Single Variable Linear Regression

Shen Hengheng

2017

该笔记是来自 Andrew Ng 的 Machine Learning 课程的第二周: **单变量线性回归**的课堂记录，主要讲解了以下几个内容:

- 模型表达
- 代价函数
- 梯度下降算法

0.1 模型表达

还是上一个关于房屋价格预测的例子，给你一组房屋的大小和对应房屋的价格的数据，让你对数据进行建模，使得对于其他的房屋的大小更好的拟合出更准确的价格出来。如图 1 所示。

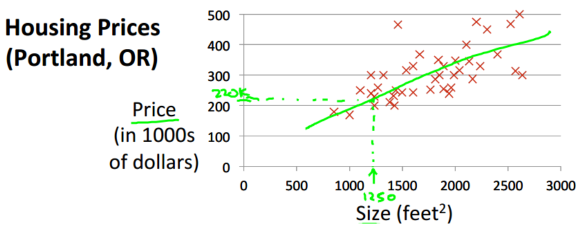


图 1: 房屋价格预测模型

题外话：首先数据是一组关于标准输入与标准答案的数据集，那么对该数据进行建模，是属于监督学习任务。又因为数据只含有一个变量（$size$）即一个特征，且找到一条直线来拟合数据集，所以该问题又被称为单变量线性回归问题。

图 2 是部分的样本数据，其中 X 表示房屋的大小，Y 表示对应房屋的价格。

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

m=47

图 2: 数据

为了后面更好的描述，所以对一些符号进行声明：

- m 训练集中实例的数量
- $x's$ 输入变量或者特征
- $y's$ 输出变量或目标变量
- (x, y) 一个训练样本
- $(x^{(i)}, y^{(i)})$ 第 i 个训练样本

有了数据之后，可以用下图就可以描述房屋价格预测问题建模的过程。

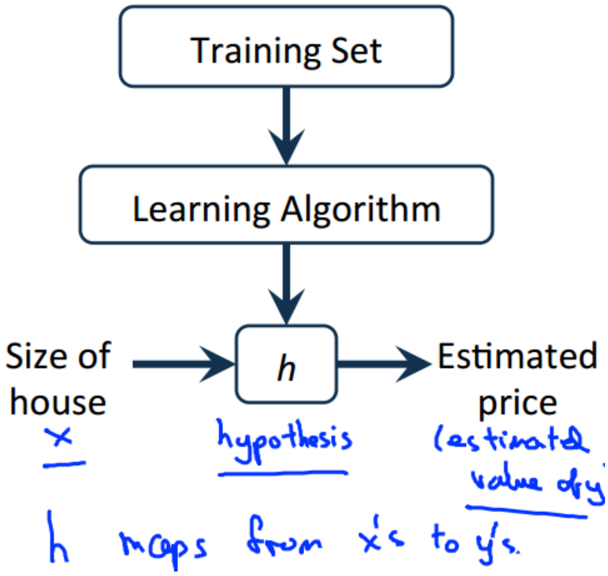


图 3: 建模的过程

图片描述：因而解决房屋价格预测问题，实际上就是要将训练集”喂给“我们的学习算法，进而学习到一个假设 h，然后将要预测的房屋尺寸作为输入变量输入给 h，预测出该房屋的交易价格作为输出结果。**其实 h 就是一个我们要学习的函数**。

我们如何表示函数呢？对于单变量线性回归问题来说，$h_\theta(x) = \theta_0 + \theta_1 x$.

0.2 代价函数

0.2.1 引入代价函数

有了训练数据，也有了模型的表示 h，但是问题是如何选择 θ_0 和 θ_1 ?

