Multivariate Linear Regression

In this homework, we will investigate multivariate linear regression using Gradient Descent and Stochastic Gradient Descent. We 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.

In [1]:

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

The following cell is going to processing the picture in this experiment.

In [2]:

```
# read train data
dataframe1 = pd.read_csv('dataForTraining.txt', sep=' ', header=None,
                         names=['train_size', 'train_distance', 'train_price'])
train_sizes = dataframe1[['train_size']].values/10
train_distances = dataframe1[['train_distance']].values
train_prices = dataframe1[['train_price']].values

# read test data
dataframe2 = pd.read_csv('dataForTesting.txt', sep=' ', header=None, names=['tes
t_size', 'test_distance', 'test_price'])
test_sizes = dataframe2[['test_size']].values/10
test_distances = dataframe2[['test_distance']].values
test_prices = dataframe2[['test_price']].values
```

Visualization of the linear regression, including observation scatter and regression plat contour.

In [3]:

```
# show the regression
def show_regression(train_sizes, train_distances, train_prices, test_sizes, test
_distances, test_prices, w1, w2, bias):

    # scatter
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    plt.title('Apartment Prices Linear Regression',fontsize=12)
    ax.set_xlabel('Size/100m^2', fontsize=14)
    ax.set_ylabel('Distance/Km', fontsize=14)
    ax.set_zlabel('Price/*1000RMB', fontsize=14)
    ax.scatter3D(train_sizes*10, train_distances, train_prices, color='#0000ff')
    ax.scatter3D(test_sizes*10, test_distances, test_prices, color='#ff0000')

    # contour
    x,y = np.meshgrid(np.linspace(6.,16.,10),np.linspace(0.,12.5,10))
    predicts = w1*x+w2*y+bias*np.linspace(1.,1.,10)
    ax = plt.gca(projection='3d')
    ax.contour(x*10,y,predicts,100,colors='#bbbbbb')
    plt.show()
```

Class linear regression defines the LR method. In this class, `lr` , `w1` (w_size), `w2` (w_dist), `bias` , `n_epoch` , `k` , and `iter_num` are defined as private variables. Two methods `train_GD` and `train_SGD` train the model based on training set, while test set are used in `test` method.

