菊安酱的机器学习第7期

菊安酱的直播间：https://live.bilibili.com/14988341

每周一晚8:00 菊安酱和你不见不散哦~(^o^)/~

更新日期：2018-12-17

作者：菊安酱

课件内容说明：

- 本文为作者原创，转载请注明作者和出处
- 如果想获得此课件及录播视频，可扫描左边二维码，回复"k"进群
- 如果想获得2小时完整版视频，可扫描右边二维码或点击如下链接
- 若有任何疑问，请给作者留言。

交流群二维码　　完整版视频及课件

直播视频及课件：http://www.peixun.net/view/1278.html

完整版视频及课件：http://edu.cda.cn/course/966

12期完整版课纲

直播时间：每周一晚8:00

直播内容：

时间	期数	算法
2018/11/05	第1期	k-近邻算法
2018/11/12	第2期	决策树
2018/11/19	第3期	朴素贝叶斯
2018/11/26	第4期	Logistic回归
2018/12/03	第5期	支持向量机
2018/12/10	第6期	AdaBoost 算法
2018/12/17	第7期	线性回归
2018/12/24	第8期	树回归
2018/12/31	第9期	K-均值聚类算法
2019/01/07	第10期	Apriori 算法
2019/01/14	第11期	FP-growth 算法
2019/01/21	第12期	奇异值分解SVD

线性回归

我们前6期的内容都在讲解分类问题，这一期我们正式进入回归问题。虽然分类问题和回归问题都属于机器学习有监督算法的范畴，但实际上，回归问题要远比分类问题复杂。首先是关于输出结果的对比，分类模型最终输出的结果是离散型变量，而离散变量本身包含信息量较少，其本身并不具备代数运算性质，因此其评价指标体系也较为简单，最常用的就是混淆矩阵和ROC曲线。而回归问题最终输出的是连续变量，其本身不仅能够进行代数运算，而且还具有统计学意义的分布特征，因此其评价指标将更为复杂。同时在描绘客观事物规律上，回归问题是一种更加"精致"的方法，希望对事物运行的更底层原理进行挖掘，也就是说回归类问题的模型更加全面、完善地描绘事物客观规律，从而能够得出更加细粒度的结论。因此回归问题的模型往往更加复杂，建模需要的数据所提供的信息量也越多，进而在建模过程中可能遇到的问题也越多。

一、什么是回归

回归的目的是预测数值型的目标值。最直接的办法是依据输入写出一个目标值的计算公式。假如你想要预测某位小姐姐男友汽车的功率大小，可能会这么计算：

$$HorsePower = 0.0015 * annualSalary - 0.99 * hoursListeningToPublicRadio$$

翻译成中文就是：

$$\text{小姐姐男友汽车功率} = 0.0015 * \text{男友年薪} - 0.99 * \text{收听公共广播时间}$$

这就是所谓的**回归方程**（regression equation）,其中的0.0015和-0.99称作**回归系数**（regression weights）,求这些回归系数的过程就是回归。一旦有了这些回归系数，再给定输入值，我们计算出回归系数与输入值的乘积之和，就可以得到最后的预测值。

回归分为线性回归和非线性回归，上述功率计算公式也可以写做：

$$HorsePower = 0.0015 * annualSalary / hoursListeningToPublicRadio$$

这就是一个非线性回归的例子，但我们不对此做深入讨论。这里主要讨论的是**线性回归。**

二、线性回归

1. 简单线性回归

简单线性回归也叫一元线性回归，先来看一元线性方程：

$$y = b + wx$$

这个直线方程相信大家都不陌生，这就是最简单的线性回归模型。其中，w是直线的**斜率**，b是直线的**截距。**

写成矩阵形式为：

$$y = \boldsymbol{X}^T \boldsymbol{\omega}$$

其中

$$\boldsymbol{X} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad (\text{令} \, x_0 = 1), \quad \boldsymbol{\omega} = \begin{bmatrix} b \\ w \end{bmatrix}$$

假如我们有m个训练样本，则

$$\boldsymbol{X}^T=\begin{bmatrix} x_0^1, x_1^1 \\ x_0^2, x_1^2 \\ \ldots \\ x_0^m, x_1^m \end{bmatrix}$$

