线性回归

线性回归简单理解就是：可以将输入项分别乘以一些常量，再将结果加起来得到输出。这些常量就是回归系数，预测的关键就是找到最佳拟合直线，也即找到最优的回归系数w。

[注]数学表示的意义：
n： 特征feature的个数
$x^{(i)}$: 第i个训练集中的输入
$x_j^{(i)}$: 第i个训练集中的特征j的值

首先确定假设函数的形式，对于线性回归，这里可以初始化采用简单的线性函数来逼近y：

$$h_\theta(x) = \theta_0 + \theta_1 + \theta_2$$

其中的θ_i表示系数（权重），也可以用下式的w_i表示：$y = w^T x$。
我们对上式做个处理：令$x_0 = 1$(即截距项),补上这一项之后有：

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

那接下来系数θ要如何选择呢?

我们的目的就是尽可能的使得假设函数的值与真实输出值y接近，为了形式化这个“尽量接近”的问题，就需要引入均方误差的概念了：对于任意样本标注 (x,y) 和模型的预测值 h(x)，均方误差表示为标注值 y 和 预测值 h(x) 差的平方：

$$Error = (h(x) - y)^2$$

接下来我们的目标就是对于训练数据中的所有 N 个样本点，使平均均方误差最小。

$$LOSS = J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{2}\sum_{i=1}^{n}\left(\theta^T\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{J}$$

上式被称之为损失函数（Cost Function），也即最小二乘回归模型中常见的最小二乘成本函数。

损失函数的值越小，假设函数和真实输出之间的误差越小，预测结果就越准确，接下里就是算法

的核心：找到最小的损失函数值

损失函数最小值

方法1：最小均方法（LMS: Least mean squares algorithm）

该算法是一种搜索算法，即算法起始于某个关于θ的“初始猜测值”，然后不断的修改θ以使J(θ)减小，直到最终θ收敛于使得J(θ)取最小处。
至此便可以很直观的联系起高数中**梯度**的概念【
1、方向导数是各个方向上的导数
2、偏导数连续才有梯度存在
3、梯度的方向是方向导数中取到最大值的方向，梯度的值是方向导数的最大值】，而这里为了找到损失函数的最小值，需要引入梯度下降（gradient descent）法（梯度下降从字面意可以看出朝着梯度的反方向变动，函数值下降最快）。
【附】梯度下降的一个直观的解释
具体的算法步骤：
算法从某个初始的θ起，不断更新$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$（:=表示赋值，该赋值运算同时作用于所有的$\theta_j$，$j = 0, \cdots, n$）。赋值运算中的$\alpha$称为学习速率，通常采用手动设置（learning rate，在本例中，它决定了我们“向下山走”时每一步的大小，过小的话收敛太慢，过大的话可能错过最小值）。这是一种很自然的算法，每一步总是寻找使J下降最“陡”的方向（就像找最快下山的路一样）。学习速率的意义在于每次移动的速率，如果移动的步伐太大就可能越过最低点，如果移动的步伐太小则会导致梯度下降的速度太慢，收敛到极值点耗时太久。
赋值运算中的求偏导数的意义：表示曲线上定点的斜率。

如果一开始θ_i就处于局部最低点，那么该点的倒数恒为0，梯度下降算法将保持不变。

随着梯度下降算法的运行，会发现$\frac{\partial}{\partial \theta_j} J(\theta)$是逐渐变小的，即使学习速率不变，$\alpha \frac{\partial}{\partial \theta_j} J(\theta)$的值也会自动变小，，即移动的幅度会越来越小，直到收敛到局部最低点。

接下里我们以含有两个参数的假设函数为例来说明：

$$h_\theta(x) = \theta_0 + \theta_1 x$$

损失函数：

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

其中j取0和1，分别对应θ_j的下标
接下来对$J(\theta_0, \theta_1)$求导：

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(\theta_0 + \theta_1 x^{(i)}) - y^{(i)}\right)^2$$

当j=0:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

当y=1:

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{n} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$$

不断的同步更新θ_0、θ_1（对于所有的j，不断的迭代下面的算式直到θ_j收敛）：

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

$\theta_1 := \theta_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x^{(i)}$

这个方法其实就是对$J(\theta)$原始的成本函数做了简单的梯度下降。该方法在每一步搜索“最陡方向”时都会遍历整个训练集，所以也被称作批量梯度下降（batch gradient descent）。值得注意的是，梯度下降可以容易的达到局部最小值，而且我们此处的优化问题也只有一个全局最优解（没有局部最小值）。因此，在本例中，梯度下降总是收敛于全局最小值（当然学习速率α不能过大）。况且成本函数J确实是一个凸二次函数（仅有一个全局最小值）。

> 图中的椭圆为某二次函数的等高线，同时也显示了梯度下降法从初始值(48，30)到最小值中间的轨迹。图中的“×”标记了梯度下降过程中经过的一系列θ值。
>
> 关于收敛，我们可以查看两侧迭代是否相差很多，如果相差无几则可以判断收敛；更常用的方法是检查$J(\theta)$，如果这个值不再发生较大变化时，也可以判断收敛。关于找如何“找”最陡下山路径，其实求偏导本身就已经给出了最陡路径。
>
> 我们运行批量梯度下降算法用θ拟合“公寓租金”训练集，以求得根据面积预测价格的函数，最终得到$\theta_0 = 71.27$，$\theta_1 = 0.1345$。当我们在最开始的“面积-价格”图中画出关于面积x的函数$h_\theta(x)$时，有：
>
> 如果将卧室数量也纳入输入特征，则会得到$\theta_0 = 89.60$，$\theta_1 = 0.1392$，$\theta_2 = -8.738$。这个结果也是通过批量梯度下降求得。

> 我们接下来介绍第二种方法：
>
> 在这个方法中，我们每次仅使用一个训练样本，根据由这个样本得到的误差梯度来更新参数θ。整个算法运行完毕时，对每一套θ参数，每个样本只使用了一次。这就是**随机梯度下降（stochastic gradient descent）**，也称作**增量梯度下降（incremental gradient descent）**。相对于批量梯度下降每走一步都需要遍历整个训练集（如果训练集样本很多，即m很大时，就是非常繁重的计算），随机梯度下降每一步就轻松很多，之后就是不断地根据遇到的每一个样本调整参数。通常情况下，随机梯度下降能够比批量梯度下降更快的使θ接近最优解。（然而，值得注意的是，随机梯度下降可能永远都不会收敛于最小值点，参数θ将在$J(\theta)$最小值附近持续摆动。不过，在实践中，最小值附近的解通常都足够接近最小值。另外，在随机梯度下降的实际操作中，随着迭代步骤的进行，我们会慢慢减小α的值至0，这样也可以保证参数收敛于全局最小值，而不是在其附近持续摆动）。也是因为效率原因，在遇到训练集中包含大量样本的情况下，我们通常会选用随机梯度下降法。