**排版并不好，请打开网页，进行拓展学习**

**（第一篇）逻辑回归(logistic regression)的本质——极大似然估计**

2017-08-14 19:36:24 [zjuPeco](https://me.csdn.net/zjuPeco" \t "_blank) 阅读数 69808更多

分类专栏： [机器学习](https://blog.csdn.net/zjupeco/article/category/7075199)

版权声明：本文为博主原创文章，遵循[CC 4.0 BY-SA](http://creativecommons.org/licenses/by-sa/4.0/)版权协议，转载请附上原文出处链接和本声明。

**本文链接：**[**https://blog.csdn.net/zjuPeco/article/details/77165974**](https://blog.csdn.net/zjuPeco/article/details/77165974)

**前言**

逻辑回归是分类当中极为常用的手段，因此，掌握其内在原理是非常必要的。我会争取在本文中尽可能简明地展现逻辑回归(logistic regression)的整个推导过程。

**什么是逻辑回归**

逻辑回归在某些书中也被称为对数几率回归，明明被叫做回归，却用在了分类问题上，我个人认为这是因为逻辑回归用了和回归类似的方法来解决了分类问题。  
假设有一个二分类问题，输出为y∈{0,1}y∈{0,1}，而线性回归模型产生的预测值为z=wTx+bz=wTx+b是实数值，我们希望有一个理想的阶跃函数来帮我们实现zz值到0/10/1值的转化。

ϕ(z)=⎧⎩⎨00.51if z < 0if z = 0if z > 0ϕ(z)={0if z < 00.5if z = 01if z > 0

然而该函数不连续，我们希望有一个单调可微的函数来供我们使用，于是便找到了Sigmoid functionSigmoid function来替代。

ϕ(z)=11+e−zϕ(z)=11+e−z

两者的图像如下图所示（图片出自文献2）

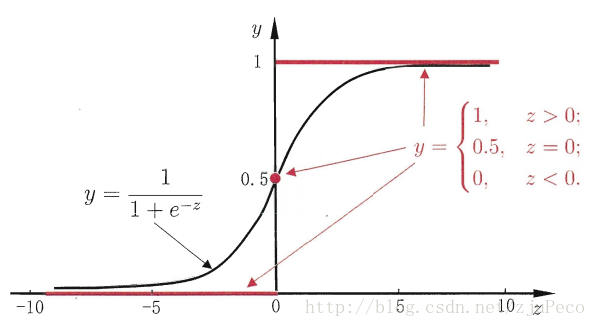


图1：sigmoid & step function

有了Sigmoid fuctionSigmoid fuction之后，由于其取值在[0,1][0,1]，我们就可以将其视为类11的后验概率估计p(y=1|x)p(y=1|x)。说白了，就是如果有了一个测试点xx，那么就可以用Sigmoid fuctionSigmoid fuction算出来的结果来当做该点xx属于类别11的概率大小。  
于是，非常自然地，我们把Sigmoid fuctionSigmoid fuction计算得到的值大于等于0.50.5的归为类别11，小于0.50.5的归为类别00。

y^={10if ϕ(z)≥0.5otherwisey^={1if ϕ(z)≥0.50otherwise

同时逻辑回归与自适应线性网络非常相似，两者的区别在于逻辑回归的激活函数是Sigmoid functionSigmoid function而自适应线性网络的激活函数是y=xy=x，两者的网络结构如下图所示（图片出自文献1）。

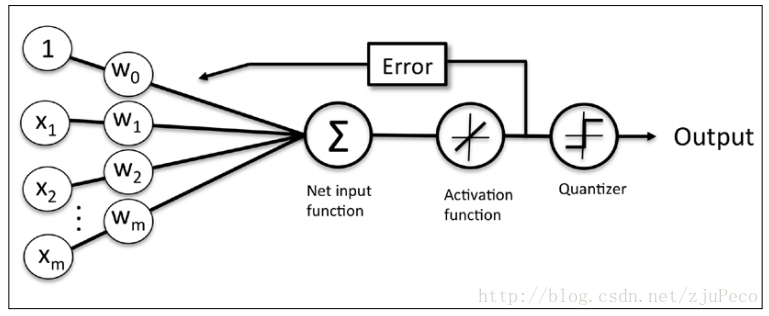


图2：自适应线性网络

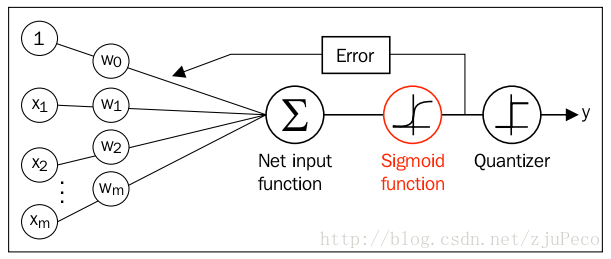


图3：逻辑回归网络

**逻辑回归的代价函数**

好了，所要用的几个函数我们都有了，接下来要做的就是根据给定的训练集，把参数ww给求出来了。要找参数ww，首先就是得把代价函数（cost function）给定义出来，也就是目标函数。  
我们第一个想到的自然是模仿线性回归的做法，利用误差平方和来当代价函数。

J(w)=∑i12(ϕ(z(i))−y(i))2J(w)=∑i12(ϕ(z(i))−y(i))2

其中，z(i)=wTx(i)+bz(i)=wTx(i)+b，ii表示第ii个样本点，y(i)y(i)表示第ii个样本的真实值，ϕ(z(i))ϕ(z(i))表示第ii个样本的预测值。  
这时，如果我们将ϕ(z(i))=11+e−z(i)ϕ(z(i))=11+e−z(i)代入的话，会发现这是一个非凸函数，这就意味着代价函数有着许多的局部最小值，这不利于我们的求解。

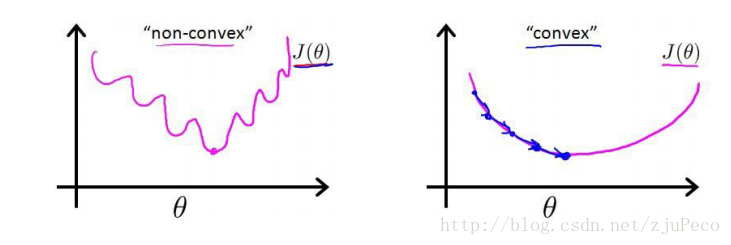


图4：凸函数和非凸函数

那么我们不妨来换一个思路解决这个问题。前面，我们提到了ϕ(z)ϕ(z)可以视为类11的后验估计，所以我们有

p(y=1|x;w)=ϕ(wTx+b)=ϕ(z)p(y=1|x;w)=ϕ(wTx+b)=ϕ(z)

p(y=0|x;w)=1−ϕ(z)p(y=0|x;w)=1−ϕ(z)

其中，p(y=1|x;w)p(y=1|x;w)表示给定ww，那么xx点y=1y=1的概率大小。  
上面两式可以写成一般形式

p(y|x;w)=ϕ(z)y(1−ϕ(z))(1−y)p(y|x;w)=ϕ(z)y(1−ϕ(z))(1−y)