2. 多元线性回归

多元线性方程：

$$y=w_0+w_1x_1+w_2x_2+\ldots+w_nx_n$$

写成矩阵形式为：

$$y=\boldsymbol{X}^T\boldsymbol{\omega}$$

假如我们有m个训练样本，则

$$\boldsymbol{X}^T=\begin{bmatrix} x_0^1, x_1^1, \ldots, x_n^1 \\ x_0^2, x_1^2, \ldots, x_n^2 \\ \ldots \\ x_0^m, x_1^m, \ldots, x_n^m \end{bmatrix}, \boldsymbol{\omega}=\begin{bmatrix} w_0 \\ w_1 \\ \ldots \\ w_n \end{bmatrix}$$

3. 线性回归的损失函数

在第4期逻辑回归中有跟大家提到过损失函数。损失函数其实就是衡量预测值与真实值之间的差距的函数。这里采用平方误差作为线性回归的损失函数：

$$\begin{aligned} SSE &= \sum_{i=1}^{m}(y_i-\hat{y}_i)^2 \\ &= \sum_{i=1}^{m}(y_i-x_i^Tw)^2 \end{aligned}$$

【注】用平方而没用误差绝对值是因为：平方对于后续求导比较方便。

用矩阵表示可以写做：

$$(y-\boldsymbol{X}w)^T(y-\boldsymbol{X}w)$$

对w求导，得到$\boldsymbol{X}^T(y-\boldsymbol{X}w)$，令其等于0，解出w如下：

$$\hat{w}=(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^Ty$$

【公式推导过程，见完整版视频】

上述的求解过程称为**最小二乘法**（ordinary least squares），简称**OLS。**

Python中对于矩阵的各种操作可以通过Numpy库的一些方法来实现，非常方便。但在这个代码实现中需要注意：**X矩阵不能为奇异矩阵，否则是无法求解矩阵的逆的。**

【补充】

名称	英文	公式	别称	英文
残差平方和SSE	Sum of Squares for Error	SSE = $\sum_{i=1}^{m}(y_i-\hat{y}_i)^2$	剩余平方和RSS	residual sum of squares
回归平方和SSR	Sum of Squares for Regression	SSR = $\sum_{i=1}^{m}(\hat{y}_i-\bar{y})^2$	解释平方和ESS	explained sum of squares
总离差平方和SST	Sum of Squares for Total	SST = $\sum_{i=1}^{m}(y_i-\bar{y})^2$	总离差平方和TSS	total sum of squares

$$SST = SSR + SSE$$
$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

4. 简单线性回归的python实现

4.1 导入相关包

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['simhei']
%matplotlib inline
```

4.2 导入数据集并探索数据

```
ex0 = pd.read_table('ex0.txt',header=None)
ex0.head()
ex0.shape
```

4.3 构建辅助函数

```
"""函数功能：输入DF数据集（最后一列为标签），返回特征矩阵和标签矩阵"""
def get_Mat(dataSet):
    xMat = np.mat(dataSet.iloc[:,:-1].values)
    yMat = np.mat(dataSet.iloc[:,-1].values).T
    return xMat,yMat
```

```
#查看函数运行结果
xMat,yMat = get_Mat(ex0)
```

```
"""函数功能：数据集可视化"""
def plotShow(dataSet):
    xMat,yMat=get_Mat(dataSet)
    plt.scatter(xMat.A[:,1],yMat.A,c='b',s=5)
    plt.show()
```

```
#可视化ex0数据集
plotShow(ex0)
```

4.5 计算回归系数

```
"""
函数功能：计算回归系数
参数说明：
    dataSet：原始数据集
返回：
    ws：回归系数
"""
def standRegres(dataSet):
    xMat,yMat =get_Mat(dataSet)
    xTx = xMat.T*xMat
    if np.linalg.det(xTx)==0:
        print('矩阵为奇异矩阵，无法求逆')
        return
    ws=xTx.I*(xMat.T*yMat)
    return ws
```

说明：det(A)指的是矩阵A的行列式（determinant），如果det(A)=0，则说明矩阵A是奇异矩阵，不可逆。

