基于红葡萄酒品质的线性回归与逻辑回归

李厚霖 520020910007 (电子信息与电气工程学院)

摘 要

本次作业主要从提供的数据中，选取了一些关于红葡萄酒品质的数据，针对数据的多种特征和品质结果进行了线性回归和逻辑回归。

在线性回归中，我将有关红酒品质的 11 种特征作为输入，以最后的品质评分（所给数据的范围为 3-8）作为输出，从而将其进行多元线性回归，并得到了最后的拟合参数 β。最后经过交叉验证，均方误差在 0.45 左右，回归效果较好。

在逻辑回归中，我以与线性回归相同的变量作为输入和输出。与线性回归不同的是，我将输入经过 sigmoid 函数进行非线性变换，将输出（品质得分）小于 6 的值设为 0（较差）、品质得分大于等于 6 的值设为 1（较好），从而完成逻辑回归的数据处理。经过逻辑回归后，可以得到损失函数平滑且收敛，并在测试集上验证得到准确率大于 70%

关键词：**红葡萄酒、线性回归、逻辑回归**

1 问题描述

葡萄酒数据集中，根据多种指标，对若干种葡萄酒进行评级。其中，指标（即输入特征）有"fixed acidity";"volatile acidity";"citric acid";"residual sugar";"chlorides";"free sulfurdioxide";"total sulfur dioxide";"density";"pH";"sulphates";"alcohol" 等共 11 种，而品质 quality 则用 0-10 的整数来表示。其中在数据集中，品质评分分布在 3-8 之间。经统计，品质评分为 5、6 的较多，且大于等于 6 与小于 6 的数量较为均衡。因此，根据 csv 文件中的葡萄酒数据及其特性，设计出以下两种任务：

- 根据 11 种特征数据，进行多元线性回归，预测葡萄酒的品质评分。
- 设葡萄酒质量 < 6 为较差，赋于数值 0；≥ 6 为较好，赋于数值 1，采用逻辑回归的算法将葡萄酒品质的好坏进行分类。

接下来的报告中，将对这两个问题的实现原理及其代码进行分析。

2 多元线性回归

2.1 多元线性回归模型

多元线性回归模型通常用来研究一个应变量依赖多个自变量的变化关系，如果二者的以来关系可以用线性形式来刻画，则可以建立多元线性模型来进行分析。

多元线性回归模型通常用来描述变量 y 和 x 之间的随机线性关系，即：

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k + \xi \tag{1}$$

式中，$x_1, \cdots, x_k$ 是非随机的变量，y 是随机的因变量；$\beta_0, \ldots, \beta_k$ 是回归系数；ξ 是随机误差项。用矩阵表示可以写成：

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{11} & \ldots & x_{k1} \\ 1 & x_{12} & \ldots & x_{k2} \\ \vdots & \vdots & & \vdots \\ 1 & x_{1n} & \ldots & x_{kn} \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}, \quad \xi = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix} \tag{2}$$

故此时模型可以写作：$y = X\beta + \xi$

在正态假定下，如果 X 是列满秩的，则普通线性回归模型的参数的最小二乘估计为：

$$\hat{\beta} = \left(x^T x\right)^{-1} x^T y \tag{3}$$

因此，我们得到了计算 $\hat{\beta}$ 的公式。

此时，出现了一个问题，当若 X 非列满秩，即方程个数小于变量个数，此时 X^TX 则是不可逆的，故不能用此方法求解。因此，我们引入正则项概念，将 X^TX 矩阵加上一个值为　常数的对角矩阵，即：

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}+\lambda\boldsymbol{I}=\boldsymbol{U}\,\mathrm{diag}\left(\lambda_1+\lambda,\lambda_2+\lambda,\cdots,\lambda\right)\boldsymbol{U}^{\top} \tag{4}$$