接下来我们就要用极大似然估计来根据给定的训练集估计出参数ww。

L(w)=∏ni=1p(y(i)|x(i);w)=∏ni=1(ϕ(z(i)))y(i)(1−ϕ(z(i)))1−y(i)L(w)=∏i=1np(y(i)|x(i);w)=∏i=1n(ϕ(z(i)))y(i)(1−ϕ(z(i)))1−y(i)

为了简化运算，我们对上面这个等式的两边都取一个对数

l(w)=lnL(w)=∑ni=1y(i)ln(ϕ(z(i)))+(1−y(i))ln(1−ϕ(z(i)))l(w)=lnL(w)=∑i=1ny(i)ln(ϕ(z(i)))+(1−y(i))ln(1−ϕ(z(i)))

我们现在要求的是使得l(w)l(w)最大的ww。没错，我们的代价函数出现了，我们在l(w)l(w)前面加个负号不就变成就最小了吗？不就变成我们代价函数了吗？

J(w)=−l(w)=−∑ni=1y(i)ln(ϕ(z(i)))+(1−y(i))ln(1−ϕ(z(i)))J(w)=−l(w)=−∑i=1ny(i)ln(ϕ(z(i)))+(1−y(i))ln(1−ϕ(z(i)))

为了更好地理解这个代价函数，我们不妨拿一个例子的来看看

J(ϕ(z),y;w)=−yln(ϕ(z))−(1−y)ln(1−ϕ(z))J(ϕ(z),y;w)=−yln(ϕ(z))−(1−y)ln(1−ϕ(z))

也就是说

J(ϕ(z),y;w)={−ln(ϕ(z))−ln(1−ϕ(z))if y=1if y=0J(ϕ(z),y;w)={−ln(ϕ(z))if y=1−ln(1−ϕ(z))if y=0

我们来看看这是一个怎么样的函数

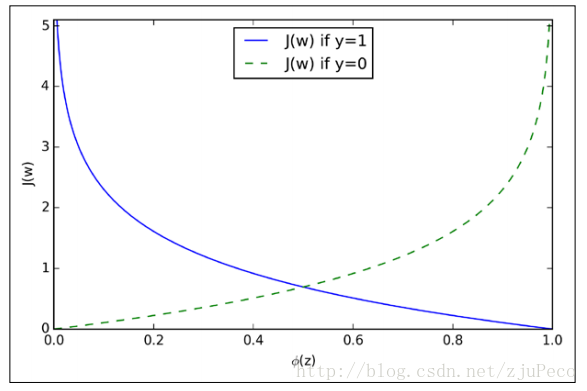


图5：代价函数

从图中不难看出，如果样本的值是11的话，估计值ϕ(z)ϕ(z)越接近11付出的代价就越小，反之越大；同理，如果样本的值是00的话，估计值ϕ(z)ϕ(z)越接近00付出的代价就越小，反之越大。

**利用梯度下降法求参数**

在开始梯度下降之前，要这里插一句，sigmoid functionsigmoid function有一个**很好的性质**就是

%FontSize=10.5
%TeXFontSize=10.5
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
\phi'(z) = \phi(z)(1 - \phi(z))
\]
\end{document}

下面会用到这个性质。  
还有，我们要明确一点，梯度的负方向就是代价函数下降最快的方向。什么？为什么？好，我来说明一下。借助于泰特展开，我们有

%FontSize=10.5
%TeXFontSize=10.5
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
f'(x) \cdot \delta = ||f'(x)|| \cdot ||\delta|| \cdot cos \theta
\]
\end{document}

其中，f′(x)f′(x)和δδ为向量，那么这两者的内积就等于

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
f'(x) \cdot \delta = ||f'(x)|| \cdot ||\delta|| \cdot cos \theta
\]
\end{document}

当θ=πθ=π时，也就是δδ在f′(x)f′(x)的负方向上时，取得最小值，也就是下降的最快的方向了~  
okay？好，坐稳了，我们要开始下降了。

%FontSize=10.5
%TeXFontSize=10.5
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
w := w + \Delta w, \ \Delta w=-\eta \nabla J(w)
\]
\end{document}

没错，就是这么下降。没反应过来？那我再写详细一些

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
w_j := w_j + \Delta w_j,\ \Delta w_j = -\eta \dfrac{\partial J(w)}{\partial w_j}
\]
\end{document}  
其中，wjwj表示第jj个特征的权重；ηη为学习率，用来控制步长。  
重点来了。

%FontSize=13
%TeXFontSize=13
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
\dfrac{\partial J(w)}{w_j} = -\sum_{i=1}^n (y^{(i)}\dfrac{1}{\phi(z^{(i)})}-(1 - y^{(i)})\dfrac{1}{1-\phi(z^{(i)})})\dfrac{\partial \phi(z^{(i)})}{\partial w_j} \] \[\\ \quad   \quad \quad =-\sum_{i=1}^n (y^{(i)}\dfrac{1}{\phi(z^{(i)})}-(1 - y^{(i)})\dfrac{1}{1-\phi(z^{(i)})})\phi(z^{(i)})(1-\phi(z^{(i)}))\dfrac{\partial z^{(i)}}{\partial w_j} \]\[\\ \quad \quad \quad =-\sum_{i=1}^n (y^{(i)}(1-\phi(z^{(i)}))-(1-y^{(i)})\phi(z^{(i)}))x_j^{(i)} \]\[\\ \quad \quad \quad =-\sum_{i=1}^n (y^{(i)}-\phi(z^{(i)}))x_j^{(i)}
\]
\end{document}

%FontSize=13
%TeXFontSize=13
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
w_j :=w_j+\eta \sum_{i=1}^n (y^{(i)}-\phi(z^{(i)}))x_j^{(i)}
\]
\end{document}  
所以，在使用梯度下降法更新权重时，只要根据下式即可

%FontSize=13
%TeXFontSize=13
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
w_j := w_j + \eta (y^{(i)}-\phi(z^{(i)}))x_j^{(i)}, for \ i \ in \ range(n)
\]
\end{document}  
此式与线性回归时更新权重用的式子极为相似，也许这也是逻辑回归要在后面加上回归两个字的原因吧。  
当然，在样本量极大的时候，每次更新权重会非常耗费时间，这时可以采用随机梯度下降法，这时每次迭代时需要将样本重新打乱，然后用下式不断更新权重。

%FontSize=12
%TeXFontSize=12
\documentclass{article}
\pagestyle{empty}
\begin{document}
\[
w_j := w_j + \eta (y^{(i)}-\phi(z^{(i)}))x_j^{(i)}, for \ i \ in \ range(n)
\]
\end{document}

也就是去掉了求和，而是针对每个样本点都进行更新。

**结束语**

以上就是我参考了基本书中的说法之后对逻辑回归整个推到过程的梳理，也不知道讲清楚没有。  
如有不足，还请指正~

**参考文献**

[1] Raschka S. Python Machine Learning[M]. Packt Publishing, 2015.  
[2] 周志华. 机器学习 : = Machine learning[M]. 清华大学出版社, 2016.