```
ws = standRegres(ex0)
ws
```

4.6 绘制最佳拟合直线

```
"""
函数功能：绘制散点图和最佳拟合直线
"""
def plotReg(dataSet):
    xMat,yMat=get_Mat(dataSet)
    plt.scatter(xMat.A[:,1],yMat.A,c='b',s=5)
    ws = standRegres(dataSet)
    yHat = xMat*ws
    plt.plot(xMat[:,1],yHat,c='r')
    plt.show()
```

```
#绘制ex0数据集的散点图和最佳拟合直线
plotReg(ex0)
```

4.7 计算相关系数

在python中，Numpy库提供了相关系数的计算方法：可以通过函数np.corrcoef（yEstimate,yActual）来计算预测值和真实值之间的相关性。这里需要保证的是，输入的两个参数都是**行向量**。

```
xMat,yMat =get_Mat(ex0)
ws=standRegres(ex0)
yHat = xMat*ws
np.corrcoef(yHat.T,yMat.T) #保证两个都是行向量
```

该矩阵包含所有两两组合的相关系数。可以看到，对角线上全部为1.0，因为自身匹配肯定是最完美的，而yHat和yMat的相关系数为0.98。看起来似乎是一个不错的结果。但是仔细观察数据集，会发现数据呈现有规律的波动，但是直线似乎没有很好的捕捉到这些波动。

三、局部加权线性回归

线性回归的一个问题时有可能出现欠拟合现象，为了解决这个问题，我们可以采用的一个方法是**局部加权线性回归**(Locally Weighted Linear Regression)，简称**LWLR**。该算法思想就是给带预测点附近的每个点赋予一定的权重，然后按照简单线性回归求解w方法求解。与KNN一样，这种算法每次预测均需要实现选取出对应的数据子集。该算法解出回归系数w的形式如下：

$$\hat{w} = (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{W}y$$

其中，$\boldsymbol{W}$ 是一个矩阵，用来给每个数据点赋予权重。

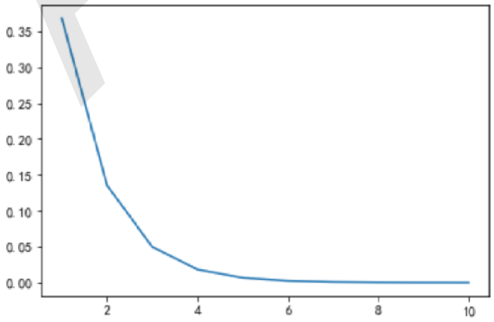
【推导过程，完整版课件里面会讲~】

LWLR使用"核"（与支持向量机中的核类似）来对附近的点赋予更高的权重。核的类型可以自由选择，最常用的就是**高斯核**，高斯核对应的权重如下：

$$w(i,i) = exp\left[\frac{|x^i - x|^2}{-2k^2}\right]$$

这样我们就构建了一个只含对角元素的权重矩阵$\boldsymbol{W}$，并且点x与x(i)越近，w(i,i)将会越大。

