# 机器学习算法系列（2）：线性回归

## 一、线性回归模型

线性回归假设特征和结果满足线性关系。其实线性关系的表达能力非常强大，每个特征对结果的影响强弱可以由前面的参数体现，而且每个特征变量可以首先映射到一个函数，然后再参与线性计算。这样就可以表达特征与结果之间的非线性关系。

我们可以有这样的模型表达：

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

其中，$x_1, x_2, \cdots, x_n$表示自变量（特征分量），$y$表示因变量，$\theta_i$表示对应自变量（特征）的权重，$\theta_0$是偏倚项（又称为截距）。

对于参数$\theta$，在物理上可以解释为：在自变量（特征）之间相互独立的前提下，$\theta_i$反映自变量$x_i$对因变量$y$的影响程度，$\theta_i$越大，说明$x_i$对结果$y$的影响越大。因此，我们可以通过每个自变量（特征）前面的参数，可以很直观的看出那些特征分量对结果的影响比较大。

如果令$x_0 = 1, y = h_\theta(x)$，可以将上述模型写成向量形式，即：

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

其中$\theta = (\theta_0, \theta_1, \cdots, \theta_n)$ , $x = (1, x_1, x_2, \cdots, x_n)$均为向量，$\theta^T$为$\theta$的转置。

在上述公式中，假设特征空间与输入空间$x$相同。准确地讲，模型表达式要建立的是特征空间与结果之间的关系。在一些应用场合中，需要将输入空间映射到特征空间中，然后建模，定义映射函数为$\Phi(x)$，因此我们可以把公式写成更通用的表达公式：

$$h_\theta(x) = \theta^T \Phi(x)$$

特征映射相关技术，包括特征哈希、特征学习、$Kernel$等。

## 二、目标函数

### 2.1 目标函数

上面的公式的参数向量$\theta$是$n+1$维的，每个参数的取值是实数集合，也就是说参数向量$\theta$在$n+1$维实数空间中取值结果有无穷种可能。

那么，如何利用一个规则或机制帮助我们评估求得的参数$\theta$，并且使得线性模型效果最佳呢？直观地认为，如果求得参数$\theta$线性求和后，得到的结果$h_\theta(x)$与真实值$y$之差越小越好。

这时我们需要映入一个函数来衡量$h_\theta(x)$表示真实值$y$好坏的程度，该函数称为损失函数（loss function，也称为错误函数）。数学表示如下：

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

$$\min_\theta J(\theta)$$

这个损失函数用的是$x^{(i)}$的预测值$h_\theta(x^{(i)})$与真实值$y^{(i)}$之差的平方和。如果不考虑诸如过拟合等其他问题，这就是我们需要优化的目标函数。

## 2.2 目标函数的概率解释

一般地，机器学习中不同的模型会有相应的目标函数。而回归模型（尤其是线性回归类）的目标函数通常用平方损失函数来作为优化的目标函数（即真实值与预测值之差的平方和）。为什么要选用误差平方和作为目标函数呢？答案可以从概率论中的中心极限定理、高斯分布等知识中找到。

### 2.2.1 中心极限定理

目标函数的概率解释需要用到中心极限定理。中心极限定理本身就是研究独立随机变量和的极限分布为正态分布的问题。

**中心极限定理的公式表示为：**
设$n$个随机变量$X_1,X_2,\cdots,X_n$相互独立，均具有相同的数学期望与方差，即$E(X_i)=\mu;D(X_i)=\sigma^2$，令$Y_n$为随机变量之和，有

$$Y_n=X_1+X_2+\cdots+X_n$$

$$Z_n=\frac{Y_n-E(Y_n)}{\sqrt{D(Y_n)}}=\frac{Y_n-n\mu}{\sqrt{n}\sigma}\rightarrow N(0,1)$$

称随机变量$Z_n$为$n$个随机变量$X_1,X_2,\cdots,X_n$的规范和。

**它的定义为：**
设从均值为$\mu$、方差为$\sigma^2$（有限）的任意一个总体中抽取样本量为$n$的样本，当$n$充分大时，样本均值的抽样分布$\frac{Y_n}{n}$近似服从于均值为$\mu$、方差为$\sigma^2$的正态分布。