In [4]:

```
class LinearRegression(object):


    # class initialization
    def __init__(self, lr, w1, w2, bias, n_epochs, k):
        self.lr = lr
        self.w1 = w1
        self.w2 = w2
        self.bias = bias
        self.n_epochs = n_epochs
        self.k = k
        self.iter_num = int(self.n_epochs/self.k)



    # train the weights using GD
    def train_GD(self):
        iter_errs = np.zeros((self.iter_num,1))
        iter_losses = np.zeros((self.iter_num,1))

        for epoch in range(self.n_epochs):
            iteration = int(epoch/self.k)
            train_predicts = np.zeros((50,1))
            train_errors = np.zeros((50,1))

            train_predicts = self.w1*train_sizes+self.w2*train_distances+self.bi
as*np.ones((50,1))
            train_errors = train_prices-train_predicts

            self.w1 += self.lr*sum(np.multiply(train_errors, train_sizes))/50
            self.w2 += self.lr*sum(np.multiply(train_errors, train_distances))/5
0
            self.bias += self.lr*sum(train_errors)/50

            # evaluate the performance
            if epoch%self.k == self.k-1:
                iter_errs[iteration] = sum(abs(train_errors)/50)
                iter_losses[iteration] = sum(np.power(train_errors,2)/100)
                print("-------------------------------------------- Iteration "+
str(iteration)+" --------------------------------------------\n")
                print("w_sizes, w_distances, w_bias = "+str(self.w1/10)+", "+str
(self.w2)+", "+str(self.bias)+" correspondingly.\n")
                print("The average train error is "+str(iter_errs[iteration])+",
the average train loss is "+str(iter_losses[iteration])+".\n")

        # plot error curve
        print("-------------------------------------------- Training Curve -----
---------------------------------------\n")
        plt.title('Error and Loss Curve',fontsize=12)
        plt.plot(range(self.iter_num), iter_errs, label='errors', c='b')
        plt.plot(range(self.iter_num), iter_losses, label='losses', c='g')
        plt.axis()
        plt.legend()
        plt.show()



    # train the weights using SGD
    def train_SGD(self):
```

```
        iter_errs = np.zeros((self.iter_num,1))
        iter_losses = np.zeros((self.iter_num,1))

        for epoch in range(self.n_epochs):
            iteration = int(epoch/self.k)
            train_predicts = np.zeros((50,1))
            train_errors = np.zeros((50,1))

            for i in range(50):
                train_predicts[i] = self.w1*train_sizes[i]+self.w2*train_distanc
es[i]+self.bias
                train_errors[i] = train_prices[i]-train_predicts[i]
                self.w1 += self.lr*(train_errors[i])*train_sizes[i]
                self.w2 += self.lr*(train_errors[i])*train_distances[i]
                self.bias += self.lr*(train_errors[i])

            # evaluate the performance
            if epoch%self.k == self.k-1:
                iter_errs[iteration] = sum(abs(train_errors)/50)
                iter_losses[iteration] = sum(np.power(train_errors,2)/100)
                print("-------------------------------------------- Iteration "+
str(iteration)+" --------------------------------------------\n")
                print("w_sizes, w_distances, w_bias = "+str(self.w1/10)+", "+str
(self.w2)+", "+str(self.bias)+" correspondingly.\n")
                print("The average train error is "+str(iter_errs[iteration])+",
the average train loss is "+str(iter_losses[iteration])+".\n")

        # plot error curve
        print("-------------------------------------------- Training Curve -----
---------------------------------------\n")
        plt.title('Error and Loss Curve',fontsize=12)
        plt.plot(range(self.iter_num), iter_errs, label='errors', c='b')
        plt.plot(range(self.iter_num), iter_losses, label='losses', c='g')
        plt.axis()
        plt.legend()
        plt.show()


    # test our model
    def test(self):
        test_predicts = np.zeros((10,1))
        test_errors = np.zeros((10,1))
        for i in range(10):
            test_predicts[i] = self.w1*test_sizes[i]+self.w2*test_distances[i]+s
elf.bias
            test_errors[i] = test_prices[i]-test_predicts[i]
        print("\n-------------------------------------------- Test Results -----
---------------------------------------\n")
        print("The average test error is : "+str(sum(abs(test_errors))/10)+".\n"
)
        print("The average test loss is : "+str(sum(np.power(test_errors,2))/20)
+".\n")
```

Exercise 1

How many parameters do you use to tune this linear regression model? Please use **Gradient Descent** to obtain the optimal parameters. 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.

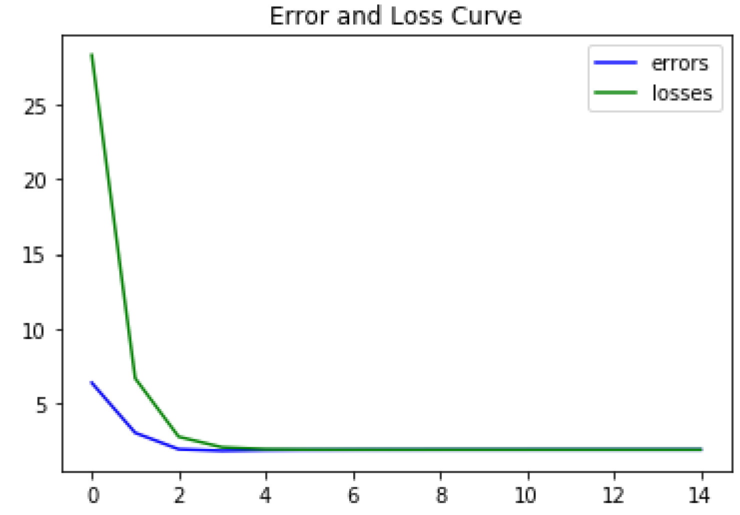
In [5]:

```
# first train, lr = 0.00015, GD
myLR1 = LinearRegression(0.00015, 0., 0., 0., 1500000, 100000)
myLR1.train_GD()
myLR1.test()
show_regression(train_sizes, train_distances, train_prices, test_sizes, test_dis
tances, test_prices, myLR1.w1, myLR1.w2, myLR1.bias)
```

