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

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

易得模型参数为 3 个。使用梯度下降的方法进行训练，对 θ_0、θ_1、θ_2 进行迭代，迭代规则如下：

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$$

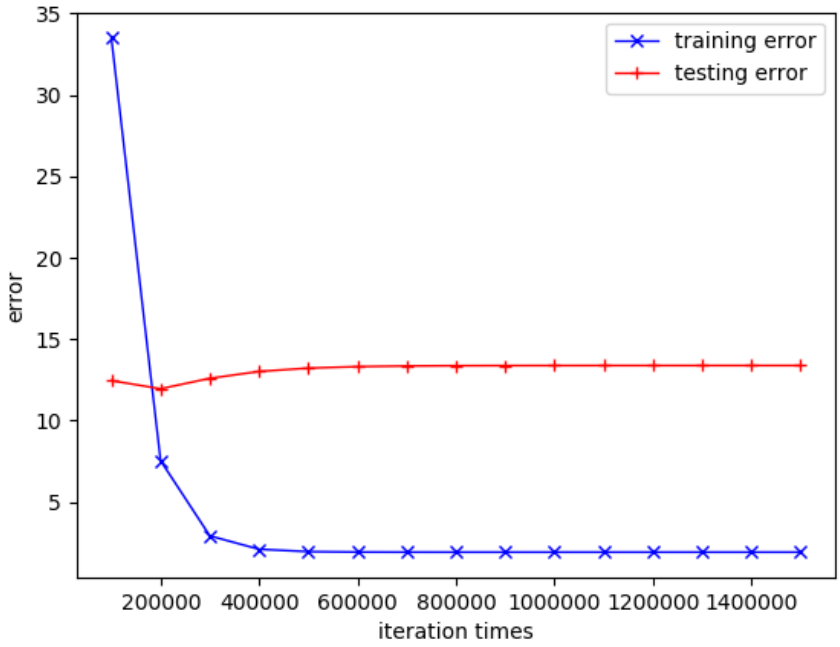
...

}

误差计算公式如下：

$$E(f;D) = \frac{1}{m} \sum_{i=1}^{m} (f(\boldsymbol{x}_i) - y_i)^2 \ .$$

迭代结果如下：



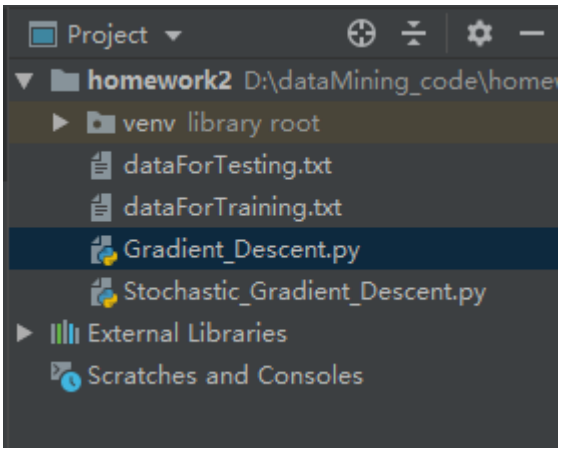
number of iterations	theta0	theta1	theta2	training_error	testing_error
100000	46.32960907409016	7.089390249476694	-72.75988620414542	33.503852166713806	12.446329858931465
200000	65.48714313030769	6.90006422168188	-72.54075315993076	7.529741700253027	11.945523365857287
300000	73.56819957267346	6.820202465355145	-72.44831816837659	2.908084273511879	12.587200620863314
400000	76.9769615267465	6.786515073305167	-72.4093271175553	2.0857378298202938	13.009638028432729
500000	78.41485001848125	6.772305012879157	-72.39287986074687	1.939415069171821	13.214835040996872
600000	79.02138205049656	6.766310906330022	-72.38594205705064	1.9133793874369462	13.306196409214206
700000	79.27723019725956	6.763782464349798	-72.38301554348719	1.9087467744354212	13.345589532783624
800000	79.38515240210221	6.762715913597816	-72.38178107765161	1.90792247863089	13.362358516463292
900000	79.43067628978632	6.76226601974406	-72.38126035363443	1.9077758090141559	13.369459099332401
1000000	79.44987923651175	6.7620762449215395	-72.38104070113839	1.9077497116151845	13.372459092904776
1100000	79.45797944898648	6.761996193854025	-72.380948047031	1.9077450680206505	13.373725411151613
1200000	79.46139629136175	6.7619624266065	-72.38090896355249	1.9077442417708625	13.37425972368003
1300000	79.46283758834896	6.761948182861316	-72.38089247730768	1.9077440947536004	13.374485135232108
1400000	79.463445558156	6.76194217454583	-72.38088552305807	1.907744068594297	13.374580223459775
1500000	79.46370201278938	6.761939640110196	-72.38088258960731	1.9077440639397065	13.374620334562037

