数据挖掘导论

第二次实验报告

——梯度下降方法求解多重线性回归问题

学号：16340192

姓名：宋晓彤

方向：嵌入式软件与系统

目录

一、 使用梯度下降法训练线性回归模型

1. 实验目的

使用梯度下降法对房价受面积及与双鸭山职业技术学院距离影响的线性回归模型进行训练，其中，参数初始化为 0，学习率为 0.00015，迭代步数每达到 100000 次时进行一次误差计算，绘制图象并进行分析。

2. 实验原理

对房价建立线性回归模型，并使用合适的方法对线性回归模型中的未知参数进行求解，以达到求得最符合训练样本变化情况的参数组成，也就是使得最终线性模型对训练样本的方差最小。

由此，使用梯度下降法，以 0.00015 作为步长，将上述线性模型的均方误差代价函数作为观测函数，从而求得能使代价函数最小的各参数值。、

由于房价受房屋面积 s，与到双鸭山职业技术学院的距离 d 两个变量影响，所以对房价 P 建立如下线性模型：

$$\mathrm{P}(\theta) = \theta_0 + \theta_1 * s + \theta_2 * d$$

对于上述模型，需要使用三个参数对模型进行训练，也就是参数 θ，其均方误差代价函数为

$$\mathrm{J} = \frac{1}{2\mathrm{m}} \sum_{1}^{m} [P(\theta) - p]^2$$

令 X, Y, θ 为三个矩阵，分别表示输入、输出、参数此时上述代价函数为

$$\mathrm{J} = \frac{1}{2\mathrm{m}} (X * \theta - Y)^T (X * \theta - Y)$$

对矩阵 θ 求微分，也就是梯度可得（推导过程见附录）

$$\nabla \mathrm{J} = \frac{1}{\mathrm{m}} X^T (X * \theta - Y)$$

对此微分结果迭代求解

3. 实验过程

将文件中的数据读到 X 矩阵 Y 矩阵中，同时将 theta 矩阵初始化，参数均为 0。然后根据初始参数值，使用上述梯度方程计算下一次循环的参数值，一直到达到了迭

代次数，此时得到的参数值向量 theta 就是要求的结果。

想要计算误差值则可以使用上述均方误差代价函数，当迭代次数达到目标次数时进行相应的计算。

其中，需要对两个函数进行定义。一个是代价函数的计算，一个是对 theta 的梯度。

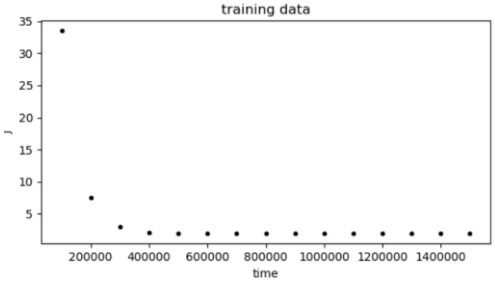
```
# 均方误差计算结果
def error(X, theta, Y, m):
    dis = (dot(X, theta) - Y)
    return dot(transpose(dis), dis) / (2*m)

# 微分计算结果
def diff(X, theta, Y, m):
    return dot(transpose(X), (dot(X, theta) - Y)) / m
```

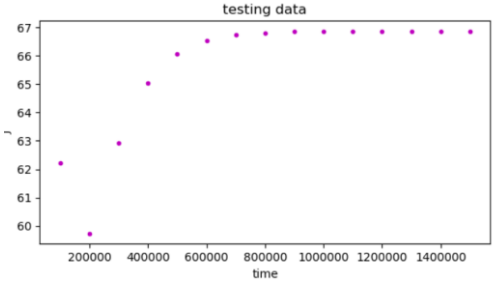
4. 实验结果

（1）θ随迭代次数变化统计

迭代次数/w	θ_0	θ_1	θ_2
1	46.329609	7.089390	-72.759886
2	65.487264	6.900063	-72.540752
3	73.568301	6.820201	-72.448317
4	76.977026	6.786514	-72.409326
5	78.414886	6.772305	-72.392879
6	79.021401	6.766311	-72.385942
7	79.277240	6.763782	-72.383015
8	79.385157	6.762716	-72.381781
9	79.430679	6.762266	-72.381260
10	79.449880	6.762076	-72.381041
11	79.457980	6.761996	-72.380948
12	79.461397	6.761962	-72.380909
13	79.462838	6.761948	-72.380892
14	79.463446	6.761942	-72.380886
15	79.463702	6.761940	-72.380883

