**实验二**

姓名: 李昀哲 学号： 20123101

**1 题目和数据：**

题目：multiple\_linear\_regression

### 实验描述

实验1：用线性回归找到最佳拟合直线

实验2：局部加权线性回归找到最佳拟合直线

实验3：使用scikit-learn实现线性回归算法

**2 算法：**

### A.线性回归

线性回归方程的最小平方函数对一个或多个[自变量](https://baike.baidu.com/item/%E8%87%AA%E5%8F%98%E9%87%8F?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92/_blank)和[因变量](https://baike.baidu.com/item/%E5%9B%A0%E5%8F%98%E9%87%8F?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92/_blank)之间关系进行建模的一种[回归分析](https://baike.baidu.com/item/%E5%9B%9E%E5%BD%92%E5%88%86%E6%9E%90?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92/_blank)。这种函数是一个或多个称为回归系数的模型参数的线性组合。只有一个自变量的情况称为简单回归，大于一个自变量情况的叫做多元回归。

线性回归模型经常用最小二乘逼近来拟合，但他们也可能用别的方法来拟合，比如用最小化“拟合缺陷”在一些其他规范里（比如最小绝对误差回归），或者在桥回归中最小化最小二乘损失函数的惩罚.相反,最小二乘逼近可以用来拟合那些非线性的模型.因此，尽管“最小二乘法”和“线性模型”是紧密相连的，但他们是不能划等号的。

这里使用的方法正是**最小二乘法**：

一般来说，线性回归都可以通过[最小二乘法](https://baike.baidu.com/item/%E6%9C%80%E5%B0%8F%E4%BA%8C%E4%B9%98%E6%B3%95?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92/_blank)求出其方程，可以计算出对于y=bx+a的直线。

一般地，影响y的因素往往不止一个，假设有x1，x2，...，xk，k个因素，通常可考虑如下的线性关系式：

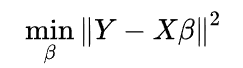


对y与x1，x2，...，xk同时作n次独立观察得n组观测值（xt1，xt2，...，xtk），t=1,2,...,n（n>k+1），它们满足关系式：



回归系数的计算如下所示：

回归系数的最小二乘估计（least square estimator of regression coefficient）简称LS估计。参数估计的一种方法。[线性回归模型](https://baike.baidu.com/item/%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92%E6%A8%A1%E5%9E%8B/19111343?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%9B%9E%E5%BD%92%E7%B3%BB%E6%95%B0/_blank)中，未知参数β的最小二乘估计为满足：



β是方程的解。此方程称为正规方程。由于线性回归模型中，X矩阵列满秩，故β可解除，记为。

### 局部线性回归

线性回归的一个问题是有可能出现欠拟合现象，因为它求的是具有小均方误差的无偏估 计。显而易见，如果模型欠拟合将不能取得好的预测效果。所以有些方法允许在估计中引入一 些偏差，从而降低预测的均方误差。

其中的一个方法是局部加权线性回归（Locally Weighted Linear Regression，LWLR）。在该方法中，我们给待预测点附近的每个点赋予一定的权重。与kNN一样，这种算法每次预测均需要事先选取出对应的数据子集。

### C.scikit-learn实现线性回归算法

sklearn.linear\_model模块提供了很多集成操作，线性回归、岭回归、贝叶斯回归等.

参数说明如下：

***fit\_intercept***：可选参数，布尔值，默认为True。是否对数据进行中心化。如果该变量为false，则表明数据已经进行了中心化，在下面的过程中不进行中心化处理，否则对输入数据进行中心化处理。

***normalize***：可选参数，布尔值，默认为False。是否对数据进行标准化处理。当fit\_intercept设置为false时，该参数会被忽略。如果该变量为true，则对数据进行标准化处理。需要在使用normalize=False的estimator调用fit之前使用sklearn.preprocessing.StandardScaler。

