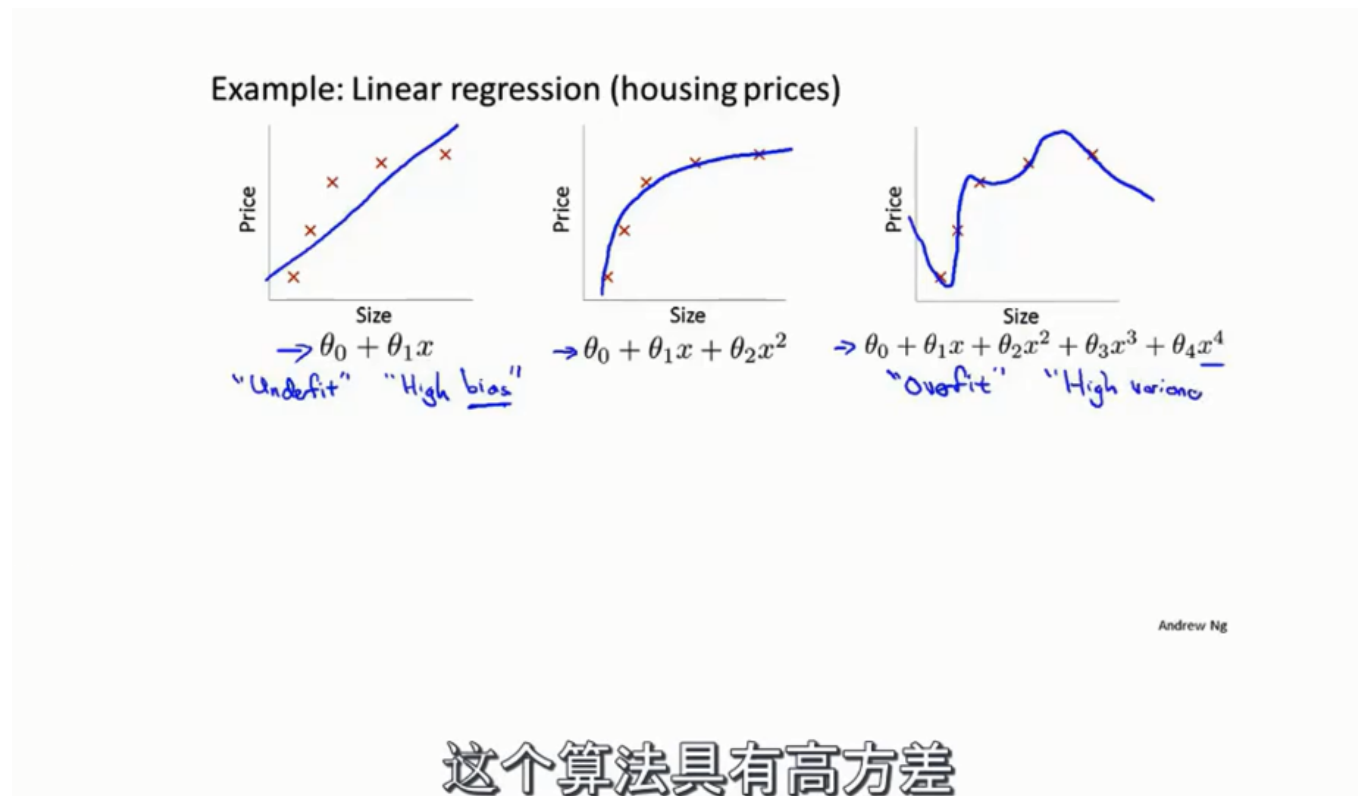


正则化_吴恩达_机器学习笔记

1、过拟合问题的产生

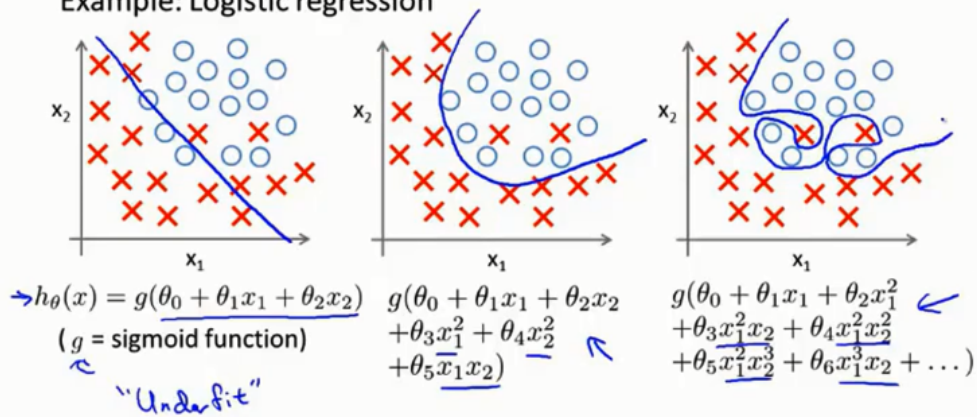
过拟合问题存在高方差，且模型泛化能力很弱，基本上在新的预测数据上，会导致无法对新数据进行正确预测。

案例一：线性回归的案例



案例二：逻辑斯蒂回归案例

Example: Logistic regression



Andrew Ng

来符合每一个训练样本

课程的后半段，将重点将如何识别过拟合和欠拟合的诊断工具和方法。

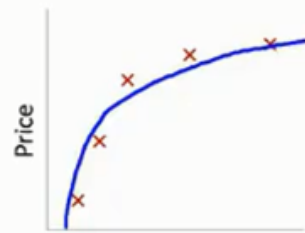
解决过拟合问题有两个方向可以发力：

- 减少特征量。但是如果特征量全部与预测结果相关，那么丢失掉的信息也对预测数据降低了帮助。
- 正则化。本次主要说正则化问题。正则化将保留所有的特征数量用于建模，但是会改变参数 θ 的大小或者是特征的量级。这就保证了特征全部被保留用于预测结果提升。

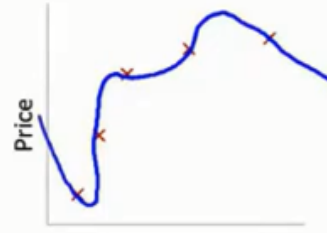
2、代价函数

先来看一个案例，能够更好的理解正则化的惩罚项的作用。从最简答的线性回归开始：

Intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Suppose we penalize and make θ_3, θ_4 really small.

$$\rightarrow \min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + 1000 \theta_3^2 + 1000 \theta_4^2$$