该方法可解决不满秩问题，且总是可逆可以防止模型退化。

在葡萄酒数据中，不存在方程个数小于变量个数的问题，但是引入合适的正则项会使预测更加精准。

2.2 代码实现

代码实现主要分为以下几部分：数据预处理、模型训练、模型预测、误差计算

2.2.1 数据预处理

```
import pandas as pd
import numpy as np

data = pd.read_csv("./winequality-red.csv", sep=';')
#1600条数据8:2
data = np.array(data)
#加上 0的系数, 加在行上
data = np.concatenate((np.ones((1599,1)),data),axis=1)  #分割数据集, 并提取
                                                        出品质评分y
train_data = data[0:1280,:]
test_data = data[1280:1600,:]
print(train_data.shape)
train_data_x = train_data[:,0:12]
train_data_y = train_data[:,12:13]
test_data_x = test_data[:,0:12]
test_data_y = test_data[:,12:13]
print(train_data_x.shape)
print(train_data_y)
print(train_data_y.shape)
```

2.2.2 模型训练

```
##变成方阵并加入正则项
lambdax = np.eye(12,12)
lambdax = 0.1*lambdax
#转置
#Xt = np.transpose(train_data_x)
XtX = np.dot(train_data_x.T, train_data_x)
print(XtX.shape)
left = XtX + lambdax
Xty = np.dot(train_data_x.T, train_data_y)
beta = np.linalg.solve(left , Xty)
print(beta)
```

2.2.3 模型预测

```
##做预测
pre_data_y = np.dot(test_data_x, beta)
compare = np.concatenate((pre_data_y, test_data_y),axis=1)  #加在行上
print(compare)
```

2.2.4 误差计算

```
##测试结果
print(pre_data_y.shape)
print(test_data_y.shape)
mae = sum(abs(pre_data_y-test_data_y))/pre_data_y.shape[0]

mse = sum((pre_data_y-test_data_y)**2)/pre_data_y.shape[0]
#mae = sum()
print(mae)
print(mse)
```

2.3 结果分析

2.3.1 运行结果

所提供的红葡萄酒数据一共有 1600 条，故我将数据集按 8：2 的比例划分为训练集与测试集，经训练得以下数据：

1) β 参数（保留五位小数）：

$$\beta = \begin{pmatrix} 2.15317 \\ 0.00708 \\ -1.07761 \\ -0.18753 \\ 0.00083 \\ -1.69507 \\ 0.00335 \\ -0.00366 \\ 1.94 \\ -0.39155 \\ 0.77288 \\ 0.30546 \end{pmatrix}$$

2) 平均绝对值误差：$\sum |ypre - ytruth| = 0.49720655$

3) 平均均方误差: $\sum |ypre - ytruth|^2 = 0.4321651$

2.3.2 交叉验证

2.3.1 中的数据是将第 1280-1600 作为测试集，因此我在这采用交叉验证，共分为 5 组，对应数据分别为：[0:320],[321：640]，[640,960],[960,1280],[1280,1600] 得到以下均方误差:

测试集	0-320	320-640	640-960	960-1280	1280-1600
mae	0.52100712	0.52452746	0.50068564	0.50641931	0.49720655
mse	0.44394825	0.44792762	0.44443104	0.41057471	0.4321651

Figure 1: 不同测试集下的误差

2.3.3 正则项 λ 的选取

我取最初的 1280-1600 条数据作为测试集，通过修改正则项 λ 的值，从中找出训练结果最好的模型：

正则项λ	0	0.0001	0.0004	0.0005	0.0006	0.0007	0.001	0.01
mae	0.49697464	0.49706443	0.49725303	0.49728673	0.49731808	0.49735349	0.49742822	0.49761696
mse	0.43193864	0.431876	0.43183228	0.4318303	0.43183004	0.43183069	0.43183472	0.43188849

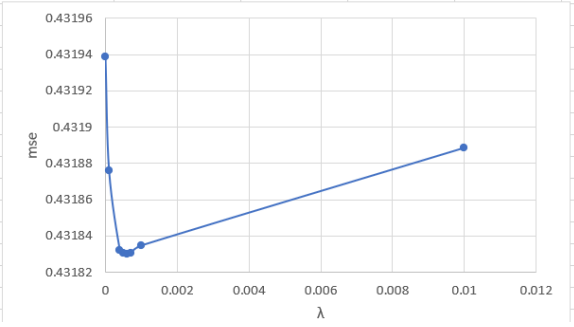


Figure 2: 不同

由上图可知，当 λ 取 0.0006 时，所得的测试集误差最小

2.3.4 总结

通过对实验结果得分析，我们得出并输出了　矩阵，我观测提取的部分数据与实际值得间误差较小，并最终五倍交叉验证下绝对值误差和均方误差均较小，故此次多元线性回归预测葡萄酒品质的任务完成较好。

