nolonely

很多时候不是我们做不好，而是没有竭尽全力……

梯度下降法和随机梯度下降法

(1)梯度下降法

在迭代问题中，每一次更新w的值，更新的增量为 η v，其中 η 表示的是步长，v表示的是方向

For $t = 0, 1, \ldots$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \mathbf{v}$$

when stop, return last $\mathbf{w}$ as g

要寻找目标函数曲线的波谷，采用贪心法：想象一个小人站在半山腰，他朝哪个方向跨一步，可以使他距离谷底更近（位置更低），就朝这个方向前进。这个方向可以通过微分得到。选择足够小的一段曲线，可以将这段看做直线段，那么有：

$$E_{\text{in}}(\mathbf{w}_t + \eta \mathbf{v}) \approx E_{\text{in}}(\mathbf{w}_t) + \eta \mathbf{v}^T \nabla E_{\text{in}}(\mathbf{w}_t)$$

an approximate greedy approach for some given small η:

$$\min_{\|\mathbf{v}\|=1} \underbrace{E_{\text{in}}(\mathbf{w}_t)}_{\text{known}} + \underbrace{\eta}_{\text{given positive}} \underbrace{\mathbf{v}^T \nabla E_{\text{in}}(\mathbf{w}_t)}_{\text{known}}$$

其中 η 表示的是步长，v表示的是方向，而梯度下降的精髓是：

an approximate greedy approach for some given small η:

$$\min_{\|\mathbf{v}\|=1} \underbrace{E_{\text{in}}(\mathbf{w}_t)}_{\text{known}} + \underbrace{\eta}_{\text{given positive}} \underbrace{\mathbf{v}^T \nabla E_{\text{in}}(\mathbf{w}_t)}_{\text{known}}$$

通过上边的例子可以看出，只要找到了后边这项的最小值即可，因此当v^T与Δ的方向相反时，该值最小，因此，最优的V的值的求法如下：

- optimal $\mathbf{v}$: opposite direction of $\nabla E_{\text{in}}(\mathbf{w}_t)$

$$\mathbf{v} = -\frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$$

- gradient descent: for small η, $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$

显然上边解决了方向问题，但是还存在步长问题，步子太小的话，速度太慢；过大的话，容易发生抖动，可能到不了谷底。

解决方案：距离谷底较远（位置较高）时，步幅大些比较好；接近谷底时，步幅小些比较好（以免跨过界）。距离谷底的远近可以通过梯度（斜率）的数值大小间接反映，接近谷底时，坡度会减小。设置步幅与梯度数值大小正相关。

< 2018年10

日	一	二	三
30	1	2	3
7	8	9	10
14	15	16	17
21	22	23	24
28	29	30	31
4	5	6	7

随笔分类

arithmetic(10)

deep learning(3)

hadoop(2)

java(9)

linux(8)

Machine Learning(69

php(12)

python(54)

spark(8)

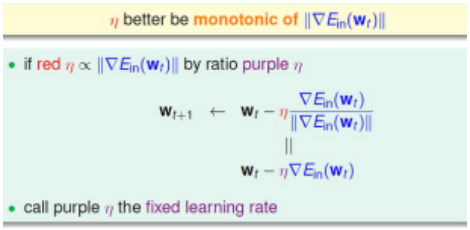
其他(15)

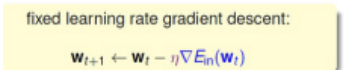
推荐系统(6)

阅读排行榜

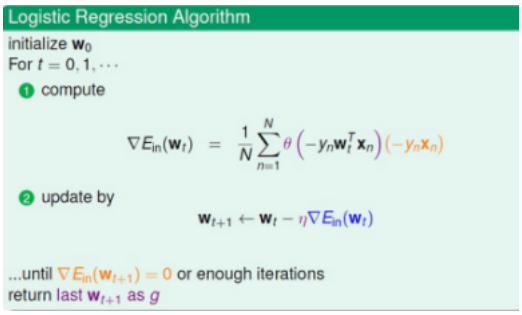
1. Java单例模式------(22178)