基本想法是选择 θ_0，θ_1 以使得在训练集上 $h_\theta(x)$ 接近 y. 用数学表达式表示出来，如下：

$$minimize_{\theta 0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

注意：在学习的时候，有一个区别，$lossfunction$ 描述的是单个样本误差，而 $cost$ $function$ 描述整个训练集的误差。

同样地，为了更好的下文描述，将其简化：

$$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$$

Goals:

$$minimize_{\theta 0,\theta 1}J(\theta_0,\theta_1)$$

其中 $J(\theta_0,\theta_1)$ 是平方误差（代价）函数，它是解决回归问题常用的手段。

0.2.2 代价函数 J 的工作原理

假说函数：$h_\theta(x)=\theta_0+\theta_1 x$
参数：θ_0,θ_1
代价函数：$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$
目标：$minimize_{\theta 0,\theta 1}J(\theta_0,\theta_1)$
简化来讲，就是找到使得误差最小的那一对参数（θ_0,θ_1）.

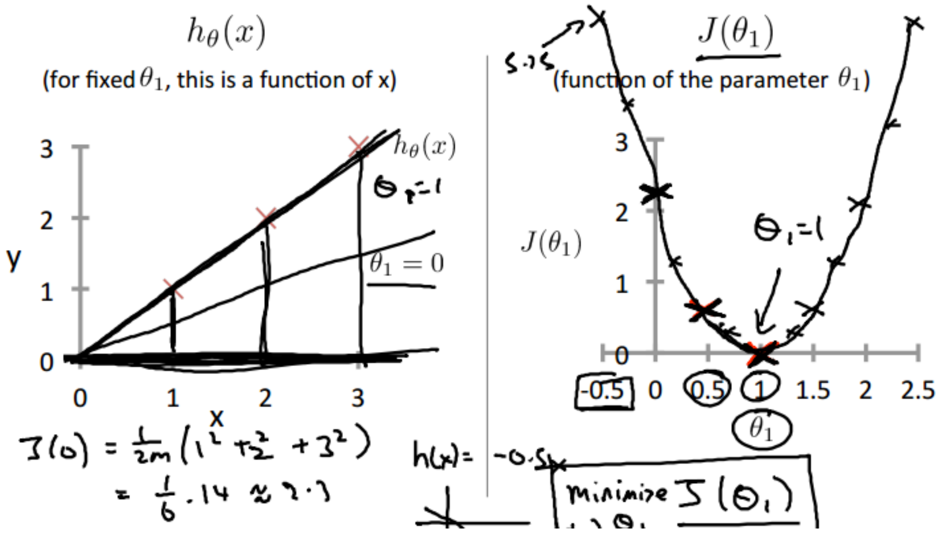


图 4: 代价函数工作原理

0.2.3 代价函数的直观理解

引入一种表示方法：轮廓图。我们在三维空间来表示（$\theta_0,\theta_1,J(\theta_0,\theta_1)$），如图5

可以直观地得到以下结论：在曲面的最低的那个点对应的代价误差最小，对应的 (θ_0,θ_1) 是我们要找的参数组合。

还有一种表示方法：等高线图，他的工作原理是，每一圈上的值都是相等的，即对应的代价误差是相同的。从外到里，代价不断减小，最里面的是最小的代价误差，也是我们算法要最终学习到的点。如图6