源代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable


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
    np.set_printoptions(suppress=True)  # 使输出不为科学计数法形式
    fr = open(filename)
    arrayOLines = fr.readlines()
    numberOFLines = len(arrayOLines)  # 获取此 list 的行数
    returnMat = np.zeros((numberOFLines, 3))  # 返回的矩阵是 numberOFLines * 3 的
    index = 0
    for line in arrayOLines:
        line = line.strip()  # 移除每一行结尾的最后符号，即'\n'符；
        listFromLine = line.split(' ')   # 移除每一行中出现的' '符；
        returnMat[index, :] = listFromLine[0:3]
        index += 1
    return returnMat


def gradient_descent():
    alpha = 0.00015  # 设定学习率
    theta0 = 0.0  # 初始化参数 1 为 0.0
    theta1 = 0.0  # 初始化参数 2 为 0.0
    theta2 = 0.0  # 初始化参数 3 位 0.0
    Mat = list_to_matrix('dataForTraining.txt')  # 读取 txt 训练集，存放在 matrix 中
    testing_mat = list_to_matrix('dataForTesting.txt')   # 读取 txt 测试集，存放在 matrix 中

    result_table = PrettyTable(["number of iterations", "theta0",
                                "theta1", "theta2", "training_error", "testing_error"])
    training_error_arr = []
    iter_times_arr = []
    testing_error_arr = []
    m = len(Mat)
    n = len(testing_mat)
    count = 1
    for j in range(1500000):   # 迭代 1500000 次
        sum_0 = 0
        sum_1 = 0
        sum_2 = 0
        for i in range(m):   # 根据迭代规则更新参数
            hx = theta0 + theta1*Mat[i][0] + theta2*Mat[i][1]
```

```
            diff = Mat[i][2] - hx  #标记 1：这里的正负号反了
            sum_0 += diff * 1
            sum_1 += diff * Mat[i][0]   # 这个数值有可能会特别大
            sum_2 += diff * Mat[i][1]
        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
        theta1 = theta1 + alpha * (1/m) * sum_1
        theta2 = theta2 + alpha * (1/m) * sum_2

        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
            iter_times_arr.append(j+count)
            temp = 0
            testing_temp = 0
            for i in range(m):  #计算训练误差
                Hx = theta0 + theta1*Mat[i][0] + theta2*Mat[i][1]
                temp += np.square(Hx-Mat[i][2])
            training_error = 0.5 * (1/m) * temp
            training_error_arr.append(training_error)

            for t in range(n):  #计算测试误差
                Hx = theta0 + theta1*testing_mat[t][0] + theta2*testing_mat[t][1]
                testing_temp += np.square(Hx-testing_mat[t][2])
            testing_error = 0.5 * (1/m) * testing_temp
            testing_error_arr.append(testing_error)

            result_table.add_row([j+count, theta0, theta1, theta2, training_error, testing_error])
            count += 1

    print(result_table)
    plt.figure()
    plt.plot(iter_times_arr, training_error_arr, 'x-', c='b', linewidth=1, label="training error")
    plt.plot(iter_times_arr, testing_error_arr, '+-', c='r', linewidth=1, label="testing error")
    plt.xlabel("iteration times")  # X 轴标签
    plt.ylabel("error")   # Y 轴标签
    plt.legend()    # 显示图例
    plt.show()


