南開大學
Nankai University

南　开　大　学

计 算 机 学 院

机器学习实验报告

实验二 回归模型

姓名：王泳鑫

学号：1911479

年级：2019级

专业：计算机科学与技术

指导教师：卫金茂

2021 年 11 月 8 日

摘要

关键字：线性回归，多元线性回归，**Machine Learning ， Deep Learning**

目录

一、 实验描述 **1**

（一） 实验内容 1

（二） 实验要求 1

二、 代码实现 **1**

（一） 实验要求1与结果展示 1

（二） 实验要求2 3

一、 实验描述

（一） 实验内容

利用课上所学实现线性回归模型，来完成以下实验要求。

（二） 实验要求

基本要求：

1.根据数据集dataset_regression.csv,求最小二乘解，用得到的回归方程生成五个测试样本，画出回归曲线，给出训练误差。

2.将数据集winequality_white.csv按照4：1划分为训练集和测试集，构造线性回归模型，采用批量梯度下降或者随机梯度下降均可；输出训练集和测试集的均方误差(MSE), 画出MSE收敛曲线。

中级要求：

尝试不同的学习率并进行MSE曲线展示、分析选择最佳的学习率。

二、 代码实现

（一） 实验要求1与结果展示

根据数据集dataset_regression.csv，在利用最小二乘法时，我们得到x和y，但是在求解逆矩阵时，发现矩阵不存在逆矩阵，如下图5所示

```
xtx=x.T*x
xtx
```

```
array([[ 4.  ,  3.  ,  2.  ,  1.  , -0.  , -1.  , -2.  , -3.  , -4.  ],
       [ 3.  ,  2.25,  1.5 ,  0.75, -0.  , -0.75, -1.5 , -2.25, -3.  ],
       [ 2.  ,  1.5 ,  1.  ,  0.5 , -0.  , -0.5 , -1.  , -1.5 , -2.  ],
       [ 1.  ,  0.75,  0.5 ,  0.25, -0.  , -0.25, -0.5 , -0.75, -1.  ],
       [-0.  , -0.  , -0.  , -0.  ,  0.  ,  0.  ,  0.  ,  0.  ,  0.  ],
       [-1.  , -0.75, -0.5 , -0.25,  0.  ,  0.25,  0.5 ,  0.75,  1.  ],
       [-2.  , -1.5 , -1.  , -0.5 ,  0.  ,  0.5 ,  1.  ,  1.5 ,  2.  ],
       [-3.  , -2.25, -1.5 , -0.75,  0.  ,  0.75,  1.5 ,  2.25,  3.  ],
       [-4.  , -3.  , -2.  , -1.  ,  0.  ,  1.  ,  2.  ,  3.  ,  4.  ]])
```

图 1: 无法求逆的矩阵

这个矩阵无法求逆，所以我们只能放弃最小二乘法，采用批量梯度下降，求解回归曲线。也可以通过scipy库里的leastsq最小二乘法，代码如下：

首先是数据集的导入，并且把数据集中的第二列和第三列划分为x和y，代码如下：

```
1  dataset = np.loadtxt('D:\\Data\\dataset_regression.csv',float,delimiter=',',
                                                skiprows=1,usecols=(1,2))
2  x=dataset[:,0]
3  y=dataset[:,1]
```

接下来就是，规范我们要拟合的曲线的形式了，代码如下：

```
    #需要拟合的函数#func 指定函数的形状:

def func(p, x):
    k, b = p

    return k * x + b


#偏差函数: #x,都是列表y这里的:x,更上面的yXi,中是一一对应的Yi

def error(p, x, y):
    return func(p, x) - y
```

leastsq函数返回值第一个表示拟合结果，第二个表示最后的误差：

```
    Para = leastsq(error, p0, args=(Xi, Yi))
```