```
a = np.arange(1,11,1)
plt.plot(a,np.exp(-a));
```

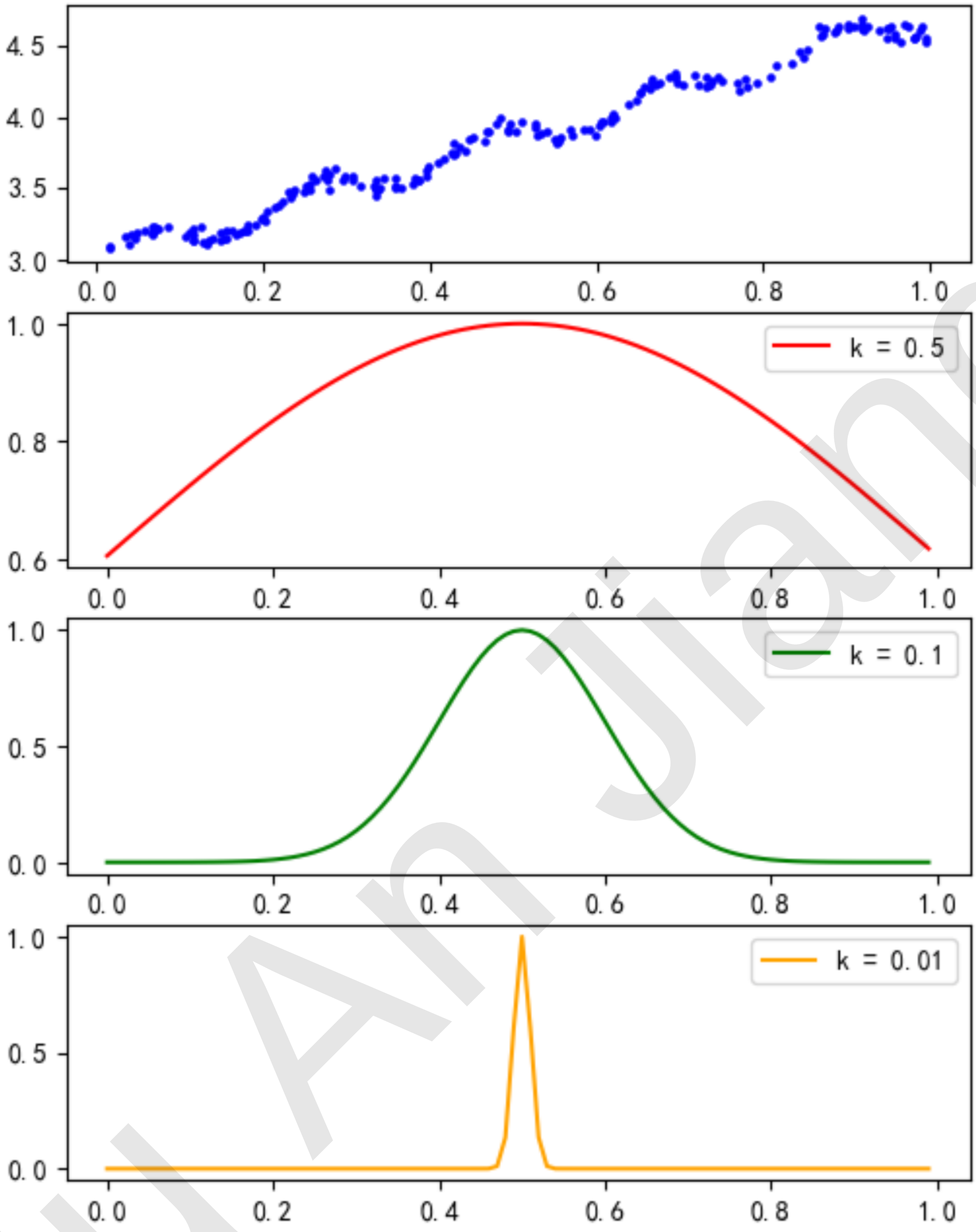


这个公式包含一个需要用户指定的参数k,它决定了对附件的带你赋予多大的权重，这也是使用局部加权线性回归（LWLR）时唯一需要考虑的参数。下面我们来看一下不同参数k与权重的关系。

```
#此段代码供大家参考
xMat,yMat = get_Mat(ex0)
x=0.5
xi = np.arange(0,1.0,0.01)
k1,k2,k3=0.5,0.1,0.01
```

```
w1 = np.exp((xi-x)**2/(-2*k1**2))
w2 = np.exp((xi-x)**2/(-2*k2**2))
w3 = np.exp((xi-x)**2/(-2*k3**2))
#创建画布
fig = plt.figure(figsize=(6,8),dpi=100)
#子画布1，原始数据集
fig1 = fig.add_subplot(411)
plt.scatter(xMat.A[:,1],yMat.A,c='b',s=5)
#子画布2，w=0.5
fig2 = fig.add_subplot(412)
plt.plot(xi,w1,color='r')
plt.legend(['k = 0.5'])
#子画布3，w=0.1
fig3 = fig.add_subplot(413)
plt.plot(xi,w2,color='g')
plt.legend(['k = 0.1'])
#子画布4，w=0.01
fig4 = fig.add_subplot(414)
plt.plot(xi,w3,color='orange')
plt.legend(['k = 0.01'])
plt.show()
```

运行结果如下所示：



这里假定我们预测的点是x=0.5，最上面的图是原始数据集，从下面三张图可以看出随着k的减小，被用于训练模型的数据点越来越少。

1. 构建LWLR函数

这个过程与简单线性函数的基本一致，唯一不同的是加入了权重weights，这里我将权重参数求解和预测yHat放在了一个函数里面。