（2）训练集以及测试集误差统计





拟合次数/w	训练集代价 J_1	测试集代价 J_2
1	33.503852	62.231649
2	7.529645	59.727646
3	2.908050	62.936059
4	2.085729	65.048234
5	1.939413	66.074202
6	1.913379	66.530997
7	1.908747	66.727955
8	1.907922	66.811796
9	1.907776	66.847297
10	1.907750	66.862296
11	1.907745	66.868627
12	1.907744	66.871299
13	1.907744	66.872426
14	1.907744	66.872901
15	1.907744	66.873102

5. 实验结论

应该使用 3 个参数对模型进行训练，分别是面积的系数 θ_1、距离的系数 θ_2 以及常数 θ_0。

对于训练集，均方误差代价呈现自然下降的趋势，到第五十万次左右就已经达到基本上的稳定，此时的方差稳定在 1.9 左右，在保留六位小数的情况下，最终结果稳定在 1.907744，此时，只查看训练集的结果，可以认为当前的线性模型对训练集进行了较好的拟合。

而对于测试集，均方误差从开始到第 200000 次迭代时降低，但是转而开始递增，最终趋于稳定，稳定在 68.87 左右。后期误差的变化比第一次还要高很多，也就是说，在 200000 次左右时，拟合效果已经达到最好，再次进行迭代之后测试效果反而变差，可以认为此时发生了过拟合。

二、 修改学习率二次实验

1. 实验目的

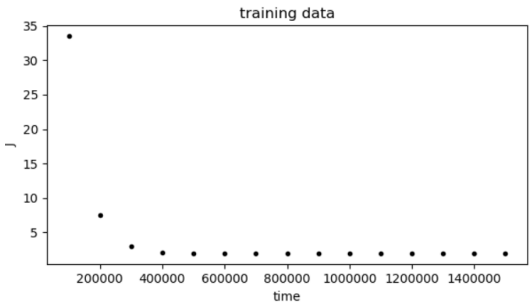
改变学习率，重复实验一过程并进行分析。

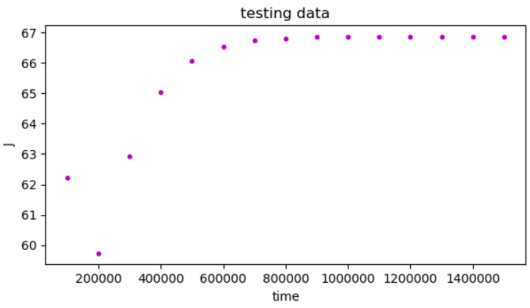
2. 实验原理

为了比较学习率大小的效果，分别将新的学习率修改为 0.0001 和 0.0002，可以判断当学习率更小或者更大时的结果。

3. 实验过程与结果

（1） 新学习率为 0.0001 时：





（2） 新学习率为 0.0002 时：

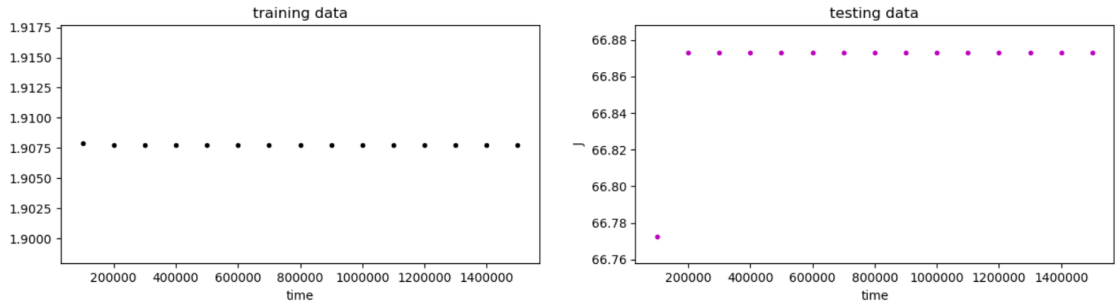
```
1.py:46: RuntimeWarning: invalid value encountered in subtract
  theta = theta - step*diff(x_input, theta, y_input, m)
100000 [[nan nan nan]]
200000 [[nan nan nan]]
300000 [[nan nan nan]]
400000 [[nan nan nan]]
```