2. sklearn提供的自带

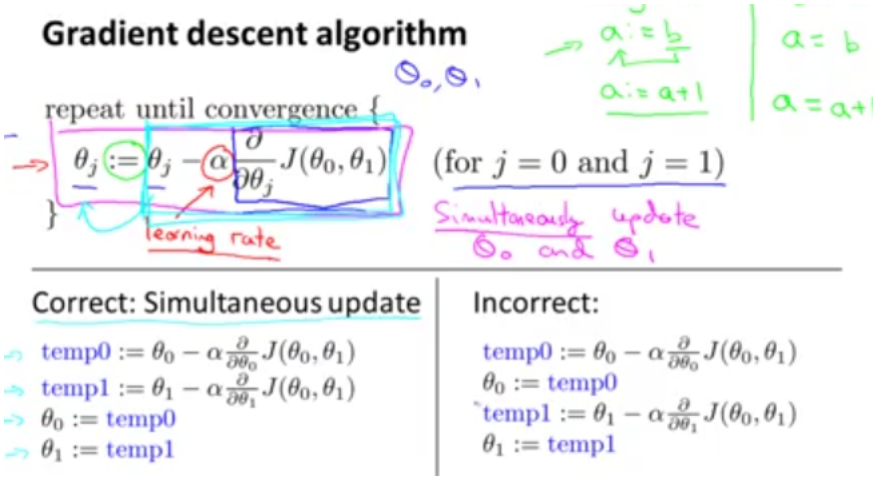




根据上边优化完成的梯度下降算法，可以得到完整的Logistic Regression Algorithm:

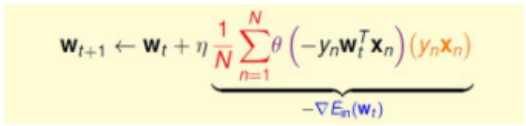


值得注意的是，梯度下降法，需要同时更新θ_1，θ_2，...，θ_n，的值



(2)随机梯度下降法

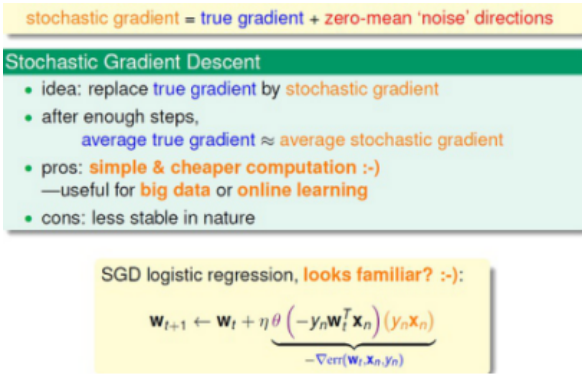
传统的随机梯度下降更新方法：



问题：每次更新都需要遍历所有data，当数据量太大或者一次无法获取全部数据时，这种方法并不可行。

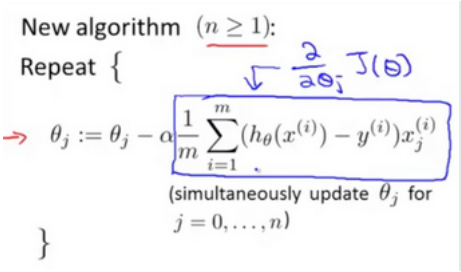
解决这个问题基本思路是：只通过一个随机选取的数据(xn, yn) 来获取“梯度”，以此对w 进行更新。这种优化方法叫做随机梯度下降。

评论排行榜



关于梯度下降法的几点说明：

①对于n》=1的梯度下降规则，n表示的是特征的数目



②

梯度下降实用技巧 1：特征缩放(Feature Scaling)以及均值归一化(mean normalization)