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：

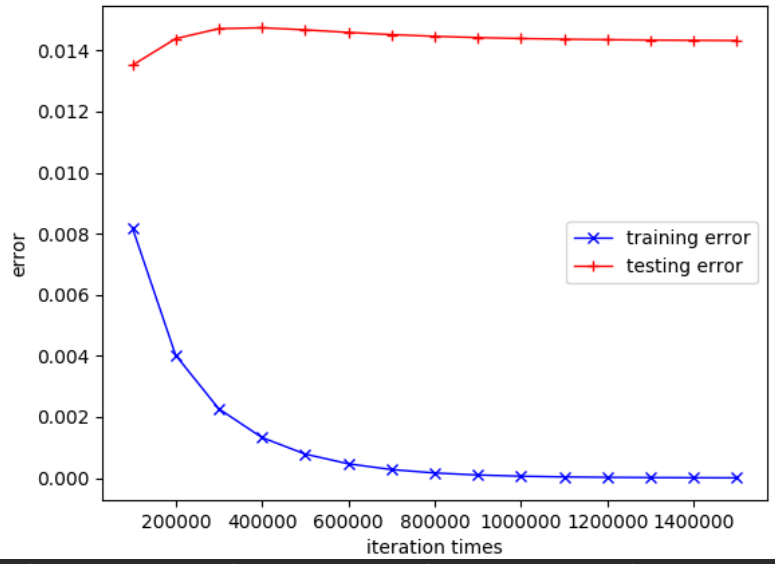
首先尝试只改变学习率为 0.0002，不改变迭代次数，结果如下：

number of iterations	theta0	theta1	theta2	training_error	testing_error
100000	nan	nan	nan	nan	nan
200000	nan	nan	nan	nan	nan
300000	nan	nan	nan	nan	nan
400000	nan	nan	nan	nan	nan
500000	nan	nan	nan	nan	nan
600000	nan	nan	nan	nan	nan
700000	nan	nan	nan	nan	nan
800000	nan	nan	nan	nan	nan
900000	nan	nan	nan	nan	nan
1000000	nan	nan	nan	nan	nan
1100000	nan	nan	nan	nan	nan
1200000	nan	nan	nan	nan	nan
1300000	nan	nan	nan	nan	nan
1400000	nan	nan	nan	nan	nan
1500000	nan	nan	nan	nan	nan

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：

number of iterations	theta0	theta1	theta2	training_error	testing_error
10000	nan	nan	nan	nan	nan
20000	nan	nan	nan	nan	nan
30000	nan	nan	nan	nan	nan
40000	nan	nan	nan	nan	nan
50000	nan	nan	nan	nan	nan
60000	nan	nan	nan	nan	nan
70000	nan	nan	nan	nan	nan
80000	nan	nan	nan	nan	nan
90000	nan	nan	nan	nan	nan
100000	nan	nan	nan	nan	nan
110000	nan	nan	nan	nan	nan
120000	nan	nan	nan	nan	nan
130000	nan	nan	nan	nan	nan
140000	nan	nan	nan	nan	nan
150000	nan	nan	nan	nan	nan

可以看到结果仍然是 nan。这个时候我采取一种新的措施：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：



