2020 年春季学期

计算学部《机器学习》课程

Lab 2 实验报告

姓名	周牧云
学号	1180300315
班号	1803501
电子邮件	zhou_mu_yun@163.com
手机号码	13912263240

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

实现两种损失函数的参数估计（1，无惩罚项；2.加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等。

验证：1.可以手工生成两个分别类别数据（可以用高斯分布），验证你的算法。考察类条件分布不满足朴素贝叶斯假设，会得到什么样的结果。
2. 逻辑回归有广泛的用处，例如广告预测。可以到 UCI 网站上，找一实际数据加以测试。

三、实验内容

1、算法原理
我们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 < $x_1, \dots, x_n$)。按照朴素贝叶斯的方法，可以用贝叶斯概率公式，将其转化为类条件概率（似然）和类概率的乘积。这次实验，是直接求该概率。
经过推导我们可以得到：

$$P(Y=1|X) = \frac{1}{1+exp(w_0+\sigma_{i=1}^{n} w_i X_i)}$$

定义 sigmoid 函数为：

$$a = \frac{1}{1+exp(-b)}$$

计算损失函数为：

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

用梯度下降法求得 $W = \text{argmax}_w l(w)$，注意要用梯度下降的话，一般要把这里的 l(w)转化为相反数，-l(w)作为损失函数，求其最小值。

$$\frac{\partial l(w)}{\partial w_i} = \sum_l X_i^l (Y^l - \frac{1}{1+exp(-(w_0+\sum_i^n w_i X_i^l))})$$

$$w_i \leftarrow w_i + \eta \sum_l X_i^l (Y^l - \frac{1}{1+exp(-(w_0+\sum_i^n w_i X_i^l))})$$

而我们加上正则项的梯度下降为

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_l X_i^l (Y^l - \frac{1}{1+exp(-(w_0+\sum_i^n w_i X_i^l))})$$

2. 算法的实现

首先是生成数据，如果要生成类条件分布满足朴素贝叶斯假设的数据，那么就对每一个类别的每一个维度都用一个独立的高斯分布生成。如果要生成类条件分布不满足朴素贝叶斯假设的数据，那么 就对每一个类别的两个维度用一个二维高斯分布生成。需要注意的是，由于高斯分布具有的特性， 多维高斯分布不相关可以推出独立性，因此，可以用二维高斯分布生成数据，如果是满足朴素贝叶斯假设的，那么协方差矩阵的非对角线元素均为 0，如果是不满足朴素贝叶斯假设的，那么协方差矩阵的非对角线元素不为 0（协方差矩阵应该是对称阵）。

计算极大似然估计:

```python
def likelihood(train_x, train_y, weight):
    total = np.size(train_x, axis=0)
    predict = np.zeros((total, 1))
    for i in range(total):
        predict[i] = np.dot(weight, train_x[i].T)
    t = 0
    for i in range(total):
        t += np.log(1 + np.exp(predict[i]))
    return np.dot(train_y, predict) - t
```

梯度下降算法：

```python
def descent_gradient(train_x, train_y, epoch, eta, eps, dimension, lam):
    total = np.size(train_x, axis=0)
    weight = np.ones((1, dimension + 1))
    epoch_list = np.zeros(epoch)
    loss_list = np.zeros(epoch)
    for i in range(epoch):
        old_loss = - 1 / total * likelihood(train_x, train_y, weight)
        t = np.zeros((total, 1))
        for j in range(total):
            t[j] = np.dot(weight, train_x[j].T)
        gradient = - 1 / total * np.dot(train_y - sigmoid(t.T), train_x)
        weight = weight - eta * lam * weight - eta * gradient # 梯度下降
        new_loss = - 1 / total * likelihood(train_x, train_y, weight)
        epoch_list[i] = i
        loss_list[i] = new_loss
        if i % 100 == 0:
            print(i, ', loss=', new_loss, ', weight=', weight, 'gradient=', gradient)
        if abs(new_loss - old_loss) < eps:
            epoch_list = epoch_list[:i+1]
            loss_list = loss_list[:i+1]
            break
    return weight, epoch_list, loss_list
```