视频地址

在这段视频以及下一段视频中，我想告诉你一些关于梯度下降运算中的实用技巧。

在这段视频中我会告诉你两个称为**特征缩放**(feature scaling)以及**均值归一化**(mean normalization)的方法。

这两个方法都是为了**保证特征的取值在合适的范围内**的。

特征缩放(feature scaling)

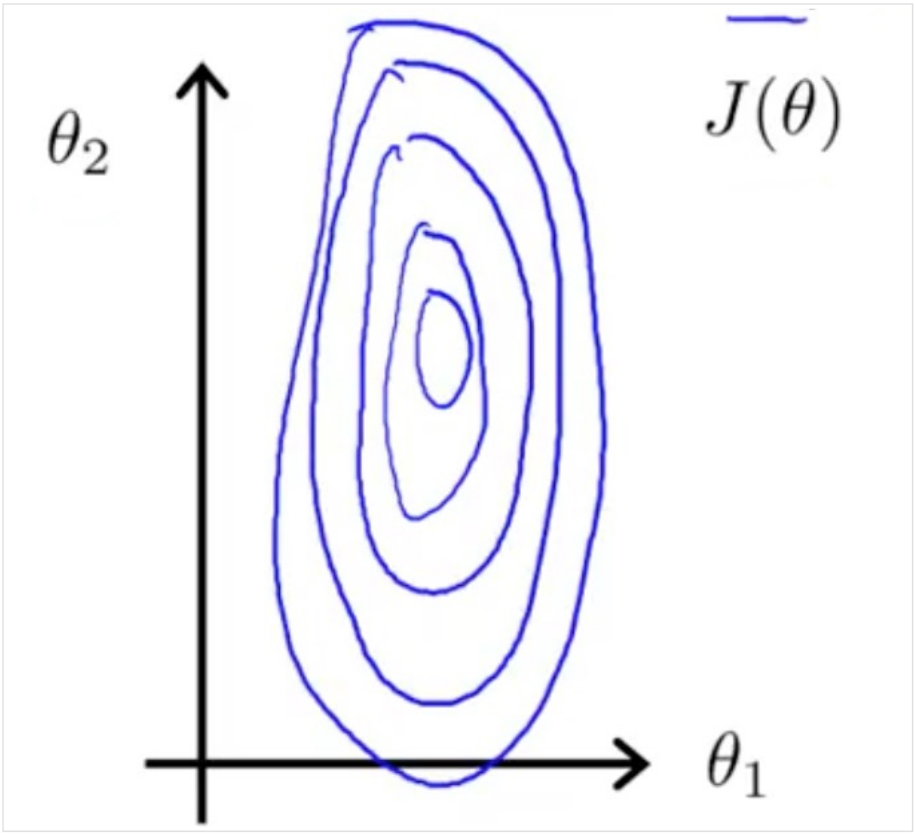
其中，**特征缩放**(feature scaling)大致的思路是这样的：**梯度下降算法中，在有多个特征的情况下，如果你能确保这些不同的特征都处在一个相近的范围，这样梯度下降法就能更快地收敛。**

举个例子来说明：

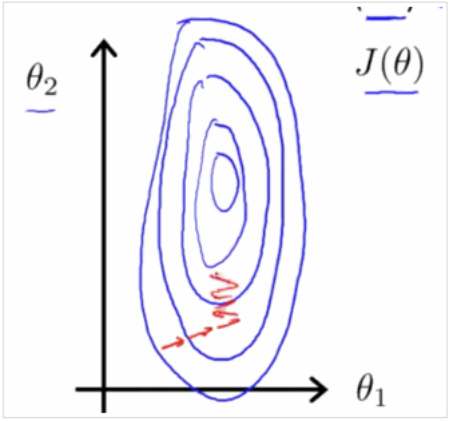
$$x_1 = size(0 - 2000feet^2)$$
$$x_2 = \text{卧室的数量}(1 - 5)$$

