**《机器学习与数据挖掘》实验二**

实验题目： 使用梯度下降法训练多元线性回归模型

实验目的： 掌握线性回归的基本原理，以及梯度下降法

实验环境（硬件和软件） Anaconda/Jupyter notebook/Pycharm

实验内容：

（1）编码实现基于梯度下降的多元线性回归算法，包括梯度的计算与验证；

（2）画数据散点图，以及得到的直线；

（3）画梯度下降过程中损失的变化图；

（4）基于训练得到的参数，输入新的样本数据，输出预测值；

**实验原理：**

多元线性回归建立在单元线性回归的基础上，其预测函数也和单元线性回归的函数大体相同。

由于变量增加，所以参数也多一个，其他都和单元线性回归一致。

求θ的过程通过对预测值和真实值的“差距”的不断缩小实现。

这个差距用方差，这里叫损失函数实现。

损失函数的形式如下：

因此只要对这个损失函数的θ求梯度，就可以得到每一个参数变化方向。

得到如下公式：

即：

注意到只知道参数变化的方向而不知道具体变化的数值，所以此处引入ETA，学习率，它是一个步长，表示每一次参数更新变化的剧烈程度。

如此，只要将所有参数并成一个矩阵，不断向梯度指向的方向变化，就可以在有限的时间内得到相对正确的参数。

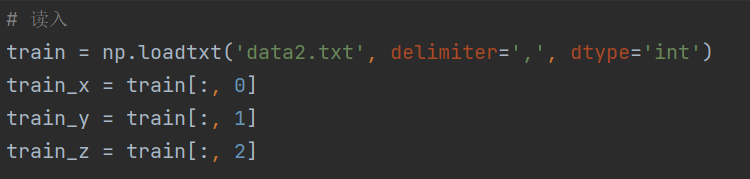
**梯度下降原理：**

在最小化损失函数时，可以通过梯度下降法来一步步的迭代求解，得到最小化的损失函数，和模型参数值。反过来，如果我们需要求解损失函数的最大值，这时就需要用梯度上升法来迭代了。

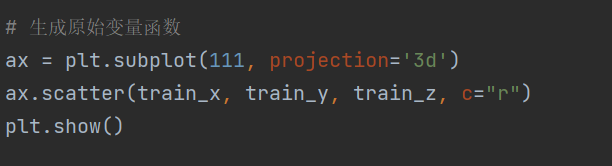
梯度是一个向量，是一个n元函数f关于n个变量的偏导数，比如二元函数f的梯度为, 简称为或者, 而三元函数的梯度为 , 然后要明白梯度的方向是函数增长最快的方向，梯度的反方向是降低最快的方向。