**（第二篇）逻辑回归与最大似然估计推导**

2018-08-01 16:42:31 [糖葫芦君](https://me.csdn.net/yinyu19950811) 阅读数 5997更多

分类专栏： [机器学习](https://blog.csdn.net/yinyu19950811/article/category/6779277)

版权声明：本文为博主原创文章，遵循[CC 4.0 BY-SA](http://creativecommons.org/licenses/by-sa/4.0/)版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/yinyu19950811/article/details/81321944>

**逻辑回归（对数几率回归）**

逻辑回归是一种分类算法，不是回归算法，因为它用了和回归类似的思想来解决了分类问题。

一句话总结逻辑回归：“逻辑回归假设数据服从伯努利分布，通过极大似然函数的方法，运用梯度下降来求解参数，来达到将数据二分类的目的”。

**1.广义线性模型**

我们先来看看线性回归模型：

y=w^Tx+b

但是假设我们认为实例所对应的输出标记是在指数尺度上变化，那么就可以将输出标记的对数作为线性模型逼近的目标：

In y=w^Tx+b

这就是“对数线性回归”，它实际上是试图让e^{w^Tx+b}逼近y，其是形式上是线性回归，实际上是在求输入空间到输出空间的非线性函数映射。这里的对数函数起到了将线性回归模型的预测值与真实标记联系起来的作用。

更一般地，考虑单调可微函数g(\cdot)=In(\cdot )，另：

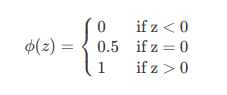
y=g^{-1}(w^Tx+b)

这样得到的模型称为“广义线性模型”，其中函数g(\cdot)称为“联系函数”。显然，对数线性回归是广义线性模型在g(\cdot )=In(\cdot )时的特例。

如上讨论了如何使用线性模型来进行回归学习，但是如果要做的是分类任务该怎么办？下面介绍如何由“广义线性模型”引出逻辑回归模型。

我们只需要找到一个联系函数，将**分类任务的真实标记y与线性回归模型的预测值联系起来**。

考虑二分类问题，其输出标记y\in \{0,1\}，而线性回归模型的预测值z=w^Tx+b是实数值，于是我们需要将实数值z转换为0/1值，可以使用单位阶跃函数：



但是该函数不连续，因此不能作为联系函数g，所以找到了一个能在一定程度上近似单位阶跃函数的“替代函数”，单调可微的对数几率函数（Logistic function），它能够将线性回归模型的预测值转化为分类任务对应的概率:

\phi (z)=\frac{1}{1+e^{-z}}

两者的图像如下图所示：

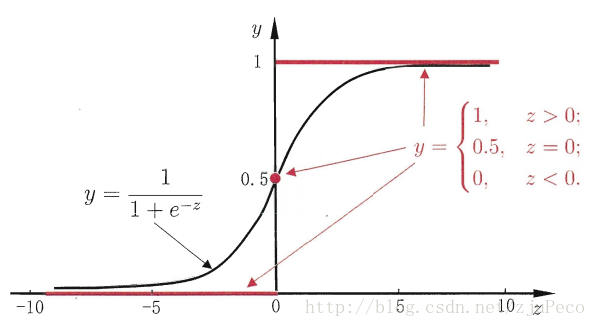


图1.单位阶跃函数与对数几率函数

对数几率其实是一种“sigmoid”函数，它将z值转化为一个接近0或1的y值：

y=\frac{1}{1+e^{-(w^Tx+b)}}\rightarrowIn\frac{y}{1-y}=w^Tx+b

若将y视为样本x作为正例的可能性，则1-y是其反例的可能性，两者的比值\frac{y}{1-y}称为“几率”，反映了x作为正例的相对可能性，对几率取对数则得到In\frac{y}{1-y}，可以看出，上式其实是在**用线性回归模型的预测结果去逼近真实标记的对数几率**。所以该模型也被称作“对数几率回归”。

由于sigmoid函数的取值在[0,1]之间，所以可以将其视为类1的后验概率估计p(y=1|x)，所以我们把sigmoid函数计算得到的值大于等于0.5的归为类别1，小于0.5的归为类别0。

\hat{y}=\left\{\begin{matrix}1,\ if\phi (z)\geq 0.5 \\0,\ otherwise \end{matrix}\right.

面经问题：为什么要使用sigmoid函数作为假设？

因为线性回归模型的预测值为实数，而样本的类标记为（0,1），我们需要将**分类任务的真实标记y与线性回归模型的预测值联系起来，**也就是找到广义线性模型中的联系函数。如果选择单位阶跃函数的话，它是不连续的不可微。而如果选择sigmoid函数，它是连续的，而且能够将z转化为一个接近0或1的值。

**2.逻辑回归的假设**

任何的模型都是有自己的假设的，在这个假设下模型才是试用的。

逻辑回归的第一个假设是：**假设数据服从伯努利分布**。第二个假设为假设模型的**输出值是样本为正例的概率**。

所以整个模型可以描述为：h_{\theta}(x;\theta)=p=\frac{1}{e^{-\theta^Tx}}

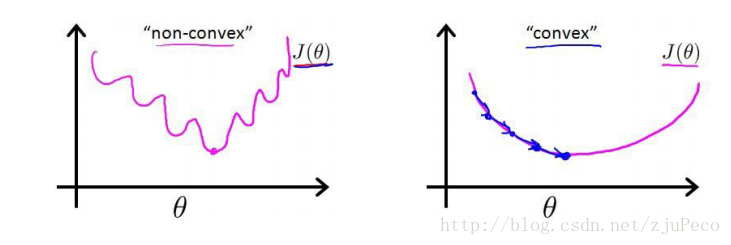
其中\theta=（w;b）为向量形式。

**2. 逻辑回归的代价函数**

接下来就是根据给定的训练集，把参数w求出来。要找到w，首先要先定义代价函数（目标函数）。首先想到的就是模仿线性回归的做法，利用误差平方和来当做代价函数：

J(\theta)=\sum_i\frac{1}{2}\left (h_{\theta}(x^i;\theta)-y^{i} \right )^2

将\phi (z^i)带入的话，会发现这是一个**非凸函数**，这就意味着代价函数有着许多的局部最小值，不利于求解。



而最大似然作为逻辑回归模型的损失函数，很容易得到参数的最优解（凸函数）。所以说选取的标准要容易测量，这就是逻辑回归损失函数为什么使用最大似然而不用最小二乘的原因。

**2.1 极大似然估计**

**逻辑回归与极大似然估计的关系：**

最大似然估计就是通过已知结果去反推最大概率导致该结果的参数。极大似然估计是概率论在统计学中的应用，它提供了一种给定观察数据来评估模型参数的方法，即 “模型已定，参数未知”，通过若干次试验，观察其结果，利用实验结果得到某个参数值能够使样本出现的概率为最大，则称为极大似然估计。逻辑回归是一种监督式学习，是有训练标签的，就是有已知结果的，从这个已知结果入手，去推导能获得最大概率的结果参数$\theta$，只要我们得出了这个参数，那我们的模型就自然可以很准确的预测未知的数据了。

