19335030_陈至雪 _HW3

Exercise 1

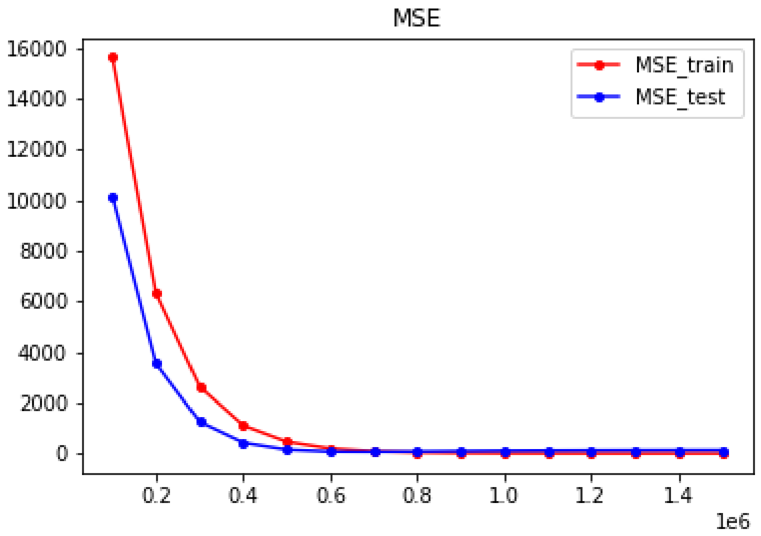
(a)

该数据集中一共有两个自变量，因此需要训练的参数有三个，此处设它们分别为 [w1, w2, w3]，其中 w1 为第一个自变量的权重，w2 为第二个自变量的权重，w3 为常量（偏置）。因为使用梯度下降法更新参数，需要手动设置的超参数有每次更新时的步幅大小，此处设为α.

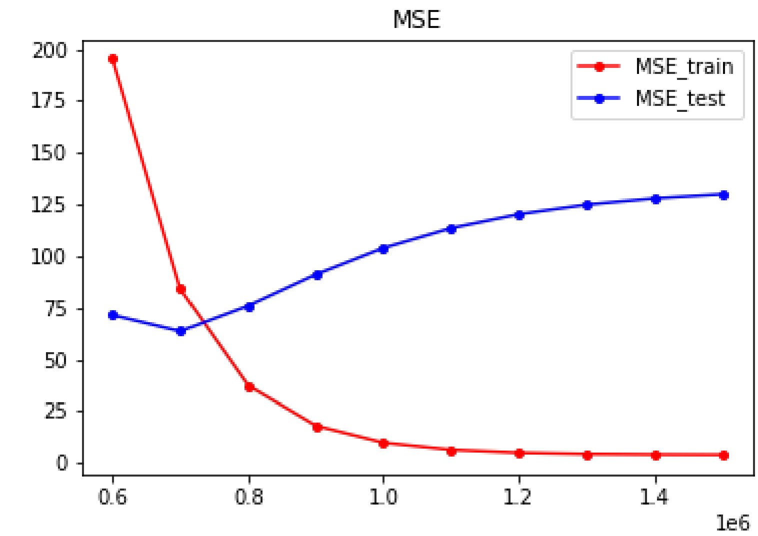
实现细节：

开始的时候，我未对数据进行预处理，直接进行线性回归，预测房价，后来计算出来的 MSE 过大，超出了python flaot能表示的精度范围，对参数 w 进行输出也发现 w 的数值很大。于是我对数据进行了输出，发现第一个自变量（房的面积）和第二个自变量（对应房子距离双鸭山职业技术学院的距离）的数值不在一个数量级上，这可能是导致 MSE 和 w 过大的原因，于是我对数据进行了归一化处理。处理后的结果是，不再出现以上现象，MSE 和 w 也在迭代过程中逐渐收敛。

