**Homework 2: Multivariate Linear Regression**

Student ID:16340305 Student Name: 郑先淇

Lectured by 梁上松, Sun Yat-sen University

In this homework, you will investigate multivariate linear regression using Gradient Descent and Stochastic Gradient Descent. You will also examine the relationship between the cost function, the convergence of gradient descent, overfitting problem, and the learning rate.

Download the file “dataForTraining.txt” in the attached files called “Homework 2”. This is a training dataset of apartment prices in Haizhu District, Guangzhou, Guangdong, China, where there are 50 training instances, one line per one instance, formatted in three columns separated with each other by a whitespace. The data in the first and the second columns are sizes of the apartments in square meters and the distances to the Double-Duck-Mountain Vocational Technical College in kilo-meters, respectively, while the data in the third are the corresponding prices in billion RMB. Please build a multivariate linear regression model with the training instances by script in any programming languages to predict the prices of the apartments. For evaluation purpose, please also download the file “dataForTesting.txt” (the same format as that in the file of training data) in the same folder.

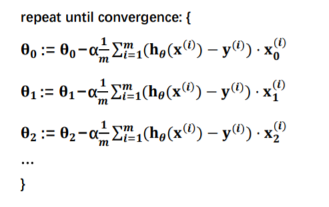
Exercise 1: How many parameters do you use to tune this linear regression model? Please use Gradient Descent to obtain the optimal parameters(最优化参数). Before you train the model, please set the number of iterations to be 1500000, the learning rate to 0.00015, the initial values of all the parameters to 0.0. During training, at every 100000 iterations, i.e., 100000 , 200000,…, 1500000, report the current training error and the testing error in a figure (you can draw it by hands or by any software). What can you find in the plots? Please analyze the plots.

Solution:

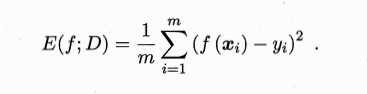
在这个模型中，我使用的假设函数为

h(x) = θ0 + θ1x1 + θ2x2

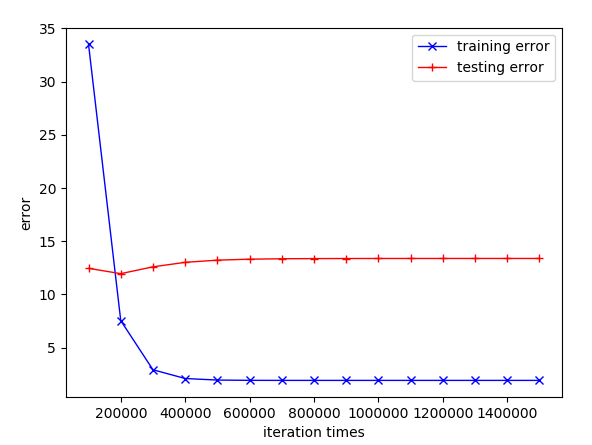
易得模型参数为3个。使用梯度下降的方法进行训练，对θ0、θ1、θ2进行迭代，迭代规则如下：

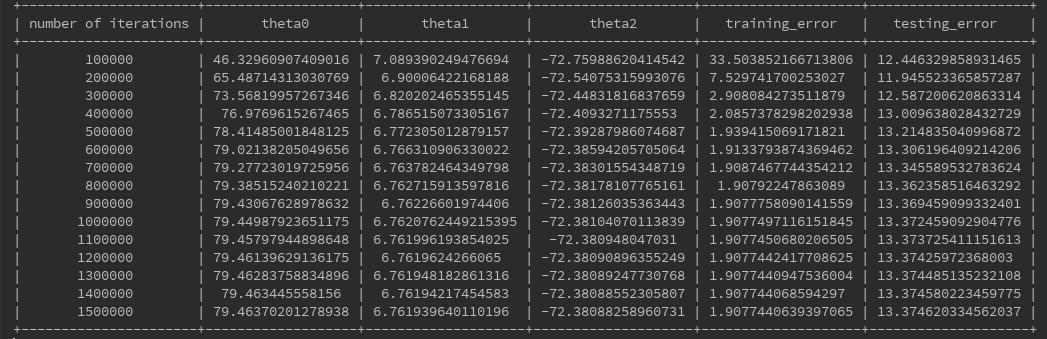


误差计算公式如下：



迭代结果如下：





源代码如下：

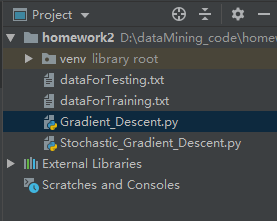
1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **from** prettytable **import** PrettyTable

