机器学习——线性回归基础

对于回归我们首先简单地理解线性回归以及局部加权线性回归，因为线性回归以及非线性回归还是有很大区别，所以我们这一次只讲线性回归。一般线性回归就可以理解为将输入项分别乘以一些常量，再将结果加起来得到输出，而我们的目的一般就是求出这些常量，也就是回归系数，一旦有了回归系数，我们就可以将之视为一个模型，做预测就可以建立在这个模型的基础上，直接乘上输入值再相加，就得到了预测值。所以我们首先理解这样的一个回归方程：

回归方程：

对于一般的线性方程，我们可以用下面的式子表示：

$$Y = X^T W$$

再把这样的矩阵形式拆分出来，我们可以有：

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = X^T W = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}^T \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

对于一般的问题我们是知道输入和输出值的，所以我们就是要求出回归系数，显然我们是要使得乘上回归系数后的输入项无限趋近于输出值，这样的回归系数就是最接近模型的系数，所以我们可以定义这样的平方误差来衡量回归系数的偏差：

$$\sum_{i=1}^{m} (y_i - x_i^T w)^2$$

如果用矩阵来表示就是：

$$(y - Xw)^T (y - Xw)$$

我们的目的就是最小化这个目标函数，也就是说，这个误差越小，拟合效果越好，因此我们对这个函数进行求导，并令它为零，得到最优的回归系数：

$$\begin{aligned} &\frac{\partial (y - Xw)^T (y - Xw)}{\partial w} \\ &= \frac{\partial (y^T y - y^T Xw - w^T X^T y + w^T X^T Xw)}{\partial w} \\ &= \frac{\partial y^T y}{\partial w} - \frac{\partial y^T Xw}{\partial w} - \frac{\partial w^T X^T y}{\partial w} + \frac{\partial w^T X^T Xw}{\partial w} \\ &= 0 + X^T y - X^T y + 2X^T Xw \\ &= 2X^T (y - Xw) = 0 \end{aligned}$$