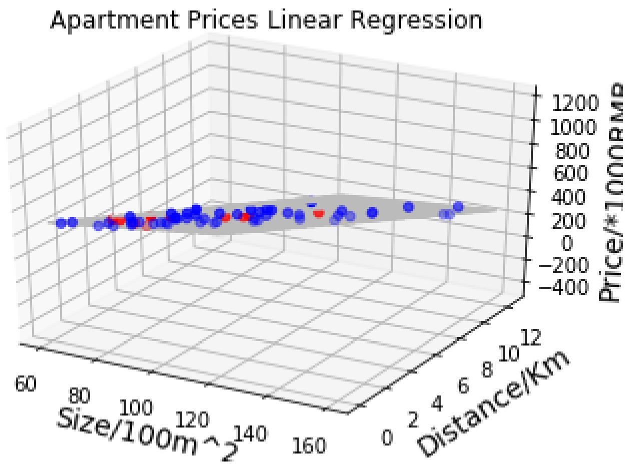
```
------------------------------------------- Iteration 0 -----------
--------------------------------

w_sizes, w_distances, w_bias = [7.06203048], [-72.74101927], [49.183
84041] correspondingly.

The average train error is [6.38585383], the average train loss is
[28.29630539].

------------------------------------------- Iteration 1 -----------
--------------------------------

w_sizes, w_distances, w_bias = [6.88957561], [-72.53405757], [66.584
93721] correspondingly.

The average train error is [3.03439445], the average train loss is
[6.68153744].

------------------------------------------- Iteration 2 -----------
--------------------------------

w_sizes, w_distances, w_bias = [6.81622573], [-72.44603096], [73.986
11054] correspondingly.

The average train error is [1.93535896], the average train loss is
[2.77134185].

------------------------------------------- Iteration 3 -----------
--------------------------------

w_sizes, w_distances, w_bias = [6.78502797], [-72.40859079], [77.134
03663] correspondingly.

The average train error is [1.82805336], the average train loss is
[2.06397226].

------------------------------------------- Iteration 4 -----------
--------------------------------

w_sizes, w_distances, w_bias = [6.77175869], [-72.39266643], [78.472
93767] correspondingly.

The average train error is [1.87234897], the average train loss is
[1.93600635].

------------------------------------------- Iteration 5 -----------
--------------------------------

w_sizes, w_distances, w_bias = [6.76611489], [-72.38589336], [79.042
40974] correspondingly.

The average train error is [1.89118914], the average train loss is
[1.91285682].

------------------------------------------- Iteration 6 -----------
--------------------------------

w_sizes, w_distances, w_bias = [6.76371443], [-72.38301258], [79.284
62214] correspondingly.

The average train error is [1.89920238], the average train loss is
```

```
[1.90866898].

-------------------------------------------- Iteration 7 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76269345], [-72.3817873], [79.3876
4186] correspondingly.

The average train error is [1.90261064], the average train loss is
[1.90791138].

-------------------------------------------- Iteration 8 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76225919], [-72.38126616], [79.431
45902] correspondingly.

The average train error is [1.90406027], the average train loss is
[1.90777433].

-------------------------------------------- Iteration 9 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76207449], [-72.3810445], [79.4500
9569] correspondingly.

The average train error is [1.90467684], the average train loss is
[1.90774954].

-------------------------------------------- Iteration 10 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76199593], [-72.38095023], [79.458
02238] correspondingly.

The average train error is [1.90493908], the average train loss is
[1.90774505].

-------------------------------------------- Iteration 11 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76196252], [-72.38091013], [79.461
39383] correspondingly.

The average train error is [1.90505062], the average train loss is
[1.90774424].

-------------------------------------------- Iteration 12 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76194831], [-72.38089307], [79.462
8278] correspondingly.

The average train error is [1.90509806], the average train loss is
[1.9077441].

-------------------------------------------- Iteration 13 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76194226], [-72.38088582], [79.463
43771] correspondingly.
```

```
The average train error is [1.90511824], the average train loss is
[1.90774407].

--------------------------------------------- Iteration 14 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76193969], [-72.38088273], [79.463
69712] correspondingly.

The average train error is [1.90512682], the average train loss is
[1.90774406].

--------------------------------------------- Training Curve --------
------------------------------------
```




```
--------------------------------------------- Test Results ----------
----------------------------------

The average test error is : [11.10900937].

The average test loss is : [66.87310085].
```



Errors and losses gradually decrease with the iteration. And the result finally converged on (w_sizes, w_distances, w_bias) = (6.76, -72.38, 79.46). After testing, we find the results pretty accurate and only suffer an error of about 2w RMB.