6. **def** list\_to\_matrix(filename):  # 这个函数将txt文件中的数据读取出来，存进一个matrix中
7. np.set\_printoptions(suppress=True)  # 使输出不为科学计数法形式
8. fr = open(filename)
9. arrayOLines = fr.readlines()
10. numberOFLines = len(arrayOLines)  # 获取此list的行数
11. returnMat = np.zeros((numberOFLines, 3))  # 返回的矩阵是numberOFLines \* 3 的
12. index = 0
13. **for** line **in** arrayOLines:
14. line = line.strip()  # 移除每一行结尾的最后符号，即'\n'符；
15. listFromLine = line.split(' ')   # 移除每一行中出现的' '符；
16. returnMat[index, :] = listFromLine[0:3]
17. index += 1
18. **return** returnMat

21. **def** gradient\_descent():
22. alpha = 0.00015  # 设定学习率
23. theta0 = 0.0  # 初始化参数1为0.0
24. theta1 = 0.0  # 初始化参数2为0.0
25. theta2 = 0.0  # 初始化参数3位0.0
26. Mat = list\_to\_matrix('dataForTraining.txt')  # 读取txt训练集，存放在matrix中
27. testing\_mat = list\_to\_matrix('dataForTesting.txt')   # 读取txt测试集，存放在matrix中
29. result\_table = PrettyTable(["number of iterations", "theta0",
30. "theta1", "theta2", "training\_error", "testing\_error"])
31. training\_error\_arr = []
32. iter\_times\_arr = []
33. testing\_error\_arr = []
34. m = len(Mat)
35. n = len(testing\_mat)
36. count = 1
37. **for** j **in** range(1500000):   # 迭代1500000次
38. sum\_0 = 0
39. sum\_1 = 0
40. sum\_2 = 0
41. **for** i **in** range(m):   # 根据迭代规则更新参数
42. hx = theta0 + theta1\*Mat[i][0] + theta2\*Mat[i][1]
43. diff = Mat[i][2] - hx  #标记1：这里的正负号反了
44. sum\_0 += diff \* 1
45. sum\_1 += diff \* Mat[i][0]   # 这个数值有可能会特别大
46. sum\_2 += diff \* Mat[i][1]
47. theta0 = theta0 + alpha \* (1/m) \* sum\_0  #根据迭代规则更新theta0，注意由于标记1处的正负号反了，这里应用加号而不是减号
48. theta1 = theta1 + alpha \* (1/m) \* sum\_1
49. theta2 = theta2 + alpha \* (1/m) \* sum\_2
51. **if** j % 99999 == 0 **and** j >= 99999:   # 当迭代次数为100000的整数倍时，计算训练误差和测试误差
52. iter\_times\_arr.append(j+count)
53. temp = 0
54. testing\_temp = 0
55. **for** i **in** range(m):  #计算训练误差
56. Hx = theta0 + theta1\*Mat[i][0] + theta2\*Mat[i][1]
57. temp += np.square(Hx-Mat[i][2])
58. training\_error = 0.5 \* (1/m) \* temp
59. training\_error\_arr.append(training\_error)
61. **for** t **in** range(n):  #计算测试误差
62. Hx = theta0 + theta1\*testing\_mat[t][0] + theta2\*testing\_mat[t][1]
63. testing\_temp += np.square(Hx-testing\_mat[t][2])
64. testing\_error = 0.5 \* (1/m) \* testing\_temp
65. testing\_error\_arr.append(testing\_error)
67. result\_table.add\_row([j+count, theta0, theta1, theta2, training\_error, testing\_error])
68. count += 1
70. **print**(result\_table)
71. plt.figure()
72. plt.plot(iter\_times\_arr, training\_error\_arr, 'x-', c='b', linewidth=1, label="training error")
73. plt.plot(iter\_times\_arr, testing\_error\_arr, '+-', c='r', linewidth=1, label="testing error")
74. plt.xlabel("iteration times")  # X轴标签
75. plt.ylabel("error")   # Y轴标签
76. plt.legend()    # 显示图例
77. plt.show()