在做 UCI 上的数据时，选取了皮肤 Skin_NonSkin.txt 数据。由于该数据量太大，这里只选取了其中一部分。

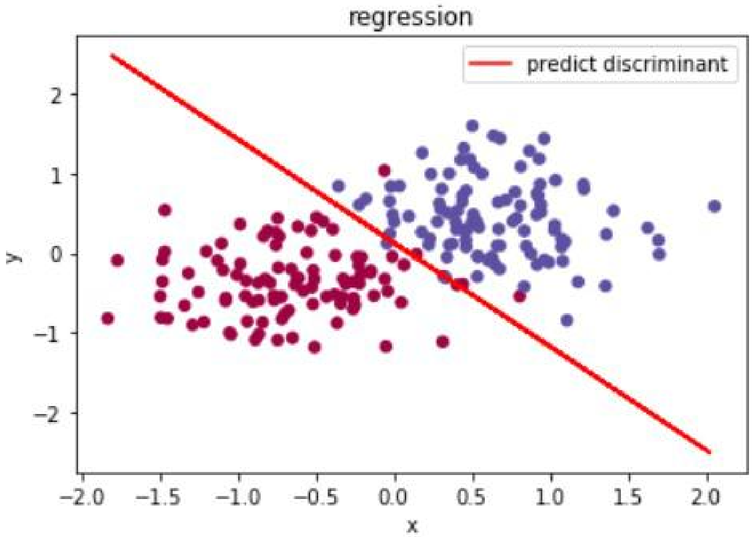
读取数据时，用 numpy 切片提取数据信息，用 50 作为步长，提取部分数据用做实验。还要对样本点进行空间平移，否则在计算 MCLE 时可能会溢出，因为计算 MCLE 时，要用参数与样本做矩阵乘法，而且还要作为的指数计算，可能会溢出。

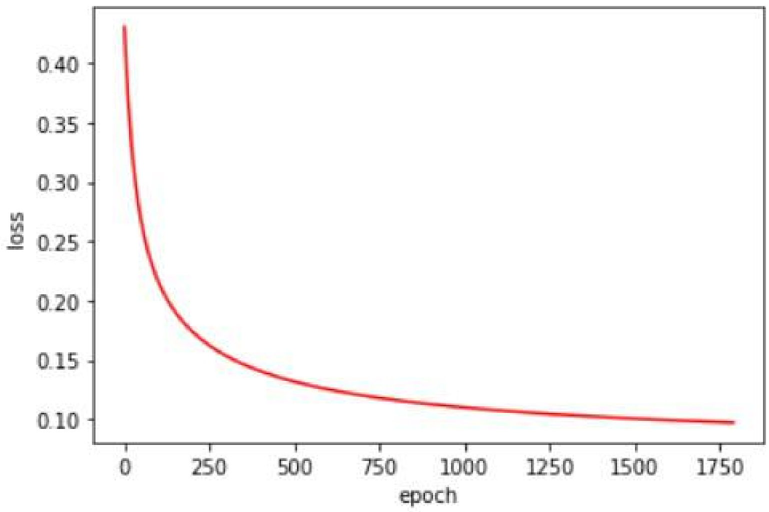
```python
def skin_gen_data():
    load_data = np.loadtxt('./Skin_NonSkin.txt', dtype=np.int32)
    np.random.shuffle(load_data) # 打乱原数据，以便分成训练集和测试集
    test_data_rate = 0.2 # 测试集比例
    load_data_size = np.size(load_data, axis=0)
    train_data = load_data[:int(test_data_rate * load_data_size), :] # 训练集数据
    test_data = load_data[int(test_data_rate * load_data_size):, :] # 测试集数据
    dim = np.size(load_data, axis=1) - 1 # 训练集样本维度

    step = 50 # 采用步长为50的方式，选取打乱的数据
    train_x = train_data[:,0:dim]
    train_x = train_x[::step]
    train_x = train_x - 100 # 对样本点进行坐标平移
    train_y = train_data[:,dim:dim + 1] - 1
    train_y = train_y[::step]
    train_size = np.size(train_x, axis=0)
    train_y = train_y.reshape(train_size) # 矩阵转化为行向量

    test_x = test_data[:,0:dim]
    test_x = test_x[::step] - 100  # 对样本点进行坐标平移
    test_y = test_data[:,dim:dim + 1] - 1
    test_y = test_y[::step]
    test_size = np.size(test_x, axis=0)
    test_y = test_y.reshape(test_size) # 矩阵转化为行向量
    return train_x, train_y, test_x, test_y
```