### 2.2.2 高斯分布

假设给定一个输入样例$x^{(i)}$根据公式得到预测值$\theta^T x^{(i)}$与真实值$y^{(i)}$之间存在误差，即为$\varepsilon^{(i)}$。那么，它们之间的关系表示如下：

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$$

而这里假设误差$\varepsilon^{(i)}$服从标准高斯分布是合理的。

**解释如下：**

回归模型的最终目标是通过函数表达式建立自变量$x$与结果$y$之间的关系，希望通过$x$能较为准确地表示结果$y$。而在实际的应用场合中，很难甚至不可能把导致$y$的所有变量（特征）都找出来，并放到回归模型中。那么模型中存在的$x$通常认为是影响结果$y$最主要的变量集合（又称为因子，在ML中称为特征集）。根据中心极限定理，把那些对结果影响比较小的变量（假设独立同分布）之和认为服从正态分布是合理的。

可以用一个示例来说明误差服从高斯分布是合理的：

$AndrewNg$的课程中第一节线性回归的例子中，根据训练数据建立房屋的面积$x$与房屋的售价$y$之间的函数表达。
它的数据集把房屋面积作为最为主要的变量。除此之外我们还知道房屋所在的地段（地铁、学区、城区、郊区），周边交通状况，当地房价、楼层、采光、绿化面积等等诸多因素会影响房价。

实际上，因数据收集问题可能拿不到所有影响房屋售价的变量，可以假设多个因素变量相互独立，根据中心极限定理，认为变量之和服从高斯分布。即：

$$\epsilon^{(i)} = y^{(i)} - \theta^T x^{(i)}$$

那么$x$和$y$的条件概率可表示为：

$$p\left(y^{(i)}|x^{(i)};\theta\right) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)$$

### 2.2.3 极大似然估计与损失函数极小化等价

根据上述公式估计得到一条样本的结果概率，模型的最终目标是希望在全部样本上预测最准，也就是概率积最大，这个概率积就是似然函数。优化的目标函数即为似然函数，表示如下：

$$\max_{\theta} L(\theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}\right)$$

对$L(x)$取对数，可得对数似然函数：

$$\max_{\theta} l(\theta) = -m \log \sqrt{2\pi}\sigma - \frac{1}{2\sigma^2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

由于$n, \sigma$都为常数，因此上式等价于

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^{m} \left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

我们可以发现，经过最大似然估计推导出来的待优化的目标函数与平方损失函数是等价的。因此可以得出结论：

线性回归误差平方损失极小化与极大似然估计等价。其实在概率模型中，目标函数的原函数（或对偶函数）极小化（或极大化）与极大似然估计等价，这是一个带有普遍性的结论。比如在最大熵模型中，有对偶函数极大化与极大似然估计等价的结论。

那上面为什么是条件概率$p(y|x;\theta)$呢？因为我们希望预测值与真实值更接近，这就意味着希望求出来的参数$\theta$，在给定输入$x$的情况下，得到的预测值等于真实值得可能性越大越好。而$\theta$，$x$均为前提条件，因此用条件概率$p(y|x;\theta)$表示。即$p(y|x;\theta)$越大，越能说明估计的越准确。当然也不能一味地只有该条件函数，还要考虑拟合过度以及模型的泛化能力问题。

# 三、参数估计

如何调整参数$\theta$使得$J(\theta)$取得最小值？方法有很多，这里介绍几种比较经典的方法，即最小二乘法、梯度下降法以及牛顿法。

## 3.1 最小二乘法

### 3.1.1 目标函数的矩阵形式

将$m$个$n$维样本组成矩阵$X$：

$$\begin{pmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(n)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(n)} \\ \cdots & \cdots & \cdots & & \\ 1 & x_m^{(1)} & x_m^{(2)} & \cdots & x_m^{(n)} \end{pmatrix}$$