80. gradient\_descent()

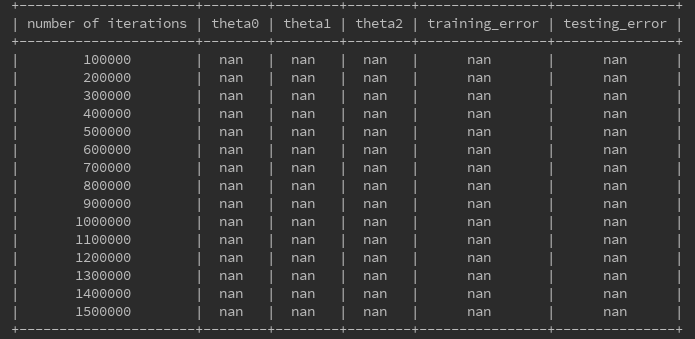
文件结构如下（py文件和txt文件在同一个文件夹下）：



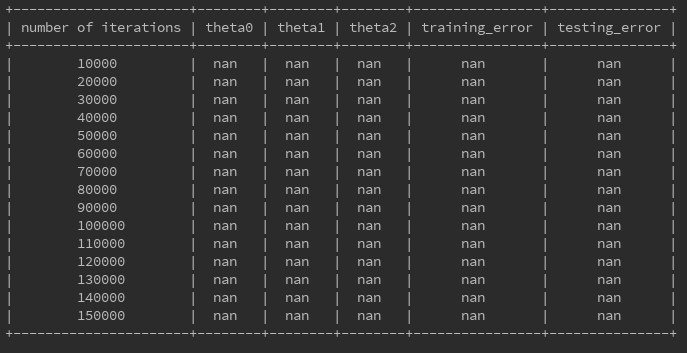
Exercise 2: Now, you change the learning rate to a number of different values, for instance, to 0.0002 (you may also change the number of iterations as well) and then train the model again. What can you find? Please conclude your findings.

Solution：

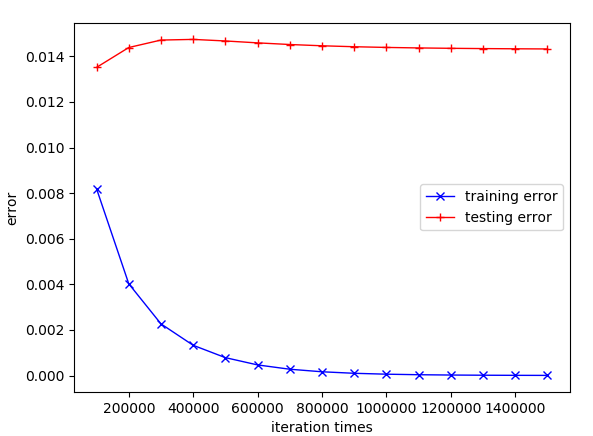
首先尝试只改变学习率为0.0002，不改变迭代次数，结果如下：

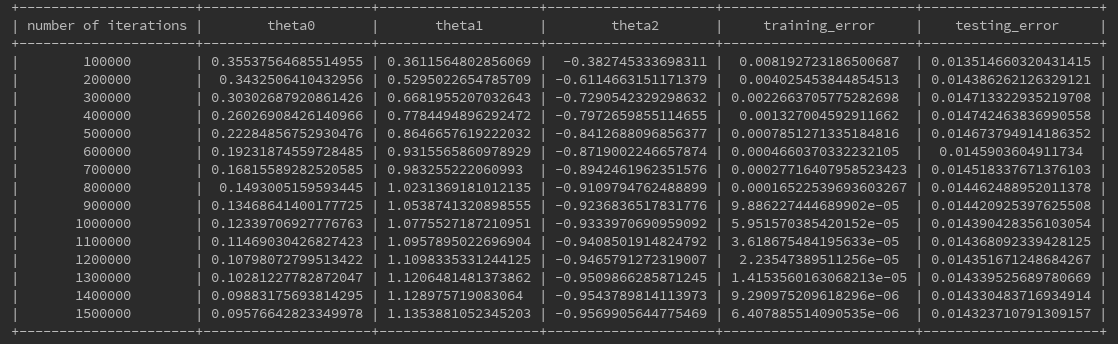


可以看到结果为nan。nan为无法表示(涉及无穷)，这里我们猜测是因为梯度下降没有收敛；尝试将迭代次数缩小10倍，结果如下：



可以看到结果仍然是nan。这个时候我开始怀疑不收敛的原因是因为初始数据不够好，所以采取一种措施：数据预处理，将给出的数据归一化，使其分布在0-1范围内，结果如下：