四、实验结果

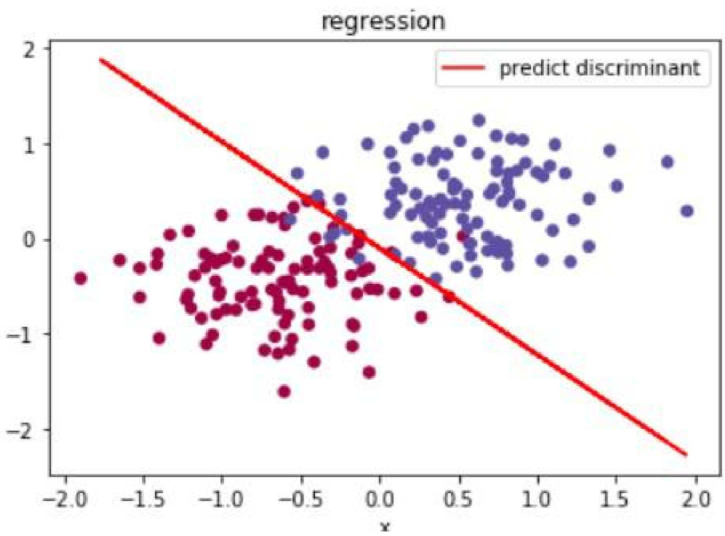
自己生成数据

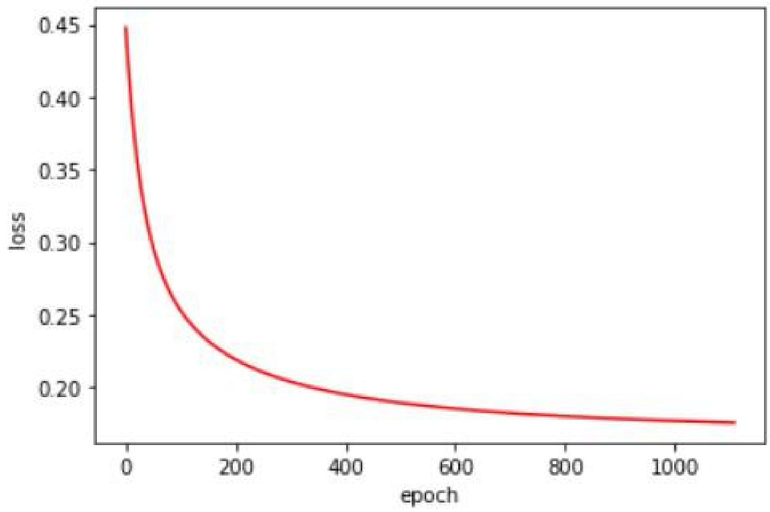
类条件概率满足朴素贝叶斯假设，正则项λ=0，size=200



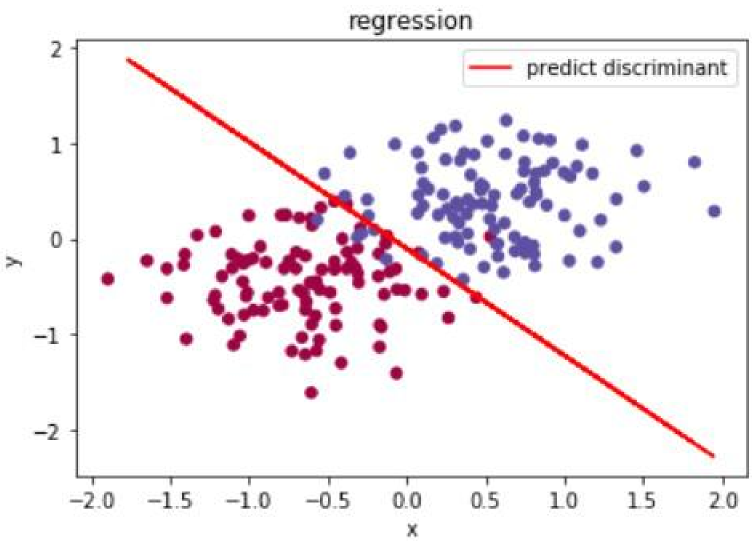


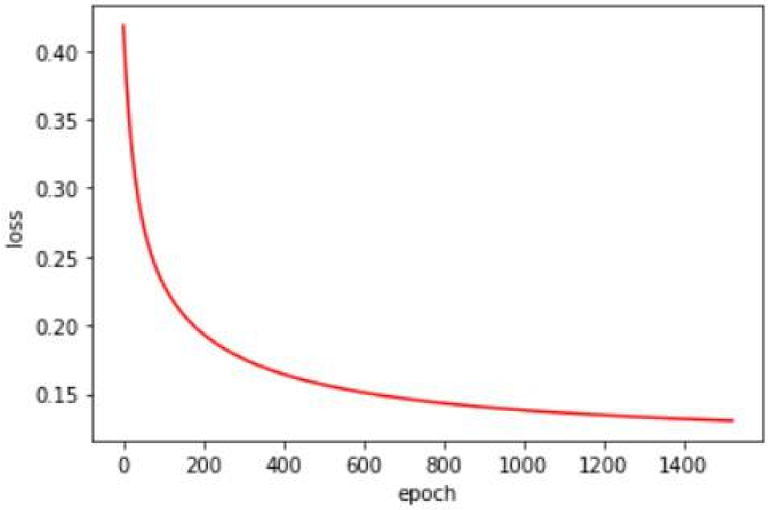
类条件概率不满足朴素贝叶斯假设，正则项λ=0，size=200