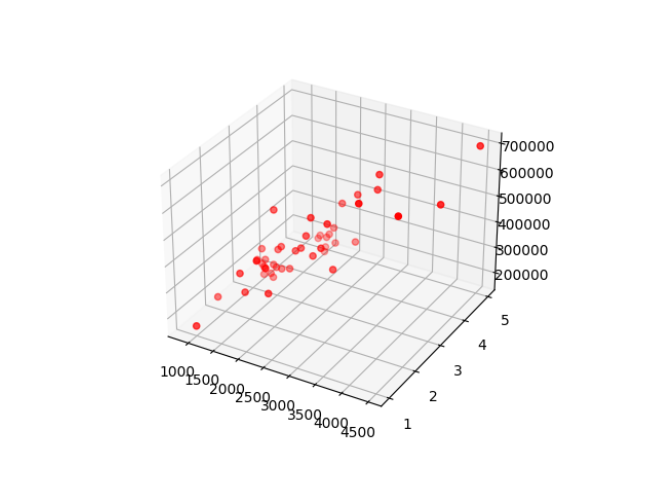
**实验步骤：**

1. 从 data.txt 文件中读取已知数据，分别存入三变量 train\_x,train\_y,train\_z中作为训练数据。

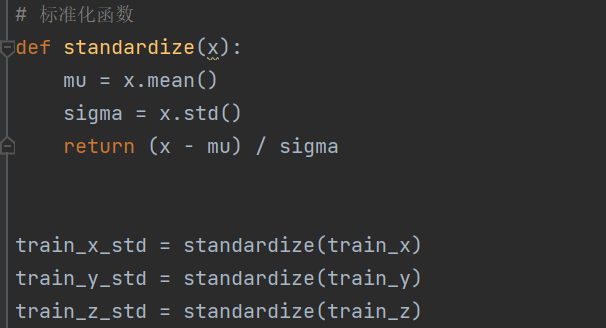


1. 生成原始变量函数，得到三维散点图

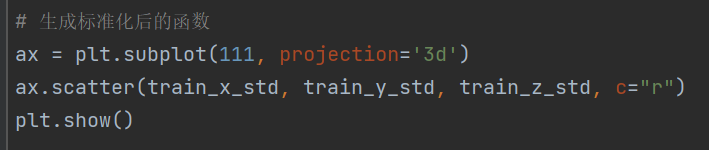


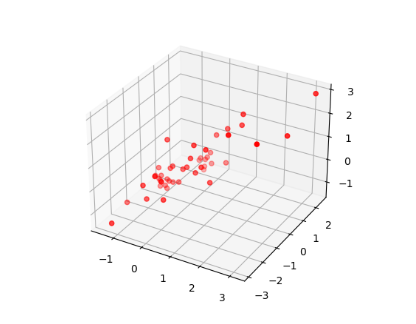


1. 对存储的数据进行标准化：