假如你有一个具有两个特征的问题，其中x_1是房屋面积大小，它的取值在0到2000之间；x_2是卧室的数量，可能这个值的取值范围在1到5之间。其代价函数$J(\theta)$是一个关于参数θ_0，θ_1和θ_2的函数。但这里我们暂时不考虑θ_0并假想一个函数的变量只有θ_1和θ_2。

如果x_1的取值范围远远大于x_2的取值范围的话，那么最终画出来的代价函数$J(\theta)$的轮廓图就会呈现出这样一种非常偏斜并且椭圆的形状：



如果你用这个代价函数来运行梯度下降的话，你要得到梯度值最终可能需要花很长一段时间，并且可能会来回波动，然后会经过很长时间最终才收敛到全局最小值。



事实上如果这些轮廓再被放大一些的话，如果你画的再夸张一些把它画的更细更长，那么可能情况会更糟糕，梯度下降的过程可能更加缓慢，需要花更长的时间反复来回振荡，最终才找到一条正确通往全局最小值的路。

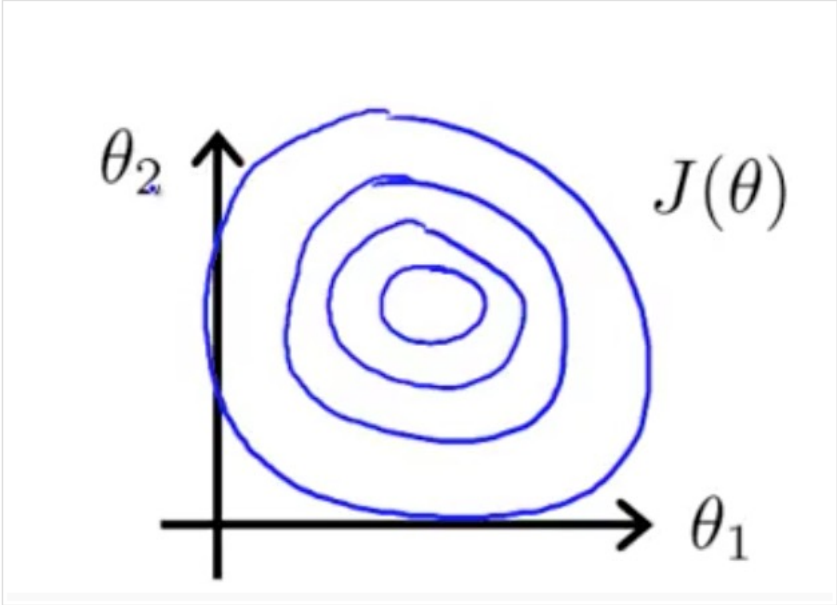
在这样的情况下一种有效的方法是**进行特征缩放(feature scaling)**。

具体来说把特征x定义为：

$$x_1 = \frac{size(feet^2)}{2000}$$

$$x_2 = \frac{\text{卧室的数量}}{5}$$

通过这样的变化，表示代价函数$J(\theta)$的轮廓图的形状就会变得偏移没那么严重，可能看起来更圆一些了。



如果你用这样的代价函数来执行梯度下降的话，那么可以从数学上来证明梯度下降算法将会找到一条更捷径的路径通向全局最小，而不是像刚才那样 沿着一条让人摸不着头脑的路径，来找到全局最小值。

因此在这个例子中，通过特征缩放，我们最终得到的两个特征x_1和 x_2都在0和1之间，这样你得到的梯度下降算法就会更快地收敛。

更一般地，**我们执行特征缩放时，我们通常的目的是将特征的取值约束到-1到$+1$的范围内**。其中，特征x_0总是等于1，因此这已经是在这个范围内了，但对于其他的特征，你可能需要通过除以不同的数来让它们处于同一范围内。

-1和$+1$这两个数字并不是太重要，所以如果你有一个特征x_1它的取值在`0 ~ 3`之间，这没问题如果你有另外一个特征取值在`-2 ~ +0.5`之间，这也没什么关系，因为这也非常接近`-1 ~ +1`的范围。