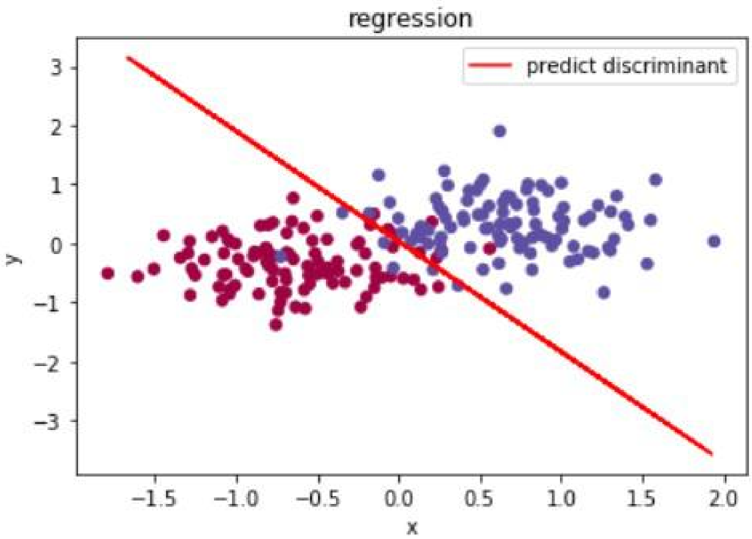


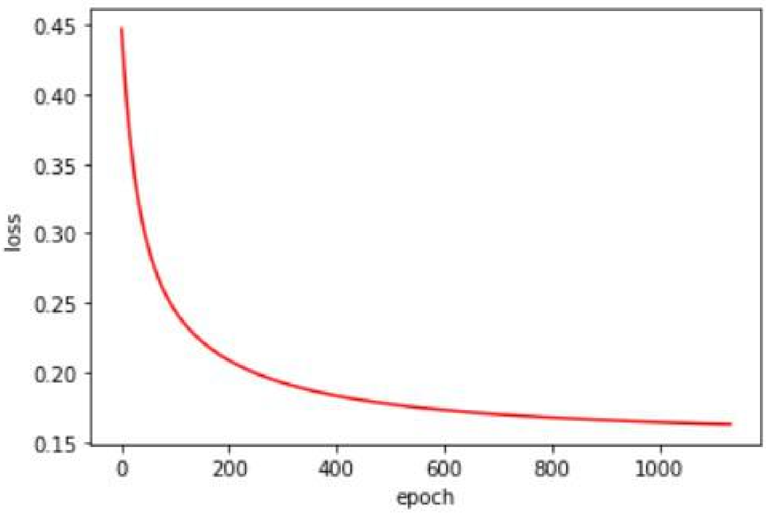
类条件分布满足朴素贝叶斯假设，正则项λ=0.001，size=200





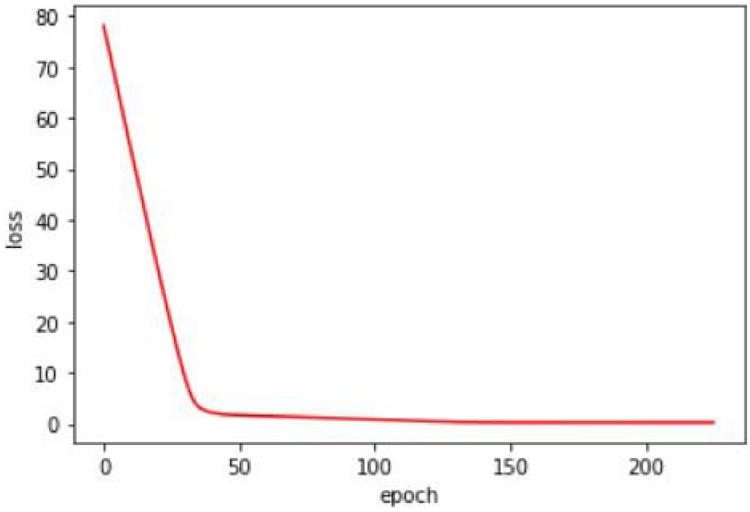
类条件概率不满足朴素贝叶斯假设，正则项λ=0.001，size=200

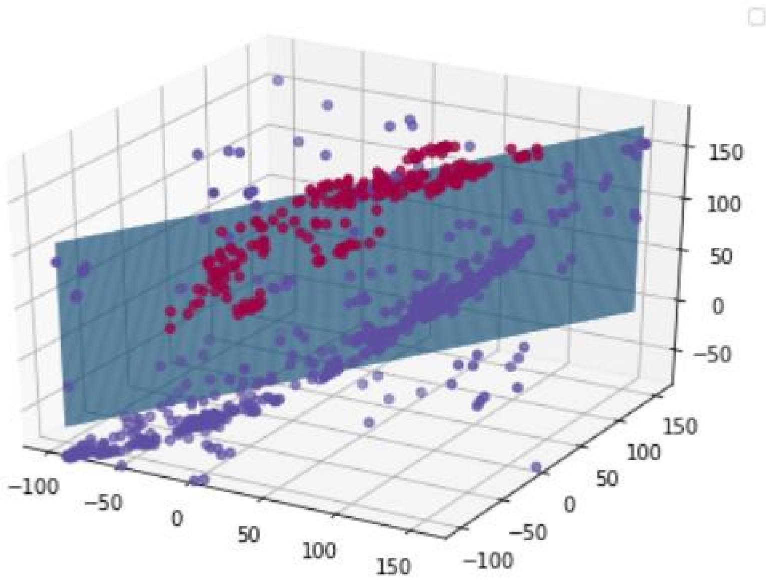