实验结果：



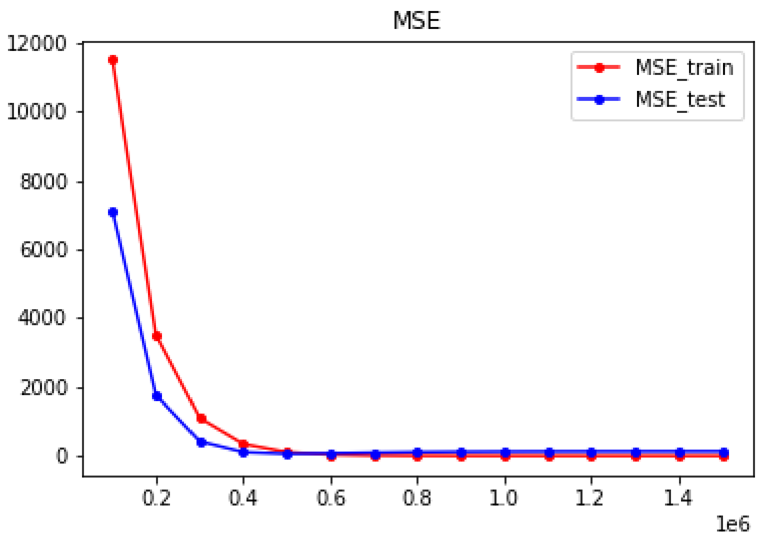
可以看到在 600000 次迭代前，训练集和测试集上的 MSE 都是下降的，但是训练集的 MSE 大于测试集的 MSE。600000 次迭代后，两个MSE的范围接近，于是对大于600000次迭代后得到的 MSE 进行二次绘图，结果如下：

