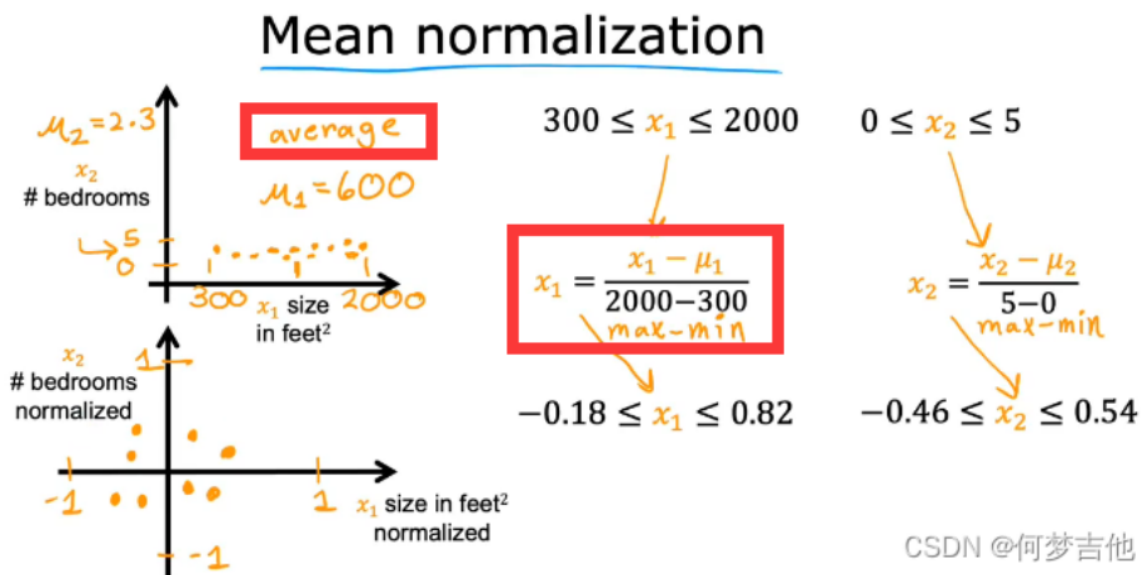


特征归一化

用于数据预处理，使数据处在合理范围内，能使偏离值较为均匀的变化，使得梯度下降更便于找到最好值



逻辑回归

sigmoid 函数

g 代表一个常用的逻辑函数 (logistic function) 为S形函数 (Sigmoid function)，公式为：

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

合起来，我们得到逻辑回归模型的假设函数：

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

代价函数

逻辑回归的损失函数Loss 即期望若落空造成的损失

Logistic loss function

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

简化代价函数

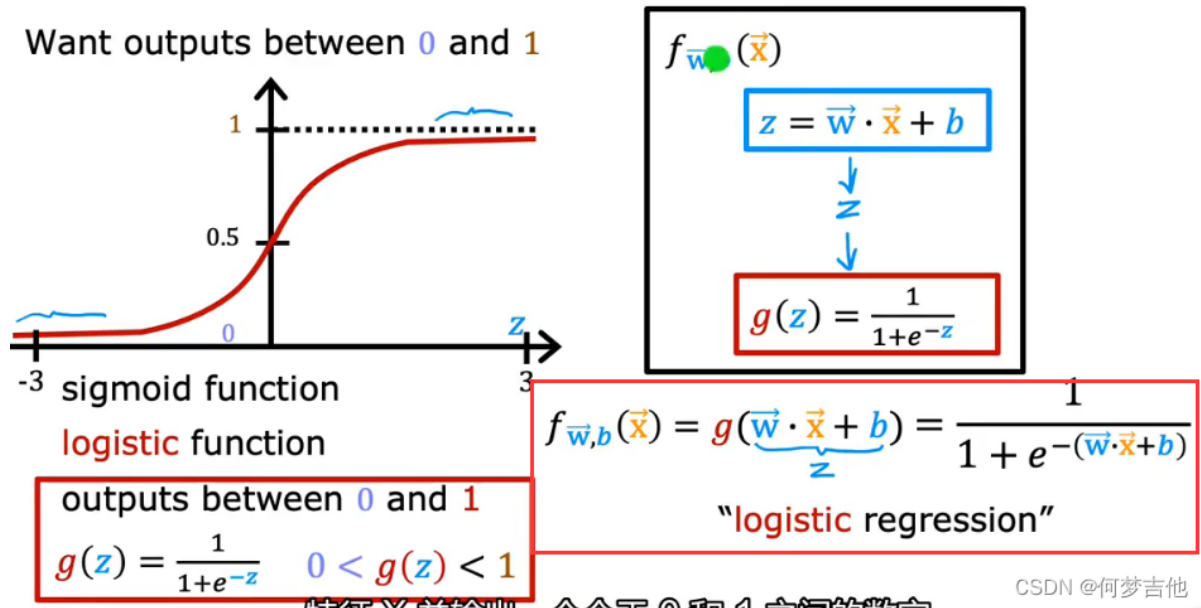
$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$$