经过拟合后，我们得到的最终结果如下图5所示：

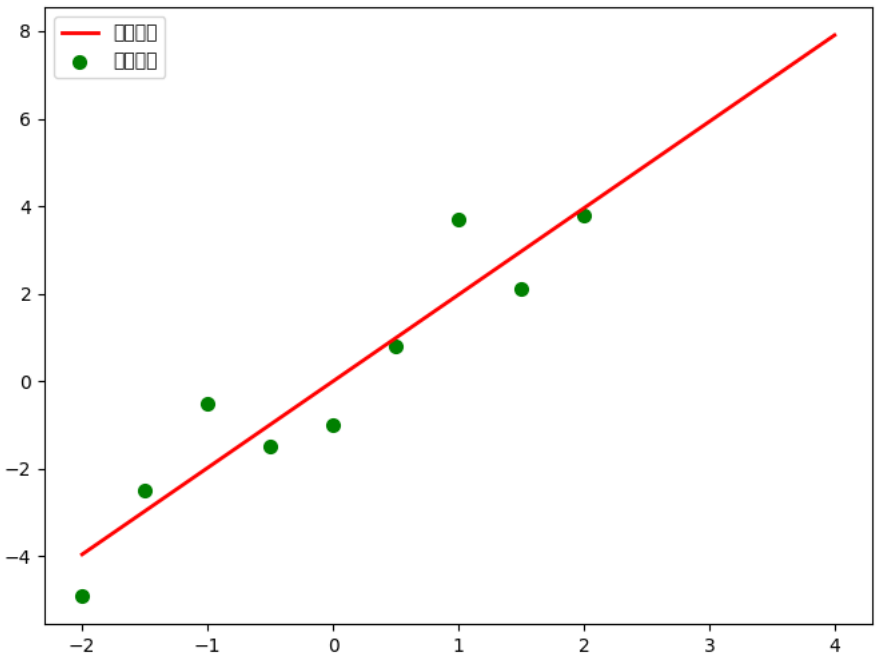

图 2: 拟合结果

拟合函数的数值和误差如下图5所示：

```
k= 1.97666666667534 b= -8.72457661671433e-12
cost: 3
```

图 3: 拟合结果

（二） 实验要求2

本实验将数据集按照4：1划分为训练集和测试集，在这里，我通过留一法来划分，代码如下：

```
def loo_split(data, label, idx):
    train_data = []
    test_data = []
    train_label = []
    test_label = []
    for i in range(0, idx):
        train_data.append(data[i])
        train_label.append(label[i])
    test_data.append(data[idx])
    test_label.append(label[idx])
    for i in range(idx+1, data.shape[0]):
        train_data.append(data[i])
        train_label.append(label[i])
    return np.array(train_data), np.array(train_label), \
           np.array(test_data), np.array(test_label)
```

在实验要求2中，我利用随机梯度下降来完成线性回归，代码如下：

```
def linear_regression(data, label, lr, epochs, reg=None, alpha=None, random=False):
    data = data_normalization(data)
    preds = []
    for i in range(data.shape[0]):  # 每一条数据都要用留一法进行预测
        train_data, train_label, test_data, test_label = loo_split(data, label, i)
        errors, weight = gradient_descent(train_data, train_label, lr, epochs, reg,
                                          alpha, random)
        test_x = np.hstack((np.array([1]).reshape((1, 1)), test_data))
        pred_y = np.dot(test_x, weight)
        preds.append(pred_y)
        if i % 100 == 0:
            print(i, 'loo')
    preds = np.array(preds).reshape(label.shape)
    rmse = np.sqrt(((preds - label) ** 2).mean())  # 计算所有预测值的RMSE
    return preds, rmse, errors
```

其中我们要先把数据标准化，代码如下：

```
def data_normalization(data):
    data_normed = data
    for i in range(data.shape[1]):
        cur_col = data[:, i]
        mean = np.mean(cur_col)
        std = np.std(cur_col)
        for j in range(cur_col.shape[0]):
            data_normed[j, i] = (data[j, i] - mean) / std
    return data_normed
```

然后对于每一条数据都要进行一次留一法进行预测，然后对训练集进行梯度下降的计算，同时返回误差和权重，代码如下：

```
def gradient_descent(data, label, lr, epochs, reg=None, alpha=None, random=False):
    weight = np.zeros((12))  # 在个特征列基础上加一个常数项11
    x = np.hstack((np.ones((data.shape[0], 1)), data))  # 给数据加上全的一列1
```

```
    y = label
    errors = []
    if reg is None:
        for epoch in range(epochs):
            error, grad = gradient(weight, x, y, random)
            weight = weight - lr * grad
            errors.append(error)
    elif reg == 'L1':
        for epoch in range(epochs):
            error, grad = gradient_L1(weight, x, y, alpha, random)
            weight = weight - lr * grad
            errors.append(error)
    else:
        for epoch in range(epochs):
            error, grad = gradient_L2(weight, x, y, alpha, random)
            weight = weight - lr * grad
            errors.append(error)
    return errors, weight
```