可以看到此时梯度下降法在训练集中很明显是收敛的，训练误差不断减少。在测试集中测试误差保持一定的数值变化不大(差不多稳定在0.014左右)，此种误差我们可认为是比较精确的，因此在对数据进行归一化处理之后，梯度下降法同样适用于学习率为0.0002的情况。

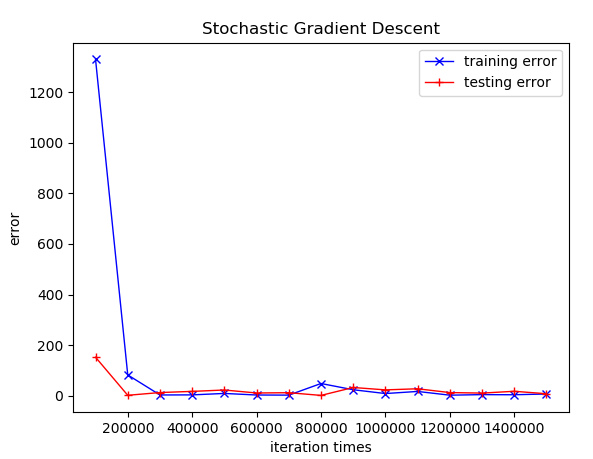
归一化处理的代码如下（其他代码与exercise 1相同）：

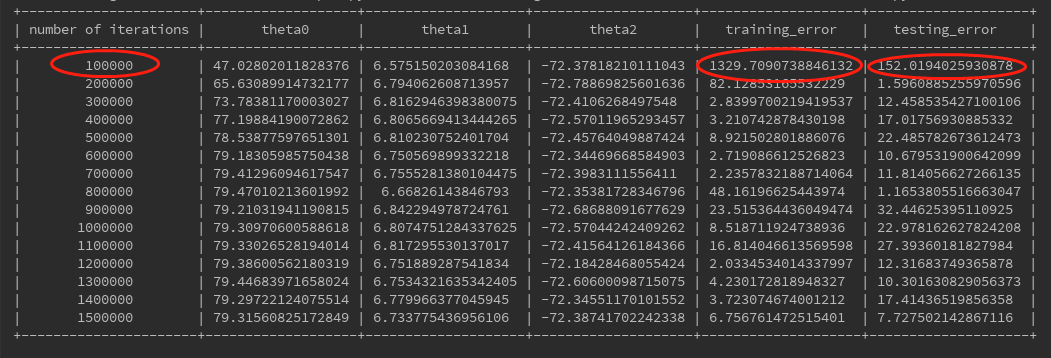


Exercise 3: Now, we turn to use other optimization methods to get the optimal parameters. Can you use Stochastic Gradient Descent to get the optimal parameters? Plots the training error and the testing error at each K-step iterations (the size of K is set by yourself). Can you analyze the plots and make comparisons to those findings in Exercise 1?

Solution：

随机梯度下降法的大体步骤和梯度下降法其实是一样的，唯一不同的地方是随机梯度下降法在更新θ的时候，并不使用所有样本的数据进行计算，而是随机选取其中一个样本的数据来更新θ。使用随机梯度下降法的结果如下（这里使用的迭代次数仍是1500000次，每隔100000次迭代计算一次误差）：





可以看到随机梯度下降法最后也获得了和梯度下降法几乎一样的结果，但是有一点值得注意的是，观察上图中使用随机梯度下降法迭代100000次之后的结果，训练误差为1329，测试误差为152；比起使用梯度下降法迭代100000次时误差要大得多，这是因为随机梯度下降每次更新参数用的都是一个随机样本的值，存在一定的随机性，因而可能其收敛速度不像梯度下降法那般快，但是当迭代次数足够多时，随机梯度下降法一般都能够获得和梯度下降法一样好的结果，这在这个例子中就体现地极为明显。并且，使用随机梯度下降法的计算量仅为梯度下降法的1/m，大大减少了计算量，因此随机梯度下降法是一个很有用的方法。

