

1、 正则化(Regularization)

7.1 过拟合的问题

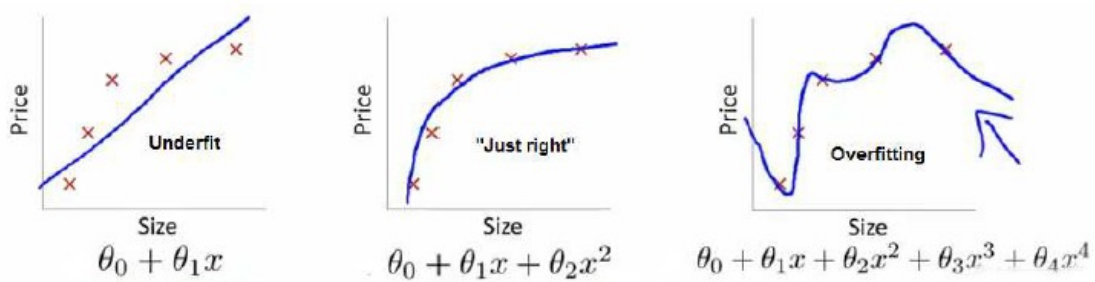
参考视频: 7 - 1 - The Problem of Overfitting (10 min).mkv

遇到**过拟合(over-fitting)**的问题，可能会导致它们效果很差。

正则化(regularization)的技术，它可以改善或者减少过度拟合问题。

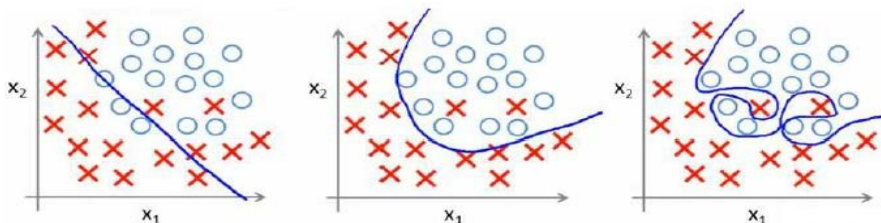
如果我们有非常多的特征，我们通过学习得到的假设可能能够非常好地适应训练集（代价函数可能几乎为0），但是可能会不能推广到新的数据。

下图是一个回归问题的例子：



第一个模型是一个线性模型，欠拟合，不能很好地适应我们的训练集；第三个模型是一个四次方的模型，过于强调拟合原始数据，而丢失了算法的本质：预测新数据。我们可以看出，若给出一个新的值使之预测，它将表现的很差，是过拟合，虽然能非常好地适应我们的训练集但在新输入变量进行预测时可能会效果不好；而中间的模型似乎最合适。

分类问题中也存在这样的问题：



就以多项式理解， x 的次数越高，拟合的越好，但相应的预测的能力就可能变差。

1. 丢弃一些不能帮助我们正确预测的特征。可以是手工选择保留哪些特征，或者使用一些模型选择的算法来帮忙（例如 **PCA**）

2. 正则化。 保留所有的特征，但是减少参数的大小（**magnitude**）。

7.2 代价函数

参考视频: 7 - 2 - Cost Function (10 min).mkv

上面的回归问题中如果我们的模型是:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$$

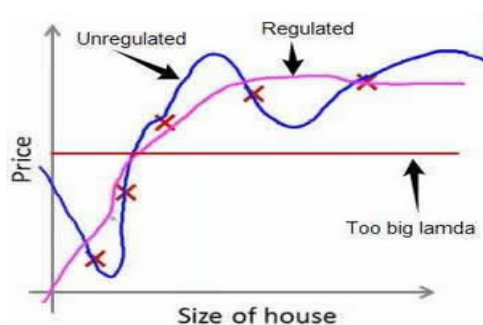
我们可以从之前的事例中看出,正是那些**高次项导致了过拟合的产生**,所以如果我们能让这些高次项的系数接近于0的话,我们就能很好的拟合了。