Andrew Ng

就是 θ_3 和 θ_4 要尽可能小

在这个案例中，后面蓝色字体的就是惩罚项，首先惩罚项的X都是一个很大的值，这样的话，我们就需要选择一个很小很小的 θ_3 和 θ_4 （两个参数的取值都接近于0），才能保证后面蓝色的项目最终的取值结果很小。类似于直接去掉了 θ_3 和 θ_4 项目。

这样的话，实质上就是第一个图的二次函数加上了一个很小的数，那么，整个过拟合的图的拟合函数，就会非常接近图一，虽然不是完整的二次函数，但是非常接近二次函数的图像。

正则化的思想，补充的就是，目的是让所有的参数的取值更小，更小的取值意味着模型更加的简单。这样拟合出来的函数越平滑，也越不会过拟合。

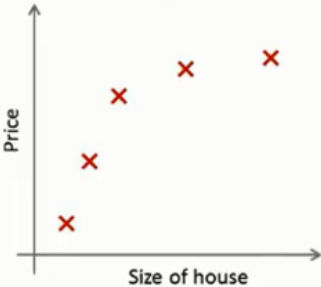
吴大佬说了，要理解这样的结果，恐怕需要自己去尝试实验一下，看看参数取值大和取值小的情况下，函数图像是不是越平滑。

经测试以后发现，确实如此，当添加的 θ 参数非常大的时候，函数的图像基本上是非常扭曲的。具体可以使用函数作图工具测试。

Regularization.

$$\min_{\theta} J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

regularization parameter



Andrew Ng

之间的平衡关系

仔细看这个图，前面部分是模型要尽可能的拟合数据，而从 λ 以后的项，是惩罚项，目的是保持参数尽可能的小，虽然这个公式为何就能实现参数尽可能小的目标很难解释，但是不如尝试做一次实验，应该就可以更信服这个结论。