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.

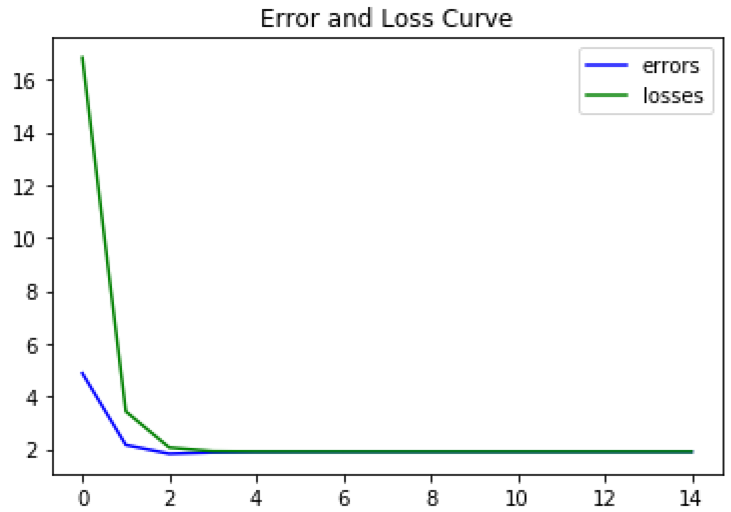
In [6]:

```
# second train, lr = 0.0002, GD
myLR2 = LinearRegression(0.0002, 0., 0., 0., 1500000, 100000)
myLR2.train_GD()
myLR2.test()
show_regression(train_sizes, train_distances, train_prices, test_sizes, test_dis
tances, test_prices, myLR2.w1, myLR2.w2, myLR2.bias)
```