所以我们要做的就是在一定程度上减小这些参数 θ 的值,这就是正则化的基本方法。我们决定要减少 θ_3 和 θ_4 的大小,我们要做的便是修改代价函数,在其中 θ_3 和 θ_4 设置一点惩罚。这样做的话,我们在尝试最小化代价时也需要将这个惩罚纳入考虑中,并最终导致选择较小一些的 θ_3 和 θ_4 。

修改后的代价函数如下: $\min_{\theta} \frac{1}{2m} [\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + 1000\theta_3^2 + 10000\theta_4^2]$

通过这样的代价函数选择出的 θ_3 和 θ_4 对预测结果的影响就比之前要小许多。假如我们有非常多的特征,我们并不知道其中哪些特征我们要惩罚,我们将**对所有的特征进行惩罚**,并且让代价函数最优化的软件来选择这些惩罚的程度。这样的结果是得到了一个较为简单的能防止过拟合问题的假设: $J(\theta) = \frac{1}{2m} [\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2]$

其中 λ 又称为**正则化参数 (Regularization Parameter)**。注:根据惯例,我们不对 θ_0 进行惩罚。经过正则化处理的模型与原模型的可能对比如下图所示:



如果选择的正则化参数 λ 过大,则会把所有的参数都最小化了,导致模型变成 $h_{\theta}(x) = \theta_0$, 也就是上图中红色直线所示的情况,造成欠拟合。

7.3 正则化线性回归

参考视频: 7 - 3 - Regularized Linear Regression (11 min).mkv

对于线性回归的求解，我们之前推导了两种学习算法：一种基于梯度下降，一种基于正规方程。

正则化线性回归的代价函数为：

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

如果我们要使用梯度下降法令这个代价函数最小化，因为我们未对进行正则化，所以梯度下降算法将分两种情形：

Repeat until convergence{

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

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

}

Repeat

对上面的算法中 $j = 1, 2, \dots, n$ 时的更新式子进行调整可得：

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

Gradient descent

Repeat {

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

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

(j = 1, 2, 3, ..., n)

}

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

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

$$0.99$$

$$\theta_j \times 0.99$$

可以看出，正则化线性回归的梯度下降算法的变化在于，每次都在原有算法更新规则的基础上令 θ 值减少了一个额外的值。

我们同样也可以利用正规方程来求解正则化线性回归模型，方法如下所示：

Non-invertibility (optional/advanced).

Suppose $m \leq n$,
(#examples) (#features)

$$\theta = (X^T X)^{-1} X^T y$$

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

Andrew Ng

图中的矩阵尺寸为 $(n + 1) * (n + 1)$ 。

7.4 正则化的逻辑回归模型

参考视频: 7 - 4 - Regularized Logistic Regression (9 min).mkv

自己计算导数同样对于逻辑回归，我们也给代价函数增加一个正则化的表达式，得到代价函数：

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

Python 代码：

```
import numpy as np
def costReg(theta, X, y, learningRate):
    theta = np.matrix(theta)
    X = np.matrix(X)
    y = np.matrix(y)
    first = np.multiply(-y, np.log(sigmoid(X*theta.T)))
    second = np.multiply((1 - y), np.log(1 - sigmoid(X*theta.T)))
    reg = (learningRate / (2 * len(X)) * np.sum(np.power(theta[:,1:theta.shape[1]],2))
    return np.sum(first - second) / (len(X)) + reg
```

要最小化该代价函数，通过求导，得出梯度下降算法为：

Repeat until convergence

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

$$\theta_j := \theta_j - a [\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j]$$

for $j = 1, 2, \dots, n$

}

注：看上去同线性回归一样，但是知道 $h_{\theta}(x) = g(\theta^T X)$ ，所以与线性回归不同。

Octave 中，我们依旧可以用 **fminuc** 函数来求解代价函数最小化的参数，值得注意的是参数 θ_0 的更新规则与其他情况不同。

注意：

1. 虽然正则化的逻辑回归中的梯度下降和正则化的线性回归中的表达式看起来一样，但由于两者的 $h_{\theta}(x)$ 不同所以还是有很大差别。
2. θ_0 不参与其中的任何一个正则化。