但如果你有另一个特征x_3，假如它的范围在`-100 ~ +100`之间，那么这个范围跟`-1 ~ +1`就有很大不同了。所以这可能是一个不那么好的特征。类似地，如果你的特征在一个非常非常小的范围内，比如另外一个特征x_4，它的范围在`-0.0001 ~ +0.0001`之间,那么这同样是一个比`-1 ~ +1`小得多的范围，因此我同样会认为这个特征也不太好。所以可能你认可的范围，也许可以大于或者小于`-1 ~ +1`，但是也别太大或太小，只要与`-1 ~ +1`范围偏差不多就可以接受。

因此，总的来说不用过于担心你的特征是否在完全相同的范围或区间内，但是只要它们足够接近的话，梯度下降法就会正常地工作。

均值归一化(mean normalization)

除了在特征缩放中将特征除以最大值以外，有时候我们也会进行一个称为**均值归一化**(mean normalization)的工作。

具体做法就是：如果你有一个特征x_i你就用$x_i - \mu_i$来替换。这样做的目的是为了**让你的特征值具有为0的平均值**。很明显 我们不需要把这一步应用到x_0中，因为x_0总是等于1的，所以它不可能有为0的的平均值。

但是对其他的特征来说，比如房子的大小取值介于`0 ~ 2000`，并且假如房子面积的平均值是等于1000的，那么你可以用这个公式

$$x_1 = \frac{size - 1000}{2000}$$

类似地，如果你的房子有五间卧室，并且平均一套房子有两间卧室，那么你可以使用这个公式来归一化你的第二个特征x_2：

$$x_2 = \frac{\text{卧室数} - 2}{5}$$

在这两种情况下你可以算出新的特征x_1和x_2，它们的范围可以在`-0.5 ~ +0.5`之间，当然这肯定不对，x_2的值实际上肯定会大于0.5。更一般的规律是用:

$$\frac{x_n - \mu_n}{S_n}$$

来替换原来的特征x_n。其中定义μ_n的意思是在训练集中特征x_n的平均值。而S_n是该特征值的范围（最大值减去最小值）。

> 最后直的一提的是：特征缩放其实并不需要太精确，其目的只是为了让梯度下降能够运行得更快一点，让梯度下降收敛所需的循环次数更少一些而已。

梯度下降实用技巧 2：学习速率(Learning Rate)

