多变量线性回归算法解析（含程序解读）

代码执行的主要流程：

1 初始化工作，插入各种所需要的模块，如numpy等。

2 打开文件，加载数据，用for循环逐行读取数据，并用逗号进行分隔，然后保存在列表中，并打印出原始数据（即特征缩放之前的数据）

3 各个子函数的编写。主要有3个：

1、特征缩放 featureNormalize(X)

2、代价函数 computeCost(X, y, theta)，

3、梯度下降gradientDescent(X, y, theta, alpha, num\_iters)

4 编写主程序对子函数进行调用

1、设置迭代次数，学习率

2、进行数据的处理，将其分成三部分：一、特征x1，x2（47行2列），并保存在列表x中，二、实际值y（47行1列），并保存在列表y中，三、样例的个数（47行1列），并保存在列表m中

3、调用函数特征缩放 featureNormalize(X)对数据归一化，返回三个参数（1、处理之后的特征数据x1，x2；2、均值mu；3、标准差sigma），然对数据进行扩充（即加入特征x0），并用hstack()进行数据的连接，最后得到特征x1，x2，x0，其shape属性为47行3列

4、初始化参数theta，即（，，），然后调用函数梯度下降gradientDescent(X, y, theta, alpha, num\_iters)，在该函数中用J\_history列表来存储历史误差（共10000个元素，即对应10000次迭代），其中使用for循环来得到这10000次的迭代值，在每一次的for循环中，即每迭代一次，就能够得到一个theta列表和一个代价函数值J\_history（其中，代价函数值调用函数computeCost(X, y, theta)来求得）。在for循环结束之后（迭代过程结束），梯度下降函数会返回完整的历史误差列表J\_history（此时该列表含有10000个元素）以及最终的theta参数列表（内含参数，，）

5 画出列表J\_history中的元素，即迭代收敛图，并输出最终的参数，，

6 用得到的参数所构成的最终模型来进行数据的预测

子函数的解读

一、加载数据函数

def load\_exdata(filename):

data = []

with open(filename, 'r') as f: # f = open(filename, 'r'),打开文件

for line in f.readlines():

line = line.split(',') #以逗号为界，将每行数据进行分割为三部分，并生成对应的列表

current = [int(item) for item in line] #数据取整

#5.5277,9.1302

data.append(current) #添加列表元素

print(data)

>>> load\_exdata('E:\\ex1data2.txt')

[[2104, 3, 399900], [1600, 3, 329900], [2400, 3, 369000], [1416, 2, 232000], [3000, 4, 539900], [1985, 4, 299900], [1534, 3, 314900], [1427, 3, 198999], [1380, 3, 212000], [1494, 3, 242500], [1940, 4, 239999], [2000, 3, 347000], [1890, 3, 329999], [4478, 5, 699900], [1268, 3, 259900], [2300, 4, 449900], [1320, 2, 299900], [1236, 3, 199900], [2609, 4, 499998], [3031, 4, 599000], [1767, 3, 252900], [1888, 2, 255000], [1604, 3, 242900], [1962, 4, 259900], [3890, 3, 573900], [1100, 3, 249900], [1458, 3, 464500], [2526, 3, 469000], [2200, 3, 475000], [2637, 3, 299900], [1839, 2, 349900], [1000, 1, 169900], [2040, 4, 314900], [3137, 3, 579900], [1811, 4, 285900], [1437, 3, 249900], [1239, 3, 229900], [2132, 4, 345000], [4215, 4, 549000], [2162, 4, 287000], [1664, 2, 368500], [2238, 3, 329900], [2567, 4, 314000], [1200, 3, 299000], [852, 2, 179900], [1852, 4, 299900], [1203, 3, 239500]]

【注意点】：split()函数的运用

二、特征缩放函数

def featureNormalize(X):

X\_norm = X;

mu = np.zeros((1,X.shape[1])) #生成1行X.shape[1]列的零数组(此处为2列),即[[ 0., 0.]]