***copy\_X***:可选参数，布尔值，默认为True。该值为true，复制X；否则，重写X。

***n\_jobs***:可选参数，整型，默认为1。计算时设置的任务个数(number of jobs)。如果选择-1则代表使用所有的CPU。这一参数的对于目标个数>1（n\_targets>1）且足够大规模的问题有加速作用。

**返回值：**

***coef\_*** :数组型变量，shape为（n\_feature）或者(n\_targets,n\_features)。线性回归问题中的回归系数。如果输入为多目标问题，即fit二维数据，则返回一个二维数组，shape为（n\_targets,n\_features）；如果输入为单目标问题，返回一个一维数组。

***intercept\_***：数组型变量。线性模型中的独立项。

**方法：**

***decision\_function(X)*** 对训练数据X进行预测

***fit(X, y[, n\_jobs])*** 对训练集X, y进行训练。是对scipy.linalg.lstsq的封装

***get\_params([deep])*** 得到该估计器(estimator)的参数。

***predict(X)*** 使用训练得到的估计器对输入为X的集合进行预测（X可以是测试集，也可以是需要预测的数据）。

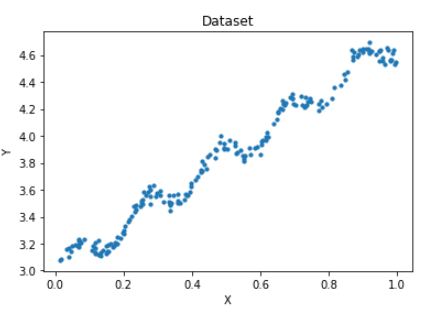
***score(X, y[,]sample\_weight)*** 返回对于以X为samples，以y为target的预测效果评分。

***set\_params(\*\*params)*** 设置估计器的参数

**3 代码及结果**

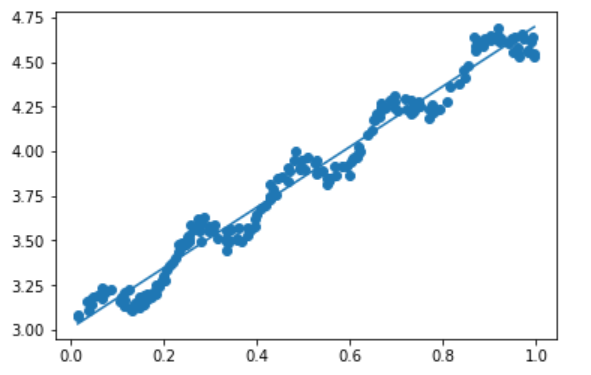
**数据读入及预处理**

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  def loadDataSet(filename):  """  函数说明:加载数据  Parameters:  fileName - 文件名  Returns:  xArr - x数据集  yArr - y数据集  """  # with open("./multiple\_linear\_regression/ex0.txt") as f:  # dataset = f.read()    xArr, yArr = [], []  for line in open(filename):  line = line.strip().split()    # 这一步中加float非常重要！！！！  xArr.append([1.0, float(line[1])])  yArr.append(float(line[2]))  return xArr, yArr  def plotDataSet():  """  函数说明:绘制数据集  Parameters:  无  Returns:  无  """  filename = "./multiple\_linear\_regression/ex0.txt"  xArr, yArr = loadDataSet(filename)  X = np.array(xArr)  Y = np.array(yArr)  amount = np.shape(X)[0]  print(amount)  x, y = [],[]  for i in range(amount):  x.append(X[i, 1])  y.append(Y[i])    plt.scatter(x, y, s=10)  plt.xlabel("X")  plt.ylabel("Y")  plt.title("Dataset")  plt.show()  if \_\_name\_\_ == '\_\_main\_\_':  xArr, yArr = loadDataSet("./multiple\_linear\_regression/ex0.txt")  plotDataSet() |