此时数据范围出现错误，从 θ 的输出结果我们也可以发现， θ 变为 nan，也就是说此时的数据超过了范围以至于程序作出提醒。对于这样的情况，一般有几种原因，一种学习率过大导致梯度爆炸，另一种是出现脏数据，使得数据出现无穷结果，可以使用归一化等方法进行调整，或者可能是求代价或梯度函数时由于使用数值过大产生了计算结果超出数据范围的情况。

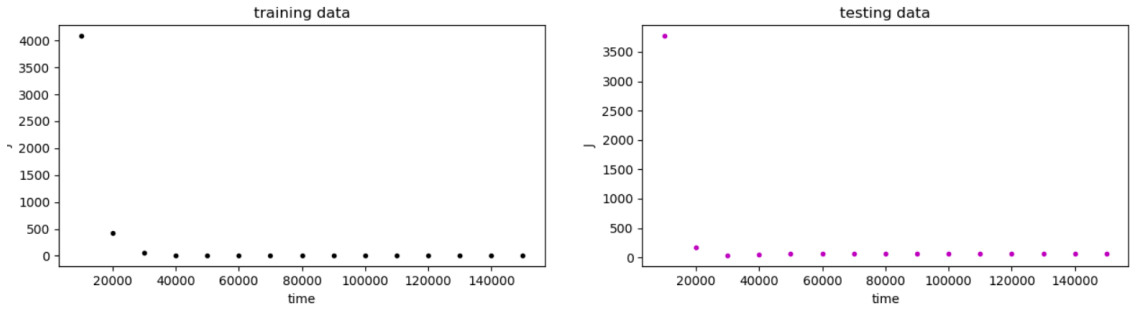
经过分析我们可以发现，全程使用矩阵相乘的算法，产生无穷脏数据的可能性不是很大，最大的可能是在计算过程中数值过大使得数据溢出，所以为了排除这个问题，我们需要对数据进行标准化：也就是用输入数据与对应测量特征平均值的差除以其标准差最后得到标准化数据

$$x_i = (x_i - \mu_i)/\sigma_i$$

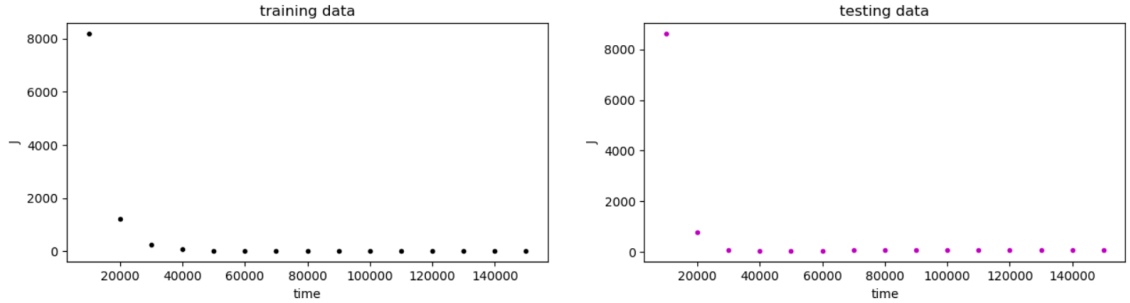
使用 0.0002 作为学习率时，最后的结果如下



此时我们可以发现由于迭代次数设置过大，导致图象观测不明显，将迭代次数 N 改为 15 万，K 改为 1 万再次实验



为了与学习率为 0.00015 对比，我们使用标准化的数据再次对学习率为 0.00015 进行实验并比对



两次实验的均方误差整理见下表

拟合次数/1w	0.00015	0.0002	0.00015	0.0002
	训练集代价 J1	训练集代价 J2	测试集代价 $J3_1$	测试集代价 $J4_2$
1	4085.080269	8187.482734	3772.263200	8619.264372
2	426.813399	1220.237313	172.007666	779.023743
3	63.099144	260.529473	28.377095	84.581666
4	11.175064	63.103317	43.401406	28.376634
5	3.320085	16.748184	56.537436	38.875783
6	2.123148	5.525169	62.659668	51.046528
7	1.940599	2.790435	65.200157	58.579291
8	1.912756	2.123176	66.2155230	62.659392
9	1.908508	1.960325	66.615705	64.762642
10	1.907861	1.920578	66.772560	65.823373
11	1.907762	1.910876	66.833908	66.352790