```
"""
```

```
函数功能：计算局部加权线性回归的预测值
参数说明：
    testMat:测试集
    xMat:训练集的特征矩阵
    yMat:训练集的标签矩阵
返回：
    yHat: 函数预测值
"""
def LWLR(testMat,xMat,yMat,k=1.0):
    n=testMat.shape[0]
    m=xMat.shape[0]
    weights =np.mat(np.eye(m))
    yHat = np.zeros(n)
    for i in range(n):
        for j in range(m):
            diffMat = testMat[i]-xMat[j]
            weights[j,j]=np.exp(diffMat*diffMat.T/(-2*k**2))
        xTx = xMat.T*(weights*xMat)
        if np.linalg.det(xTx)==0:
            print('矩阵为奇异矩阵，不能求逆')
            return
        ws = xTx.I*(xMat.T*(weights*yMat))
        yHat[i]= testMat[i]*ws
    return yHat
```

2. 不同k值的结果图

我们调整k值，然后查看不同k值对模型的影响

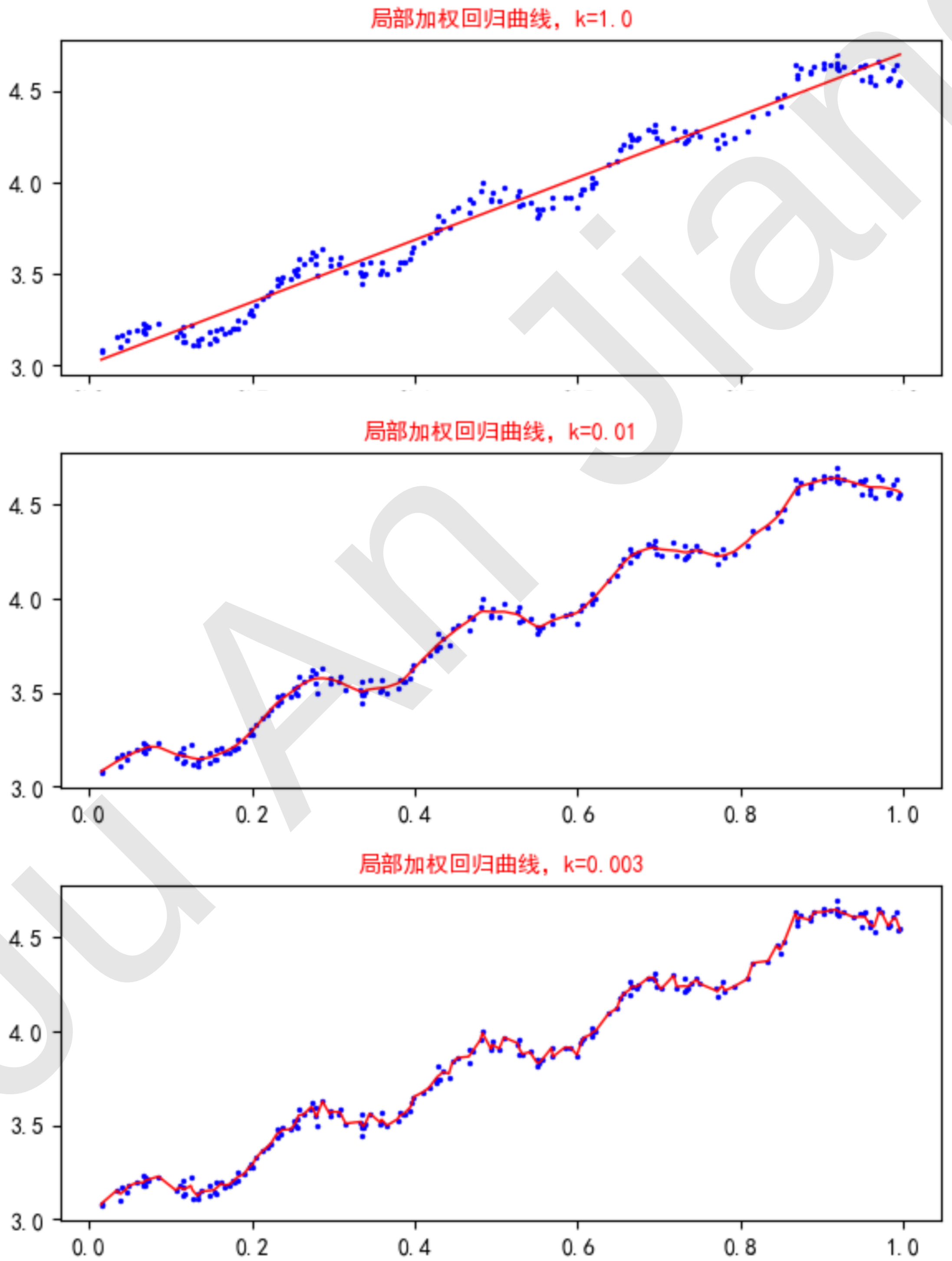
```
xMat,yMat = get_Mat(ex0)
#将数据点排列（argsort()默认升序排列，返回索引）
srtInd = xMat[:,1].argsort(0)
xSort=xMat[srtInd][:,0]