之前提到了h_{\theta}(x;\theta)可以视为类1的后验概率，所以有：

h_\theta(x;\theta)=p(y=1|x;\theta)=\phi (w^Tx+b)=\phi(z)=\frac{1}{1+e^{-\theta ^Tx}}

p(y=0|x;w)=1-\phi (z)

将上面两式写为一般形式：

p(y|x;\theta)=h_\theta(x;\theta)^y(1-h_\theta(x;\theta))^{(1-y)}

接下来使用极大似然估计来根据给定的训练集估计出参数w：

为了简化运算，我们对上述等式两边取一个对数：

l(\theta)=InL(\theta)=\sum_{i=1}^{n}y^iIn(h_\theta(x^i;\theta))+(1-y^i)In(1-h_\theta(x^i;\theta))

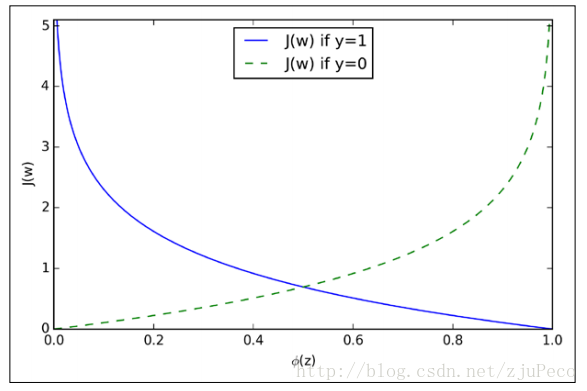
现在要求使得l(w)最大的w，在l(w)前面加一个负号就变为最小化负对数似然函数：

J(\theta)=-l(\theta)=-\left ( \sum_{i=1}^{n}y^iIn(h_\theta(x^i;\theta))+(1-y^i)In(1-h_\theta(x^i;\theta)) \right )

如此就得到了代价函数。让我们更好地理解这个代价函数：

J(h_\theta(x;\theta),y;\theta)=-yIn(h_\theta(x;\theta))-(1-y)In(1-h_\theta(x;\theta))

等价于：



可以看出，如果样本的类别为1，估计值\phi (z)越接近1付出的代价越小，反之越大。

同理，如果样本的值为0的话，估计值\phi (z)越接近于0付出的代价越小，反之越大。

**3. 利用梯度下降法求解参数w**

首先解释一些问什么梯度的负方向就是代价函数下降最快的方向，借助于泰勒展开：

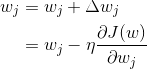
f(x+\delta )-f(x)\approx f'(x)\cdot \delta

f'(x)和\delta均为向量，那么两者的内积就等于：

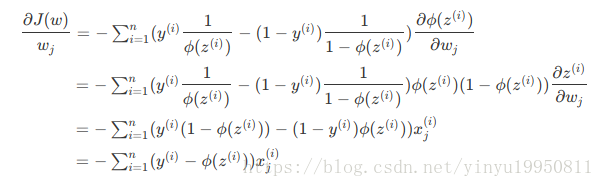
f'(x)\cdot \delta =\left \| f'(x) \right \|\cdot \left \| \delta \right \|\cdot cos \theta

当\theta =\pi时，也就是在f'(x)的负方向时，取得最小值，也就是下降的最快方向了。

梯度下降：



\eta为学习率，用来控制步长



所以，在使用梯度下降法更新权重时，只要根据下式即可：

w_j=w_j+\eta \sum^{n}_{i=1}\left(y_i- \phi(z^i)\right )x_j

（x_j代表第j列特征，w_j代表第j个特征对应的参数）

当样本量极大的时候，每次更新权重都需要遍历整个数据集，会非常耗时，这时可以采用随机梯度下降法：

w_j=w_j+\eta\left(y_i- \phi(z^i)\right )x_j,for\ i\ in\ range(n)

每次仅用一个样本点来更新回归系数，这种方法被称作随机梯度上升法。（由于可以在新样本到来时对分类器进行增量式更新，因此随机梯度算法是一个在线学习算法。）它与梯度上升算法效果相当，但占用更少的资源。

**3.1 三种梯度下降方法的选择**

1. 批量梯度下降BGD（Batch Gradient Descent）：优点：会获得全局最优解，易于并行实现。缺点：更新每个参数时需要遍历所有的数据，计算量会很大并且有很多的冗余计算，导致当数据量大的时候每个参数的更新都会很慢。
2. 随机梯度下降SGD：优点：训练速度快；缺点：准确率下降，并不是全局最优，不易于并行实现。它的具体思路是更新没一个参数时都是用一个样本来更新。（以高方差频繁更新，优点是使得sgd会跳到新的和潜在更好的局部最优解，缺点是使得收敛到局部最优解的过程更加的复杂。？）
3. small batch梯度下降：结合了上述两点的优点，每次更新参数时仅使用一部分样本，减少了参数更新的次数，可以达到更加稳定的结果，一般在深度学习中采用这种方法。

在实际应用时根据样本量的大小选择不同的梯度更新方法。

**4.逻辑回归优缺点：**

优点：

1.~~直接对分类可能性进行建模，无需实现假设数据分布，这样就避免了假设分布不准确所带来的问题（周志华.机器学习）~~

（其实很多机器学习模型本身都是对数据分布有一定的假设的，在这个假设前提之下去进行理论研究有助于我们关注主要矛盾，忽律次要矛盾。但是在工程当中，很多时候我们对数据的分布其实是不了解的，贸然对数据进行假设容易造成模型无法无法拟合真实的分布。）

2.形式简单，模型的可解释性非常好，特征的权重可以看到不同的特征对最后结果的影响。

2.除了类别，还能得到近似概率预测，这对许多需利用概率辅助决策的任务很有用。

3.对率函数是任意阶可导的凸函数，有很好的数学性质。

缺点：

1.准确率不是很高，因为形势非常的简单，很难去拟合数据的真实分布？

2.本身无法筛选特征。

参考资料：

1.周志华.机器学习

2.[逻辑回归(logistic regression)的本质——极大似然估计](https://blog.csdn.net/zjuPeco/article/details/77165974)

3.[逻辑回归的常见面试点总结](https://www.cnblogs.com/ModifyRong/p/7739955.html)

**（第三篇）逻辑回归模型（Logistic Regression）及Python实现**

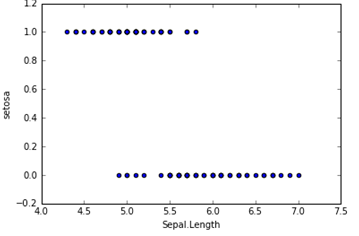
2018-08-01 17:07:22 [wang603603](https://me.csdn.net/wang603603) 阅读数 3281更多

分类专栏： [python](https://blog.csdn.net/wang603603/article/category/7318249)

