数据挖掘第二次作业

exercise1

训练该线性回归模型的参数个数为8个，分别为迭代次数，用来训练的输入矩阵和标准的输出矩阵，初始的用于构建线性回归模型的参数列表，学习率，样本个数，用于测试的输入矩阵和标准的输出矩阵。

使用的方法为梯度下降法，算法的实现为：

1. 设线性方程为：$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
2. 为了得到目标线性方程，只需要确定公示中的参数θ值即可，这里我们要使用一个损失函数即：

   $J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$

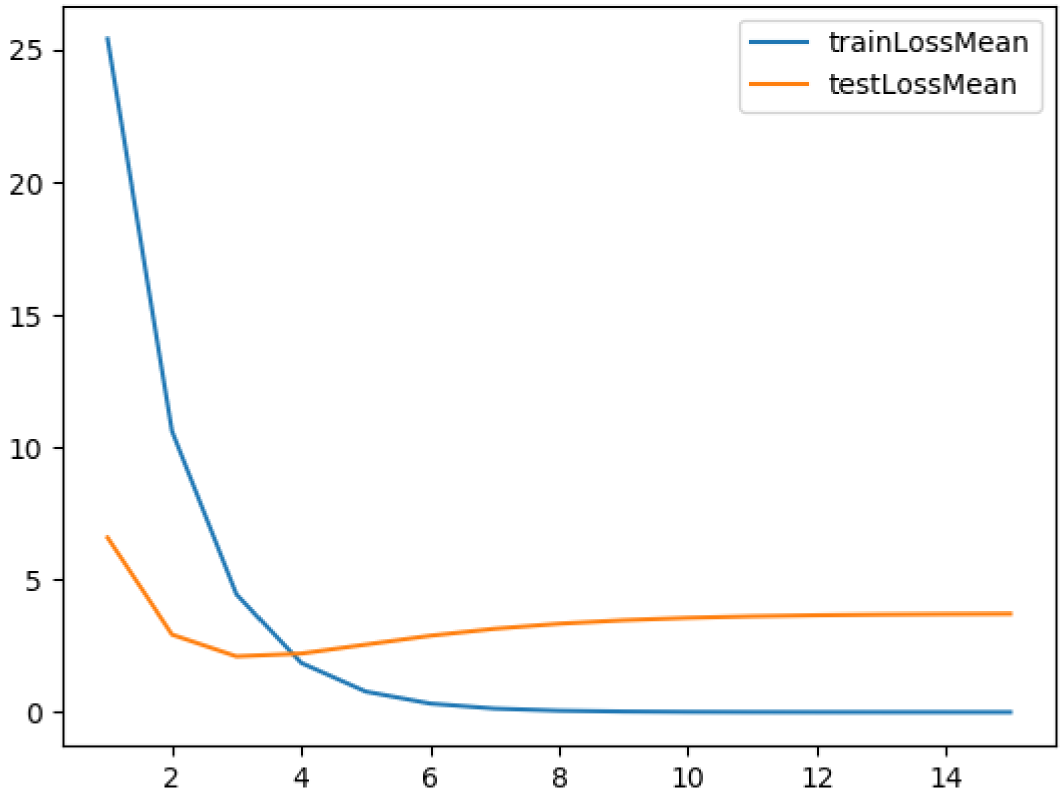
   通过迭代，调整参数θ 值来使$J(\theta)$ 值最小。
3. 这里我们批量梯度下降法，参数的迭代公式为：

   $\theta_j = \theta_j + \alpha\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$

代码实现如下：

```
def GradientDescent(maxiter,x,y,theta,alpha, m, testX, testY):
    TrainLoss = []
    TestLoss = []
    xTrains = x.transpose()
    count = 1
    for i in range(0,maxiter):
        hypothesis = np.dot(x,theta)
        loss = (hypothesis-y)
        gradient = np.dot(xTrains,loss)/m
        theta = theta - alpha * gradient
        if(count == 100000):
            TrainLoss.append(1.0/2*m*np.sum(np.square(np.dot(x,np.transpose(theta))-y)))
            TestLoss.append(1.0/2*m*np.sum(np.square(np.dot(testX,np.transpose(theta))-
testY)))
            count = 0
        count += 1
    return theta, TrainLoss, TestLoss
```