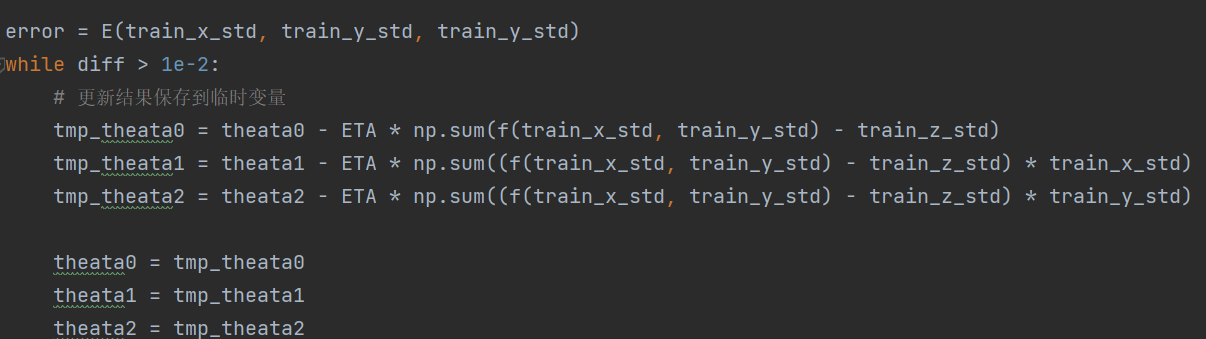


1. 将标准化的函数生成三维散点图

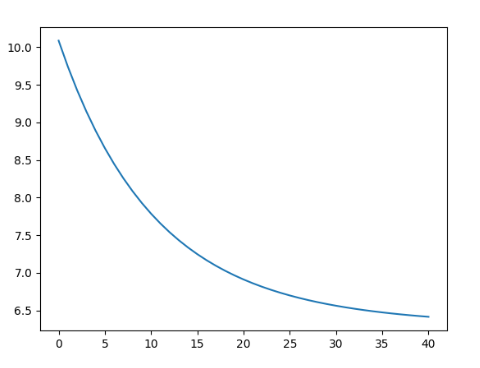




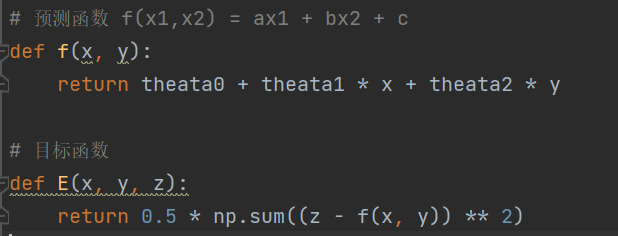
1. 设置合适步长（即学习率），不断迭代，减少误差，更新

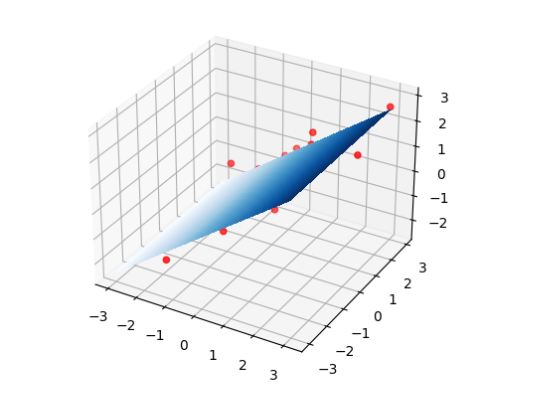


1. 生成迭代过程中损失变化图

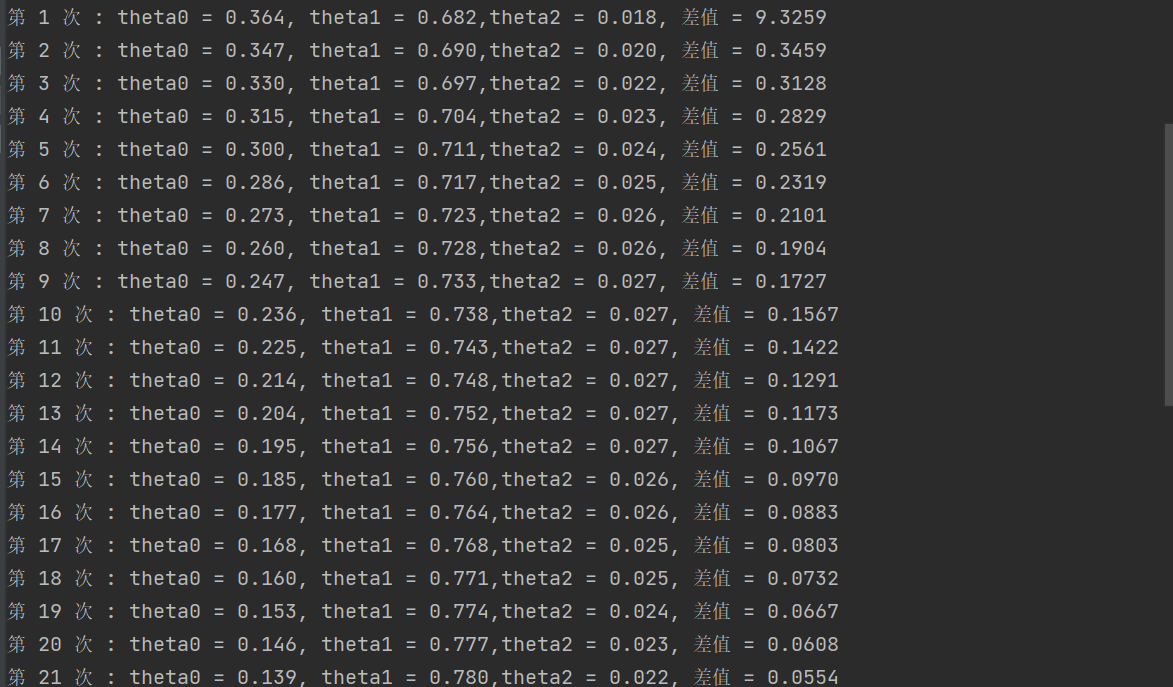
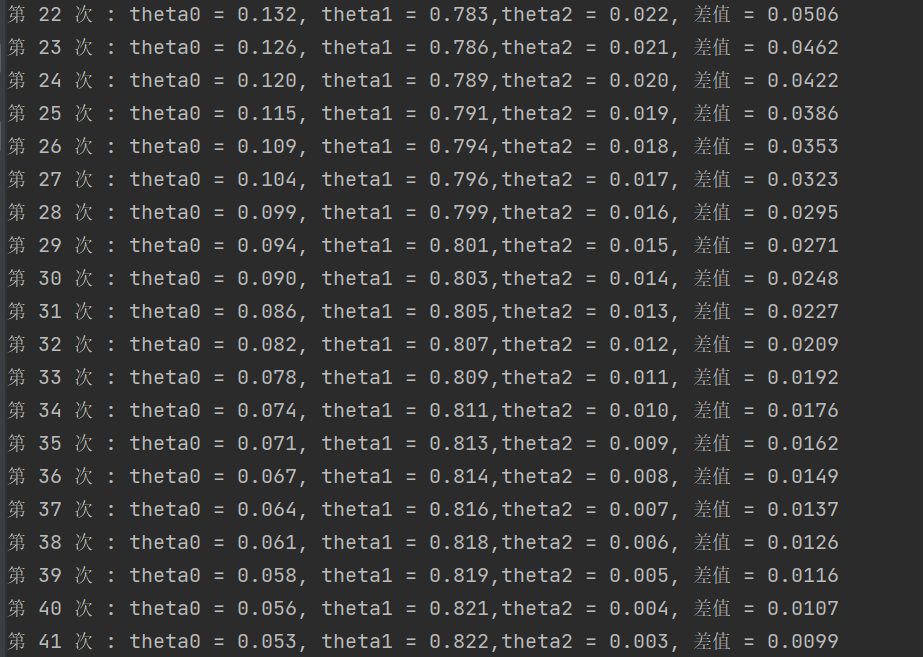