UCI 皮肤颜色数据集

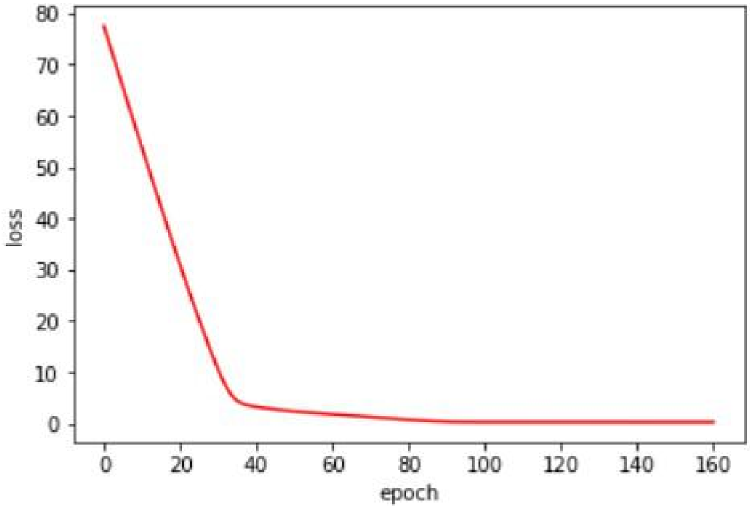
正则项 λ =0

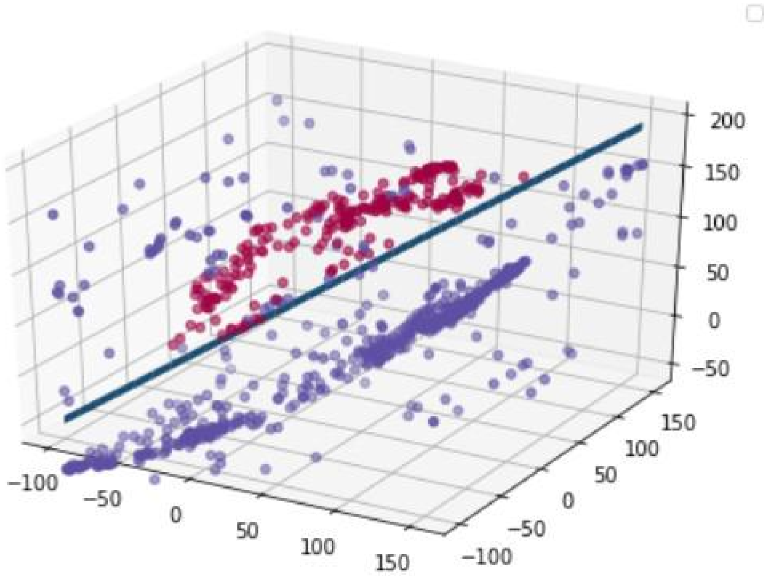




0.9466972711043101

正则项λ=0.01

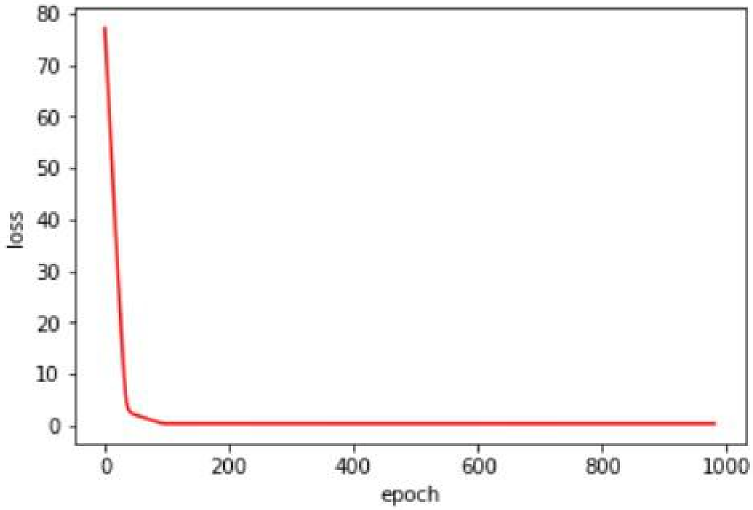




0.9380260137719969

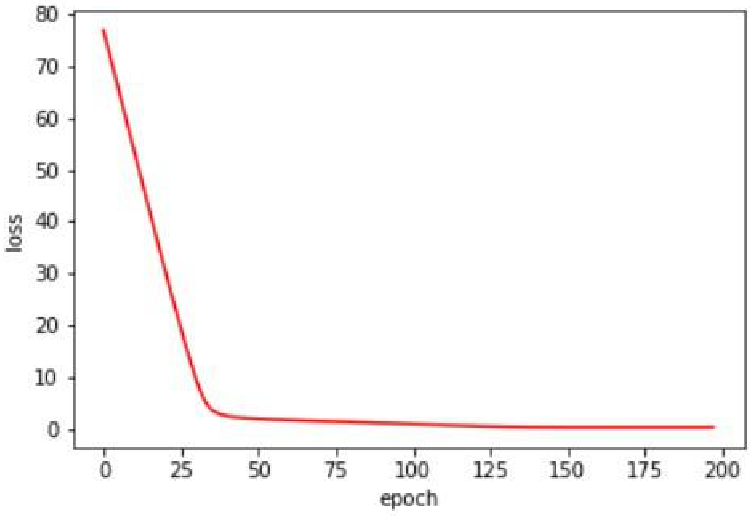
UCI banknote 数据集

正则项λ=0



0.9382810507523591

正则项λ=0.01



0.9410864575363428

实验发现，UCI 的数据的 20%测试集的准确率基本稳定在 93%-94%。 正则项在数据量较大时，对结果的影响不大，在数据量较小时， 应可以有效解决过拟合问题。 类条件分布在满足朴素贝叶斯假设时的分类表现，要比不满足假设时略好。 logistics 回归可以很好地解决简单的线性分类问题，而且收敛速度较快。