机器学习第一次实验实验报告

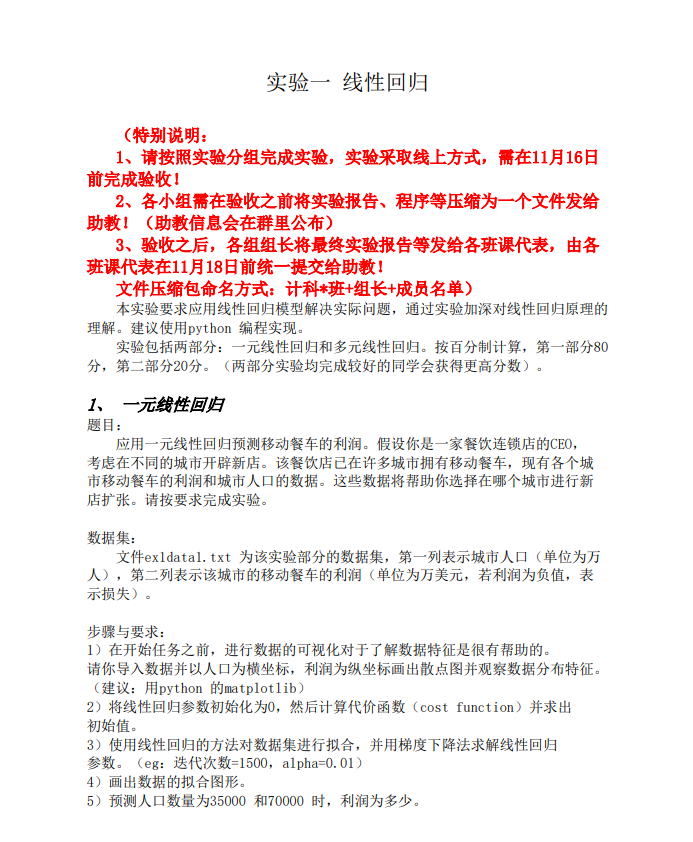
**实验成员：**

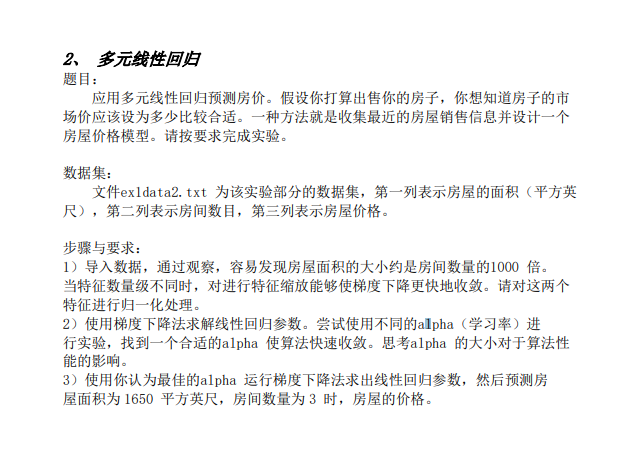
计科1804张曦辰 201808010425

**完成日期：**

2020.11.10

**实验题目：**

****

****

**一、实验描述：**

第一题：

观察数据散点图，考虑用线性回归拟合数据。用梯度下降法求解线性回归参数（alpha和迭代次数已给出），利用矩阵处理数据集

第二题：

考虑到数据特征数量级不同时，对进行特征缩放能够使梯度下降更快地收敛，因此对数据进行归一化处理。固定迭代次数变化学习率（alpha）测试收敛效率，固定学习率拟合并预测结果

