# 线性回归

1. 模型概述
2. 概念描述

[回归分析](https://baike.baidu.com/item/%E5%9B%9E%E5%BD%92%E5%88%86%E6%9E%90/2625498)中，只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为[一元线性回归](https://baike.baidu.com/item/%E4%B8%80%E5%85%83%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92)分析。如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为[多元线性回归](https://baike.baidu.com/item/%E5%A4%9A%E5%85%83%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92/10702248)分析。

我们需要一个评判标准，来评判直线拟合的好坏，因此引入了最小二乘法“参数估计”以及梯度下降法。

1. 公式描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **数据集** | x= | y= | | W=[ ]  =b |
| **拟合函数** | =x\* | | | |
| **损失函数** |  | | | |
| **W和b**  **最优解** | = | | b= | |
| **梯度下降法** | = | W=W- | |  |

1. 重要代码详解

np.average为求矩阵平均值，该式用来求。

np.sum(np.power(x,2))用来求矩阵中每个元素平方然后求和即。

np.power(np.sum(x),2)/len(x)用来求。

y-theta[0,1]\*x为矩阵每一个元素为。

之后使用np.sum()对其求和再除以其len()即为b的值。

x\_=np.average(x)

theta[0,1]=(y.T\*(x-x\_))/(np.sum(np.power(x,2))-np.power(np.sum(x),2)/len(x))

theta[0,0]=np.sum(y-theta[0,1]\*x)/len(x)

拟合的函数h(x)：返回x\*

损失函数cost()：描述,这样使用的目的是为了简化运算使其求导项中消去元素,同样使用矩阵运算。

def h(X,theta):

    return X\*theta.T

def cost(X,y,theta):

    temp=np.power(h(X,theta)-y,2)

    return np.sum(temp)/(2\*X.shape[0])

梯度下降：error为每个与y的差值。

(X.T\*error).T为然后进行迭代最终求得theta的最优解。

def graddescent(X,y,theta,alpha,count):

    J=np.matrix(np.ones(count))

    for i in range(count):

        error=h(X,theta)-y

        theta=theta-alpha\*(X.T\*error).T

        J[0,i]=cost(X,y,theta)

    return theta,J

绘图：x2表示从x最小值到最大值，平均分为100个点，然后利用求出的参数得出y，绘图查看拟合结果。

x2=np.linspace(X[:,1:].min(),X[:,1:].max(),100)

y2=x1\*theta1[0,1]+theta1[0,0]

plt.plot(x2,y2,'r',label='test2')

plt.plot(X[:,1:],y,'+')

plt.title("test2")

plt.show()

三.结果分析

使用最小二乘法求导获得

w=15.35300994

b=0.79251899

= 40.48314825638028

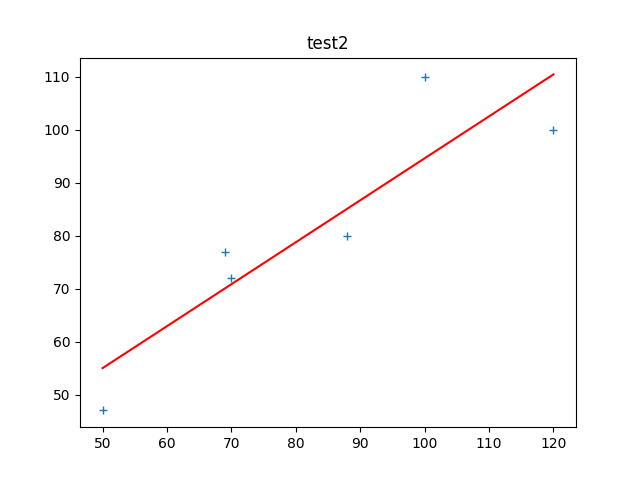
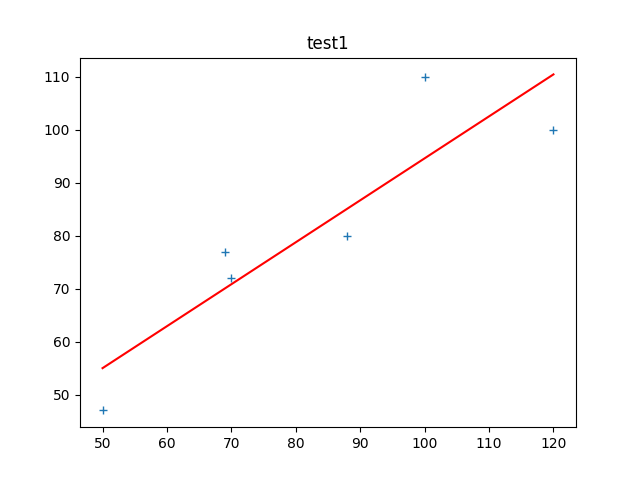
使用梯度下降法得到的结果为

w=15.35300994

b=0.79251899

= 40.48314826

从结果上来看两组结果相同，下面看图示



可以看出在结果图中拟合效果还可以

四.源码

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data=pd.read\_csv("线性回归\data.txt",names=['x1','y'])

data.insert(0,'x0',1)

X=np.matrix(data.iloc[:,0:2])

y=np.matrix(data.iloc[:,2:])

theta=np.matrix(np.ones(X.shape[1]))

#各个矩阵的形状为X(6,2) y(6,1) theta(1,2)

def example(X,y,theta):

    x=X[:,1:2]

    ##使用矩阵运算减少循环的次数,np.average()为矩阵所有元素的平均值

    x\_=np.average(x)

    #np.sum()为矩阵所有元素和,np.power(x,y)为x的y次方

    theta[0,1]=(y.T\*(x-x\_))/(np.sum(np.power(x,2))-np.power(np.sum(x),2)/len(x))

    theta[0,0]=np.sum(y-theta[0,1]\*x)/len(x)

    return theta

theta1=example(X,y,theta)

#以X第一列中从最小值到最大值为x轴,以新求得的theta最为参数求出y作为y轴画图

x1=np.linspace(X[:,1:].min(),X[:,1:].max(),100)

y1=x1\*theta1[0,1]+theta1[0,0]

plt.plot(x1,y1,'r',label='test1')

plt.plot(X[:,1:],y,'+')

plt.title("test1")

plt.show()

def h(X,theta):

    return X\*theta.T

def cost(X,y,theta):

    temp=np.power(h(X,theta)-y,2)

    return np.sum(temp)/(2\*X.shape[0])

def graddescent(X,y,theta,alpha,count):

    J=np.matrix(np.ones(count))

    for i in range(count):

        error=h(X,theta)-y

        #进行迭代,X.T为(2.6),error为(6,1)结果为(2,1)转置后可以直接用于迭代

        #(X.T\*error).T中的每个元素也为对应的偏导数,减少循环次数

        theta=theta-alpha\*(X.T\*error).T

        J[0,i]=cost(X,y,theta)

    return theta,J

#设定参数alpha和迭代次数count的值

alpha=0.00001

count=1000

J1=cost(X,y,theta1)

theta2,J2=graddescent(X,y,theta,alpha,count)

x2=np.linspace(X[:,1:].min(),X[:,1:].max(),100)

y2=x1\*theta1[0,1]+theta1[0,0]

plt.plot(x2,y2,'r',label='test2')

plt.plot(X[:,1:],y,'+')

plt.title("test2")

plt.show()

#查看结果，两组结果一致

print(theta1,J1)

print(theta2,J2[0,-1:])