Classification and logistic regression

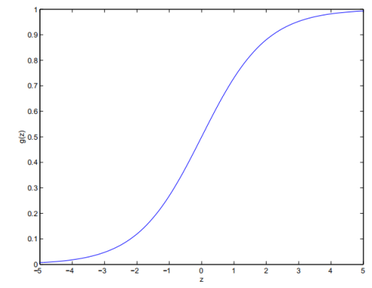
逻辑回归

Logistic regression

分类问题用线性函数效率很低，效果也很差，所以我们提出sigmoid函数

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

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}},$$



寻找拟合theta的方法

sigmoid函数的导数性质

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\ &= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

在一系列假设的前提下，最小二乘法公式可以用最大似然估计来推出，我们同样用最大似然估计来推出逻辑回归拟合theta的公式

1. 概率假设

$$\begin{aligned} P(y=1 \mid x;\theta) &= h_\theta(x) \\ P(y=0 \mid x;\theta) &= 1-h_\theta(x) \end{aligned}$$

也就是

$$p(y \mid x;\theta) = (h_\theta(x))^y \left(1-h_\theta(x)\right)^{1-y}$$

2. 写出似然函数

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X;\theta) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1-h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

3. 取对数似然

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log(1-h(x^{(i)})) \end{aligned}$$

4. 求最大似然函数（求导）

$$\begin{aligned} \frac{\partial}{\partial\theta_j}\ell(\theta) &= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right)\frac{\partial}{\partial\theta_j} g(\theta^T x) \\ &= \left(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)}\right) g(\theta^T x)(1-g(\theta^T x))\frac{\partial}{\partial\theta_j}\theta^T x \\ &= \left(y(1-g(\theta^T x)) - (1-y)g(\theta^T x)\right)x_j \\ &= \left(y-h_\theta(x)\right)x_j \end{aligned}$$

5. theta更新规则采用梯度上升法（stochastic gradient ascent rule）

$$\theta := \theta + \alpha\nabla_\theta\ell(\theta)$$

6. 最终得到随机梯度上升规则

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}$$

与最小二乘法形式上一样，但完全不同，因为这里的h(x)是sigmoid函数，而不是线性函数

感知机

The perceptron learning algorithm

threshold function

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}.$$

另一种最大化ℓ(θ)的算法——牛顿法

用于求方程零点

不断迭代，更新theta，直到f(theta) = 0

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}.$$

1. 牛顿法是用来求函数零点的，如果我们令f(θ) = ℓ'(θ)的导数，也就是说用牛顿法来求ℓ'(θ)导数的零点，就可以求出最大化ℓ(θ)的θ值

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}.$$

2. 在逻辑回归中，θ是一个向量，所以要把牛顿法扩展到多维空间，叫做牛顿-拉普森法（Newton-Raphson method）

$$\theta := \theta - H^{-1}\nabla_\theta\ell(\theta).$$

where H is Hessian matrix,

$$H_{ij} = \frac{\partial^2\ell(\theta)}{\partial\theta_i\partial\theta_j}.$$

代表二阶导数

总结

牛顿法是二次收敛，要比梯度下降收敛的更快，但性能消耗更大，因为每次迭代都需要计算一个Hessian 矩阵