**二、实验结果及分析：**

1. **开发语言及运行环境**

开发语言：python

运行环境：

实验平台：Windows

解释器：python3.8

IDE：Pycharm

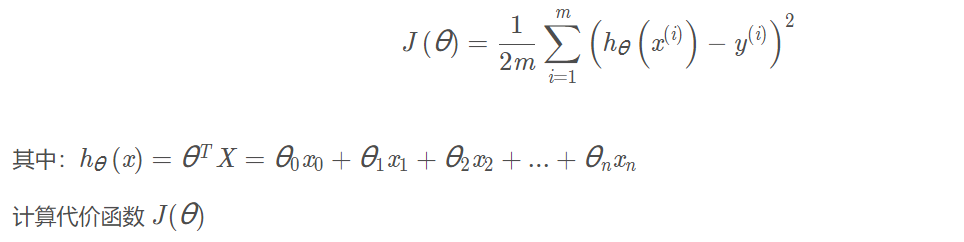
导入的包：matplotlib、numpy、pandas、sklearn

1. **实验的具体步骤**

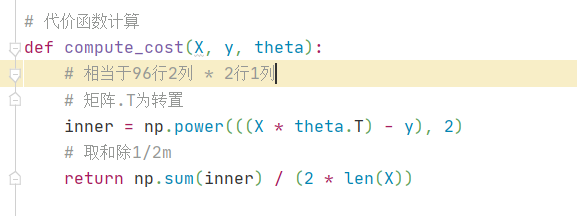
**第一题：**

1. pd.read\_csv读入数据
2. 利用matplotlib库绘制散点图，查看数据特征
3. 把数据转换为矩阵，方便处理
4. 参数初始化为0，调用代价函数求出初始值。

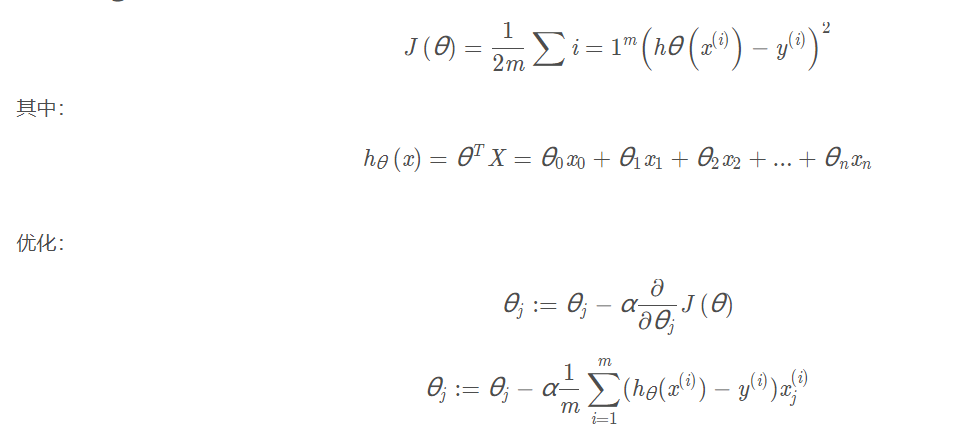
代价函数计算方法如下：



代码



1. 调用梯度下降函数求解回归参数



代码：



返回值为代价数据和回归参数

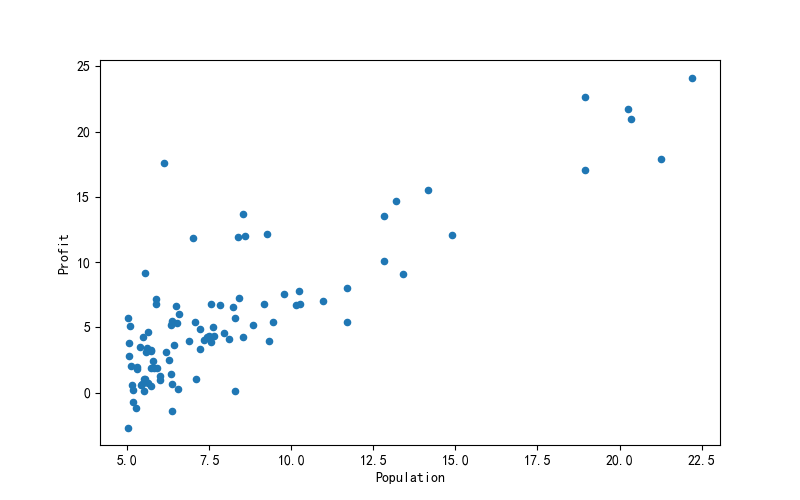
1. 画出拟合直线和代价值随迭代次数的变化图
2. 根据回归参数预测数据
3. 调用sklearn的线性回归模型model = LinearRegression()，拟合后画图，发现和自己写的结果相近