12	1.907747	1.908509	66.857881	66.615680
13	1.907744	1.907931	66.867246	66.745889
14	1.907744	1.907790	66.870904	66.810301
15	1.907744	1.907755	66.872332	66.842143

4. 实验结论

当使用 0.0001 作为新的学习率时，过拟合的现象没有得到改善，不难分析，当学习率为 0.00015 时已经出现过度拟合，也就是步长过小的情况，使用更小的学习率一定不会对过拟合现象产生改善。

当使用 0.0002 作为学习率的时候，由于没有对数据进行标准化的预处理，theta 数值产生 nan 的情况，而对于标准化、以及减少迭代次数后的实验现象，我们可以发现当使用 0.0002 作为学习率时，拟合收敛的速度要高于使用 0.00015 时的速度，也就是学习率越大，收敛的速率相对而言就会越快一些。

三、 随机梯度下降方法训练模型

1. 实验目的

使用随机梯度下降法对线性回归模型进行训练并与实验一进行对比得出结论。

2. 实验原理

随机梯度下降法对每次数值的代入并没有使用全部数据，而是从训练样本中随机取得一个数据代入计算。

与实验一及实验二相比，X Y theta 可以不采取矩阵的形式，代价函数和微分方程的形式也随之发生变化，

$$\mathrm{J} = \frac{1}{2}\,[P(\theta) - p]^2$$

对 θ_0 θ_1 θ_2 求偏导

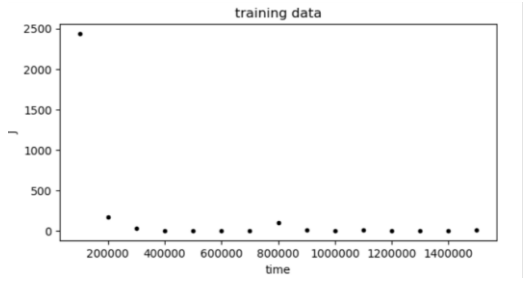
$$\frac{\partial \mathrm{J}}{\partial \theta_0} = 1 * (\mathrm{P}(\theta) - \mathrm{p});\quad \frac{\partial \mathrm{J}}{\partial \theta_1} = s * (\mathrm{P}(\theta) - \mathrm{p});\quad \frac{\partial \mathrm{J}}{\partial \theta_2} = d * (\mathrm{P}(\theta) - \mathrm{p})$$