则目标函数的矩阵形式为

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

#### 3.1.2 最小二乘法求解

对$\theta$求导，梯度（矩阵求导）：

$$\nabla_\theta J(\theta)=\nabla_\theta\left(\frac{1}{2}(X\theta-y)^T(X\theta-y)\right)$$

$$=\nabla_\theta\left(\frac{1}{2}\left(\theta^TX^TX\theta-\theta^TX^Ty-y^Ty\right)\right)$$

$$=\frac{1}{2}\left(2X^TX\theta-X^Ty-\left(y^TX\right)^T\right)$$

$$=X^TX\theta-X^Ty$$

令其为零，求得驻点：

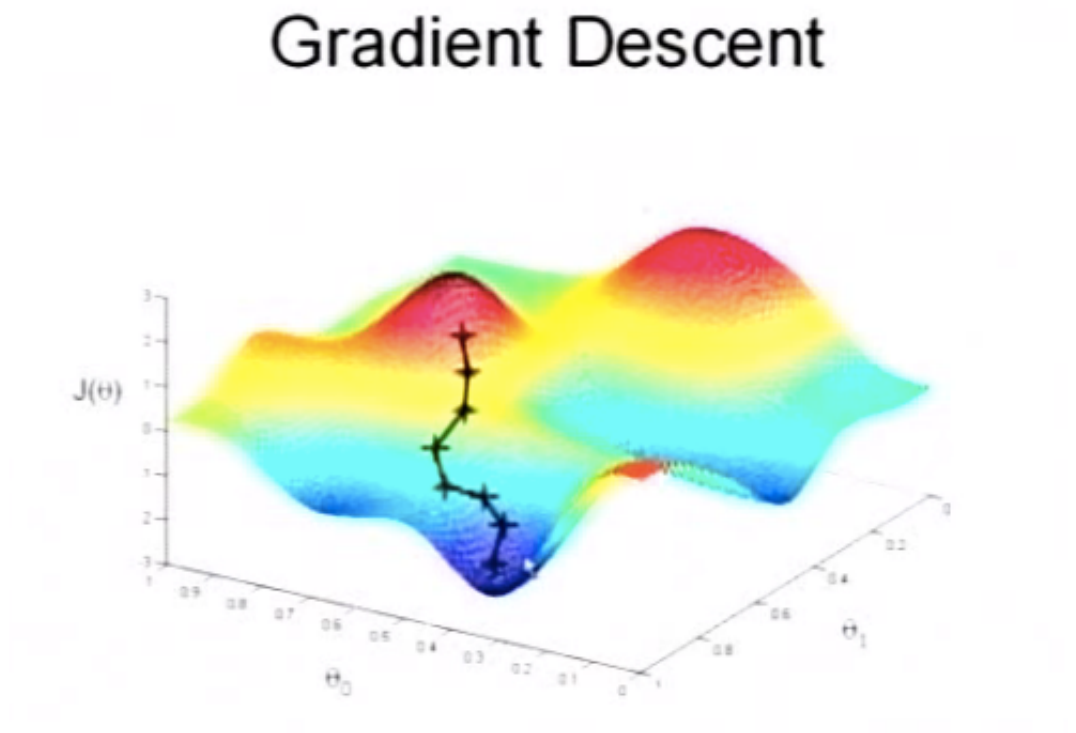
$$\theta=\left(X^TX\right)^{-1}X^Ty$$

## 3.2 梯度下降法

梯度下降法是按下面的流程进行的：

- 1）首先对$\theta$赋值，这个值可以是随机的，也可是让$\theta$是一个全零的向量；
- 2）改变$\theta$的值，使得$J(\theta)$按梯度下降的方向进行减少。