源代码如下：

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
3. **from** prettytable **import** PrettyTable
4. **import** random

7. **def** list\_to\_matrix(filename):  # 这个函数将txt文件中的数据读取出来，存进一个matrix中
8. np.set\_printoptions(suppress=True)  # 使输出不为科学计数法形式
9. fr = open(filename)
10. arrayOLines = fr.readlines()
11. numberOFLines = len(arrayOLines)  # 获取此list的行数
12. returnMat = np.zeros((numberOFLines, 3))  # 返回的矩阵是numberOFLines \* 3 的
13. index = 0
14. **for** line **in** arrayOLines:
15. line = line.strip()  # 移除每一行结尾的最后符号，即'\n'符；
16. listFromLine = line.split(' ')  # 移除每一行中出现的' '符；
17. returnMat[index, :] = listFromLine[0:3]
18. index += 1
19. **return** returnMat

22. **def** gradient\_descent():
23. alpha = 0.00015  # 设定学习率
24. theta0 = 0.0  # 初始化参数1为0.0
25. theta1 = 0.0  # 初始化参数2为0.0
26. theta2 = 0.0  # 初始化参数3位0.0
27. Mat = list\_to\_matrix('dataForTraining.txt')  # 读取txt训练集，存放在matrix中
28. testing\_mat = list\_to\_matrix('dataForTesting.txt')  # 读取txt测试集，存放在matrix中
30. result\_table = PrettyTable(["number of iterations", "theta0",
31. "theta1", "theta2", "training\_error", "testing\_error"])
32. training\_error\_arr = []
33. iter\_times\_arr = []
34. testing\_error\_arr = []
35. m = len(Mat)
36. n = len(testing\_mat)
37. count = 1
38. **for** j **in** range(1500000):  # 迭代1500000次
39. index = random.randint(0, m-1)  # 随机选取一个样本进行梯度下降
40. hx = theta0 + theta1 \* Mat[index][0] + theta2 \* Mat[index][1]
41. theta0 = theta0 - alpha \* (hx - Mat[index][2]) \* 1  # 标记1的符号是反的所以这里的减要变成加
42. theta1 = theta1 - alpha \* (hx - Mat[index][2]) \* Mat[index][0]
43. theta2 = theta2 - alpha \* (hx - Mat[index][2]) \* Mat[index][1]
45. **if** j % 99999 == 0 **and** j >= 99999:  # 当迭代次数为100000的整数倍时，计算训练误差和测试误差
46. iter\_times\_arr.append(j + count)
47. temp = 0
48. testing\_temp = 0
49. **for** i **in** range(m):
50. Hx = theta0 + theta1 \* Mat[i][0] + theta2 \* Mat[i][1]
51. temp += np.square(Hx - Mat[i][2])
52. training\_error = 0.5 \* (1 / m) \* temp
53. training\_error\_arr.append(training\_error)
55. **for** t **in** range(n):
56. Hx = theta0 + theta1 \* testing\_mat[t][0] + theta2 \* testing\_mat[t][1]
57. testing\_temp += np.square(Hx - testing\_mat[t][2])
58. testing\_error = 0.5 \* (1 / m) \* testing\_temp
59. testing\_error\_arr.append(testing\_error)
61. result\_table.add\_row([j + count, theta0, theta1, theta2, training\_error, testing\_error])
62. count += 1
64. **print**(result\_table)
65. plt.figure()
66. plt.plot(iter\_times\_arr, training\_error\_arr, 'x-', c='b', linewidth=1, label="training error")
67. plt.plot(iter\_times\_arr, testing\_error\_arr, '+-', c='r', linewidth=1, label="testing error")
68. plt.xlabel("iteration times")  # X轴标签
69. plt.ylabel("error")  # Y轴标签
70. plt.legend()  # 显示图例
71. plt.title("Stochastic Gradient Descent")
72. plt.show()

75. gradient\_descent()