#计算不同k取值下的y估计值yHat
yHat1 = LWLR(xMat,xMat,yMat,k=1.0)
yHat2 = LWLR(xMat,xMat,yMat,k=0.01)
yHat3 = LWLR(xMat,xMat,yMat,k=0.003)

#创建画布
fig = plt.figure(figsize=(6,8),dpi=100)
#子图1绘制k=1.0的曲线
fig1=fig.add_subplot(311)
plt.scatter(xMat[:,1].A,yMat.A,c='b',s=2)
plt.plot(xSort[:,1],yHat1[srtInd],linewidth=1,color='r')
plt.title('局部加权回归曲线，k=1.0',size=10,color='r')
#子图2绘制k=0.01的曲线
fig2=fig.add_subplot(312)
plt.scatter(xMat[:,1].A,yMat.A,c='b',s=2)
plt.plot(xSort[:,1],yHat2[srtInd],linewidth=1,color='r')
plt.title('局部加权回归曲线，k=0.01',size=10,color='r')
```

```
#子图3绘制k=0.003的曲线
fig3=fig.add_subplot(313)
plt.scatter(xMat[:,1].A,yMat.A,c='b',s=2)
plt.plot(xSort[:,1],yHat3[srtInd],linewidth=1,color='r')
plt.title('局部加权回归曲线，k=0.003',size=10,color='r')
#调整子图的间距
plt.tight_layout(pad=1.2)
plt.show()
```

运行结果如下：



这三个图是不同平滑值绘出的局部加权线性回归结果。当k=1.0时，模型的效果与最小二乘法差不多；k=0.01时，该模型基本上已经挖出了数据的潜在规律，当继续减小到k=0.003时，会发现模型考虑了太多的噪音，进而导致了过拟合现象。

```
#四种模型相关系数比较
np.corrcoef(yHat.T,yMat.T)      #最小二乘法
np.corrcoef(yHat1.T,yMat.T)     #k=1.0模型
np.corrcoef(yHat2.T,yMat.T)     #k=0.01模型
np.corrcoef(yHat3.T,yMat.T)     #k=0.003模型
```

局部加权线性回归也存在一个问题——增加了计算量，因为它对每个点预测都要使用整个数据集。从不同k值的结果图中可以看出，当k=0.01时模型可以很好地拟合数据潜在规律，但是同时看一下，k值与权重关系图，可以发现，当k=0.01时，大部分数据点的权重都接近0，也就是说他们基本上可以不用带入计算。所以如果一开始就能去掉这些数据点的计算，那么就可以大大减少程序的运行时间了，从而缓解计算量增加带来的问题。后面我们会讲解这个操作。

四、案例：预测鲍鱼的年龄

接下来，我们将回归用于真实数据。此案例所用数据集来自UCI数据集，记录了鲍鱼（一种介壳类水生生物）的一些相关属性，根据这些属性来预测鲍鱼的年龄。

鲍鱼年龄的计算主要是通过一些比较容易获得的量测数据，以及通过锥体切割贝壳，染色并在显微镜下数出环数来确定的 - 这是一项无聊且耗时的任务（数据提供者的吐槽……）。

Name	数据类型	单位	描述
性别	离散型	——	公，母，婴儿（1,-1,0）
长度	连续型	毫米	贝壳最长的部分
直径	连续型	毫米	垂直于长度
高度	连续型	毫米	壳里的肉的高度
整体重量	连续型	克	整个鲍鱼的重量
肉重量	连续型	克	鲍鱼肉的重量
内脏重量	连续型	克	内脏的重量 (去血后)
壳重	连续型	克	干了之后的壳重
年龄	整数型	——	鲍鱼的年龄

1. 导入数据集