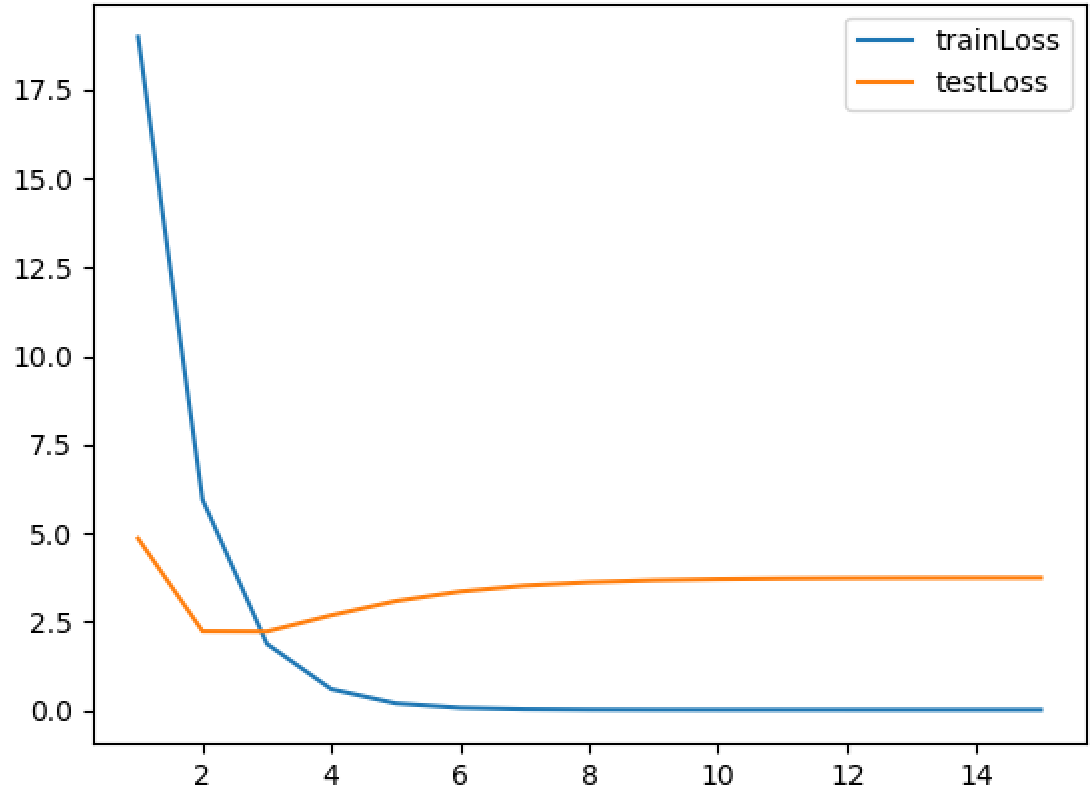
一共迭代1500000步，每100000步计算测试集和训练集的损失函数，画出的曲线图如下：



可以看出，随着迭代步数的增加，训练集的损失函数逐渐减小，到大约800000步左右的时候趋于稳定，不再下降。对于测试集而言，损失函数先下降然后有一定的提升，这说明随着迭代步数的增加，每次迭代的参数对于测试集而言不是最优参数。对这一现象，我分析是训练的样本太少，导致训练出来的参数容错性比较差。而这一容错性我觉得无法通过增加迭代次数来进行改进，而是应该增加训练集中训练数据的数量，使该多元线性回归模型的容错性提升。但无论是对于训练集还是测试集，最后的损失函数都会趋于稳定，这说明随着迭代次数的增加，参数最终收敛，使损失函数最终稳定。

exercise2

同样使用批量梯度下降法，通过改变学习率再一次画出结果图作分析：



在本次试验中，我发现如果不做数据标准化处理的话，会使计算的参数无法收敛，猜测原因是在梯度下降的过程中错过了最小值点，而当我对数据做标准化处理后就可以在1500000内得到收敛的参数。相较于选择学习率为0.00015的exercise1，当学习率选择为0.0002时，损失函数有所减小，收敛速度加快，在600000步左右收敛，但是整体趋势与学习率为0.00015时的损失函数变化保持一致。

exercise3

使用随机梯度下降法，随机梯度下降法的算法如下：

Loop {

For i=1 to m {

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \qquad \text{(for every j)}$$