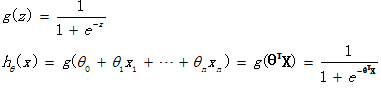
<https://www.cnblogs.com/sumai/p/5221067.html>

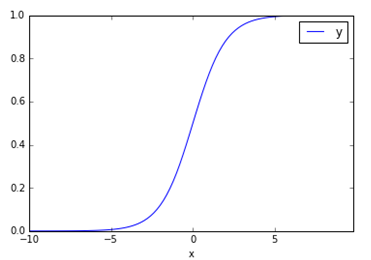
1.模型

　　在分类问题中，比如判断邮件是否为垃圾邮件，判断肿瘤是否为阳性，目标变量是离散的，只有两种取值，通常会编码为0和1。假设我们有一个特征X，画出散点图，结果如下所示。这时候如果我们用线性回归去拟合一条直线：hθ(X) = θ0+θ1X，若Y≥0.5则判断为1，否则为0。这样我们也可以构建出一个模型去进行分类，但是会存在很多的缺点，比如稳健性差、准确率低。而逻辑回归对于这样的问题会更加合适。



　　逻辑回归假设函数如下，它对**θTX**作了一个函数g变换，映射至0到1的范围之内，而函数g称为sigmoid function或者logistic function，函数图像如下图所示。当我们输入特征，得到的hθ(x)其实是这个样本属于1这个分类的概率值。也就是说，逻辑回归是用来得到样本属于某个分类的概率。



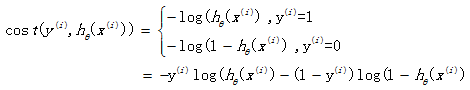


2.评价

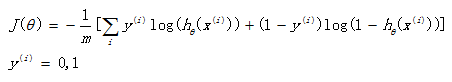
   回想起之前线性回归中所用到的损失函数：

https://images2015.cnblogs.com/blog/876358/201602/876358-20160226170451380-696933077.png

　如果在逻辑回归中也运用这种损失函数，得到的函数J是一个非凸函数，存在多个局部最小值，很难进行求解，因此需要换一个cost函数。重新定义个cost函数如下：



   当实际样本属于1类别时，如果预测概率也为1，那么损失为0，预测正确。相反，如果预测为0，那么损失将是无穷大。这样构造的损失函数是合理的，并且它还是一个凸函数，十分方便求得参数θ，使得损失函数J达到最小。



3.优化

    我们已经定义好了损失函数J(θ)，接下来的任务就是求出参数θ。我们的目标很明确，就是找到一组θ，使得我们的损失函数J(θ)最小。最常用的求解方法有两种：批量梯度下降法(batch gradient descent)， 牛顿迭代方法((Newton's method)。两种方法都是通过迭代求得的数值解，但是牛顿迭代方法的收敛速度更加快。

   批量梯度下降法： https://images2015.cnblogs.com/blog/876358/201602/876358-20160226170527786-1761009911.png

   牛顿迭代方法： https://images2015.cnblogs.com/blog/876358/201602/876358-20160226170537193-738173486.png (**H**为海瑟矩阵)

4.python代码实现

1. *# -\*- coding: utf-8 -\*-*
2. """
3. Created on Wed Feb 24 11:04:11 2016
4. @author: SumaiWong
5. """
7. import numpy as np
8. import pandas as pd
9. from numpy import dot
10. from numpy.linalg import inv
12. iris = pd.read\_csv('D:\iris.csv')
13. dummy = pd.get\_dummies(iris['Species']) *# 对Species生成哑变量*
14. iris = pd.concat([iris, dummy], axis =1 )
15. iris = iris.iloc[0:100, :] *# 截取前一百行样本*
17. *# 构建Logistic Regression , 对Species是否为setosa进行分类 setosa ~ Sepal.Length*
18. *# Y = g(BX) = 1/(1+exp(-BX))*
19. def logit(x):
20. return 1./(1+np.exp(-x))
22. temp = pd.DataFrame(iris.iloc[:, 0])
23. temp['x0'] = 1.
24. X = temp.iloc[:,[1,0]]
25. Y = iris['setosa'].reshape(len(iris), 1) *#整理出X矩阵 和 Y矩阵*
27. *# 批量梯度下降法*
28. m,n = X.shape *#矩阵大小*
29. alpha = 0.0065 *#设定学习速率*
30. theta\_g = np.zeros((n,1)) *#初始化参数*
31. maxCycles = 3000 *#迭代次数*
32. J = pd.Series(np.arange(maxCycles, dtype = float)) *#损失函数*
34. for i in range(maxCycles):
35. h = logit(dot(X, theta\_g)) *#估计值*
36. J[i] = -(1/100.)\*np.sum(Y\*np.log(h)+(1-Y)\*np.log(1-h)) *#计算损失函数值*
37. error = h - Y *#误差*
38. grad = dot(X.T, error) *#梯度*
39. theta\_g -= alpha \* grad
40. print theta\_g
41. print J.plot()
43. *# 牛顿方法*
44. theta\_n = np.zeros((n,1)) *#初始化参数*
45. maxCycles = 10 *#迭代次数*
46. C = pd.Series(np.arange(maxCycles, dtype = float)) *#损失函数*
47. for i in range(maxCycles):
48. h = logit(dot(X, theta\_n)) *#估计值*
49. C[i] = -(1/100.)\*np.sum(Y\*np.log(h)+(1-Y)\*np.log(1-h)) *#计算损失函数值*
50. error = h - Y *#误差*
51. grad = dot(X.T, error) *#梯度*
52. A = h\*(1-h)\* np.eye(len(X))
53. H = np.mat(X.T)\* A \* np.mat(X) *#海瑟矩阵, H = X`AX*
54. theta\_n -= inv(H)\*grad
55. print theta\_n

**（第四篇）极大似然估计详解，写的太好了！**

2018-08-18 15:42:08 [鹏大大大](https://me.csdn.net/qq_39355550) 阅读数 39326更多

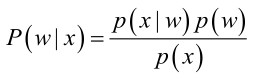
分类专栏： [Python](https://blog.csdn.net/qq_39355550/article/category/7945081)

**极大似然估计**

        以前多次接触过极大似然估计，但一直都不太明白到底什么原理，最近在看贝叶斯分类，对极大似然估计有了新的认识，总结如下：

**贝叶斯决策**

        首先来看贝叶斯分类，我们都知道经典的贝叶斯公式：



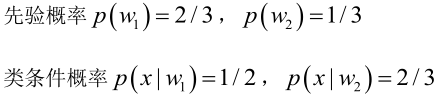
        其中：p(w)：为先验概率，表示每种类别分布的概率；https://img-blog.csdn.net/20170528002108539：类条件概率，表示在某种类别前提下，某事发生的概率；而https://img-blog.csdn.net/20170528002145196为后验概率，表示某事发生了，并且它属于某一类别的概率，有了这个后验概率，我们就可以对样本进行分类。后验概率越大，说明某事物属于这个类别的可能性越大，我们越有理由把它归到这个类别下。