对于梯度下降的计算，我采用了三个不同loss函数计算来优化梯度下降，代码如下：

```
   def gradient(weight, x, y, random=False):
    if random is False:
        error = np.dot(x, weight) - y
        return abs(error.mean()), (1/x.shape[0]) * np.dot(np.transpose(x), error)  # 注意各个矩阵维度的匹配
    else:
        rand = np.random.randint(0, x.shape[0] - 1)  # 用中的方法生成随机数numpy
        x_rand = x[rand, :]
        y_rand = y[rand]
        error = np.dot(x_rand, weight) - y_rand
        return abs(error.mean()), (1/x.shape[0]) * np.dot(x_rand, error)


def gradient_L1(weight, x, y, alpha, random=False):
    if random is False:
        error = np.dot(x, weight) - y
        return abs(error.mean()), (1/x.shape[0]) * (np.dot(np.transpose(x), error)
                                                    + alpha * np.sign(weight))  # 正则项求导后为sign(x)
    else:
        rand = np.random.randint(0, x.shape[0] - 1)
        x_rand = x[rand, :]
        y_rand = y[rand]
        error = np.dot(x_rand, weight) - y_rand
        return abs(error.mean()), (1/x.shape[0]) * (np.dot(x_rand, error) + alpha * np
                                                    .sign(weight))


def gradient_L2(weight, x, y, alpha, random=False):
    if random is False:
```

```
        error = np.dot(x, weight) - y
        # 正则项求导后为值的一次形式weight
        return abs(error.mean()), (1/x.shape[0]) * (np.dot(np.transpose(x), error) +
                                                    alpha * weight)
    else:
        rand = np.random.randint(0, x.shape[0] - 1)
        x_rand = x[rand, :]
        y_rand = y[rand]
        error = np.dot(x_rand, weight) - y_rand
        return abs(error.mean()), (1/x.shape[0]) * (np.dot(x_rand, error) + alpha *
                                                    weight)
```

实验结果如下图5所示：

```
rmse gradient descent: 0.7533526439305198
time cost gradient descent: 1069.351390361786 s
rmse L1 regularization gradient descent: 0.7533515480501173
time cost L1 regularization gradient descent: 1065.2556941509247 s
rmse L2 regularization gradient descent: 0.7533518212752452
time cost L2 regularization gradient descent: 1145.1894817352295 s
rmse random gradient descent: 5.825422934553118
time cost random gradient descent: 126.22532963752747 s
rmse L1 regularization random gradient descent: 5.828116819244598
time cost L1 regularization random gradient descent: 127.10175609588623 s
rmse L2 regularization random gradient descent: 5.8260752016775355
time cost L2 regularization random gradient descent: 126.08375358581543 s
```

图 4: 实验结果

误差和迭代次数的关系如下图5所示：

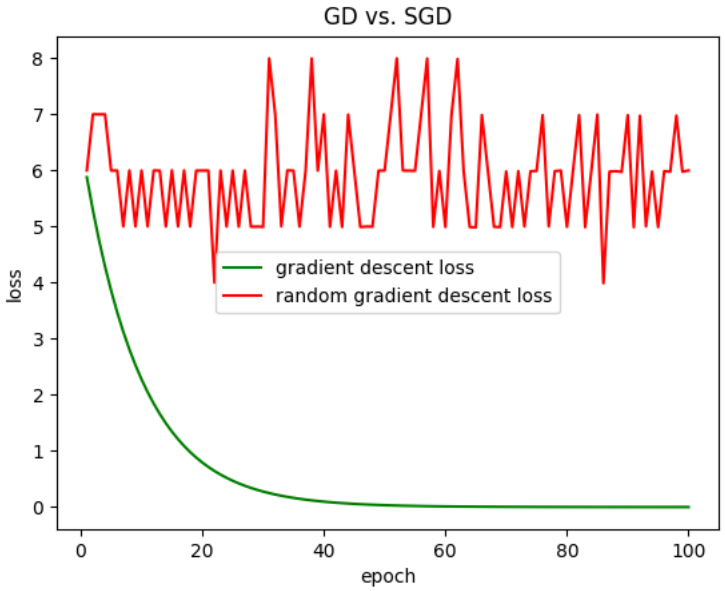

图 5: 关系