}

}

即每读取一条样本，就迭代对ΘT进行更新，然后判断其是否收敛，若没收敛，则继续读取样本进行处理，如果所有样本都读取完毕了，则循环重新从头开始读取样本进行处理。

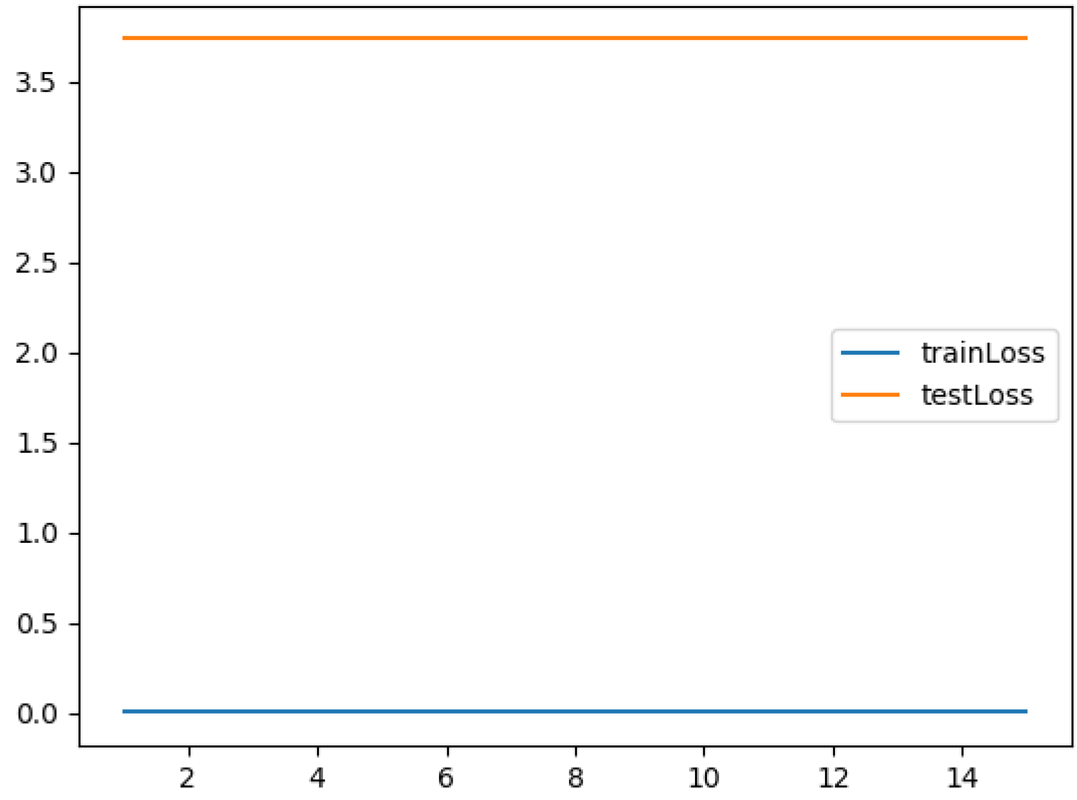
这样迭代一次的算法复杂度为O(n)。对于大数据集，很有可能只需读取一小部分数据，函数J(Θ)就收敛了。比如样本集数据量为100万，有可能读取几千条或几万条时，函数就达到了收敛值。所以当数据量很大时，更倾向于选择随机梯度下降算法。

代码实现如下：

```python
def StochGradientDescent(maxiter,x,y,theta,alpha, m, testX, testY):
    TrainLoss = []
    TestLoss = []
    xTrains = x.transpose()
    count = 1
    for i in range(0,maxiter):
        for j in range(0, m):
            loss = theta.dot(xTrains[:, j])-y[j]
            theta = theta - alpha*loss*x[j, :]

        if(count == 100000):
            TrainLoss.append(1.0/2*m*np.sum(np.square(np.dot(x,np.transpose(theta))-y)))
            TestLoss.append(1.0/2*m*np.sum(np.square(np.dot(testX,np.transpose(theta))-
testY)))
            count = 0
        count += 1
    return theta, TrainLoss, TestLoss
```

使用随机梯度下降法的曲线图如下：



在本次试验中我是用的总共迭代次数为1500000次，其中每100000次计算损失函数，结果如上图所示，可以看到当使用随机梯度下降法的时候，收敛速度更快了，而且损失函数更小。相对于实验一的收敛速度更快了。而且无论是测试数据的损失函数还是训练数据的损失函数都更小了。求出的参数结果如下：