        我们来看一个直观的例子：**已知：**在夏季，某公园男性穿凉鞋的概率为1/2，女性穿凉鞋的概率为2/3，并且该公园中男女比例通常为2:1，**问题：**若你在公园中随机遇到一个穿凉鞋的人，请问他的性别为男性或女性的概率分别为多少？

        从问题看，就是上面讲的，某事发生了，它属于某一类别的概率是多少？即后验概率。

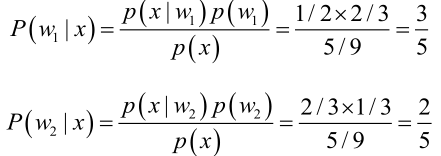
        设：https://img-blog.csdn.net/20170528002248527

        由已知可得：



        男性和女性穿凉鞋相互独立，所以https://img-blog.csdn.net/20170528002436496

（若只考虑分类问题，只需要比较后验概率的大小，的取值并不重要）。

        由贝叶斯公式算出：

**问题引出**

        但是在实际问题中并不都是这样幸运的，我们能获得的数据可能只有有限数目的样本数据，而先验概率https://img-blog.csdn.net/20170528002627998和类条件概率(各类的总体分布)https://img-blog.csdn.net/20170528002633154都是未知的。根据仅有的样本数据进行分类时，一种可行的办法是我们需要先对先验概率和类条件概率进行估计，然后再套用贝叶斯分类器。

        先验概率的估计较简单，1、每个样本所属的自然状态都是已知的（有监督学习）；2、依靠经验；3、用训练样本中各类出现的频率估计。

        类条件概率的估计（非常难），原因包括：概率密度函数包含了一个随机变量的全部信息；样本数据可能不多；特征向量x的维度可能很大等等。总之要直接估计类条件概率的密度函数很难。解决的办法就是，把估计完全未知的概率密度https://img-blog.csdn.net/20170528002633154转化为估计参数。这里就将概率密度估计问题转化为参数估计问题，极大似然估计就是一种参数估计方法。当然了，概率密度函数的选取很重要，模型正确，在样本区域无穷时，我们会得到较准确的估计值，如果模型都错了，那估计半天的参数，肯定也没啥意义了。

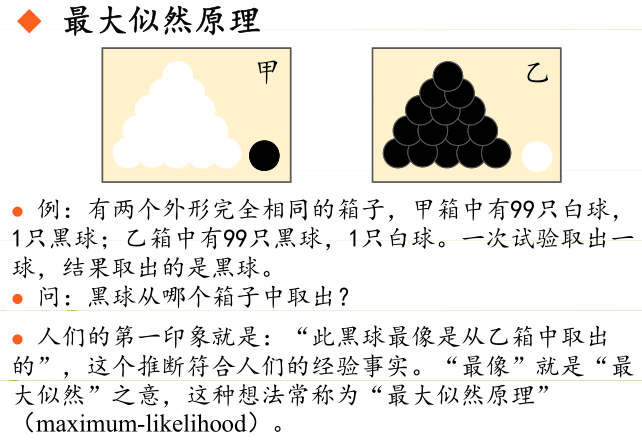
**重要前提**

        上面说到，参数估计问题只是实际问题求解过程中的一种简化方法（由于直接估计类条件概率密度函数很困难）。所以能够使用极大似然估计方法的样本必须需要满足一些前提假设。

        重要前提：训练样本的分布能代表样本的真实分布。每个样本集中的样本都是所谓独立同分布的随机变量 (iid条件)，且有充分的训练样本。

**极大似然估计**

        极大似然估计的原理，用一张图片来说明，如下图所示：



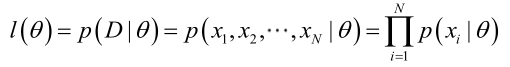
        总结起来，最大似然估计的目的就是：利用已知的样本结果，反推最有可能（最大概率）导致这样结果的参数值。

        原理：极大似然估计是建立在极大似然原理的基础上的一个统计方法，是概率论在统计学中的应用。极大似然估计提供了一种给定观察数据来评估模型参数的方法，即：“模型已定，参数未知”。通过若干次试验，观察其结果，利用试验结果得到某个参数值能够使样本出现的概率为最大，则称为极大似然估计。

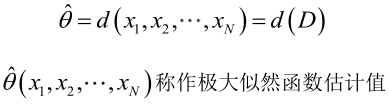
        由于样本集中的样本都是独立同分布，可以只考虑一类样本集D，来估计参数向量θ。记已知的样本集为：

https://img-blog.csdn.net/20170528003138251

        似然函数（linkehood function）：联合概率密度函数https://img-blog.csdn.net/20170528003212360称为相对于https://img-blog.csdn.net/20170528003218392的θ的似然函数。

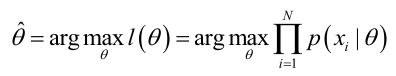


        如果https://img-blog.csdn.net/20170528003231366是参数空间中能使似然函数https://img-blog.csdn.net/20170528003236220最大的θ值，则https://img-blog.csdn.net/20170528003231366应该是“最可能”的参数值，那么https://img-blog.csdn.net/20170528003231366就是θ的极大似然估计量。它是样本集的函数，记作：



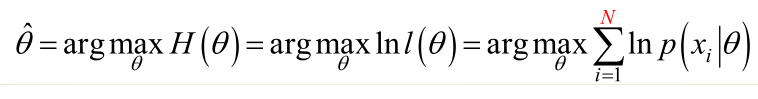
**求解极大似然函数**

        ML估计：求使得出现该组样本的概率最大的θ值。



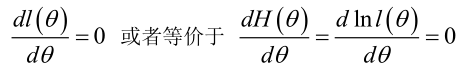
         实际中为了便于分析，定义了对数似然函数：

https://img-blog.csdn.net/20170528003844453



        1. 未知参数只有一个（θ为标量）

        在似然函数满足连续、可微的正则条件下，极大似然估计量是下面微分方程的解：

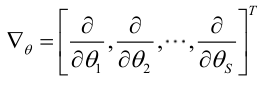


        2.未知参数有多个（θ为向量）

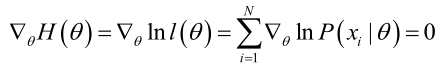
        则θ可表示为具有S个分量的未知向量：

https://img-blog.csdn.net/20170528003901066

         记梯度算子：



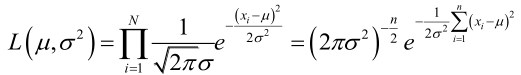
         若似然函数满足连续可导的条件，则最大似然估计量就是如下方程的解。



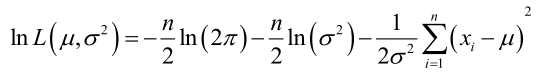
         方程的解只是一个估计值，只有在样本数趋于无限多的时候，它才会接近于真实值。

**极大似然估计的例子**

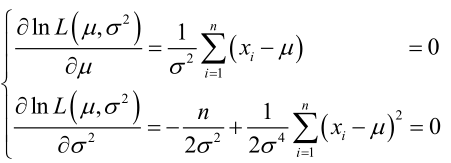
        例1：设样本服从正态分布https://img-blog.csdn.net/20170528003917176，则似然函数为：