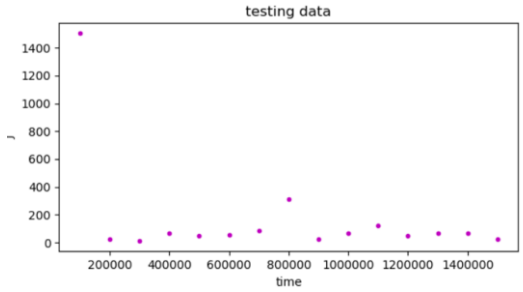
3. 实现过程

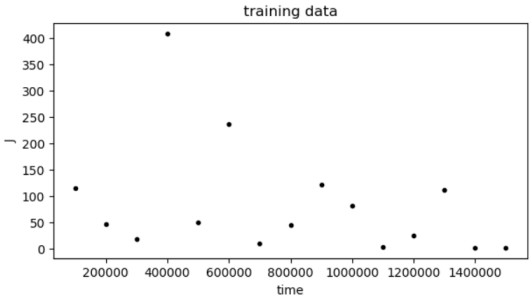
与实验一同理，当无法正常拟合时，可以考虑减小学习率

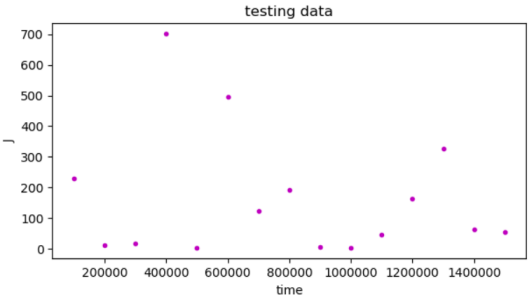
4. 实验结果

(1) 使用与实验一和实验二相同的迭代次数 N 与 K 时，

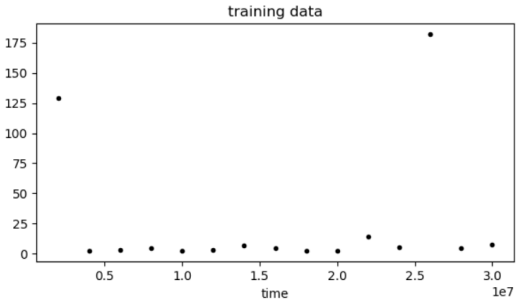


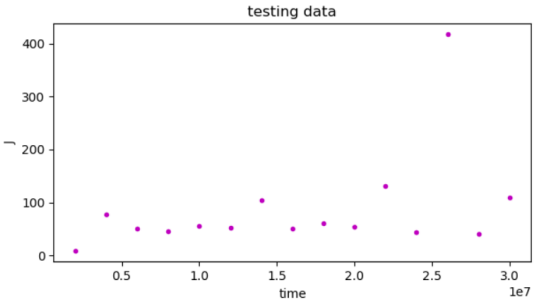




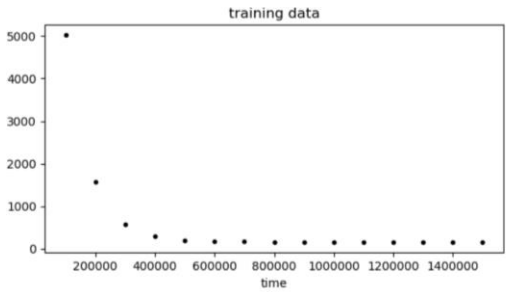


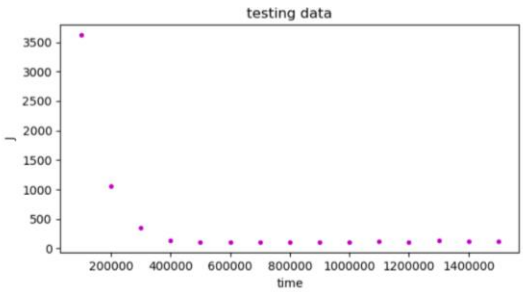
(2) 分别将 N 与 K 扩大 20 倍

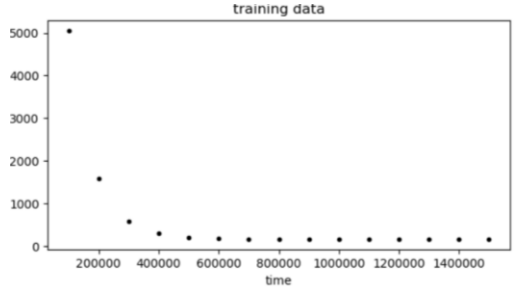


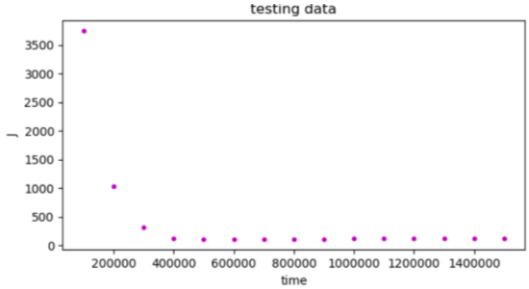


(3) 使用与实验一和实验二相同的迭代次数 N 与 K，但是缩小学习率









5. 实验结论

当使用 1500000 次迭代、K 取 100000，学习率仍为 0.00015 时，数据产生训练的结果仍旧出现问题，需要用 longlong 记录预测值，而且结果相对来说很不稳定，方差结果点很多时候散布得没有规律，此时，不认为这是正确拟合的结果。

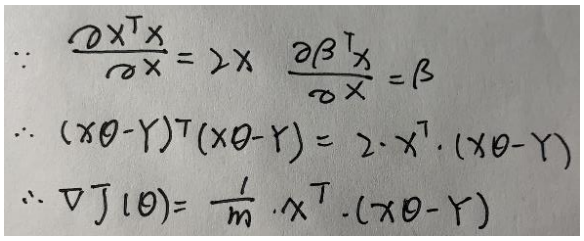
此时考虑扩大迭代次数，将迭代次数扩大 20 倍也就是 3000 万次，每 200 万次取点，最后的结果可以发现，最终的均方误差在某个范围内波动，偶尔由于随机数据的选取产生跳变。

第三次尝试缩小学习率，并将学习率设定为 0.000001，此时发现，多次实验时，每次都能够达到收敛的结果。。

所以此时，通过前面的实验现象，可以认为，当迭代次数与采样频率选取适当时，最终的误差能够在某个范围内波动，偶尔产生较高的跳变值，而如果合理降低学习率，会较为容易得到收敛的效果。