**第二题：**

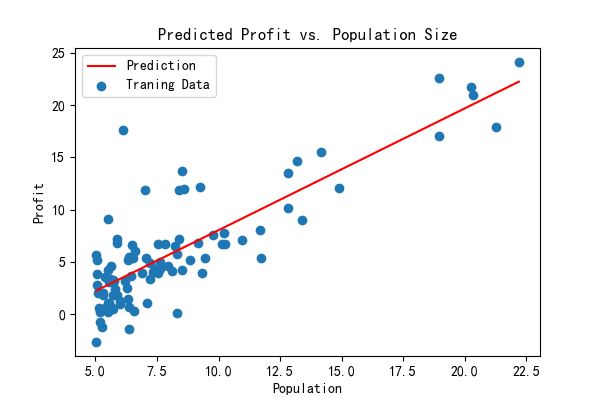
1. 代价计算函数和梯度下降函数和第一问相似
2. 用pandas库读入数据，并调用库方法对数据进行归一化
3. 对alpha取不同值进行迭代计算并计算收敛效率并绘图
4. 确定alpha值对数据集进行多元回归，计算代价和回归参数
5. 根据回归参数进行数据预测
6. **根据实验数据集，按实验要求给出相应的结果（截图）**

第一问：

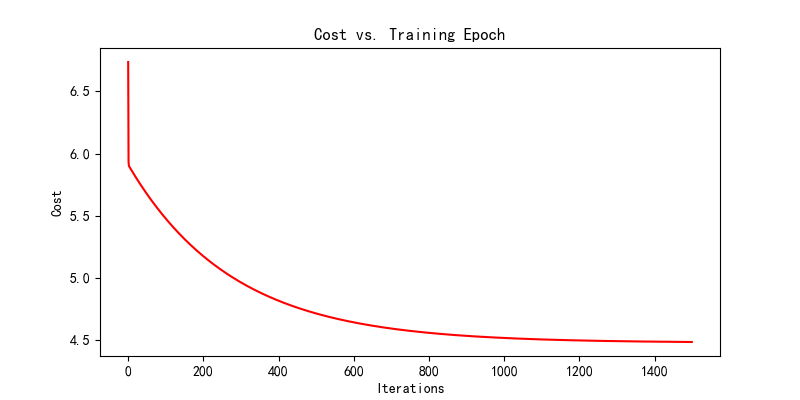
原始数据



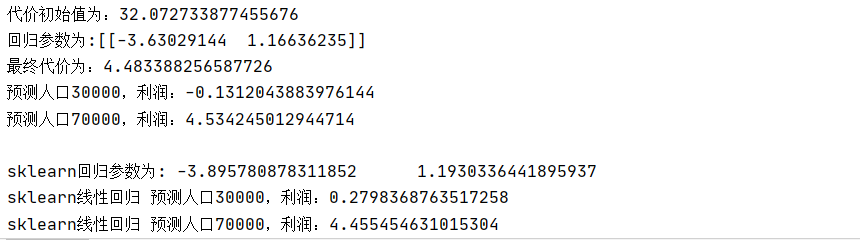
梯度下降拟合



梯度下降拟合代价

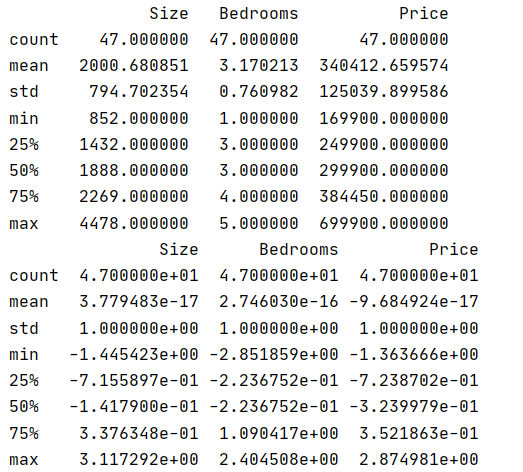


代价初始值和预测结果



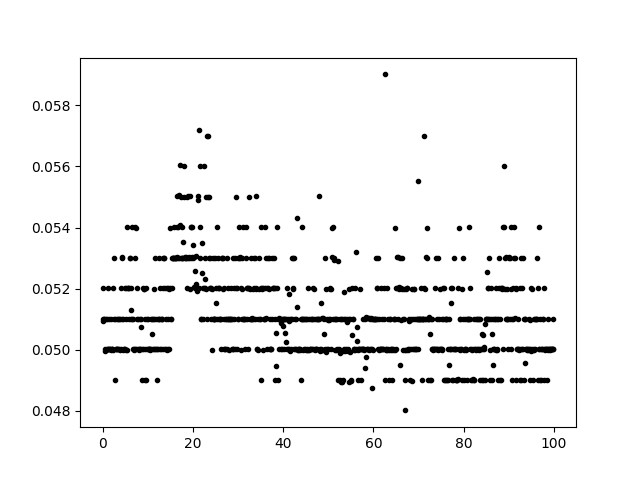