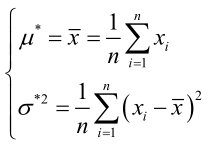
        它的对数：



        求导，得方程组：

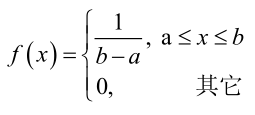


        联合解得：

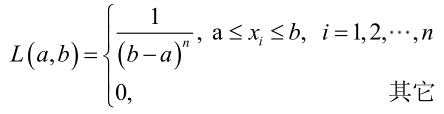


        似然方程有唯一解https://img-blog.csdn.net/20170528004743185：，而且它一定是最大值点，这是因为当https://img-blog.csdn.net/20170528004747290或https://img-blog.csdn.net/20170528004751982时，非负函数https://img-blog.csdn.net/20170528004756951。于是U和https://img-blog.csdn.net/20170528004801165的极大似然估计为https://img-blog.csdn.net/20170528004743185。

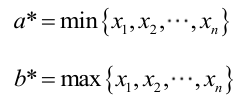
        例2：设样本服从均匀分布[a, b]。则X的概率密度函数：



        对样本https://img-blog.csdn.net/20170528005300323：



        很显然，L(a,b)作为a和b的二元函数是不连续的，这时不能用导数来求解。而必须从极大似然估计的定义出发，求L(a,b)的最大值，为使L(a,b)达到最大，b-a应该尽可能地小，但b又不能小于https://img-blog.csdn.net/20170528005307589，否则，L(a,b)=0。类似地a不能大过https://img-blog.csdn.net/20170528005311058，因此，a和b的极大似然估计：



**总结**

        求最大似然估计量https://img-blog.csdn.net/20170528003231366的一般步骤：

        （1）写出似然函数；

        （2）对似然函数取对数，并整理；

        （3）求导数；

        （4）解似然方程。

        最大似然估计的特点：

        1.比其他估计方法更加简单；

        2.收敛性：无偏或者渐近无偏，当样本数目增加时，收敛性质会更好；

        3.如果假设的类条件概率模型正确，则通常能获得较好的结果。但如果假设模型出现偏差，将导致非常差的估计结果。

正态分布ML估计的Matlab实例：[点击打开链接](http://blog.csdn.net/zengxiantao1994/article/details/72793608)

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/zengxiantao1994/article/details/72787849

**（第五篇）逻辑回归的推导及python实现**

2019-05-10 11:56:10 [song430](https://me.csdn.net/song430) 阅读数 49更多

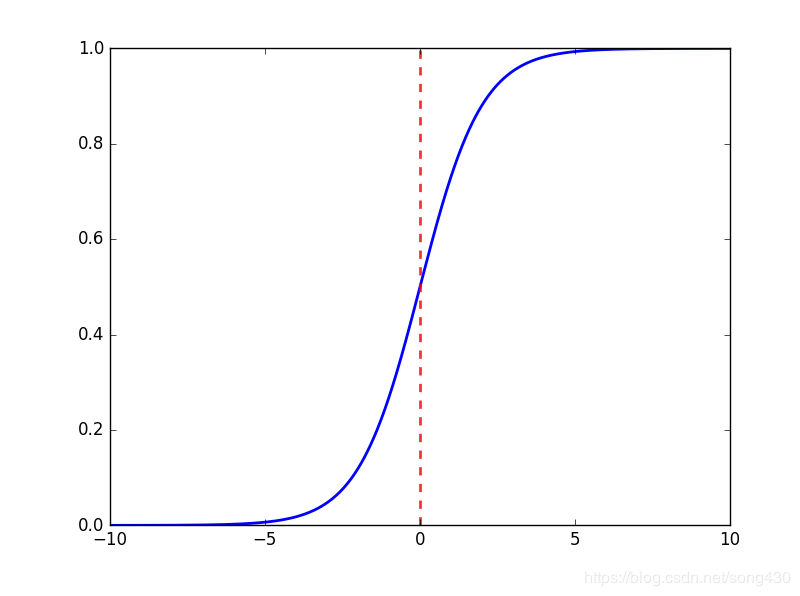
分类专栏： [机器学习算法](https://blog.csdn.net/song430/article/category/8808850)

版权声明：本文为博主原创文章，遵循[CC 4.0 BY-SA](http://creativecommons.org/licenses/by-sa/4.0/)版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/song430/article/details/89764563>

逻辑回归可以用于解决二分类问题，之前写的感知机也是用于二分类问题的，不过感知机将特征的线性组合映射到了（-1，+1）两个离散值，logistic regression则将这个线性组合的输出映射到了（0,1）区间之内，并且一般认为输出≥0.5\geq0.5≥0.5就取为1，反之则取为0（其实也就是正例和反例的分类）。

**逻辑回归**

逻辑回归的本质其实是线性回归，不过是把线性回归的值映射到了(0,1)区间之内，这个映射函数称之为sigmoid函数，也称为Logistic函数，这个函数的公式如下：  
g(z)=11+e−zg(z)=\frac{1}{1+e^{-z}}*g*(*z*)=1+*e*−*z*1​  
sigmod函数当zz*z*趋近于无穷大时，g(z)g(z)*g*(*z*)趋近于1；当zz*z*趋近于无穷小时，g(z)g(z)*g*(*z*)趋近于0。函数的图像如下所示：  
  
可能这时候会有点好奇，为什么要把线性回归的值给映射到（0,1）之间呢？这样做有什么好处？我们可以联想之前的感知机模型，其损失函数是为了让误分类的个数最小，但是个数这种离散值不容易计算梯度，所以才会有一个相同功能的函数的替换。sigmoid函数也是类似，当特征线性求和之后，我们要把正例和反例的结果相差特别远，才会对新的实例产生一个较好的分类结果，换句话说，也就是使得属于某一类的概率非常高，这就弥补了之前感知机分类的可能存在多个超平面的情形。  
一个特征线性求和函数如下,如果令b=w0,x0=1b = w\_0,x\_0=1*b*=*w*0​,*x*0​=1，则：  
wTx+b=w0+∑ni=1wixi=∑ni=0wixi=θTxw^Tx+b = w\_0+\sum\limits\_{i=1}^nw\_ix\_i=\sum\limits\_{i=0}^nw\_ix\_i=\theta^Tx*wTx*+*b*=*w*0​+*i*=1∑*n*​*wi*​*xi*​=*i*=0∑*n*​*wi*​*xi*​=*θTx*  
(w,b)(w,b)(*w*,*b*)就可以写成一个参数θ\theta*θ*，将上面这个线性特征和带入到sigmoid函数，就能得到逻辑回归的表达式：  
hθ(x)=g(θTx)=11+e−θTxh\_\theta(x)=g(\theta^Tx)=\frac{1}{1+e^{-\theta^Tx}}*hθ*​(*x*)=*g*(*θTx*)=1+*e*−*θTx*1​  
现在我们就可以将y的取值hθ(x)h\_\theta(x)*hθ*​(*x*)通过sigmoid函数归一化到（0,1）之间，y的取值表示取1的概率：  
P(y=1∣x;θ)=hθ(x)P(y=0∣x;θ)=1−hθ(x)\begin{array}{lr}P(y=1|x;\theta) = h\_\theta(x) &amp; \\ P(y=0|x;\theta) = 1-h\_\theta(x)\end{array}*P*(*y*=1∣*x*;*θ*)=*hθ*​(*x*)*P*(*y*=0∣*x*;*θ*)=1−*hθ*​(*x*)​  
对上面的表达式合并就是  
P(y∣x;θ)=(hθ(x))y(1−hθ(x))1−yP(y|x;\theta) = (h\_\theta(x))^y(1-h\_\theta(x))^{1-y}*P*(*y*∣*x*;*θ*)=(*hθ*​(*x*))*y*(1−*hθ*​(*x*))1−*y*