四、 附录

1. 矩阵微分推导过程



2. 实验一源代码

```
#-*- coding: utf-8 -*-
import random
from numpy import *
import matplotlib.pyplot as plt

x_input, x_test = ones((50,3)), ones((10,3))
y_input, y_test = zeros((50,1)), ones((10,1))

# 初始化输入矩阵和输出矩阵
```

```
file = open('dataForTraining.txt', 'r').readlines()
file2 = open('dataForTesting.txt', 'r').readlines()
for i in range(50):
    data = file[i].strip('\n').split(' ')
    x_input[i][1] = data[0]
    x_input[i][2] = data[1]
    y_input[i][0] = data[2]
    if i < 10:
        data2 = file2[i].strip('\n').split(' ')
        x_test[i][1] = data2[0]
        x_test[i][2] = data2[1]
        y_test[i][0] = data2[2]
# 初始化 theta
theta = [[0], [0], [0]]
# 均方误差计算结果
def error(X, theta, Y, m):
    dis = (dot(X, theta) - Y)
    return dot(transpose(dis), dis) / (2*m)

# 微分计算结果
def diff(X, theta, Y, m):
    return dot(transpose(X), (dot(X, theta) - Y)) / m

# 预测值
def pred(X, theta, r):
    y = zeros((r, 1))
    for i in range(r):
        y[i] = theta[0][0] + theta[1][0]*X[i][1] + theta[2][0]*X[i][2]
    return y

time = 1500000
K = 100000
step = 0.001
m = 50
axis_x, axis_y_train, axis_y_test = [], [], []
for i in range(1, time+1):
    theta = theta - step*diff(x_input, theta, y_input, m)
    if i % K == 0:
        axis_x.append(i)
        axis_y_train.append(error(x_input, theta, y_input, m)[0][0])
        axis_y_test.append(error(x_test, theta, y_test, 10)[0][0])
print(theta)
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
```

```
ax2 = fig.add_subplot(2, 2, 2)

ax1.set_title('training data')
ax2.set_title('testing data')
ax1.set_xlabel('time')
ax2.set_xlabel('time')
ax1.set_ylabel('J')
ax2.set_ylabel('J')
ax1.scatter(axis_x, axis_y_train, c='k', marker='.')
ax2.scatter(axis_x, axis_y_test, c='m', marker='.')
plt.show()
```

3. 实验二标准化过程

```
for i in range(50):
    s.append(float(data[0]))
    d.append(float(data[1]))
saver = mean(s)
daver = mean(d)
svar = std(s)
dvar = std(d)

for i in range(50):
    x_input[i][1] = (float(data[0])-saver)/svar
    x_input[i][2] = (float(data[1])-daver)/dvar
    y_input[i][0] = float(data[2])
    if i < 10:
        x_test[i][1] = (float(data2[0]) - saver)/svar
        x_test[i][2] = (float(data2[1]) - daver)/dvar
        y_test[i][0] = float(data2[2])
```

4. 实验三源代码

```
# 均方误差计算结果
def error(X, theta, Y, m):
    dis = (dot(X, theta) - Y)
    return dot(transpose(dis), dis) / (2*m)

# 微分计算结果
def diff(X, theta, m, Y, step):
    choose = random.randint(0, m-1)
    pre = longlong(pred(X, choose, theta, m))
    theta[0][0] = theta[0][0] - step*X[choose][0] * (pre-Y[choose][0])
```

```
    theta[1][0] = theta[1][0] - step*X[choose][1] * (pre-Y[choose][0])
    theta[2][0] = theta[2][0] - step*X[choose][2] * (pre-Y[choose][0])
    return theta

# 预测值
def pred(X, choose, theta, r):
    y = X[choose][0] * theta[0][0] + X[choose][1]*theta[1][0] +
X[choose][2] * theta[2][0]
    return y

time = 1500000
K = 100000
step = 0.00015
m = 50
axis_x, axis_y_train, axis_y_test = [], [], []
for i in range(1, time+1):
    theta = diff(x_input, theta, m, y_input, step)
    if i % 100000 == 0:
        axis_x.append(i)
        axis_y_train.append(error(x_input, theta, y_input, m)[0][0])
        axis_y_test.append(error(x_test, theta, y_test, 10)[0][0])
```