第二题：

归一化结果

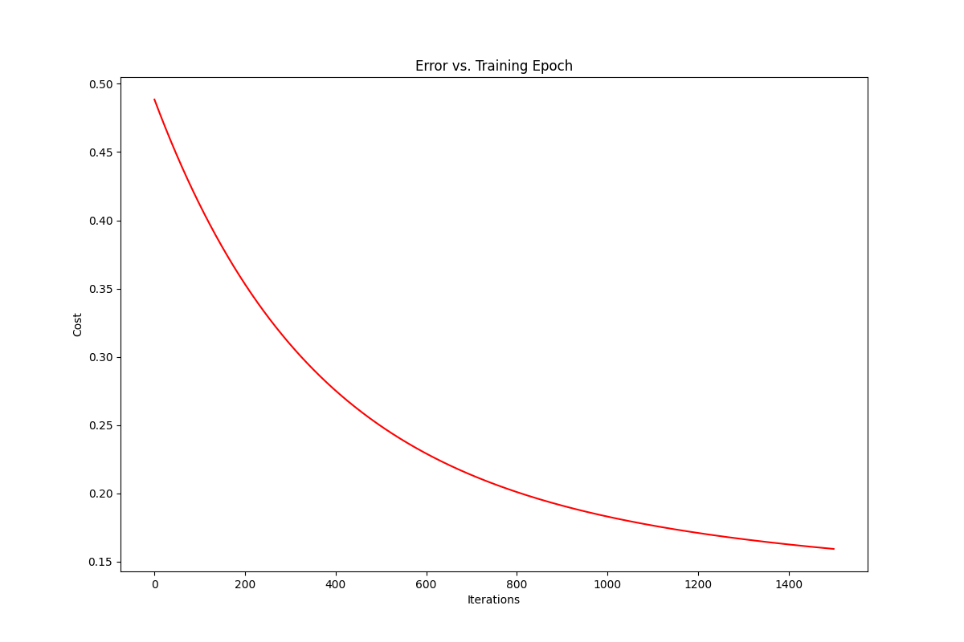


在alpha 0.01 – 10遍历，收敛时间

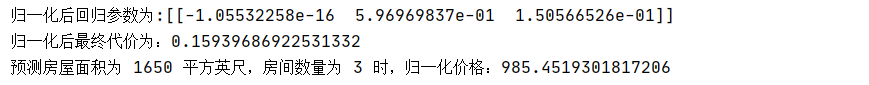
alpha越大学习速度越快，但是拟合越不准确



alpha=0.001迭代次数-代价曲线



预测结果



1. **对实验结果进行简要分析**

第一题：

1）数据可视化后考虑用线性回归拟合数据

2）将线性回归参数初始化为0，求出代价初始值为32.072733877455676

3）求解线性回归参数后利用sklearn库线性回归拟合进行结果对比，发现结果十分相近，拟合图形高度相似，实验成功

4）根据拟合参数预测人口

第二题：

1. 查看原始数据和归一化数据的区别
2. 遍历alpha在0.01-10，发现alpha越大学习速度越快，但是拟合越不准确
3. 利用梯度下降进行迭代拟合，并预测结果