λ 在这里叫做正则化参数，一般而言，目前的算法基本上都能够自动选择正则化参数的大小，不需要自己设置。当然，如果设置的正则化系数过大，那么不好意思，会造成欠拟合的情况，原因就是所有的参数都为0了，那么最后参数项就只剩下 θ_0 一个常数项了，就是一条直线，直线是不能拟合好复杂的数据的。

3、线性回归的正则化

3.1 梯度下降算法的正则化

线性回归的正则化中，由于求解极值问题，有两种方式，一个是梯度下降算法，一个是正规方程算法。以下分别对两种算法加上正则化项后的效果进行阐述：

Gradient descent

Repeat {

$$\rightarrow \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad (j = 1, 2, 3, \dots, n)$$

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$1 - \alpha \frac{\lambda}{m} < 1$$

0.99

 $\theta_j \times 0.99$

$$\theta_j^2$$

Andrew Ng

我们每次都把参数缩小一点

首先去看加入正则化项后的偏导数结果，其中后面的结果是 $\lambda/m * \theta_j$

这里还需要看到，其实 θ_0 的迭代策略和其他 θ 参数的迭代策略是不一致的，这个在线性回归的时候讲过为什么，可以回去看视频。

但是这里重点说一下这个公式变形以后的结果，那么 $1 - \alpha * \lambda/m$ 这一项，实际上是一个非常接近于1的项，可以理解为0.99这样的值，那么实际上，每次迭代， θ_j 都是乘以一个0.99的值来让这个参数变小的，经过几百次，几千次，甚至几百万次迭代以后，那么每个参数 θ ，就会变得比以前小很多。

3.2 正规方程算法的正则化

使用正规方程添加正则化项目，会在原来的公式里面添加一个很像单位矩阵的矩阵，具体示例如图

Normal equation

$$\begin{aligned}
 \underline{X} &= \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \quad \begin{matrix} \leftarrow \\ \leftarrow \end{matrix} \\
 &\quad m \times (n+1) \\
 &\rightarrow \min_{\theta} J(\theta) \\
 &\rightarrow \Theta = \left(X^T X + \lambda \underbrace{\begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}}_{(n+1) \times (n+1)} \right)^{-1} X^T y \\
 &\quad \text{e.g. } n=2 \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 &\quad \text{对角线外其他元素都是0}
 \end{aligned}$$

$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \mathbb{R}^m$
 $\frac{\partial}{\partial \theta_j} J(\theta) \stackrel{\text{set}}{=} 0$

实际上这个图上的lambda后面，就是一个类似单位矩阵的东西。

同时，由于之前沟通了正规方程的不可逆问题，就是奇异矩阵。那么这次，加上这个正则化项以后，能够保证，这个矩阵无论是什么样的奇异矩阵，都不需要再做奇异值分解了，直接就会变成一个非奇异矩阵，那么一定有逆矩阵，就是这么神奇。就是这么有尿性。

4、Logistic回归的正则化

对于逻辑斯蒂回归，仍然用一张图来表示正则化项和更新规则：

Advanced optimization

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ $\theta_0 \leftarrow \text{theta}(1)$
 $\theta_1 \leftarrow \text{theta}(2)$
 $\theta_n \leftarrow \text{theta}(n+1)$

```

function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(theta)];
    →  $J(\theta) = \left[ -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \left[ \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2 \right]$ 
    → gradient(1) = [code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$ ];
         $\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \leftarrow$ 
    → gradient(2) = [code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$ ];
         $\left( \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)} \right) + \frac{\lambda}{m} \theta_1 \leftarrow$ 
    → gradient(3) = [code to compute  $\frac{\partial}{\partial \theta_2} J(\theta)$ ];
         $\vdots \left( \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)} \right) + \frac{\lambda}{m} \theta_2$ 
    gradient(n+1) = [code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$ ];
  
```

Andrew Ng

正则化逻辑回归的解

后面的 $\lambda/2m \sum \theta_j^2$ 项就是正则化项。

上面的图还展示了如何求偏导数的顺序和偏导数求出来以后的结果。可以进行了了解，实质上就是前面的结果加上一个对应参数的正则化项。