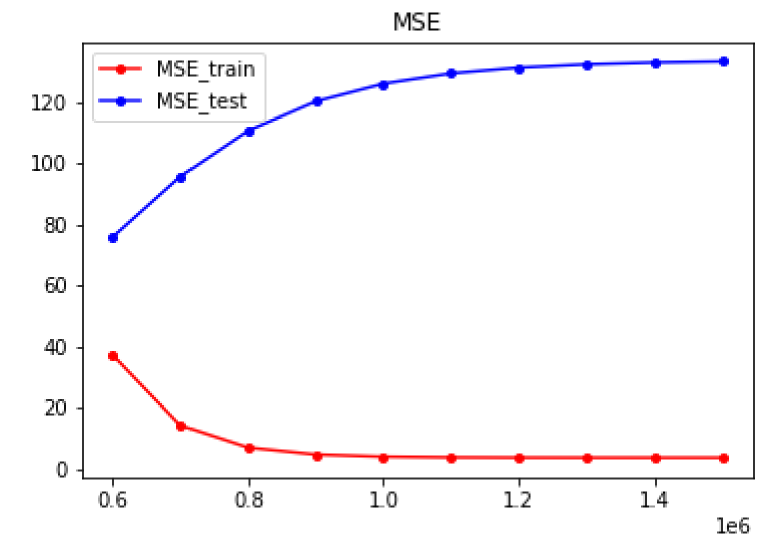


可以看到继续训练训练集上的 MSE 继续下降，但是约在 700000 次迭代后，测试集上的 MSE 不降反升，说明此时模型已经出现了过拟合现象。

（b）

将学习率由（a）中的0.00015改为 0.0002 后可以发现

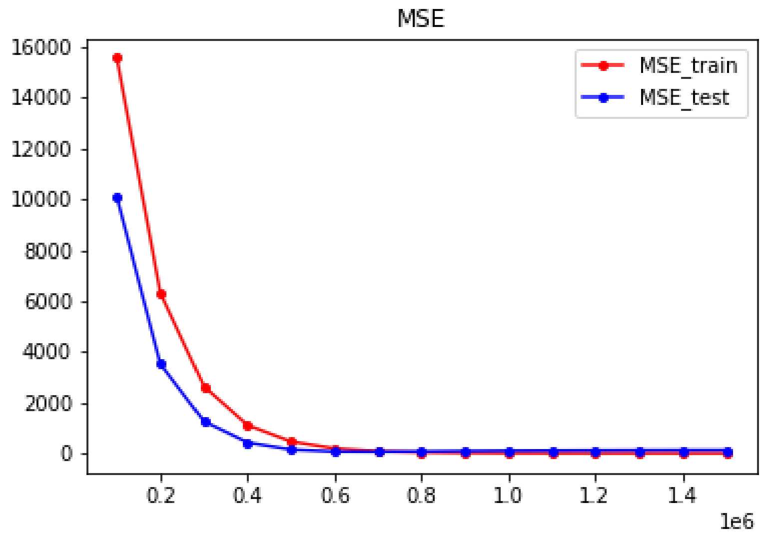




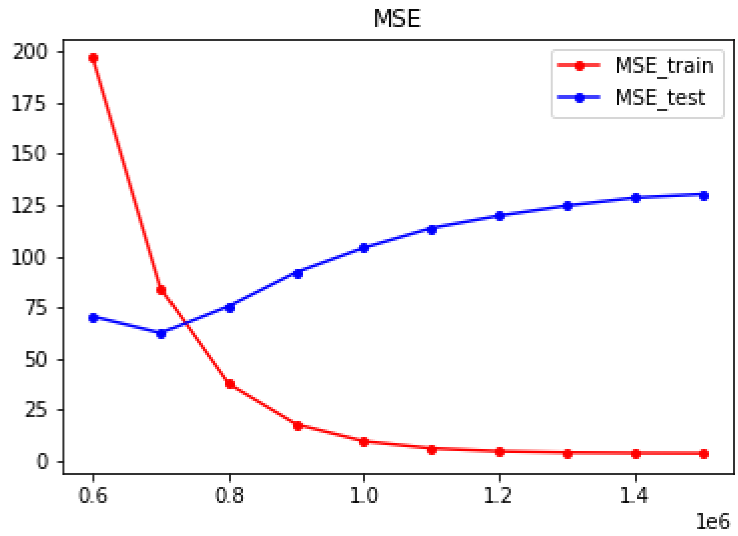
在前 500000 次迭代中训练集和测试集的 MSE 都在变小，在 500000 次迭代后训练集的 MSE 继续变小，但是测试集的 MSE 开始变大。并且可以发现，学习率为 0.00015 的 MSE 比学习率为 0.0002的 MSE 下降得快，说明学习率的选取会影响模型的最终表现。

（c）

此处采用了随机梯度下降的方法。我可能可以用梯度下降法得到最优结果，但是不能保证。采用随机梯度下降法，查看训练集和测试集上的 MSE。学习率为 0.00015. 迭代次数为 1500000.



可以看到在 600000 次迭代前，训练集和测试集上的 MSE 都是下降的，但是训练集的 MSE 大于测试集的 MSE。600000 次迭代后，两个MSE的范围接近，于是对大于600000次迭代后得到的 MSE 进行二次绘图，结果如下：



可以看到继续训练训练集上的 MSE 继续下降，但是约在 700000 次迭代后，测试集上的 MSE 不降反升，说明此时模型已经出现了过拟合现象。

对比同样参数下的梯度下降法，即题（a）中 数据，可以发现两者相差甚微，但是从更新时间上看，随机梯度下降比梯度下降快，因此随机梯度下降是梯度下降的一个不错的替代方法。但两者均无法保证可以取得全局最优解。

代码细节：

读取数据的代码如下：

```
def openreadtxt(file_name):
    data = []
    file = open(file_name,'r')  #打开文件
    file_data = file.readlines() #读取所有行
    for row in file_data:
        tmp_list = row.split(' ') #按‘，’切分每行的数据
        tmp_list[-1] = tmp_list[-1].replace('\n','') #去掉换行符
        for i in range(len(tmp_list)): #将字符串转化为浮点数
            tmp_list[i] = float(tmp_list[i])
        data.append(tmp_list) #将每行数据插入data中
    return data
```