**三、实验心得**

张曦辰：这次实验我主要负责理解内容的基础上写出代码，本次实验不仅对线性回归和梯度下降发有了更深的理解，更是学习了numpy和pandas库的相关操作，并了解到了矩阵在机器学习计算中的妙用。

**四、附录**

第一问代码如下：

import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
from sklearn.linear\_model import LinearRegression  
  
plt.rcParams[**'font.sans-serif'**] = [**'SimHei'**] # 显示中文标签  
plt.rcParams[**'axes.unicode\_minus'**] = False # 这两行需要手动设置  
  
  
# 代价函数计算  
def compute\_cost(X, y, theta):  
 # 相当于96行2列 \* 2行1列  
 # 矩阵.T为转置  
 inner = np.power(((X \* theta.T) - y), 2)  
 # 取和除1/2m  
 return np.sum(inner) / (2 \* len(X))  
  
  
def gradient\_descent(X, y, theta, alpha, epoch):  
 """return theta, cost"""  
  
 cost = np.zeros(epoch) # 初始化一个np.array，包含每次epoch的cost  
 m = X.shape[0] # 样本数量m  
  
 for i in range(epoch):  
 # 迭代次数  
 # 利用向量化一步求解  
 temp = theta - (alpha / m) \* (X \* theta.T - y).T \* X  
  
 theta = temp  
 cost[i] = compute\_cost(X, y, theta)  
  
 return theta, cost  
  
  
def create\_by\_sklearn():  
 city\_population = []  
 profit = []  
 # 读入数据  
 with open(**"ex1data1.txt"**, **"r"**, encoding=**'utf-8'**) as file:  
 for line in file.readlines():  
 line = line.strip()  
 city\_population.append(float(line.split(**","**)[0]))  
 profit.append(float(line.split(**","**)[1]))  
  
 # 数组转换  
 city\_population = np.array(city\_population)  
 profit = np.array(profit)  
  
 # 定义模型  
 model = LinearRegression()  
 city = city\_population  
 profit = profit  
  
 # 变为一列  
 x = city.reshape((-1, 1))  
 y = profit  
  
 # 拟合  
 model.fit(x, y)  
 print(**"sklearn回归参数为: {}** \t **{}"**.format(model.intercept\_, model.coef\_[0]))  
 # print(model.intercept\_) # 截距  
 # print(model.coef\_) # 线性模型的系数  
 x2 = [[-1], [3.5], [7], [22.5]] # 取两个预测值  
 y2 = model.predict(x2) # 进行预测  
 print(**"sklearn线性回归 预测人口30000，利润：{}"**.format(y2[1]))  
 print(**"sklearn线性回归 预测人口70000，利润：{}"**.format(y2[2]))  
  
 plt.title(**"sklearn线性回归"**)  
 plt.xlabel(**'city\_population'**, fontsize=15, color=**'b'**)  
 plt.ylabel(**'profit'**, fontsize=15, color=**'b'**)  
 plt.plot(city, profit, **'k.'**) # 黑点  
 plt.plot(x2, y2, **'g-'**) # 画出拟合曲线，绿色实线  
 plt.savefig(**"题目一拟合结果/sklearn拟合.png"**)  
 plt.show()  
  
 plt.xlabel(**'city\_population'**, fontsize=15, color=**'b'**)  
 plt.ylabel(**'profit'**, fontsize=15, color=**'b'**)  
 plt.title(**"sklearn线性回归残差"**)  
 yr = model.predict(x)  
 for index, x in enumerate(x):  
 plt.plot([x, x], [y[index], yr[index]], **'r-'**)  
 plt.plot(city, profit, **'k.'**) # 黑点  
 plt.plot(x2, y2, **'g-'**) # 画出拟合曲线  
 plt.savefig(**"题目一拟合结果/sklearn线性回归残差.png"**)  
 plt.show()  
  
  
def create\_by\_local():  
 path = **'ex1data1.txt'** # names添加列名，header用指定的行来作为标题，若原无标题且指定标题则设为None  
 data = pd.read\_csv(path, header=None, names=[**'Population'**, **'Profit'**])  
  