为了更清楚，给出下面的图：

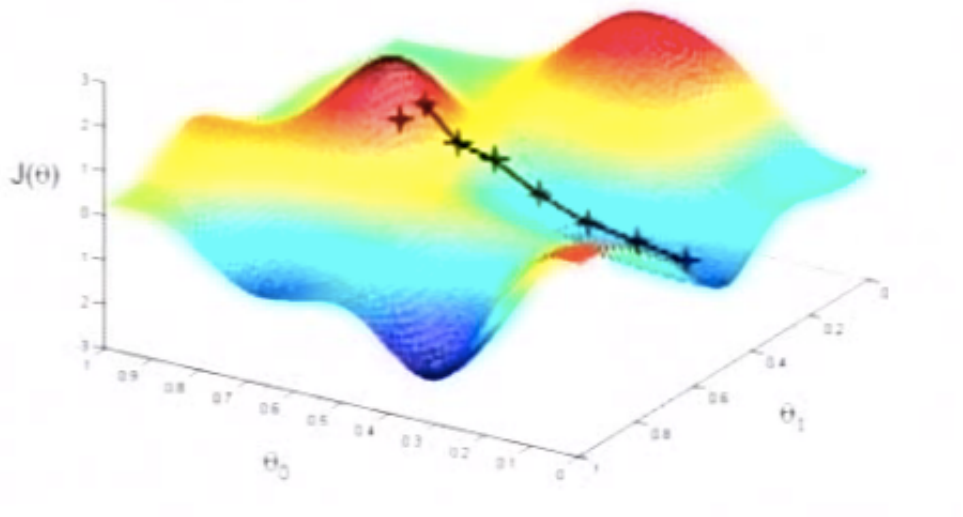



这是一个表示参数$\theta$与目标函数$J(\theta)$的关系图，红色的部分是表示$J(\theta)$有比较高的取值，我们需要的是，能够让$J(\theta)$的值尽量的低。也就是深蓝色的部分。$\theta_0$和$\theta_1$表示$\theta$向量的两个维度。

在上面提到梯度下降法的第一步是给$\theta$一个初值，假设随机给的初值是在图上的十字点。然后我们将$\theta$按照梯度下降的方向进行调整，就会使得$J(\theta)$往更低的方向进行变化，如图所示，算法的结束将是在$\theta$下降到无法继续下降为止。

当然，可能梯度下降的最终点并非是全局最小点，可能是一个局部最小点，比如下面这张图中描述的就是一个局部最小点，这是我们重新选择了一个初始点得到的，看来我们这个算法会在很大程度上被初始点的选择影响而陷入局部最小点。

# Gradient Descent




下面对于目标函数$J(\theta)$求偏导数：

$$\frac{\partial}{\partial\theta_j}J\left(\theta\right)=\frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta\left(x\right)-y)^2$$

$$=2\cdot\frac{1}{2}(h_\theta\left(x\right)-y)\frac{\partial}{\partial\theta_j}(h_\theta\left(x\right)-y)$$

$$=(h_\theta\left(x\right)-y)x_j$$

下面是更新的过程，也就是$\theta_i$会向着梯度最小的方向进行减少。$\theta$表示更新之前的值，$a$表示步长，也就是每次按照梯度减少的方向变化多少，由于求得是极小值，因此梯度方向是偏导数的反方向，结果为

$$\theta:=\theta_j+a\left(h_\theta\left(x\right)-y\right)x_j$$

一个很重要的地方值得注意的是，梯度是有方向的，对于一个向量$\theta$，每一维分量$\theta_i$都可以求出一个梯度的方向，我们就可以找到一个整体的方向，在变化的时候，我们就朝着下降最多的方向进行变化就可以达到一个最小点，不管他是全局的还是局部的。

在对目标函数$J(\theta)$求偏导时，可以用更简单的数学语言（倒三角表示梯度）进行描述：

$$\nabla_\theta J=\begin{bmatrix}\frac{\partial}{\partial\theta_0}J\\\cdots\\\cdots\\\frac{\partial}{\partial\theta_n}J\end{bmatrix}$$