number of iterations	theta0	theta1	theta2	training_error	testing_error
100000	0.35537564685514955	0.3611564802856069	-0.382745333698311	0.008192723186500687	0.013514660320431415
200000	0.3432506410432956	0.5295022654785709	-0.6114663151171379	0.004025453844854513	0.014386262126329121
300000	0.30302687920861426	0.6681955207032643	-0.7290542329298632	0.0022663705775282698	0.014713322935219708
400000	0.26026908426140966	0.7784494896292472	-0.7972659855114655	0.001327004592911662	0.014742463836990558
500000	0.22284856752930476	0.8646657619222032	-0.8412688096856377	0.0007851271335184816	0.014673794914186352
600000	0.19231874559728485	0.9315565860978929	-0.8719002246657874	0.0004660370332232105	0.014590360491734
700000	0.16815589282520585	0.983255222060993	-0.8942461962351576	0.00027716407958523423	0.014518337671376103
800000	0.1493005159593445	1.0231369181012135	-0.9109794762488899	0.00016522539693603267	0.014462488952011378
900000	0.13468641400177725	1.0538741320898555	-0.9236836517831776	9.886227444689902e-05	0.014420925397625508
1000000	0.12339706927776763	1.0775527187210951	-0.9333970690959092	5.951570385420152e-05	0.014390428356103054
1100000	0.11469030426827423	1.0957895022696904	-0.9408501914824792	3.618675484195633e-05	0.014368092339428125
1200000	0.10798072799513422	1.1098335331244125	-0.9465791272319007	2.23547389511256e-05	0.014351671248684267
1300000	0.10281227782872047	1.1206481481373862	-0.9509866285871245	1.4153560163068213e-05	0.014339525689780669
1400000	0.09883175693814295	1.128975719083064	-0.9543789814113973	9.290975209618296e-06	0.014330483716934914
1500000	0.09576642823349978	1.1353881052345203	-0.9569905644775469	6.407885514090535e-06	0.014323710791309157

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。

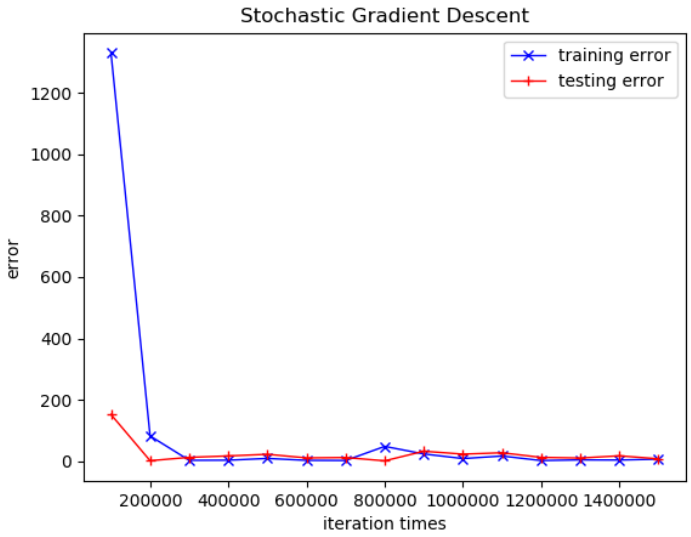
归一化处理的代码如下（其他代码与 exercise 1 相同）：

```
Mat = list_to_matrix('dataForTraining.txt')  # 读取 txt 训练集，存放在 matrix 中
testing_mat = list_to_matrix('dataForTesting.txt')   # 读取 txt 测试集，存放在 matrix 中
m = len(Mat)
n = len(testing_mat)
training_col_max_arr = np.max(Mat, axis=0)  # 找出训练集中每个 feature 的最大值
testing_col_max_arr = np.max(testing_mat, axis=0)  # 找出测试集中每个 feature 的最大值
print(testing_col_max_arr)
for t in range(m):    # 特征缩放，归一化
    Mat[t][0] = Mat[t][0] / training_col_max_arr[0]
    Mat[t][1] = Mat[t][1] / training_col_max_arr[1]
    Mat[t][2] = Mat[t][2] / training_col_max_arr[2]
for t in range(n):    # 特征缩放，归一化
    testing_mat[t][0] = testing_mat[t][0] / testing_col_max_arr[0]
    testing_mat[t][1] = testing_mat[t][1] / testing_col_max_arr[1]
    testing_mat[t][2] = testing_mat[t][2] / testing_col_max_arr[2]
```