对数据做归一化处理函数实现如下：

```
#归一化处理
def autoNorm(data, mins, maxs):         #传入一个矩阵
    ranges = maxs - mins    #最大值列表 - 最小值列表 = 差值列表
    normData = np.zeros(np.shape(data))     #生成一个与 data矩阵同规格的normData全0矩阵，用于装归一化后的数据
    row = data.shape[0]                     #返回 data矩阵的行数
    normData = data - np.tile(mins,(row,1)) #data矩阵每一列数据都减去每一列的最小值
    normData = normData / np.tile(ranges,(row,1))   #data矩阵每一列数据都除去每一列的差值（差值 = 某列的最大值- 某列最小值）
    return normData
```

归一化的公式是$data' = \frac{data - data_{min}}{data_{max} - data_{min}}$. 测试集上的归一化处理使用的是训练集上的最小值和最大值。

对每个输入自变量x，预测 y 的函数实现如下：

```
def pred(x, w):
    pred_y = 0
    for i in range(len(w)):
        if(i == len(w)-1): #w[i]是偏置
            pred_y += w[i]
        else:
            pred_y += (x[i] * w[i])
        #print(pred_y)
    return pred_y
```

对输入一系列 X， 预测一系列 Y 的函数实现如下：

```
def pred_Y(X,w):
    Y_pred = []
    for i in range(len(X)):
        y = pred(X[i],w)
        Y_pred.append(y)
    return Y_pred
```

随机梯度下降更新参数实现如下：

```
def SGD(Y, X, w, alpha):
    '''
    w: 参数
    alpha: 学习率
    '''
    i = randint(0,len(Y)-1)
    for j in range(len(w)):
        if( j == len(w) - 1 ):
            w[j] = w[j] + alpha * (Y[i] - pred(X[i],w))
        else:
            w[j] = w[j] + alpha * (Y[i] - pred(X[i],w)) * X[i][j]
```

梯度下降实现如下:

```
def GD(Y, X, w, alpha):
    sum0 = 0
    sum1 = 0
    sum2 = 0
    for i in range(len(X)):
        y_pred = pred(X[i],w)
        sum0 += (Y[i] - y_pred) * X[i][0]
        sum1 += (Y[i] - y_pred) * X[i][1]
        sum2 += (Y[i] - y_pred)
    w[0] += alpha * (sum0 / len(X))
    w[1] += alpha * (sum1 / len(X))
    w[2] += alpha * (sum2 / len(X))
```

计算均方误差函数实现如下:

```
def MSE(Y_pred, Y):
    MSE = 0
    #for i in range(len(Y)):
    #    #print(round((Y_pred[i] - Y[i]),2))
    #    tmp = pow((Y_pred[i] - Y[i]),2)
    #    MSE += tmp
        #print(MSE)
    #MSE = MSE/len(Y)
    MSE = metrics.mean_squared_error(np.array(Y), np.array(Y_pred))
    return MSE
```

绘图代码如下:

```
xx = [100000,200000,300000,400000,500000,600000,700000,800000,900000,1000000,1100000,1200000,1300000,1400000,1500000]
plt.plot(xx, MSE_train,color = 'red',marker='o',markersize=4, label='MSE_train')
plt.plot(xx,MSE_test, color = 'blue',marker='o',markersize=4, label='MSE_test')
plt.legend()
plt.title("MSE")
plt.savefig('MSE.png')
```

2、

(a)

因为是二分类问题，可使用 sigmoid 函数作为逻辑函数，表示样本属于某个类别的概率。

$$Sigmoid(y) = \frac{1}{1+e^{-y}}$$

用$Sigmoid(y)$表示样本属于正类的概率，用$1 - Sigmoid(y)$表示样本属于负类的概率。

条件似然函数可写成：

$$\begin{aligned} l(w) &= \sum_l y^l lnP(y^l = 1|x^l, w) + (1-y^l)lnP(y^l = 0|x^l, w) \\ &= \sum_l y^l(w0 + \sum_{i=1}^{p} w_i x_i^l) - ln(1 + exp(w0 + \sum_{i=1}^{p} w_i x_i^l)) \end{aligned}$$

