label=1
157.                if label==train_y[i]:
158.                    acc +=1
159.            return acc/size
160.        def uci_read(path):
161.            '''''
162.            :param path:数据集
```

```
163.            :return: 训练集和训练集标签
164.            '''
165.            column_names = ['Sample code number', 'Clump Thickness',
166.                            'Uniformity of Cell Size', 'Uniformity of Cell Shape
    ',
167.                            'Marginal Adhesion', 'Single Epithelial Cell Size',

168.                            'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli',
     'Mitoses', 'Class']
169.            data = pd.read_csv(path,names=column_names)
170.            data = data.replace(to_replace='?', value=np.nan)  # 非法字符的替代
171.            data = data.dropna(how='any')  # 去掉空值，any：出现空值行则删除
172.            x=data[column_names[1:10]].values.copy() ## 转换为 array 格式
173.            ##将标签中的 2 和 4 改为 0 和 1
174.            y=data[column_names[10:]]
175.            y.loc[y['Class']==2]=0
176.            y.loc[y['Class'] == 4] = 1
177.            y=y.values.copy()
178.            #乱序输出
179.            train_data = np.column_stack((x, y))
180.            np.random.shuffle(train_data)
181.            x=train_data[:,:-1]
182.            y=train_data[:,-1:]
183.            x.astype(np.float64)
184.            y.astype(np.float64)
185.            ##生成训练集和测试集，比例为 7:3
186.            total_num=x.shape[0]
187.            train_num=int(0.7*total_num)
188.            train_x=x[:train_num,:]
189.            train_y=y[:train_num,:]
190.            test_x=x[train_num:,:]
191.            test_y=y[train_num:,:]
192.            return train_x,train_y,test_x,test_y
193.        def uci_run(path):
194.            '''''
195.            uci 数据集运行
196.            :param path: 数据集存储路径
197.            :return: 在命令行打印结果
198.            '''
199.            train_x,train_y,test_x,test_y=uci_read(path)
200.            lamda=0
201.            epsilon=1e-6
202.            train_num=10000
203.            alpha=0.01
204.            fit,w=gradient_optim(lamda,train_x,train_y,alpha,epsilon,train_num)

205.            uci_acc=acc(train_x,train_y,w)
206.            print('train_acc:'+str(uci_acc))
207.            print('test_acc:'+str(acc(test_x,test_y,w)))
208.
209.        def figure_2D(x,y,fit,title):
210.            '''''
211.            画图——二维
212.            :param x: 特征集
213.            :param y: 标签集
214.            :param fit: 分类曲线
215.            :param title: 图名
216.            :return:
217.            '''
```

```
            y=y.reshape(x.shape[0])
            for i in range(x.shape[0]):
                if y[i]==0:
                    s1=plt.scatter(x[i, 0], x[i, 1], s=50, lw=3, color='r')
                elif y[i]==1:
                    s2=plt.scatter(x[i, 0], x[i, 1], s=50, lw=3, color='g')
            plt.xlabel('x')
            plt.ylabel('y')
            plt.legend((s1,s2),('0','1'),loc='best')
            start=min(x[:, 0])
            end=max(x[:, 0])
            line_x=np.linspace(start,end,100)
            line_y=fit(line_x)
            plt.plot(line_x,line_y)
            plt.title(title)
            plt.show()
        def run(optim_method):
            '''''
            手工生成数据集运行
            :param optim_method: 运行的优化方法，gradient 时为梯度下降，newton 的时候为牛顿迭代法
            :return:
            '''
            mean0 = np.array([-0.5, -0.5])
            var0 = np.array([0.1, 0.1])
            size0 = 50
            mean1 = np.array([0.5, 0.5])
            var1 = np.array([0.1, 0.1])
            size1 = 50
            cov = 0
            lamda = 1e-9
            alpha = 0.1
            epsilon = 1e-6
            train_num = 10000
            train_x, train_y = generateData(mean0, var0, size0, mean1, var1, size1, cov)
            if optim_method=='gradient':
                fit, w = gradient_optim(lamda, train_x, train_y, alpha, epsilon, train_num)
            elif optim_method=='newton':
                fit, w = newton_optim(train_x, train_y, 200)
            train_acc = acc(train_x, train_y, w)
            figure_2D(train_x, train_y, fit, 'train_data,regulation:yes,acc:' + str(train_acc))
            ##测试集
            test_x, test_y = generateData(mean0, var0, 2 * size0, mean1, var1, size1, cov)
            test_acc = acc(test_x, test_y, w)
            figure_2D(test_x, test_y, fit, 'test_data,regulation:yes,acc:' + str(test_acc))
            print(fit)
        if __name__=="__main__":
            run('gradient')
            run('newton')
            # uci_run('./breast-cancer-wisconsin.data')
```