3 逻辑回归

3.1 逻辑回归模型

3.1.1 逻辑回归建立

逻辑回归（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。逻辑回归因其简单、可并行化、可解释强深受工业界喜爱。逻辑回归的本质是：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。

由于在此问题中，前面提到，我们只需要将红酒品质进行二分类，故考虑二分类问题，给定数据集：

$$D = (x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N), x_i \subseteq R^n, y_i \in 0, 1, i = 1, 2, \cdots, N \tag{5}$$

考虑到 $w^T x + b$ 取值是连续的，因此它不能拟合离散变量。可以考虑用它来拟合条件概率 $p(Y = 1 \mid x)$ ，因为概率的取值也是连续的。

但是对于 $w \neq 0$ （若等于零向量则没有什么求解的价值），$w^T x + b$ 取值为 R ，不符合概率取值为 0 到 1，因此考虑采用广义线性模型。

最理想的是单位阶跃函数：

$$p(y = 1 \mid x) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases}, \quad z = w^T x + b \tag{6}$$

但是这个阶跃函数不可微，对数几率函数是一个常用的替代函数（即 sigmoid 函数）：

$$y = \frac{1}{1 + e^{-(w^T x + b)}} \tag{7}$$

于是有：

$$\ln \frac{y}{1 - y} = w^T x + b \tag{8}$$

我们将 y 视为 x 为正例的概率，则 1-y 为 x 为其反例的概率。两者的比值称为几率（odds），指该事件发生与不发生的概率比值，若事件发生的概率为 p。则对数几率：

$$\ln(\text{ odds }) = \ln \frac{y}{1 - y} \tag{9}$$

将 y 视为类后验概率估计，重写公式有：

$$w^T x + b = \ln \frac{P(Y = 1 \mid x)}{1 - P(Y = 1 \mid x)} \tag{10}$$

$$P(Y = 1 \mid x) = \frac{1}{1 + e^{-(w^T x + b)}} \tag{11}$$

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。

3.1.2 **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。

设：

$$\begin{aligned} P(Y=1 \mid x) &= p(x) \\ P(Y=0 \mid x) &= 1-p(x) \end{aligned} \tag{12}$$

似然函数：

$$L(w)=\prod\left[p\left(x_i\right)\right]^{y_i}\left[1-p\left(x_i\right)\right]^{1-y_i} \tag{13}$$

为了更方便求解，我们对等式两边同取对数，写成对数似然函数：

$$\begin{aligned} L(w) &= \sum\left[y_i \ln p\left(x_i\right)+\left(1-y_i\right) \ln \left(1-p\left(x_i\right)\right)\right] \\ &= \sum\left[y_i \ln \frac{p\left(x_i\right)}{1-p\left(x_i\right)}+\ln \left(1-p\left(x_i\right)\right)\right] \\ &= \sum\left[y_i\left(w \cdot x_i\right)-\ln \left(1+e^{w \cdot x_i}\right)\right] \end{aligned} \tag{14}$$

如果取整个数据集上的平均对数似然损失，我们可以得到:

$$J(w)=-\frac{1}{N} \ln L(w) \tag{15}$$

在逻辑回归模型中，最大化似然函数和最小化损失函数是等价的。

3.1.3 求解

在求解中，主要采用梯度下降的方法，将损失函数 $L(W)$ 对 w, b 进行偏导，经过一系列计算，最终可得逻辑回归损失函数的求偏导结果为为：

$$\frac{\partial \mathrm{L}}{\partial \omega}=\mathrm{X}^{\mathrm{T}}(\mathrm{y}-\hat{\mathrm{y}}); \quad \frac{\partial \mathrm{L}}{\partial \mathrm{b}}=\mathrm{y}-\hat{\mathrm{y}} \tag{16}$$

由于我加入正则项后准确率反而更低（可能是我代码的问题），因此在此不再叙述加入正则项之后的模型以及结果。

3.2 代码实现

同样，代码实现主要分为以下几部分：数据预处理、模型基本函数定义、模型训练及预测、主函数。对于 1600 条数据，我依照网上推荐的方法，将训练集与测试集 8：2 的比例进行分割，具体代码如下：