得到：

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

因此我们得到一个普适性的线性回归解就是如上面的公式得到的。

我们将对一个简单地数据集进行线性回归：

```
import numpy as np
import matplotlib.pyplot as plt
```

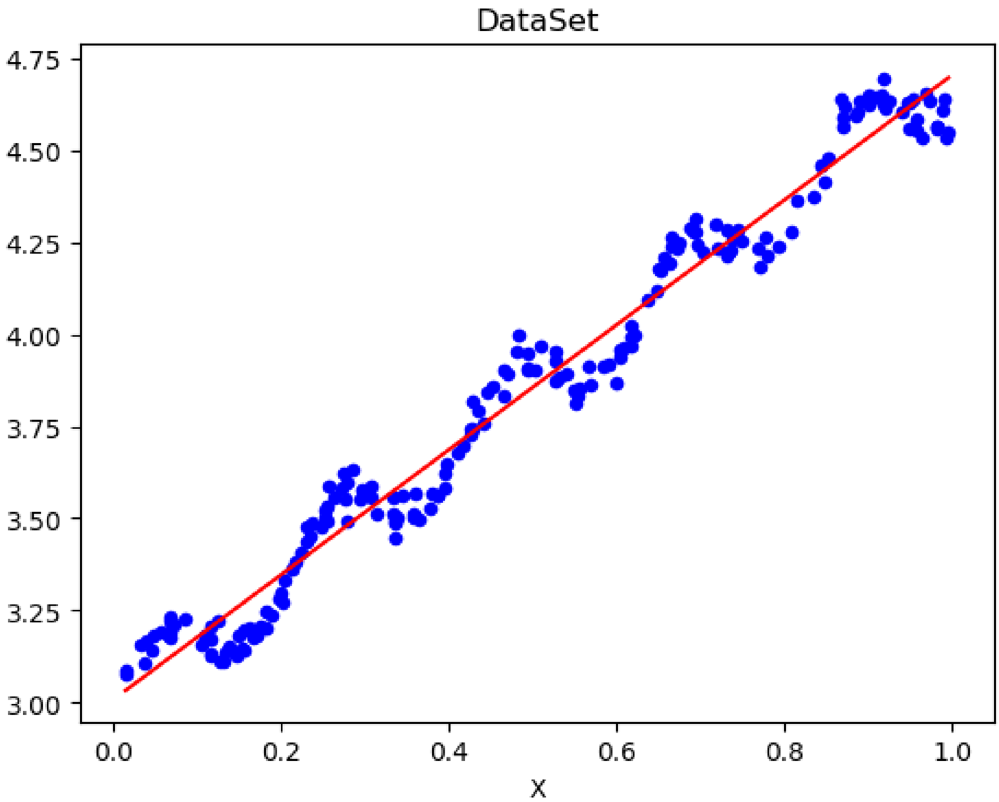
```python
def loadDataSet(filename):
    fr=open(filename)
    xArr=[]
    yArr=[]
    numFeat=len(open(filename).readline().split('\t'))-1
    for line in fr.readlines():
        lineArr=[]
        curline=line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curline[i]))
        xArr.append(lineArr)
        yArr.append(float(curline[-1]))
    return xArr,yArr

def standRegres(xArr,yArr):
    xMat=np.mat(xArr)
    yMat=np.mat(yArr).T
    XTX=xMat.T*xMat
    if np.linalg.det(XTX)==0.0:
        print("矩阵奇异")
    w=XTX.I*(xMat.T*yMat)
    return w

def plotRegress():
    xArr,yArr=loadDataSet('ex0.txt')
    w=standRegres(xArr,yArr)
    xMat=np.mat(xArr)
    yMat=np.mat(yArr)
    xCopy=xMat.copy()
    xCopy.sort(0)
    yHat=xCopy*w
    fig=plt.figure()
    ax=fig.add_subplot(111)
    ax.plot(xCopy[:,1],yHat,c='red')
    ax.scatter(xMat[:,1].flatten().A[0],yMat.flatten().A[0],s=20,c='blue')
    plt.title('DataSet')
    plt.xlabel('X')
    plt.show()

plotRegress()
```

可以得到如下拟合:



看完了线性回归，我们再深入一点来看一下局部加权的线性回归：

局部加权线性回归：

线性加权的问题还是有的，它可能出现欠拟合现象，因为它所求的是最小均方误差的无偏估计，如果我们引入一些偏差，可以降低预测的均方误差，那将改善这里面的预测效果，因此我们有了局部加权线性回归。这个方法对每一个预测点赋予了一定的权重，形式如下：

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

这其中的W就是一个加权矩阵，同样涉及到与支持向量机相似的概念，它使用核来对附近的点赋予更加高的权重，我们同样选择高斯核，权重形式如下：

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

因此我们可以根据调整k值来调节回归效果，同样的对上面例子的数据我们进行一个局部加权线性回归，代码如下：

```
import numpy as np
import matplotlib.pyplot as plt

def loadDataSet(filename):
    fr=open(filename)
    xArr=[]
    yArr=[]
    numFeat=len(open(filename).readline().split('\t'))-1
    for line in fr.readlines():
        lineArr=[]
        curline=line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curline[i]))
```