**梯度上升**

得到了逻辑回归的表达式，我们就可以来进行求解θ\theta*θ*了，回想一下概率论，如果我们想在模型参数未知，但数据已知的情形下，怎么才能找到最有可能产生这些数据的参数呢？  
答案就是采用极大似然法。采用梯度上升使得概率最大。  
我们可以写出这里的似然函数：  
L(θ)=∏mi=1p(y(i)∣x(i);θ)=∏mi=1(hθ(x(i)))y(i)(1−hθ(x(i)))1−y(i)L(\theta) = \prod\_{i=1}^mp(y^{(i)}|x^{(i)};\theta)=\prod\_{i=1}^m(h\_\theta(x^{(i)}))^{y^{(i)}}(1-h\_\theta(x^{(i)}))^{1-y^{(i)}}*L*(*θ*)=*i*=1∏*m*​*p*(*y*(*i*)∣*x*(*i*);*θ*)=*i*=1∏*m*​(*hθ*​(*x*(*i*)))*y*(*i*)(1−*hθ*​(*x*(*i*)))1−*y*(*i*)  
对似然函数取对数：  
l(θ)=logL(θ)=∑mi=1y(i)logh(x(i))+(1−y(i))log(1−h(x(i)))l(\theta) = \log L(\theta) = \sum\_{i=1}^my^{(i)}\log h(x^{(i)})+(1-y^{(i)})\log (1-h(x^{(i)}))*l*(*θ*)=log*L*(*θ*)=*i*=1∑*m*​*y*(*i*)log*h*(*x*(*i*))+(1−*y*(*i*))log(1−*h*(*x*(*i*)))  
转换后的似然函数对θ\theta*θ*取偏导，在这里以一个训练样本为例：  
∂∂θjl(θ)=(y1g(θTx)−(1−y)11−g(θTx))∂∂θjg(θTx)=(y1g(θTx)−(1−y)11−g(θTx))g(θTx)(1−g(θTx))∂∂θjθTx=(y(1−g(θTx))−(1−y)g(θTx))xj=(y−hθ(x))xj\begin{aligned} \frac{\partial}{\partial\theta\_j}l(\theta)&amp;= (y\frac{1}{g(\theta^Tx)}-(1-y)\frac{1}{1-g(\theta^Tx)})\frac{\partial}{\partial\theta\_j}g(\theta^Tx) \\ &amp;=(y\frac{1}{g(\theta^Tx)}-(1-y)\frac{1}{1-g(\theta^Tx)})g(\theta^Tx)(1-g(\theta^Tx))\frac{\partial}{\partial\theta\_j}\theta^Tx \\&amp;=(y(1-g(\theta^Tx))-(1-y)g(\theta^Tx))x\_j \\&amp;=(y-h\_\theta(x))x\_j\end{aligned}∂*θj*​∂​*l*(*θ*)​=(*yg*(*θTx*)1​−(1−*y*)1−*g*(*θTx*)1​)∂*θj*​∂​*g*(*θTx*)=(*yg*(*θTx*)1​−(1−*y*)1−*g*(*θTx*)1​)*g*(*θTx*)(1−*g*(*θTx*))∂*θj*​∂​*θTx*=(*y*(1−*g*(*θTx*))−(1−*y*)*g*(*θTx*))*xj*​=(*y*−*hθ*​(*x*))*xj*​​  
求出偏导之后，我们就可以通过迭代来更新θ\theta*θ*了，更新公式如下：  
θj:=θj+α(y(i)−hθ(x(i)))xij\theta\_j :=\theta\_j+\alpha(y^{(i)}-h\_\theta(x^{(i)}))x\_j^{i}*θj*​:=*θj*​+*α*(*y*(*i*)−*hθ*​(*x*(*i*)))*xji*​  
如果有细心的同学注意的话，这个函数的梯度其实跟最小二乘函数的梯度是一致的。

**Python实现**

import numpy as np

import random

import math

import matplotlib.pyplot as plt

#sigmoid function

def sigmoid(z):

return 1/(1+math.e\*\*(-z))

# train function to get weight and bias

def training():

train\_data1 = [[1,3,1], [2,5,1], [3,8,1], [2,6,1]] #positive sample

train\_data2 = [[3,1,0], [4,1,0], [6,2,0], [7,3,0]] #negative sample

train\_data = train\_data1 + train\_data2;

theta = [0,0,0]

learning\_rate = 0.1

train\_num = int(input("train num:"))

for i in range(train\_num):

train = random.choice(train\_data)

x1,x2,y = train;

y\_predict = sigmoid(theta[0]\*1 + theta[1]\*x1 + theta[2]\*x2 )

print("train data:x:(%d, %d) y:%d ==>y\_predict:%d" %(x1,x2,y,y\_predict))

theta[0] = theta[0] + learning\_rate\*(y-y\_predict)\*1

theta[1] = theta[1] + learning\_rate\*(y-y\_predict)\*x1

theta[2] = theta[2] + learning\_rate\*(y-y\_predict)\*x2

print("update theta:")

print(theta[0], theta[1], theta[2])

print("stop training :")

print(theta[0], theta[1], theta[2])

#plot the train data and the hyper curve

plt.plot(np.array(train\_data1)[:,0], np.array(train\_data1)[:,1],'ro')

plt.plot(np.array(train\_data2)[:,0], np.array(train\_data2)[:,1],'bo')

x\_1 = []

x\_2 = []

for i in range(-10,10):

x\_1.append(i)

x\_2.append((-theta[1]\*i-theta[0])/theta[2])

plt.plot(x\_1,x\_2)

plt.show()

return theta

#test function to predict

def test():

theta = training()

while True:

test\_data = []

data = input("enter q to quit,enter test data (x1, x2):")

if data == 'q':

break

test\_data += [int(n) for n in data.split(',')]

predict = sigmoid(theta[1]\*test\_data[0] + theta[2]\*test\_data[1] + theta[0])

if(predict>0.5):

print("predict==>1,probability==>%.3f" %predict)

else:

print("predict==>0,probability==>%.3f" %predict)

if \_\_name\_\_ == "\_\_main\_\_":

test()

* 1
* 2输出结果如下：  
  