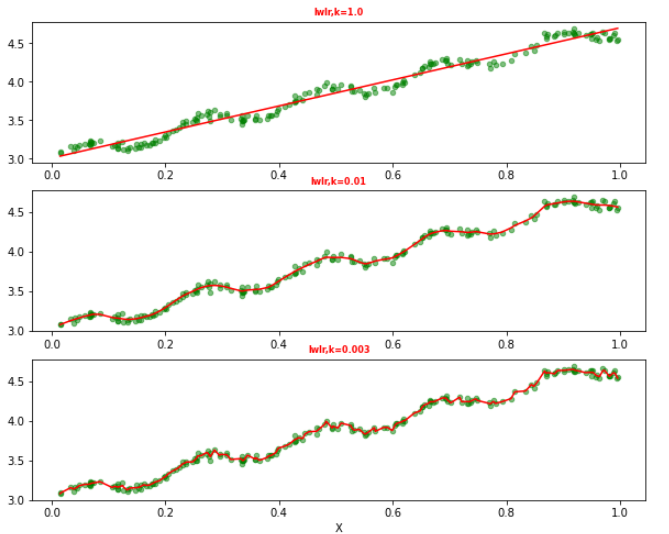
**最小二乘法拟合曲线**

|  |
| --- |
| from numpy import \*  import matplotlib.pyplot as plt  def standRegres(xArr,yArr):  """  计算最佳拟合直线  parameters:  xArr -给定的输入值  yArr -给定的输出值  return:  ws -回归系数  """  xMat=mat(xArr)#转化为矩阵  yMat=mat(yArr).T#求转置  xTx=xMat.T\*xMat  #判断是否行列式为0  if linalg.det(xTx) == 0.0:  print("This matrix is singular")  return  #行列式不为0，可逆，根据公式计算回归系数  ws=xTx.I\*(xMat.T\*yMat)  return ws    def showLinerRegre():  """  绘制最佳拟合直线  parameters:  null  return:  null  """  xArr ,yArr = loadDataSet('multiple\_linear\_regression/ex0.txt') #加载数据集  ws = standRegres(xArr,yArr) #得到回归系数  xMat=mat(xArr)  yMat=mat(yArr)  yHat=xMat\*ws  plt.scatter(xMat[:,1].flatten().A[0],yMat.T[:,0].flatten().A[0])  xCopy=xMat.copy()  xCopy.sort(0)  yHat=xCopy\*ws  plt.plot(xCopy[:,1],yHat)  plt.show()  # ...  # plt.show()    if \_\_name\_\_ =='\_\_main\_\_':  showLinerRegre() |