3.2.1 数据预处理

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
#数据处理
data = pd.read_csv("./winequality-red.csv", sep = ';')
data = np.array(data)
data = np.concatenate((np.ones((1599,1)),data),axis=1)  #加在行上
# dd = data[:,-1]
# dd = pd.Series(dd)
# dd = dd.value_counts()
# print(dd)
train_data = data[0:1280,:]
test_data = data[1280:1600,:]
train_data_x = train_data[:,0:12]
train_data_y = train_data[:,12:13]
print(train_data_y)
test_data_x = test_data[:,0:12]
test_data_y = test_data[:,12:13]
```

3.2.2 模型基本函数定义

```
#初始化w, b
def initial(dim):
    w  = np.zeros((dim, 1))
    b = 0
    return w, b
    #print(w)
    #print(w.shape)

def sigmoid(x):
    return 1.0/(1+np.exp(-x))

#initial(train_data_x.shape[1])
def logistics(X, y, w, b):
    num_of_data = X.shape[0]
    num_of_feature = X.shape[1]
    y_mat = sigmoid(np.dot(X, w)+ b)
```

```
    loss_total = np.sum(y*np.log(y_mat)+(1-y)*np.log(1-y_mat))
    loss_mean = -loss_total/num_of_data
    dw = np.dot(X.T, y-y_mat)/num_of_data
    db = sum(y-y_mat)/num_of_data
    #print(loss_mean)
    return y_mat, loss_mean, dw, db
#w,b = initial(train_data_x.shape[1])
#logistics(train_data_x,train_data_y,w,b)
```

3.2.3 模型训练及预测

```
##训练
def train(X, y, learning_rate=0.001, epochs=10000):
    epochs_loss = []
    w,b = initial(X.shape[1])
    for i in range(epochs):
        y_hat, loss, dw, db = logistics(X, y, w, b)
        w += learning_rate*dw
        b += learning_rate*db
        epochs_loss.append(loss)
        #if(i%5000 ==0):
            #learning_rate = learning_rate/2.0
            #print(learning_rate)
    params = {'w':w, 'b':b}
    grades = {'dw':dw, 'db':db}
    return epochs_loss, params, grades

#预测
def predict(X, params):
    w = params['w']
    b = params['b']
    y_pre = sigmoid(np.dot(X, w)+b)
    for i in range(len(y_pre)):
        if y_pre[i]>0.5:
            y_pre[i] = 1
        else:
            y_pre[i] =0
```

```
    return y_pre
```

3.2.4 主函数

```
##主函数
#print(train_data_y)
for i in range(len(train_data_y)):
    if train_data_y[i] >=6:
        train_data_y[i]=1
    else:
        train_data_y[i]=0
for i in range(len(test_data_y)):
    if test_data_y[i] >=6:
        test_data_y[i]=1
    else:
        test_data_y[i]=0