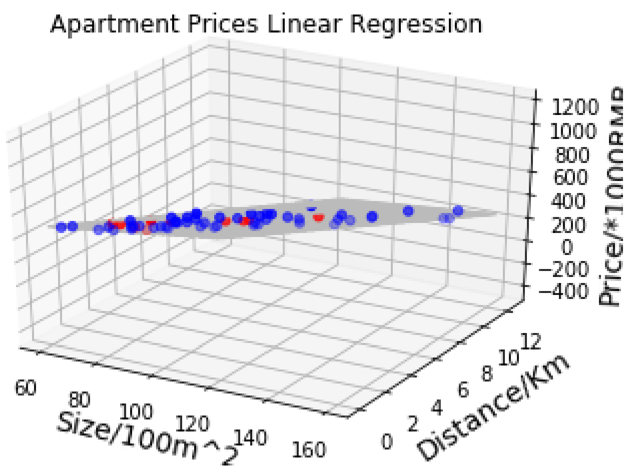
```
-------------------------------------------- Iteration 0 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.98761933], [-72.65171904], [56.692
09853] correspondingly.

The average train error is [4.8810339], the average train loss is [1
6.83223109].

-------------------------------------------- Iteration 1 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.83412506], [-72.46751181], [72.180
02651] correspondingly.

The average train error is [2.16244761], the average train loss is
[3.43470716].

-------------------------------------------- Iteration 2 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.78502786], [-72.40859065], [77.134
04798] correspondingly.

The average train error is [1.82805352], the average train loss is
[2.06397163].

-------------------------------------------- Iteration 3 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76932346], [-72.38974393], [78.718
65818] correspondingly.

The average train error is [1.88047821], the average train loss is
[1.92372811].

-------------------------------------------- Iteration 4 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.7643002], [-72.38371555], [79.2255
1701] correspondingly.

The average train error is [1.89724695], the average train loss is
[1.90937943].

-------------------------------------------- Iteration 5 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76269344], [-72.3817873], [79.3876
426] correspondingly.

The average train error is [1.90261066], the average train loss is
[1.90791138].

-------------------------------------------- Iteration 6 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76217949], [-72.38117052], [79.439
50064] correspondingly.

The average train error is [1.90432631], the average train loss is
```

```
[1.90776118].

-------------------------------------------- Iteration 7 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.7620151], [-72.38097323], [79.4560
8813] correspondingly.

The average train error is [1.90487509], the average train loss is
[1.90774581].

-------------------------------------------- Iteration 8 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76196252], [-72.38091013], [79.461
39387] correspondingly.

The average train error is [1.90505062], the average train loss is
[1.90774424].

-------------------------------------------- Iteration 9 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.7619457], [-72.38088994], [79.4630
9098] correspondingly.

The average train error is [1.90510677], the average train loss is
[1.90774408].

-------------------------------------------- Iteration 10 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76194032], [-72.38088349], [79.463
63382] correspondingly.

The average train error is [1.90512473], the average train loss is
[1.90774406].

-------------------------------------------- Iteration 11 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.7619386], [-72.38088142], [79.4638
0745] correspondingly.

The average train error is [1.90513047], the average train loss is
[1.90774406].

-------------------------------------------- Iteration 12 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76193805], [-72.38088076], [79.463
86299] correspondingly.

The average train error is [1.90513231], the average train loss is
[1.90774406].

-------------------------------------------- Iteration 13 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76193787], [-72.38088055], [79.463
88076] correspondingly.
```

```
The average train error is [1.9051329], the average train loss is
[1.90774406].

-------------------------------------------- Iteration 14 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76193782], [-72.38088048], [79.463
88644] correspondingly.

The average train error is [1.90513308], the average train loss is
[1.90774406].

-------------------------------------------- Training Curve --------
------------------------------------
```




```
-------------------------------------------- Test Results ----------
----------------------------------

The average test error is : [11.1090291].

The average test loss is : [66.87324595].
```



Errors and losses gradually decrease with the iteration. And the result finally converged on (w_sizes, w_distances, w_bias) = (6.76, -72.38, 79.46). After testing, we find the results pretty accurate and only suffer an error of about 2w RMB. **Same results as lr = 0.00015, but this model converges a little faster.**

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?

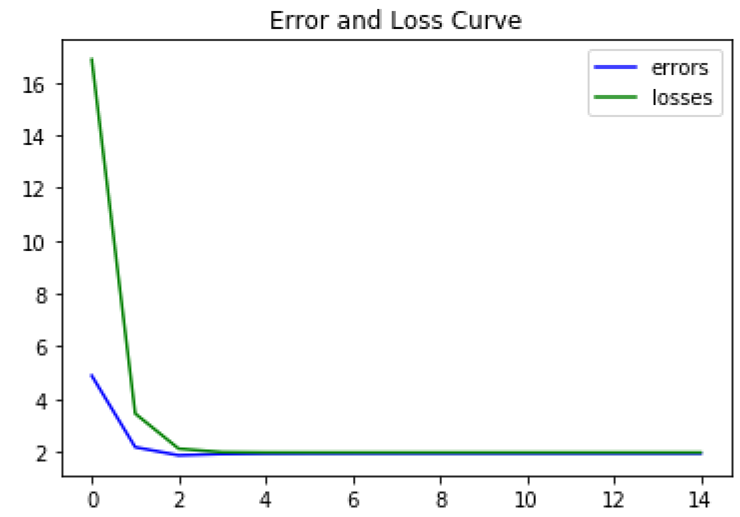
In [7]:

```
# third train, lr = 0.0002, SGD
myLR3 = LinearRegression(0.0002, 0., 0., 0., 30000, 2000)
myLR3.train_SGD()
myLR3.test()
show_regression(train_sizes, train_distances, train_prices, test_sizes, test_dis
tances, test_prices, myLR3.w1, myLR3.w2, myLR3.bias)
```