$$\theta := \theta + a\nabla_\theta J$$

将梯度下降法应用到线性回归有三种方式：批处理梯度下降法、随机梯度下降法。

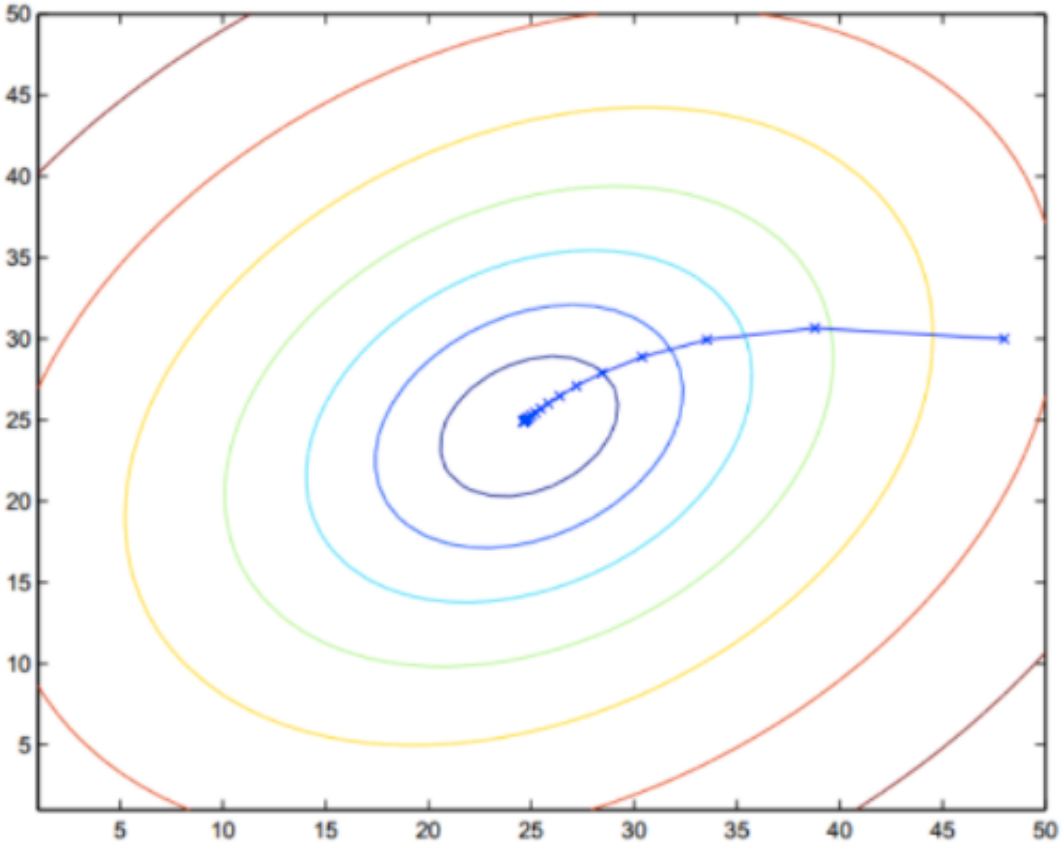
### 3.2.1 批量梯度下降法（BGD）

$$\text{Repeat until convergence } \{$$

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

$$\}$$

可以看出，参数$\theta$的值每更新一次都要遍历样本集中的所有的样本，得到新的$\theta_j$，看是否满足阈值要求，若满足，则迭代结束，根据此值就可以得到；否则继续迭代。注意到，虽然梯度下降法易受到极小值的影响，但是一般的线性规划问题只有一个极小值，所以梯度下降法一般可以收敛到全局的最小值。例如，$J$是二次凸函数，则梯度下降法的示意图为：




图中，一圈上表示目标函数的函数值类似于地理上的等高线，从外圈开始逐渐迭代，最终收敛全局最小值。

### 3.2.2 随机梯度下降算法（SGD）

$$
\begin{aligned}
&\text{Loop } \{ \\
&\quad \text{for i=1 to m, } \{ \\
&\quad\quad \theta_j := \theta_j + \alpha \left( y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)} \\
&\quad \} \\
&\}
\end{aligned}
$$

在这个算法中，我们每次更新只用到一个训练样本，若根据当前严格不能进行迭代得到一个，此时会得到一个，有新样本进来之后，在此基础上继续迭代，又得到一组新的和，以此类推。

> This algorithm is called **stochastic gradient descent** (also **incremental gradient descent**). Whereas batch gradient descent has to scan through the entire training set before taking a single step—a costly operation if $m$ is large—stochastic gradient descent can start making progress right away, and continues to make progress with each example it looks at. Often, stochastic gradient descent gets $\theta$ "close" to the minimum much faster than batch gradient descent. (Note however that it may never "converge" to the minimum, and the parameters $\theta$ will keep oscillating around the minimum of $J(\theta)$; but in practice most of the values near the minimum will be reasonably good approximations to the true minimum.[2]) For these reasons, particularly when the training set is large, stochastic gradient descent is often preferred over batch gradient descent.

批量梯度下降法，每更新一次，需要用到样本集中的所有样本；随机梯度下降法，每更新一次，只用到训练集中的一个训练样本，所以一般来说，随机梯度下降法能更快地使目标函数达到最小值（新样本的加入，随机梯度下降法有可能会使目标函数突然变大，迭代过程中在变小。所以是在全局最小值附近徘徊，但对于实际应用俩说，误差完全能满足要求）。另外，对于批量梯度下降法，如果样本集增加了一些训练样本，就要重新开始迭代。由于以上原因，当训练样本集较大时，一般使用随机梯度下降法。

# 四、参考资料

对线性回归，logistic回归和一般回归的认识