#print(train_data_y)
# n = np.sum(train_data_y == 0)
# print(n)
learning_rate = 0.001
ephochs = 100000
train_loss, params, grads = train(train_data_x,train_data_y, learning_rate,
                                  ephochs)
pre_train_data_y = predict(train_data_x, params)
pre_data_y = predict(test_data_x, params)
ephochs_list = np.arange(0, ephochs,1)
train_loss = np.array(train_loss)
plt.plot(ephochs_list, train_loss)
plt.xlabel('Epochs')
plt.ylabel('Cost')
```

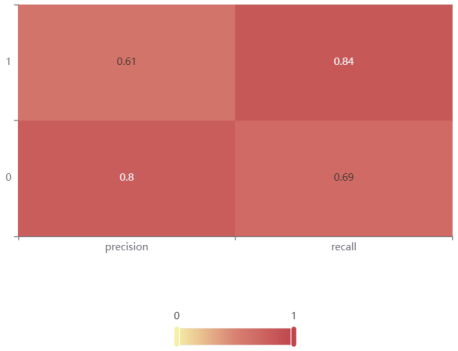
3.3 结果分析

选取 learning-rate=0.001、epochs=100000，以提取前 1280 组数据作为训练集、第 1280-1600 组数据作为测试集的为例，此时精确率为 73%，部分结果如下:

3.3.1 准确率、召回率等

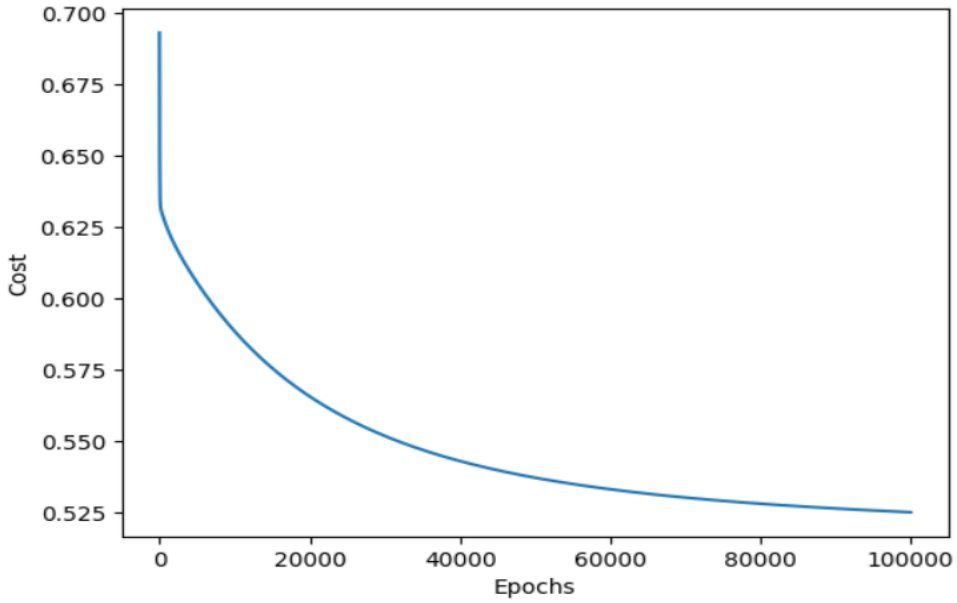
根据准确率、召回率的定义，我们可以得到如下表格结果以及对应的热力图（调了下 sklearn 的包哈哈哈）

	precision	recall	f1-score	support
0.0	0.80	0.61	0.69	159
1.0	0.69	0.84	0.76	160
accuracy			0.73	319
macro avg	0.74	0.73	0.72	319
weighted avg	0.74	0.73	0.72	319



3.3.2 学习率与迭代次数的选取

通过程序，我们可以得到损失函数与迭代次数间的关系，做出如下图像：



我们可以看到，当迭代次数超过 100000 次时，损失函数趋于稳定，并且收敛，故我选取迭代次数为 100000。

同时，我通过调节学习率，经过修改不同的学习率，得到精确率与学习率间的关系，如下表所示：

learning-rate	0.1	0.01	0.001	0.005
accuracy	65	70	73	72

可见，当学习率为 0.001 时，精确率最高，即模型效果最好。

本来此处还想根据迭代次数的增加逐渐缩小学习率，但是发现效果没有不变化学习率的好，因此也不在报告中展出。

3.3.3 交叉验证

以上的结果是以 0-1280 组数据作为训练集，1280-1600 组作为测试集得到的结论，为验证该方法的正确性，我们分别将 0-320、320-640、640-960、960-1280、1280-1600 作为测试集，其余作为训练集，计算正确率，如下表:

测试集	0-320	320-640	640-960	960-1280	1280-1600
accuracy	67	75	74	75	74

五倍交叉验证之后，平均准确率为 73.2%，任务完成得较为理想。

3.3.4 总结

通过对实验结果得分析，我们得出并输出了预测值和实际值，并最终五倍交叉验证下准确率较高，故此次多元线性回归预测葡萄酒品质的任务完成较好。

4 总结与展望

本次用多元线性回归和逻辑回归的方式对葡萄酒数据集进行预测和分类，其中多元线性回归是预测葡萄酒的具体品质，逻辑回归是判断酒的品质是较好还是较坏。两个实验中，虽然预测的结果存在一些误差，但总体预测效果较好。同时在这次作业中，也让我更加深入了解了线性回归和逻辑回归算法，加深了我对课程内容的理解。