线性回归之——最小二乘法

2013-08-11 00:04:35

标签：[线性回归](http://blog.51cto.com/tag-%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92.html) [Logistic 回归](http://blog.51cto.com/tag-Logistic%20%E5%9B%9E%E5%BD%92.html) [最小二乘法](http://blog.51cto.com/tag-%E6%9C%80%E5%B0%8F%E4%BA%8C%E4%B9%98%E6%B3%95.html)

原创作品，允许转载，转载时请务必以超链接形式标明文章 [原始出处](http://sbp810050504.blog.51cto.com/2799422/1269572) 、作者信息和本声明。否则将追究法律责任。<http://sbp810050504.blog.51cto.com/2799422/1269572>

**一、引言**

这段时间学习《机器学习》，学到第5章的“Logistic回归”，感觉相当吃力。追本溯源，从“Logistic回归”到“线性回归”，再到“最小二乘法”。最终定格到了《高等数学》（第六版·下册）第九章第10节“最小二乘法”，这才了解到最小二乘法背后的数学原理出自哪里。

“最小二乘法”是最优化问题中建立经验公式的一种实现方法。了解它的原理，对于了解“Logistic回归”和“支持向量机的学习”都很有裨益。

**二、背景知识**

“最小二乘法”出现的历史背景是很有意思的。（以下文字摘录维基百科）

1801年，意大利天文学家[朱赛普·皮亚齐](http://zh.wikipedia.org/w/index.php?title=%E6%9C%B1%E8%B5%9B%E6%99%AE%C2%B7%E7%9A%AE%E4%BA%9A%E9%BD%90&action=edit&redlink=1)发现了第一颗小行星[谷神星](http://zh.wikipedia.org/wiki/%E8%B0%B7%E7%A5%9E%E6%98%9F)。经过40天的跟踪观测后，由于谷神星运行至太阳背后，使得皮亚齐失去了谷神星的位置。随后全世界的科学家利用皮亚齐的观测数据开始寻找谷神星，但是根据大多数人计算的结果来寻找谷神星都没有结果。时年24岁的[高斯](http://zh.wikipedia.org/wiki/%E5%8D%A1%E5%B0%94%C2%B7%E5%BC%97%E9%87%8C%E5%BE%B7%E9%87%8C%E5%B8%8C%C2%B7%E9%AB%98%E6%96%AF)也计算了谷神星的轨道。奥地利天文学家[海因里希·奥尔伯斯](http://zh.wikipedia.org/w/index.php?title=%E6%B5%B7%E5%9B%A0%E9%87%8C%E5%B8%8C%C2%B7%E5%A5%A5%E5%B0%94%E4%BC%AF%E6%96%AF&action=edit&redlink=1)根据高斯计算出来的轨道重新发现了谷神星。

[高斯](http://zh.wikipedia.org/wiki/%E9%AB%98%E6%96%AF)使用的最小二乘法的方法发表于1809年他的著作《天体运动论》中，而法国科学家[勒让德](http://zh.wikipedia.org/wiki/%E5%8B%92%E8%AE%93%E5%BE%B7)于1806年独立发现“最小二乘法”，但因不为时人所知而默默无闻。两人曾为谁最早创立最小二乘法原理发生争执。

1829年，[高斯](http://zh.wikipedia.org/wiki/%E9%AB%98%E6%96%AF)提供了最小二乘法的优化效果强于其他方法的证明，见[高斯-马尔可夫定理](http://zh.wikipedia.org/wiki/%E9%AB%98%E6%96%AF-%E9%A9%AC%E5%B0%94%E5%8F%AF%E5%A4%AB%E5%AE%9A%E7%90%86)。

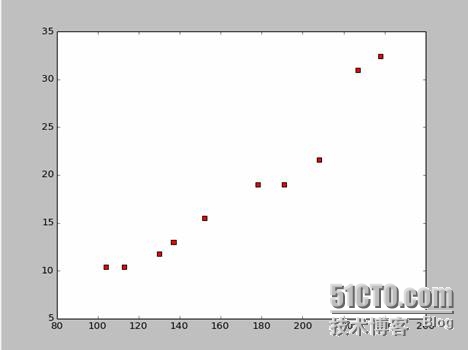
**三、知识运用**

“最小二乘法”的核心就是保证所有数据偏差的平方和最小。（“平方”的在古时侯的称谓为“二乘”）

假设我们收集到一些战舰的长度与宽度数据

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 长度（m） | 208 | 152 | 113 | 227 | 137 | 238 | 178 | 104 | 191 | 130 |
| 宽度(m) | 21.6 | 15.5 | 10.4 | 31.0 | 13.0 | 32.4 | 19.0 | 10.4 | 19.0 | 11.8 |