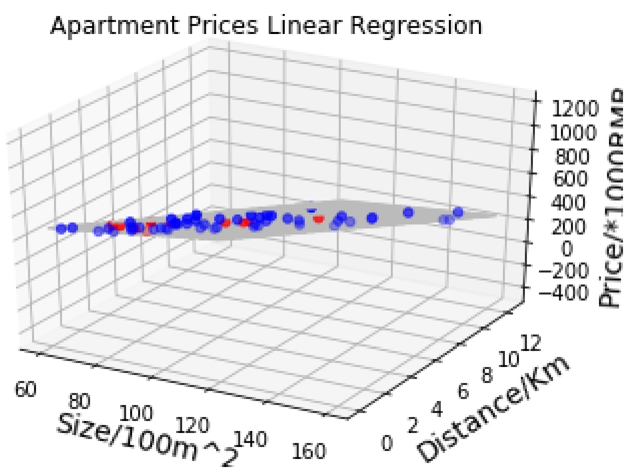
```
-------------------------------------------- Iteration 0 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.98570229], [-72.65869529], [56.995
93167] correspondingly.

The average train error is [4.88101026], the average train loss is
[16.86762276].

-------------------------------------------- Iteration 1 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.83259187], [-72.46998117], [72.374
90051] correspondingly.

The average train error is [2.16644415], the average train loss is
[3.4415893].

-------------------------------------------- Iteration 2 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.78430648], [-72.41046768], [77.224
86115] correspondingly.

The average train error is [1.85611188], the average train loss is
[2.10832439].

-------------------------------------------- Iteration 3 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76907905], [-72.39169932], [78.754
3602] correspondingly.

The average train error is [1.90778812], the average train loss is
[1.97635888].

-------------------------------------------- Iteration 4 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76427688], [-72.38578047], [79.236
70789] correspondingly.

The average train error is [1.9240849], the average train loss is
[1.963434].

-------------------------------------------- Iteration 5 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76276245], [-72.38391388], [79.388
8226] correspondingly.

The average train error is [1.92922431], the average train loss is
[1.96221152].

-------------------------------------------- Iteration 6 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76228485], [-72.38332523], [79.436
79398] correspondingly.

The average train error is [1.93084509], the average train loss is
```

```
[1.96210979].

-------------------------------------------- Iteration 7 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76213424], [-72.38313959], [79.451
92239] correspondingly.

The average train error is [1.93135622], the average train loss is
[1.96210593].

-------------------------------------------- Iteration 8 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76208674], [-72.38308104], [79.456
69333] correspondingly.

The average train error is [1.93151741], the average train loss is
[1.96210752].

-------------------------------------------- Iteration 9 -----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76207176], [-72.38306258], [79.458
19791] correspondingly.

The average train error is [1.93156825], the average train loss is
[1.9621083].

-------------------------------------------- Iteration 10 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76206704], [-72.38305676], [79.458
6724] correspondingly.

The average train error is [1.93158428], the average train loss is
[1.96210858].

-------------------------------------------- Iteration 11 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76206555], [-72.38305492], [79.458
82203] correspondingly.

The average train error is [1.93158934], the average train loss is
[1.96210867].

-------------------------------------------- Iteration 12 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76206508], [-72.38305434], [79.458
86922] correspondingly.

The average train error is [1.93159093], the average train loss is
[1.96210869].

-------------------------------------------- Iteration 13 ----------
----------------------------------

w_sizes, w_distances, w_bias = [6.76206493], [-72.38305416], [79.458
88411] correspondingly.
```

```
The average train error is [1.93159143], the average train loss is
[1.9621087].

------------------------------------------- Iteration 14 ----------
---------------------------------

w_sizes, w_distances, w_bias = [6.76206488], [-72.3830541], [79.4588
888] correspondingly.

The average train error is [1.93159159], the average train loss is
[1.96210871].

------------------------------------------- Training Curve --------
-----------------------------------
```




```
------------------------------------------- Test Results ----------
---------------------------------

The average test error is : [11.10925859].

The average test loss is : [66.88540843].
```



As for SGD, we update weights 50 times each iteration, so we only need 1500000/50 = 30000 epochs to train. This model has almost the same accuracy compared with the former two. But apparently, this model needs a shorter computation time.