```
abalone = pd.read_table('abalone.txt',header=None)
abalone.columns=['性别','长度','直径','高度','整体重量','肉重量','内脏重量','壳重','年龄']
abalone.head()
abalone.shape
abalone.info()
```

2. 切分训练集和测试集

```
"""
函数功能：切分训练集和测试集
参数说明：
    dataSet：原始数据集
    rate:训练集比例
返回：
    train,test：切分好的训练集和测试集
"""
def randSplit(dataSet,rate):
    m = dataSet.shape[0]
    n = int(m*rate)
    train = dataSet.iloc[:n,:]
    test = dataSet.iloc[n:m,:]
    test.index=range(test.shape[0])
    return train,test
```

```
train,test = randSplit(abalone,0.8)
train.head()
train.shape
test.head()
test.shape
```

3. 构建辅助函数

```
"""
函数功能：计算误差平方和SSE
参数说明：
    yMat：真实值
    yHat：估计值
返回：
    SSE：误差平方和
"""
def sseCal(yMat, yHat):
    sse = ((yMat.A.flatten()-yHat)**2).sum()
    return sse
```

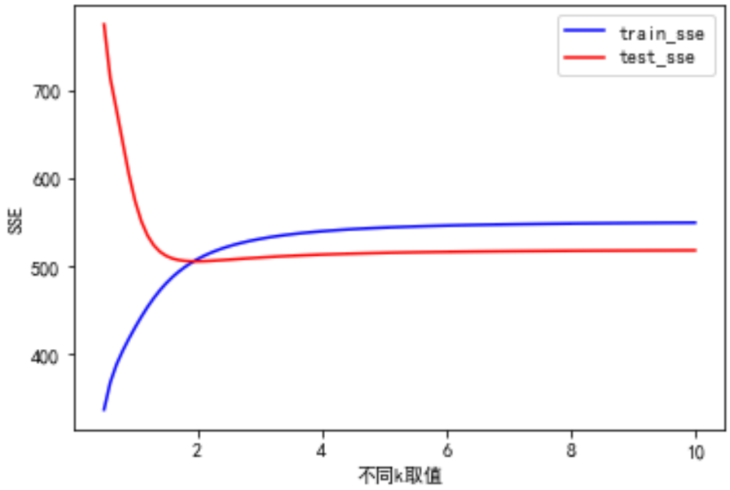
4. 构建加权线性模型

因为数据量太大，计算速度极慢，所以此处选择前100个数据作为训练集，第100-200个数据作为测试集。

```
"""
函数功能：绘制不同k取值下，训练集和测试集的SSE曲线
"""
def showPlot(abalone):
    abX,abY = get_Mat(abalone)
    train_sse = []
    test_sse = []
    for k in np.arange(0.5,10.1,0.1):
        yHat1 = LWLR(abX[:99],abX[:99],abY[:99],k)
        sse1 = sseCal(abY[:99], yHat1)
        train_sse.append(sse1)

        yHat2 = LWLR(abX[100:199],abX[:99],abY[:99],k)
        sse2 = sseCal(abY[100:199], yHat2)
        test_sse.append(sse2)
    plt.plot(np.arange(0.5,10.1,0.1),train_sse,color='b')
    plt.plot(np.arange(0.5,10.1,0.1),test_sse,color='r')
    plt.xlabel('不同k取值')
    plt.ylabel('SSE')
    plt.legend(['train_sse','test_sse'])
```

运行结果：



【以下将会在完整版视频中给大家讲解】

数学公式的推导过程

五、岭回归

六、lasso

七、向前逐步回归

八、案例：预测乐高玩具套装的价格

其他

- 菊安酱的直播间：https://live.bilibili.com/14988341
- 下周一（2018/12/24）将讲解**树回归**，欢迎各位进入菊安酱的直播间观看直播
- 如有问题，可以给我留言哦~