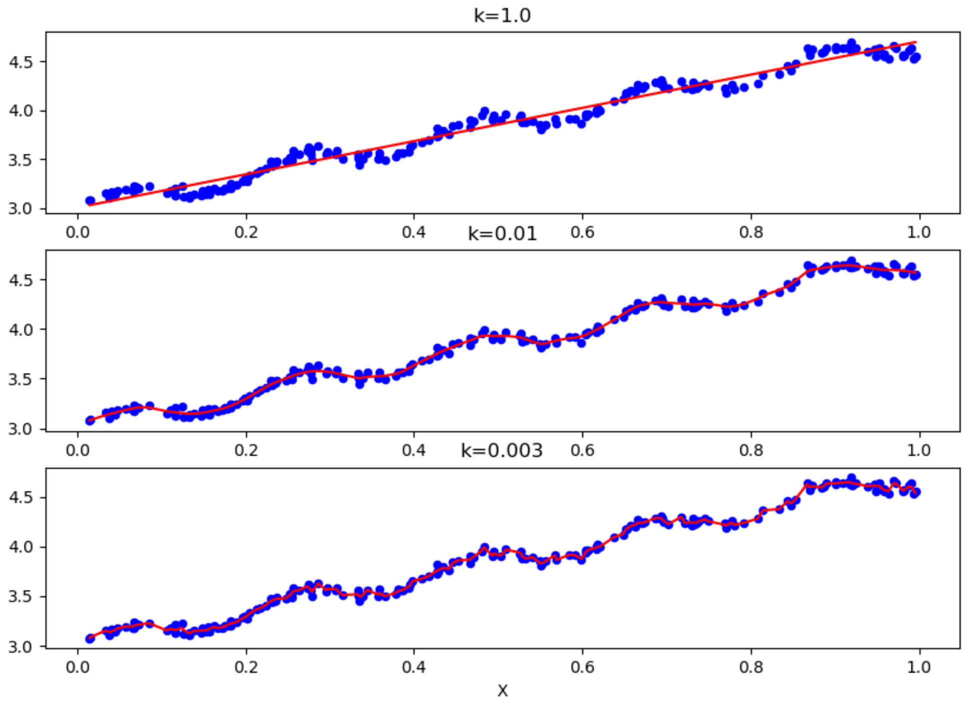
```
        xArr.append(lineArr)
        yArr.append(float(curline[-1]))
    return xArr,yArr

def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat=np.mat(xArr)
    yMat=np.mat(yArr).T
    m=np.shape(xMat)[0]
    weight=np.mat(np.eye((m)))
    for j in range(m):
        diffMat=testPoint-xMat[j,:]
        weight[j,j]=np.exp(diffMat*diffMat.T/(-2.0*k**2))
    xTx=xMat.T*weight*xMat
    if np.linalg.det(xTx)==0.0:
        print('矩阵奇异')
        return
    w=xTx.I*(xMat.T*(weight*yMat))
    return testPoint*w
def lwlrTest(testArr,xArr,yArr,k=1.0):
    m=np.shape(testArr)[0]
    yHat=np.zeros(m)
    for i in range(m):
        yHat[i]=lwlr(testArr[i],xArr,yArr,k)
    return yHat

def plotlwlr():
    xArr,yArr=loadDataSet('ex0.txt')
    yHat_1=lwlrTest(xArr,xArr,yArr,1.0)
    yHat_2=lwlrTest(xArr,xArr,yArr,0.01)
    yHat_3=lwlrTest(xArr,xArr,yArr,0.003)
    xMat=np.mat(xArr)
    yMat=np.mat(yArr)
    srtInd=xMat[:,1].argsort(0)
    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')
    axs[1].plot(xSort[:,1],yHat_2[srtInd],c='red')
    axs[2].plot(xSort[:,1],yHat_3[srtInd],c='red')
    axs[0].scatter(xMat[:,1].flatten().A[0],yMat.flatten().A[0],s=20,c='blue')
    axs[1].scatter(xMat[:,1].flatten().A[0],yMat.flatten().A[0],s=20,c='blue')
    axs[2].scatter(xMat[:,1].flatten().A[0],yMat.flatten().A[0],s=20,c='blue')
    axs0_title_text = axs[0].set_title(u'k=1.0')
    axs1_title_text = axs[1].set_title(u'k=0.01')
    axs2_title_text = axs[2].set_title(u'k=0.003')
    plt.xlabel('X')
    plt.show()

plotlwlr()
```