sigma = np.zeros((1,X.shape[1])) #sigma的初始值[[ 0., 0.]]

for i in range(X.shape[1]):

mu[0,i] = np.mean(X[:,i]) # 分别计算两列(x1,x2)的均值

sigma[0,i] = np.std(X[:,i]) # 分别计算两列(x1,x2)的标准差

# print(mu) 此处可用来验证均值

# print(sigma) 同上

X\_norm = (X - mu) / sigma #特征缩放公式，此处因为X，mu，sigma的列数相同(都为2)，故可以进行运算

return X\_norm,mu,sigma #返回缩放后的数据以及均值mu，标准差sigma

需要进行特征缩放的数据是：

array([[2104, 3],

[1600, 3],

[2400, 3],

[1416, 2],

[3000, 4],

[1985, 4],

[1534, 3],

[1427, 3],

[1380, 3],

[1494, 3],

[1940, 4],

[2000, 3],

[1890, 3],

[4478, 5],

[1268, 3],

[2300, 4],

[1320, 2],

[1236, 3],

[2609, 4],

[3031, 4],

[1767, 3],

[1888, 2],

[1604, 3],

[1962, 4],

[3890, 3],

[1100, 3],

[1458, 3],

[2526, 3],

[2200, 3],

[2637, 3],

[1839, 2],

[1000, 1],

[2040, 4],

[3137, 3],

[1811, 4],

[1437, 3],

[1239, 3],

[2132, 4],

[4215, 4],

[2162, 4],

[1664, 2],

[2238, 3],

[2567, 4],

[1200, 3],

[ 852, 2],

[1852, 4],

[1203, 3]])

【注意点】：

1、均值和标准差的形状应该是“一行两列”。

2、return语句应该返回三个值(语法上是合法的)。

三、代价函数

def computeCost(X, y, theta):

m = y.shape[0]

# J = (np.sum((X.dot(theta) - y)\*\*2)) / (2\*m)#每次取一组theta值，并与所有样例的值相乘。

C = X.dot(theta) - y

J2 = (C.T.dot(C))/ (2\*m) #C.T.dot(C)，表示C的转置乘C，即得C的平方

return J2

【注意点】：

1、在这个函数中的参数X已经变为了3列(因为在此之前已经通过hstack()函数添加了一列全1列)，故此时X为47行3列，theta为3行1列，即(X.dot(theta)得到47行1列的元素（这些样例即为m=47个所有的样例的预测值），之后在与真实值y作差得到误差值C。

2、在上一步处理之后J就是47行1列的数组，在通过“转置乘以自身”就可以得到数组的“平方和”，最后得到一个47行1列（即为47个样例的代价函数值）的代价函数数组。

四、梯度下降

def gradientDescent(X, y, theta, alpha, num\_iters): #num\_iters为迭代次数

m = y.shape[0]

#print(m)

# J\_history存储历史误差，1000行1列

J\_history = np.zeros((num\_iters, 1))

for iter in range(num\_iters):

# 对J求导，得到 alpha/m \* (WX - Y)\*x(i)， (3,m)\*(m,1) X (m,3)\*(3,1) = (m,1)

theta = theta - (alpha/m) \* (X.T.dot(X.dot(theta) - y)) #更新theta值

J\_history[iter] = computeCost(X, y, theta)#存储更新的损失函数值(因为产生了新的theta)

return J\_history,theta

【注意点】：

1、J\_history用于存储每次迭代之后的代价函数值

2、其中theta = theta - (alpha/m) \* (X.T.dot(X.dot(theta) - y))，X.dot(theta) – y为代价函数值C（见上面的代价函数），之后用数据X的转置乘以代价函数值X.dot(theta) – y，即对应公式中的

那么这个公式的最后一项中，这x的下标n的含义是指特征的编号。即每次迭代求每一个参数，就要计算所有样例的误差C，并乘以这个参数所对应的特征