其中

$$P(y^l|x^l, w) = Sigmoid(\sum_{i=1}^{p} w_i x_i^l)$$

(b)

$$\begin{aligned} \frac{\partial}{\partial w_i} l(w) &= \sum_l y^l xi - \frac{\sum_{i=1}^{p} x_i}{1 + exp(w0 + w_i x_i^l)} \\ &= \sum_l x_i^l (y^l - \hat{P}(y^l = 1|x^l, w)) \end{aligned}$$

同理，对w0求导可得

$$\frac{\partial}{\partial w_0} l(w) = \sum_l (y^l - \hat{P}(y^l = 1|x^l, w))$$

(c)

使用梯度下降法更新参数。

$$w_i = w_i + \alpha \sum_l x_i^l (y^l - \hat{P}(y^l = 1|x^l, w))$$

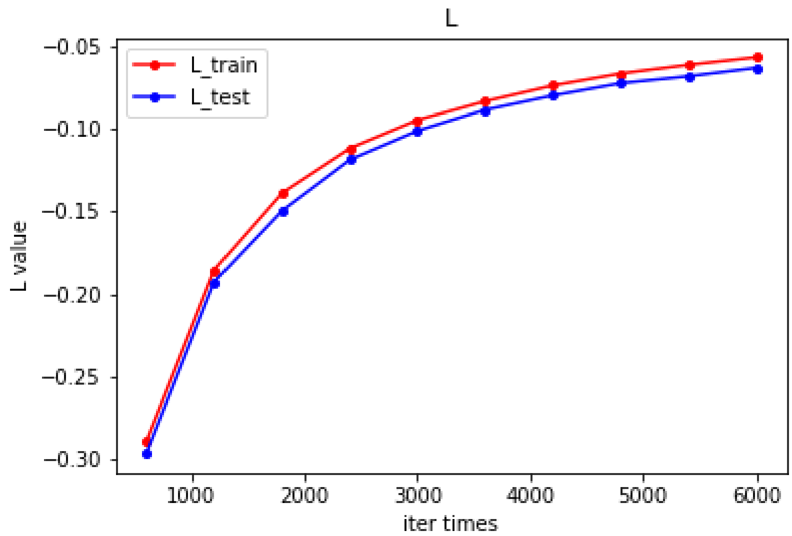
其中α为学习率。

代码实现如下：

```
def GD(Y, X, w, alpha):
    sum0 = 0
    sum1 = 0
    sum2 = 0
    sum3 = 0
    sum4 = 0
    sum5 = 0
    sum6 = 0
    for i in range(len(X)):
        y_pred = pred(X[i],w)
        sum0 += (Y[i] - y_pred) * X[i][0]
        sum1 += (Y[i] - y_pred) * X[i][1]
        sum2 += (Y[i] - y_pred) * X[i][2]
```

```
            sum3 += (Y[i] - y_pred) * X[i][3]
            sum4 += (Y[i] - y_pred) * X[i][4]
            sum5 += (Y[i] - y_pred) * X[i][5]
            sum6 += (Y[i] - y_pred)
        w[0] += alpha * (sum0 )
        w[1] += alpha * (sum1 )
        w[2] += alpha * (sum2 )
        w[3] += alpha * (sum3 )
        w[4] += alpha * (sum4 )
        w[5] += alpha * (sum5 )
        w[6] += alpha * (sum6 )
```

实验结果：



将迭代次数调至 6001，学习率调为 0.02，将结果绘制成图，其中纵坐标为（a）中的条件对数似然函数，可以看到随着训练次数增加，L值在变大。

（d）

将上题的参数视为最优，在测试集上进行预测和对结果进行统计。

实现代码如下：

```
#在测试集上统计结果
test_pred = pred_Y(X_test,w)
sum = 0
for i in range(len(test_pred)):
    tmp = 0
    if(test_pred[i] > 0.5):
        tmp = 1
    sum += abs(tmp - Y_test[i])
print(sum)
```