 # 查看数据简介  
 # print(data.describe())  
  
 # 可视化数据  
 data.plot(kind=**'scatter'**, x=**'Population'**, y=**'Profit'**, figsize=(8, 5))  
 plt.savefig(**"题目一拟合结果/原始数据.png"**)  
 plt.show()  
  
 # print(data)  
 # 在训练集中添加一列，以便我们可以使用向量化的解决方案来计算代价和梯度  
 data.insert(0, **'Ones'**, 1)  
 # print(data)  
  
 # 列数  
 cols = data.shape[1]  
  
 # 取前cols-1列，即输入向量(ONE和人口列)  
 X = data.iloc[:, 0:cols - 1]  
  
 # 取最后一列，即目标向量(利润列）  
 y = data.iloc[:, cols - 1:cols]  
  
 X = np.matrix(X.values)  
 y = np.matrix(y.values)  
  
 # theta = [[0 0]]  
 theta = np.matrix([0, 0])  
 # 截距和斜率  
  
 # 维度，可以进行输出  
 X.shape, theta.shape, y.shape  
 # ((97, 2), (1, 2), (97, 1))  
  
 print(**"代价初始值为：{}"**.format(compute\_cost(X, y, theta))) # 代价初始值：32.072733877455676  
  
 alpha = 0.01  
 epoch = 1500  
  
 # 梯度下降求参数  
 final\_theta, cost = gradient\_descent(X, y, theta, alpha, epoch)  
 print(**"回归参数为:{}"**.format(final\_theta))  
 print(**"最终代价为：{}"**.format(compute\_cost(X, y, final\_theta)))  
  
 # np.linspace()在指定的间隔内返回均匀间隔的数字  
 x = np.linspace(data.Population.min(), data.Population.max(), 100) # 横坐标  
 f = final\_theta[0, 0] + (final\_theta[0, 1] \* x) # 纵坐标，利润  
 # 截距 + 斜率\*横坐标  
  
 # 画出拟合直线  
 fig, ax = plt.subplots(figsize=(6, 4))  
 ax.plot(x, f, **'r'**, label=**'Prediction'**)  
 ax.scatter(data[**'Population'**], data.Profit, label=**'Traning Data'**)  
 ax.legend(loc=2) # 2表示在左上角  
 ax.set\_xlabel(**'Population'**)  
 ax.set\_ylabel(**'Profit'**)  
 ax.set\_title(**'Predicted Profit vs. Population Size'**)  
 plt.savefig(**"题目一拟合结果/梯度下降拟合.png"**)  
 plt.show()  
  
 # 画出代价值随迭代次数的变化  
 fig, ax = plt.subplots(figsize=(8, 4))  
 ax.plot(np.arange(epoch), cost, **'r'**) # np.arange()返回等差数组  
 ax.set\_xlabel(**'Iterations'**)  
 ax.set\_ylabel(**'Cost'**)  
 ax.set\_title(**'Cost vs. Training Epoch'**)  
 plt.savefig(**"题目一拟合结果/梯度下降拟合代价.png"**)  
 plt.show()  
  
 print(**"预测人口30000，利润：{}"**.format(final\_theta[0, 0] + (final\_theta[0, 1] \* 3)))  
 print(**"预测人口70000，利润：{}"**.format(final\_theta[0, 0] + (final\_theta[0, 1] \* 7)))  
 print()  
  
  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 # 手写预测  
 create\_by\_local()  
 # 利用sklearn预测  
 create\_by\_sklearn()