1. 利用预测函数与误差分析来对数据进行预测





**迭代次数与误差日志：**

**实验代码：**

from matplotlib import projections  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib import cm  
from mpl\_toolkits.mplot3d import axes3d  
  
# 读入  
train = np.loadtxt('data2.txt', delimiter=',', dtype='int')  
train\_x = train[:, 0]  
train\_y = train[:, 1]  
train\_z = train[:, 2]  
  
# 生成原始变量函数  
ax = plt.subplot(111, projection='3d')  
ax.scatter(train\_x, train\_y, train\_z, c="r")  
plt.show()  
  
# 随机生成 theata  
theata0 = np.random.rand()  
theata1 = np.random.rand()  
theata2 = np.random.rand()  
  
  
# 预测函数 f(x1,x2) = ax1 + bx2 + c  
def f(x, y):  
 return theata0 + theata1 \* x + theata2 \* y  
  
# 目标函数  
def E(x, y, z):  
 return 0.5 \* np.sum((z - f(x, y)) \*\* 2)  
  
  
# 标准化函数  
def standardize(x):  
 mu = x.mean()  
 sigma = x.std()  
 return (x - mu) / sigma  
  
  
train\_x\_std = standardize(train\_x)  
train\_y\_std = standardize(train\_y)  
train\_z\_std = standardize(train\_z)  
  
# 生成标准化后的函数  
ax = plt.subplot(111, projection='3d')  
ax.scatter(train\_x\_std, train\_y\_std, train\_z\_std, c="r")  
plt.show()  
  
ETA = 1e-3 # 学习率  
diff = 1 # 误差大小  
count = 0 # 迭代次数  
cnt = [] # 次数列表  
errs = [] # 误差列表  
  
error = E(train\_x\_std, train\_y\_std, train\_y\_std)  
while diff > 1e-2:  
 # 更新结果保存到临时变量  
 tmp\_theata0 = theata0 - ETA \* np.sum(f(train\_x\_std, train\_y\_std) - train\_z\_std)  
 tmp\_theata1 = theata1 - ETA \* np.sum((f(train\_x\_std, train\_y\_std) - train\_z\_std) \* train\_x\_std)  
 tmp\_theata2 = theata2 - ETA \* np.sum((f(train\_x\_std, train\_y\_std) - train\_z\_std) \* train\_y\_std)  
  
 theata0 = tmp\_theata0  
 theata1 = tmp\_theata1  
 theata2 = tmp\_theata2  
  
 current\_error = E(train\_x\_std, train\_y\_std, train\_z\_std)  
  
 diff = error - current\_error  
 error = current\_error  
  
 cnt.append(count)  
 errs.append(current\_error)  
 count += 1  
  
 log = '第 {} 次 : theta0 = {:.3f}, theta1 = {:.3f},theta2 = {:.3f}, 差值 = {:.4f}'  
 print(log.format(count, theata0, theata1, theata2, diff))  
  
ax = plt.subplot(111, projection='3d')  
ax.scatter(train\_x\_std, train\_y\_std, train\_z\_std, c='r')  
  
x = np.arange(-3, 3, 0.1)  
y = np.arange(-3, 3, 0.1)  
x, y = np.meshgrid(x, y)  
z = f(x, y)  
  
surf = ax.plot\_surface(x, y, z, cmap=cm.Blues, linewidth=1, antialiased=False)  
  
plt.show()  
  
plt.plot(cnt, errs)  
plt.show()