回归效果根据k值变化：



接下来我们就来一个真实例子进行实战演练，该例子是一个鲍鱼年龄的预测，同样给了一堆数据集（类似于上面的样式，只是维数多了一些），我们只要对这个回归函数的系数求出来就可以根据输入数进行预测：

```
import numpy as np
import matplotlib.pyplot as plt

def loadDataSet(filename):
    fr=open(filename)
    xArr=[]
    yArr=[]
    numFeat=len(open(filename).readline().split('\t'))-1
    for line in fr.readlines():
        lineArr=[]
        curline=line.strip().split('\t')
        for i in range(numFeat):
            lineArr.append(float(curline[i]))
        xArr.append(lineArr)
        yArr.append(float(curline[-1]))
    return xArr,yArr

def lwlr(testPoint,xArr,yArr,k=1.0):
    xMat=np.mat(xArr)
    yMat=np.mat(yArr).T
    m=np.shape(xMat)[0]
    weight=np.mat(np.eye((m)))
    for j in range(m):
        diffMat=testPoint-xMat[j,:]
        weight[j,j]=np.exp(diffMat*diffMat.T/(-2.0*k**2))
    xTx=xMat.T*weight*xMat
    if np.linalg.det(xTx)==0.0:
        print('矩阵奇异')
        return
    w=xTx.I*(xMat.T*(weight*yMat))
```

```
    return testPoint*w
def lwlrTest(testArr,xArr,yArr,k=1.0):
    m=np.shape(testArr)[0]
    yHat=np.zeros(m)
    for i in range(m):
        yHat[i]=lwlr(testArr[i],xArr,yArr,k)
    return yHat

def standRegres(xArr,yArr):
    xMat=np.mat(xArr)
    yMat=np.mat(yArr).T
    XTX=xMat.T*xMat
    if np.linalg.det(XTX)==0.0:
        print("矩阵奇异")
    w=XTX.I*(xMat.T*yMat)
    return w

def rssError(yArr,yHatArr):
    return ((yArr-yHatArr)**2).sum()

abX,abY=loadDataSet('abalone.txt')
yHat01=lwlrTest(abX[0:99],abX[0:99],abY[0:99],0.1)
yHat02=lwlrTest(abX[0:99],abX[0:99],abY[0:99],1)
yHat03=lwlrTest(abX[0:99],abX[0:99],abY[0:99],10)
print('训练集相同的情况：')
print('k=0.1,误差是：',rssError(abY[0:99],yHat01.T))
print('k=1,误差是：',rssError(abY[0:99],yHat02.T))
print('k=10,误差是：',rssError(abY[0:99],yHat03.T))
yHat01=lwlrTest(abX[100:199],abX[0:99],abY[0:99],0.1)
yHat02=lwlrTest(abX[100:199],abX[0:99],abY[0:99],1)
yHat03=lwlrTest(abX[100:199],abX[0:99],abY[0:99],10)
print('训练集不同的情况：')
print('k=0.1,误差是：',rssError(abY[100:199],yHat01.T))
print('k=1,误差是：',rssError(abY[100:199],yHat02.T))
print('k=10,误差是：',rssError(abY[100:199],yHat03.T))
w=standRegres(abX[0:99],abY[0:99])
yHat=np.mat(abX[100:199])*w
print('简单的线性回归误差：',rssError(abY[100:199],yHat.T.A))
```

> 训练集相同的情况：
> k=0.1,误差是： 56.78290824031715
> k=1,误差是： 429.8905618699896
> k=10,误差是： 549.1181708826314
> 训练集不同的情况：
> k=0.1,误差是： 23390.250910905706
> k=1,误差是： 573.5261441898399
> k=10,误差是： 517.5711905382227
> 简单的线性回归误差： 518.6363153249638

我们简单地对这个现象进行分析，当k=0.1时，训练集误差很小但是当引入新的测试集时误差变得很大，这就是我们所谓的过拟合，我们要确保自己的模型有足够的鲁棒性，应该确保既不过拟合又不会欠拟合，例如我们比较k=1时局部线性与普通线性效果差不多，也就是说，我们需要在未知数据的比对下才能选取比较好的模型。