根据这些数据我们用python画出散点图：

[](http://img1.51cto.com/attachment/201308/235813706.jpg)

画散点图的代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # -\*- coding: utf-8 -\*  import numpy as np  import os  import matplotlib.pyplot as plt  def drawScatterDiagram(fileName):      #改变工作路径到数据文件存放的地方      os.chdir("d:/workspace\_ml")      xcord=[];ycord=[]      fr=open(fileName)      for line in fr.readlines():          lineArr=line.strip().split()          xcord.append(float(lineArr[1]));ycord.append(float(lineArr[2]))      plt.scatter(xcord,ycord,s=30,c='red',marker='s')      plt.show() |

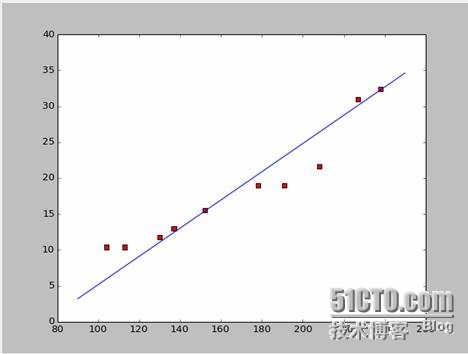
假如我们取前两个点（238,32.4）（152, 15.5）就可以得到两个方程

152\*a+b=15.5

328\*a+b=32.4

解这两个方程得a=0.197,b=-14.48

那样的话，我们可以得到这样的拟合图：

[](http://img1.51cto.com/attachment/201308/235920498.jpg)

好了，新的问题来了，这样的a,b是不是最优解呢？用专业的说法就是：a,b是不是模型的最优化参数？在回答这个问题之前，我们先解决另外一个问题：a,b满足什么条件才是最好的？答案是：保证所有数据偏差的平方和最小。至于原理，我们会在后面讲，先来看看怎么利用这个工具来计算最好的a和b。

假设所有数据的平方和为M，则[235957223.gif](http://img1.51cto.com/attachment/201308/235957223.gif)

我们现在要做的就是求使得M最小的a和b。请注意这个方程中，我们已知yi和xi

那其实这个方程就是一个以（a,b）为自变量，M为因变量的二元函数。

回想一下高数中怎么对一元函数就极值。我们用的是导数这个工具。那么在二元函数中，

我们依然用导数。只不过这里的导数有了新的名字“偏导数”。偏导数就是把两个变量中的一个视为常数来求导。

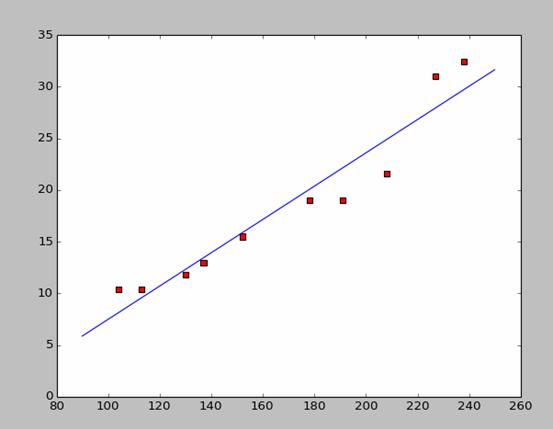
通过对M来求偏导数，我们得到一个方程组

[000017288.gif](http://img1.51cto.com/attachment/201308/000017288.gif)=0

[000042889.gif](http://img1.51cto.com/attachment/201308/000042889.gif)=0

这两个方程中xi和yi都是知道的。

很容易就求得a和b了。由于采用的是维基百科的数据，我这里就直接用答案来画出拟合图像：

[](http://img1.51cto.com/attachment/201308/000132871.jpg)

代码如下：

|  |  |
| --- | --- |
| 1 | # -\*- coding: utf-8 -\*importnumpy as npimportosimportmatplotlib.pyplot as pltdefdrawScatterDiagram(fileName):#改变工作路径到数据文件存放的地方os.chdir("d:/workspace\_ml")xcord=[];ycord=[]fr=open(fileName)forline infr.readlines():lineArr=line.strip().split()xcord.append(float(lineArr[1]));ycord.append(float(lineArr[2]))plt.scatter(xcord,ycord,s=30,c='red',marker='s')#a=0.1965;b=-14.486a=0.1612;b=-8.6394x=np.arange(90.0,250.0,0.1)y=a\*x+bplt.plot(x,y)plt.show() |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | # -\*- coding: utf-8 -\*  import numpy as np  import os  import matplotlib.pyplot as plt  def drawScatterDiagram(fileName):      #改变工作路径到数据文件存放的地方      os.chdir("d:/workspace\_ml")      xcord=[];ycord=[]      fr=open(fileName)      for line in fr.readlines():          lineArr=line.strip().split()          xcord.append(float(lineArr[1]));ycord.append(float(lineArr[2]))      plt.scatter(xcord,ycord,s=30,c='red',marker='s')      #a=0.1965;b=-14.486      a=0.1612;b=-8.6394      x=np.arange(90.0,250.0,0.1)      y=a\*x+b      plt.plot(x,y)      plt.show() |