结果显示为 0，即没有样本被分错类。

（e）

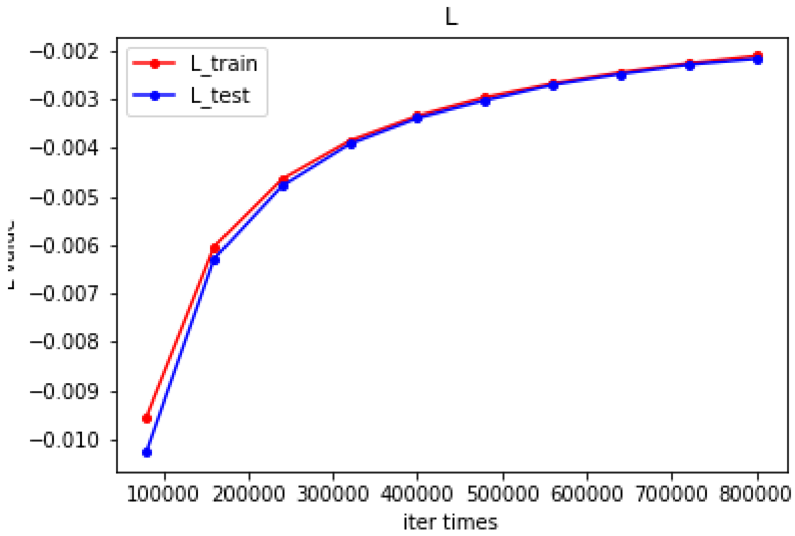
使用梯度下降法更新参数。

实现代码如下：

```python
def SGD(Y, X, w, alpha):
    '''
    w: 参数
    alpha: 学习率
    '''
    i = randint(0,len(Y)-1)
    for j in range(len(w)):
        if( j == len(w) - 1 ):
            w[j] = w[j] + alpha * (Y[i] - sigmoid(pred(X[i],w)))
        else:
            w[j] = w[j] + alpha * (Y[i] - sigmoid(pred(X[i],w))) * X[i][j]
```

经过多次测试，我将学习率α设置为 0.02，将迭代次数设置为 500001。

实验结果如下图：



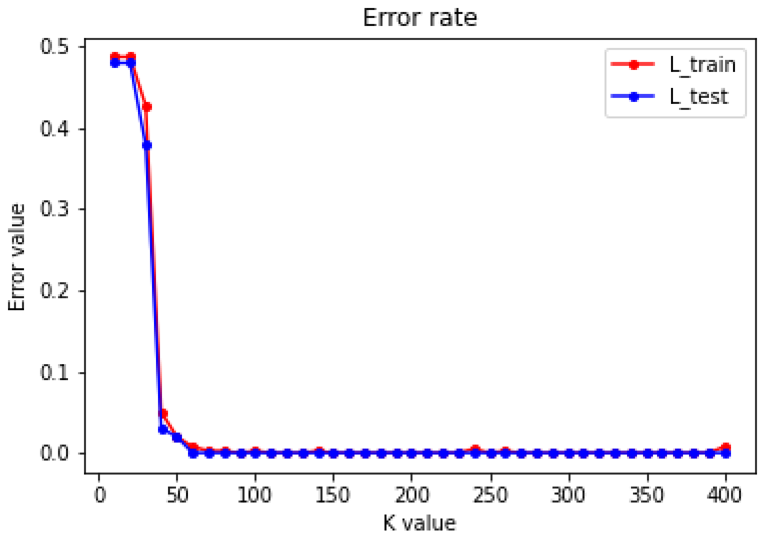
可以看到训练集和测试集上结果都变好，训练集上的结果略好于测试集上的结果。迭代收敛需要400000次左右。

（f）

错误率的计算函数：

```python
def error(y_pred,y):
    sum = 0
    for i in range(len(y_pred)):
        tmp = 0
        if(y_pred[i] > 0.5):
            tmp = 1
        sum += abs(tmp - y[i])
    return sum / len(y)
```

设置训练迭代次数为 100 次，学习率为 0.02，实验结果如下：



可以看到训练集和测试集上的错误率变化比较一致，当 k 小于 50 时，有较高的错误率，当 k 大于 50 后，基本能预测准确。

注：第2题的代码详细见`Exercise 2.ipynb`.