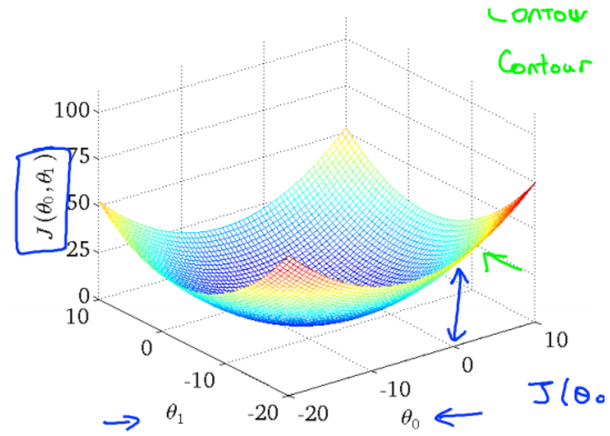


图 5: 轮廓图

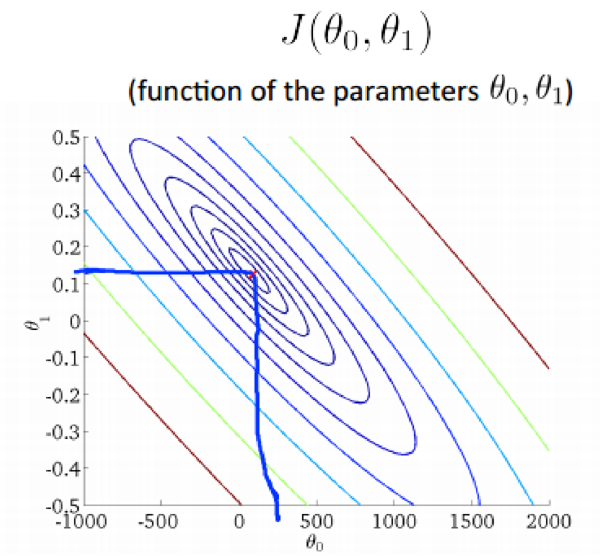


图 6: 等高线图

0.3 梯度下降算法

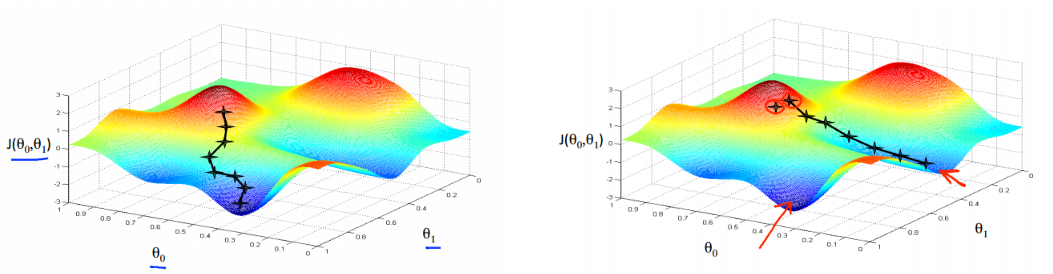
0.3.1 算法描述

我们很迫切想求得最佳拟合参数（θ_0, θ_1），但是有不想通过枚举的方式求得，因此需要一个算法能够自动求出使得代价函数 $J(\theta_0, \theta_1)$ 取得最小值的参数 θ_0, θ_1.

所以在这里引入梯度下降算法。主要思想是想象你在一个山丘上，怎么样以最快的速度从山上到山脚下？在这里就引入了梯度的概念，即下降速度最快的方向，所以人没走一步就检查一下，当前是否为下降最快的方向？若不是则将重新求梯度，按照梯度指示的方向走，则为最快的方式！将这种思想应用到求使得代价误差函数最小，也是这么做的。但是这样往往带来几个问题？

1. 以多大的脚步往下走，因为如果脚步过大，则会造成错过最佳的下山路线。（学习速率选择问题）
2. 由于不一定你的目标函数是凸函数，所以每次走可能走到不同”的山脚“。（局部最小值，凸优化问题）如图7

这些问题将会在后面一一解决！



(a) 梯度下降　　(b) 走到局部最低点

图 7: 梯度下降算法

下面是梯度下降算法的为伪代码。

Algorithm 1 梯度下降算法简化版

Require: α 学习率; $\theta_0 \in R, \theta_1 \in R$ 参数; J 代价误差函数; := 赋值;

repeat

　$\theta_j := \theta_j - \alpha \frac{\partial}{\partial_j} J(\theta_0, \theta_1)$ for $j = 0$ 和 $j = 1$;

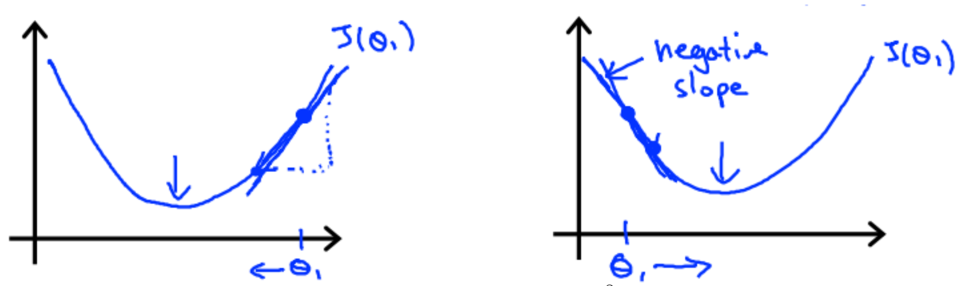
　(向量版本)$\theta := \theta - \alpha \nabla J(\theta_0, \theta_1)$;

until 收敛

下图8 直观的描述 θ 是如何变化的. 这里 $\alpha > 0$.