第二问代码如下：

import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import time  
from sklearn.linear\_model import LinearRegression  
  
  
def compute\_cost(X, y, theta):  
 # 相当于96行2列 \* 2行1列  
 # 矩阵.T为转置  
 inner = np.power(((X \* theta.T) - y), 2)  
 # 取和除1/2m  
 return np.sum(inner) / (2 \* len(X))  
  
  
def gradient\_descent(X, y, theta, alpha, epoch):  
 """return theta, cost"""  
  
 cost = np.zeros(epoch) # 初始化一个np.array，包含每次epoch的cost  
 m = X.shape[0] # 样本数量m  
  
 for i in range(epoch):  
 # 迭代次数  
 # 利用向量化一步求解  
 temp = theta - (alpha / m) \* (X \* theta.T - y).T \* X  
  
 theta = temp  
 cost[i] = compute\_cost(X, y, theta)  
  
 return theta, cost  
  
  
def multiple\_linear\_regression(alpha, data2, is\_draw):  
 # set X (training data) and y (target variable)  
 cols = data2.shape[1]  
 X2 = data2.iloc[:, 0:cols - 1]  
 y2 = data2.iloc[:, cols - 1:cols]  
  
 # convert to matrices and initialize theta  
 X2 = np.matrix(X2.values)  
 y2 = np.matrix(y2.values)  
 theta2 = np.matrix(np.array([0, 0, 0]))  
  
 epoch = 1500  
  
 # perform linear regression on the data set  
 g2, cost2 = gradient\_descent(X2, y2, theta2, alpha, epoch)  
  
 # get the cost (error) of the model  
 compute\_cost(X2, y2, g2)  
  
 if is\_draw:  
 fig, ax = plt.subplots(figsize=(12, 8))  
 ax.plot(np.arange(epoch), cost2, **'r'**)  
 ax.set\_xlabel(**'Iterations'**)  
 ax.set\_ylabel(**'Cost'**)  
 ax.set\_title(**'Error vs. Training Epoch'**)  
 plt.savefig(**"题目二拟合结果/alpha=0.001迭代次数-代价曲线.png"**)  
 plt.show()  
 return g2, cost2  
  
  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 path = **'ex1data2.txt'** data2 = pd.read\_csv(path, names=[**'Size'**, **'Bedrooms'**, **'Price'**])  
 # print(data2.head())  
 print(data2.describe())  
  
 # 预处理步骤 - 特征归一化  
 # 减去平均值除以标准差,归一化后标准差为1  
 data2 = (data2 - data2.mean()) / data2.std()  
 print(data2.describe())  
  
 # add ones column  
 data2.insert(0, **'Ones'**, 1)  
  
 time\_cost = []  
 x = []  
 # alpha 0.01 - 10  
 # alpha越大学习速度越快，但是拟合越不准确  
  
 # for alpha in range(0,10000,10):  
 # time\_start = time.time()  
 # multiple\_linear\_regression(alpha/1000, data2,False)  
 # x.append(alpha/100)  
 # time\_end = time.time()  
 # time\_cost.append(time\_end-time\_start)  
 # print(alpha,"\t",time\_end-time\_start)  
 # plt.plot(x, time\_cost, 'k.') # 画出拟合曲线  
 # plt.savefig("题目二拟合结果/alpha-收敛时间.png")  
 # plt.show()  
  
 alpha = 0.001  
 final\_theta, cost = multiple\_linear\_regression(alpha, data2, True)  
 print(**"归一化后回归参数为:{}"**.format(final\_theta))  
 print(**"归一化后最终代价为：{}"**.format(cost[-1]))  
 # alpha越大学习速度越快，但是拟合越不准确  
 print(**"预测房屋面积为 1650 平方英尺，房间数量为 3 时，归一化价格：{}"**.format((final\_theta[0, 0] + (final\_theta[0, 1] \* 1650) + (final\_theta[0, 2] \* 3))))

**五、程序文件名清单**

question1.py

question2.py

ex1data1.txt

ex1data2.txt

两个拟合结果的文件夹