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）：



number of iterations	theta0	theta1	theta2	training_error	testing_error
100000	47.02802011828376	6.575150203084168	-72.37818210111043	1329.7090738846132	152.0194025930878
200000	65.63089914732177	6.794062608713957	-72.78869825601636	82.12853165532229	1.5960885255970596
300000	73.78381170003027	6.8162946398380075	-72.4106268497548	2.8399700219419537	12.458535427100106
400000	77.19884190072862	6.8065669413444265	-72.57011965293457	3.210742878430198	17.01756930885332
500000	78.53877597651301	6.810230752401704	-72.45764049887424	8.921502801886076	22.485782673612473
600000	79.18305985750438	6.750569899332218	-72.34469668584903	2.719086612526823	10.679531900642099
700000	79.41296094617547	6.7555281380104475	-72.3983111556411	2.2357832188714064	11.814056627266135
800000	79.47010213601992	6.66826143846793	-72.35381728346796	48.16196625443974	1.1653805516663047
900000	79.21031941190815	6.842294978724761	-72.68688091677629	23.515364436049474	32.44625395110925
1000000	79.30970600588618	6.8074751284337625	-72.57044242409262	8.518711924738936	22.978162627824208
1100000	79.33026528194014	6.817295530137017	-72.41564126184366	16.814046613569598	27.39360181827984
1200000	79.38600562180319	6.751889287541834	-72.18428468055424	2.0334534014337997	12.31683749365878
1300000	79.44683971658024	6.7534321635342405	-72.60600098715075	4.230172818948327	10.301630829056373
1400000	79.29722124075514	6.779966377045945	-72.34551170101552	3.723074674001212	17.41436519856358
1500000	79.31560825172849	6.733775436956106	-72.38741702242338	6.756761472515401	7.727502142867116

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
5.
6.
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)
```

```
    arrayOLines = fr.readlines()
    numberOFLines = len(arrayOLines)  # 获取此 list 的行数
    returnMat = np.zeros((numberOFLines, 3))  # 返回的矩阵是 numberOFLines * 3 的
    index = 0
    for line in arrayOLines:
        line = line.strip()  # 移除每一行结尾的最后符号，即'\n'符；
        listFromLine = line.split(' ')  # 移除每一行中出现的' '符；
        returnMat[index, :] = listFromLine[0:3]
        index += 1
    return returnMat


def gradient_descent():
    alpha = 0.00015  # 设定学习率
    theta0 = 0.0  # 初始化参数 1 为 0.0
    theta1 = 0.0  # 初始化参数 2 为 0.0
    theta2 = 0.0  # 初始化参数 3 位 0.0
    Mat = list_to_matrix('dataForTraining.txt')  # 读取 txt 训练集，存放在 matrix 中
    testing_mat = list_to_matrix('dataForTesting.txt')  # 读取 txt 测试集，存放在
matrix 中

    result_table = PrettyTable(["number of iterations", "theta0",
                                "theta1", "theta2", "training_error", "testing_erro
r"])
    training_error_arr = []
    iter_times_arr = []
    testing_error_arr = []
    m = len(Mat)
    n = len(testing_mat)
    count = 1
    for j in range(1500000):  # 迭代 1500000 次
        index = random.randint(0, m-1)  # 随机选取一个样本进行梯度下降
        hx = theta0 + theta1 * Mat[index][0] + theta2 * Mat[index][1]
        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
        theta1 = theta1 - alpha * (hx - Mat[index][2]) * Mat[index][0]
        theta2 = theta2 - alpha * (hx - Mat[index][2]) * Mat[index][1]

        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
            iter_times_arr.append(j + count)
            temp = 0
            testing_temp = 0
            for i in range(m):
                Hx = theta0 + theta1 * Mat[i][0] + theta2 * Mat[i][1]
                temp += np.square(Hx - Mat[i][2])
            training_error = 0.5 * (1 / m) * temp
            training_error_arr.append(training_error)

            for t in range(n):
                Hx = theta0 + theta1 * testing_mat[t][0] + theta2 * testing_mat[t][
1]
                testing_temp += np.square(Hx - testing_mat[t][2])
            testing_error = 0.5 * (1 / m) * testing_temp
            testing_error_arr.append(testing_error)

            result_table.add_row([j + count, theta0, theta1, theta2, training_error
, testing_error])
            count += 1

    print(result_table)
    plt.figure()
    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()
73.
74.
75. gradient_descent()
```