0.3.2 学习率 α

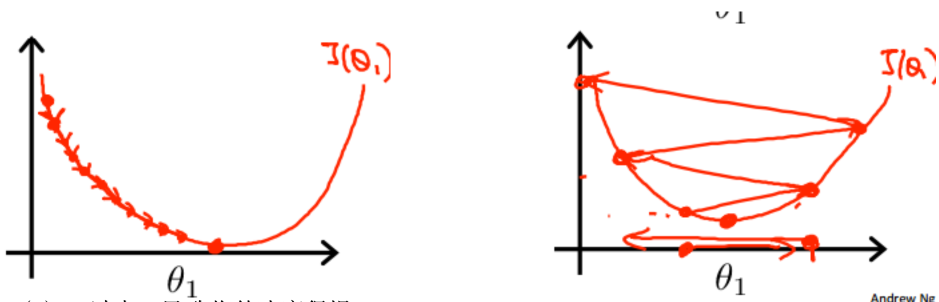
学习率在算法迭代时起着很大的作用。



(a) $\frac{\partial}{\partial_{\theta_1}}J(\theta_0,\theta_1)>0$ 时，θ_1 往右移动　(b) $\frac{\partial}{\partial_{\theta_1}}J(\theta_0,\theta_1)<0$ 时，θ_1 往左移动

图 8: 代价误差不断接近最低处

1. 如果 α 过小，将会导致算法的收敛速度很慢，算法的效率低;
2. 如果 α 过大，在梯度下降过程中，很容易掠过最小值，可能会无法收敛甚至发散。如图9.



(a) α 过小，导致收敛速度很慢　(b) α 过大，导致无法到达最低点

图 9: 不同的 α 对算法收敛的影响

虽然选择的是合适的 $alpha$，但是算法还是会掉到局部最小值，对于局部最小值的描述如图10

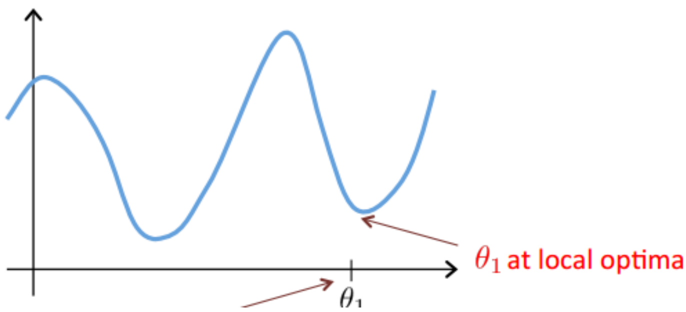


图 10: 局部最小值

在算法迭代时，随着逐渐接近最小值，步子会慢慢地变小，直到接近最小值。所以在算法的迭代过程中不需要对 α 改变。

0.3.3 算法详细版

假说函数: $h_\theta(x) = \theta_0 + \theta_1 x$

参数：θ_0, θ_1

代价函数：$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

目标：$minimize_{\theta 0, \theta 1} J(\theta_0, \theta_1)$

求解：$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1)$ for $j = 0$ 和 $j = 1$

结合上面的公式可以求得

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

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

算法的伪代码如下：

Algorithm 2 梯度下降算法简化版

Require: α 学习率; $\theta_0 \in R, \theta_1 \in R$ 参数; := 赋值;

repeat

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

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

until 收敛

1. 由于代价误差函数是二次函数，二次项为负所以图像为“弓（拱）”形，所以是**凸函数**，所以算法总能收敛到全局最小值.
2. 由于训练的过程中，将全部的训练数据喂给我们的模型，所以该过程又称**“批量训练过程”**.