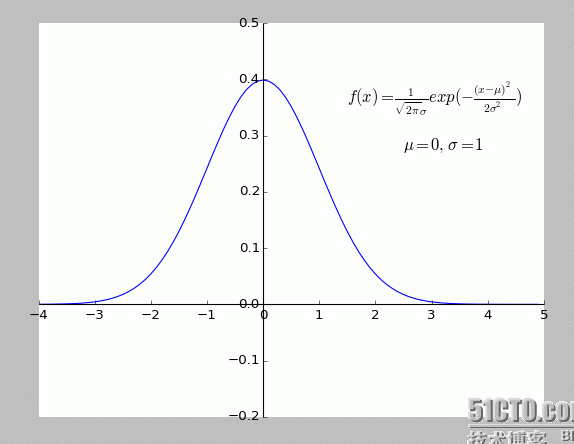
**四、原理探究**

数据拟合中，为什么要让模型的预测数据与实际数据之差的平方而不是绝对值和最小来优化模型参数？

这个问题已经有人回答了，见链接（<http://blog.sciencenet.cn/blog-430956-621997.html>）

个人感觉这个解释是非常有意思的。特别是里面的假设：所有偏离f(x)的点都是有噪音的。

一个点偏离越远说明噪音越大，这个点出现的概率也越小。那么偏离程度x与出现概率f(x)满足什么关系呢？——正态分布。

[](http://img1.51cto.com/attachment/201308/000149653.png)

已知N个点（用D来表示），求直线（用h来表示）出现的概率就可以表示为：P(h|D)

根据贝叶斯定理：P(h|D)=P(D|h)\*P(h)/P(D)即P(h|D)∝P(D|h)\*P(h) (∝表示“正比于”)

这就是一个生成模型了——由直线h生成点集D。

我们再作一个假设：h生成D中的每一个点都是独立的（如果了解贝叶斯文本分类的话，这里就很好理解了），那么P(D|h)=p(d1|h)\*p(d2|h)…

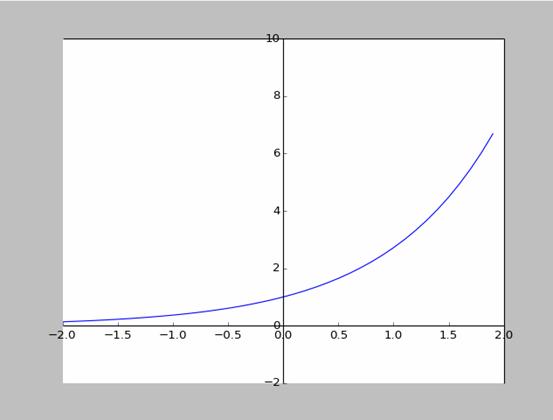
结合前面正态分布，我们可以写出这样的式子：p(di|h)∝ exp(-(ΔYi)^2)

那么P(D|h)∝EXP[-(ΔY1)^2]\* EXP[-(ΔY2)^2] \* EXP[-(ΔY3)^2] \* ..

又因为：ea\*eb\*ec=ea+b+c

所以p(D|h)∝ EXP{-[(ΔY1)^2 +(ΔY2)^2 + (ΔY3)^2 + ..]}

我们知道f(x)=ex的分布图像为：

[](http://img1.51cto.com/attachment/201308/000207654.jpg)

因为e的指数永远小于0，所以，想要p(D|h)最大，就必须使[(ΔY1)^2 + (ΔY2)^2 + (ΔY3)^2 + ..]无限接近于0，即：最大化p(D|h)就是要最小化[(ΔY1)^2 + (ΔY2)^2 + (ΔY3)^2 + ..]

到此，最小二乘法的原理得到了诠释。

**五、拓展延伸**

上面讲的都是二维的情况，也就是只有一个自变量。但现实世界中影响最后结果的都是多种因素的叠加，即自变量会有多个的情况。

对于一般N元线性函数，用《线性代数》中的逆矩阵来求解就OK了；由于暂时没有找到合适的例子，就作为一个引子，留在这里了。

当然自然界更多的是多项式拟合，而非简单的线性，那就是更高级的内容了。