**局部线性回归**

|  |
| --- |
| def lwlr(testPoint, xArr, yArr, k=1.0):  """  计算回归系数  parameters:  testPoint -待预测数据  xArr -给定输入值  yArr -给定输出值  k -高斯核的k值，决定对附近的点赋予多大的权重  return:  testPoint \* ws -回归系数的估计值  """  #转为矩阵形式  xMat = mat(xArr); yMat = mat(yArr).T  #样本个数  m = shape(xMat)[0]  weights = mat(eye((m))) #创建m\*m的单位矩阵，作为初始化对角权重矩阵  for j in range(m): #遍历数据集计算每个样本的权重  #计算预测点与该样本的偏差  diffMat =testPoint-xMat[j, :]  #根据偏差利用函数赋予该样本相应的权重  weights[j, j] = exp(diffMat \* diffMat.T/(-2.0 \* k\*\*2))  #将权重矩阵应用到公式中  xTx = xMat.T \* (weights \* xMat) #m\*m矩阵  #计算行列式值是否为0，即确定是否可逆  if linalg.det(xTx) == 0.0:  return  ws = xTx.I \* (xMat.T \* (weights \* yMat)) #计算回归系数，m\*1矩阵  #返回测试点的预测值  return testPoint\*ws  def lwlrTest(testArr, xArr, yArr,k=1.0):  """  测试函数  parameters:  testArr -测试数据集  xArr -给定输入值  yArr -给定输出值  k -高斯核的k值  return:  yHat -预测值  """  m=np.shape(testArr)[0]  yHat=np.zeros(m)#用于存储预测结果  for i in range(m):  yHat[i]=lwlr(testArr[i],xArr,yArr,k)  return yHat  def plotlwlrRegression():  """  绘制多条局部加权回归曲线  parameters:  无  returns:  无  """  xArr, yArr = loadDataSet('multiple\_linear\_regression/ex0.txt') #加载数据集  yHat\_1 = lwlrTest(xArr, xArr, yArr, 1.0) #根据局部加权线性回归计算yHat  yHat\_2 = lwlrTest(xArr, xArr, yArr, 0.01) #根据局部加权线性回归计算yHat  yHat\_3 = lwlrTest(xArr, xArr, yArr, 0.003) #根据局部加权线性回归计算yHat  xMat = mat(xArr) #创建xMat矩阵  yMat = mat(yArr) #创建yMat矩阵  srtInd = xMat[:, 1].argsort(0) #排序，返回索引值  xSort = xMat[srtInd][:,0,:]  fig, axs = plt.subplots(nrows=3, ncols=1,sharex=False, sharey=False, figsize=(10,8))  axs[0].plot(xSort[:, 1], yHat\_1[srtInd], c = 'red') #绘制回归曲线  axs[1].plot(xSort[:, 1], yHat\_2[srtInd], c = 'red') #绘制回归曲线  axs[2].plot(xSort[:, 1], yHat\_3[srtInd], c = 'red') #绘制回归曲线  axs[0].scatter(xMat[:,1].flatten().A[0], yMat.flatten().A[0], s = 20, c = 'green', alpha = .5) #绘制样本点  axs[1].scatter(xMat[:,1].flatten().A[0], yMat.flatten().A[0], s = 20, c = 'green', alpha = .5) #绘制样本点  axs[2].scatter(xMat[:,1].flatten().A[0], yMat.flatten().A[0], s = 20, c = 'green', alpha = .5) #绘制样本点  #设置标题,x轴label,y轴label  axs0\_title\_text = axs[0].set\_title(u'lwlr,k=1.0')  axs1\_title\_text = axs[1].set\_title(u'lwlr,k=0.01')  axs2\_title\_text = axs[2].set\_title(u'lwlr,k=0.003')  plt.setp(axs0\_title\_text, size=8, weight='bold', color='red')  plt.setp(axs1\_title\_text, size=8, weight='bold', color='red')  plt.setp(axs2\_title\_text, size=8, weight='bold', color='red')  plt.xlabel('X')  plt.show()  if \_\_name\_\_ == '\_\_main\_\_':  plotlwlrRegression() |



**Scikit-learn包算法调用**

|  |
| --- |
| #!/usr/bin/env python  #-\*- coding:utf-8 -\*-  import matplotlib.pyplot as plt  from numpy import \*  from sklearn import datasets, linear\_model  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error, r2\_score  def loadDataSet(fileName):  numFeat=len(open(fileName).readline().split('\t'))-1  with open(fileName) as fr:  #~ numFeat=len(fr.readline().split('\t'))-1  dataMat=[];labelMat=[]  for line in fr.readlines():  lineArr=[]  curLine=line.strip().split('\t')  for i in range(numFeat):  lineArr.append(float(curLine[i]))  dataMat.append(lineArr)  labelMat.append(float(curLine[-1]))  return dataMat,labelMat  return dataMat,labelMat  if \_\_name\_\_ =='\_\_main\_\_':  dataX, dataY =loadDataSet('multiple\_linear\_regression/ex0.txt')  matX=mat(dataX);matY=mat(dataY).T #将数据保存到矩阵中  regr = LinearRegression() #生成线性回归模型  regr.fit(matX,matY)#对训练集X, y进行训练。  #填充训练数据 matX(n\_samples,n\_features);matY(n\_samples,n\_targets)  xCopy = matX.copy()  xCopy.sort(0)  predictY = regr.predict(xCopy) #得到模型预测值 predict使用训练得到的估计器对输入为X的集合进行预测  plt.scatter(matX[:,1].flatten().A[0],matY[:,0].flatten().A[0],s=20,color='green',alpha=.5) #绘制散点图  plt.plot(xCopy[:,1],predictY,color='red',linewidth=1) #绘制最佳拟合直线    plt.xticks(())  plt.yticks(())  plt.show() |