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) \right]$$

这使得Loss也具有了碗的形状，使得逻辑回归也能使用梯度下降算法寻找极值

逻辑回归模型



简化代价函数与梯度下降

关注 Simplified loss function

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))$$

进而得到：

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) \right]$$

梯度下降：

Gradient descent for logistic regression

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) \right]$$

} simultaneous updates

最后发现与正常的线性拟合公式相同

过拟合

当变量过多时，训练出来的假设能很好地拟合训练集，所以代价函数实际上可能非常接近于0，但得到的曲线为了千方百计的拟合数据集，导致它无法泛化到新的样本中，无法预测新样本数据（冗余项）

正则化

正则化项：

$$\lambda \sum_{j=1}^n \theta_j^2$$

如果有很多参数，我们不清楚哪个参数是高级项，即不知道惩罚哪个能获得更好拟合的结果，因此引入正则化项统一惩罚参数以得到较为简单的函数

正则化后：

Regularized linear regression

$$\min_{\vec{w}, b} J(\vec{w}, b) = \min_{\vec{w}, b} \left[\frac{1}{2m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2 \right]$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

$j=1, \dots, n$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

$$= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j$$

$$= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

don't have to

$$\begin{aligned}
 &\text{repeat } \{ \\
 &w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m \left[(f_{\bar{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right] \\
 &b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\bar{w},b}(\vec{x}^{(i)}) - y^{(i)}) \\
 &\} \text{ simultaneous update } j=1 \dots n \\
 &w_j = \underbrace{1w_j - \alpha \frac{\lambda}{m} w_j}_{w_j \left(1 - \alpha \frac{\lambda}{m}\right)} - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}}_{\text{usual update}}
 \end{aligned}$$

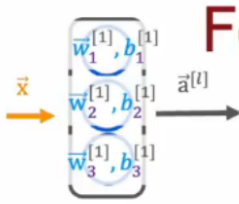
$$\begin{aligned}
 &\alpha \frac{\lambda}{m} \\
 &0.01 \frac{1}{50} = 0.0002 \\
 &w_j (1 - 0.0002) \\
 &0.9998
 \end{aligned}$$

得到最终公式

神经网络

输入前一层的激活，给定当前层的参数，它会输出下一层激活值 自定义sequential函数
中间层为网络层、隐藏层，最后输出为激活层

前向传播



Forward prop in NumPy

$\vec{w}_1^{[1]} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
 $\vec{w}_2^{[1]} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}$
 $\vec{w}_3^{[1]} = \begin{bmatrix} 5 \\ -6 \end{bmatrix}$

$W = \text{np.array}(\begin{bmatrix} 1 & -3 & 5 \\ 2 & 4 & -6 \end{bmatrix})$ 2 by 3

$b_1^{[1]} = -1$
 $b_2^{[1]} = 1$
 $b_3^{[1]} = 2$

$b = \text{np.array}([-1, 1, 2])$

$\vec{a}^{[0]} = \vec{x}$

```

def dense(a_in, W, b, g):
    units = W.shape[1]
    a_out = np.zeros(units)
    for j in range(units):
        w = W[:, j]
        z = np.dot(w, a_in) + b[j]
        a_out[j] = g(z)
    return a_out

def sequential(x):
    a1 = dense(x, W1, b1)
    a2 = dense(a1, W2, b2)
    a3 = dense(a2, W3, b3)
    a4 = dense(a3, W4, b4)
    f_x = a4
    return f_x
    
```

capital W refers to a matrix