视频地址

在本段视频中，我将集中讨论**学习率**α。具体来说这是梯度下降算法的更新规则。

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_0} J(\theta)$$

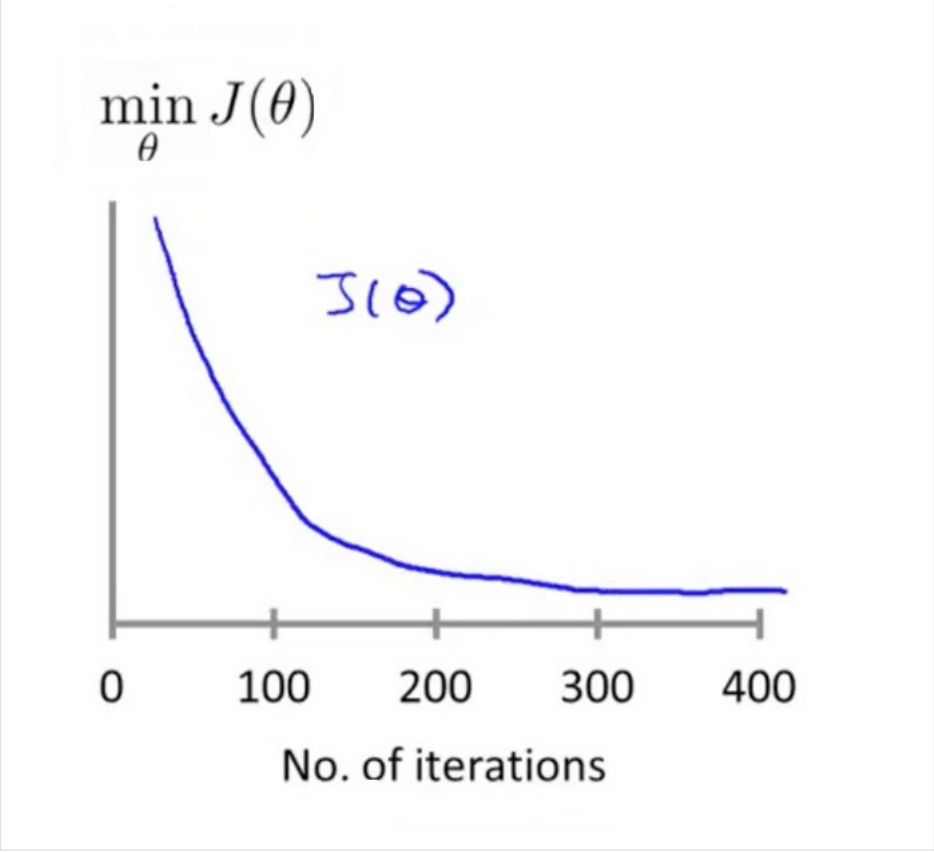
这里我想要告诉大家两点：

- 如何调试(Debugging)：也就是我认为应该如何确定梯度下降是正常工作的。
- 如何选择学习率α：如何选择这个参数才能保证梯度下降正常工作。

收敛的判断

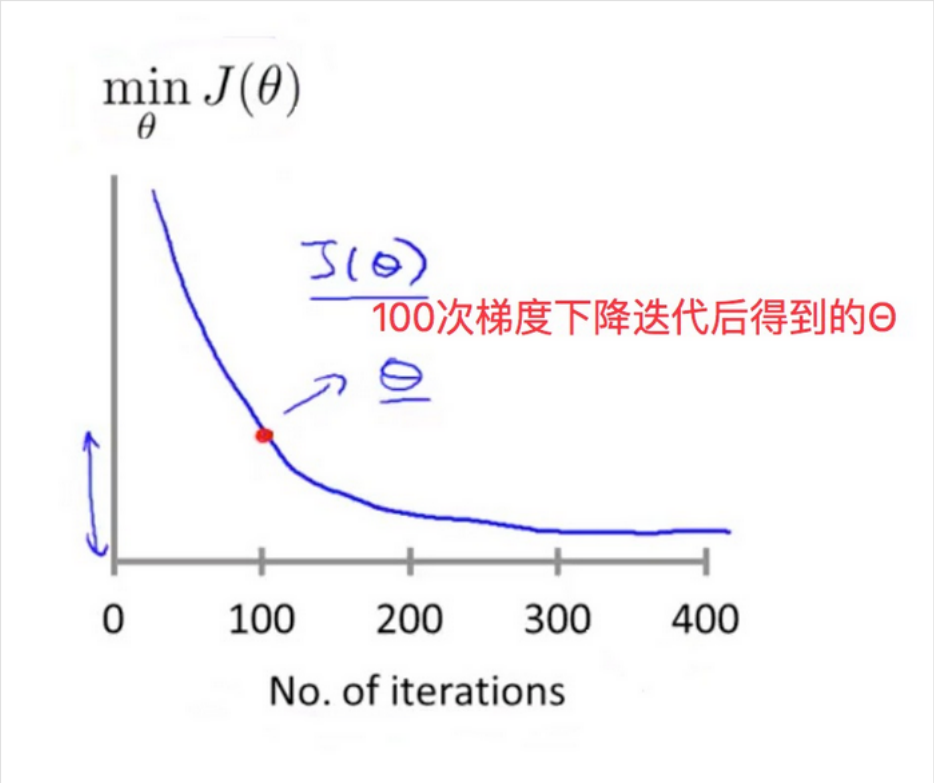
梯度下降算法所做的事情就是为你找到一个θ值，并希望它能够最小化代价函数$J(\theta)$。

我通常会在梯度下降算法运行时，绘出代价函数$J(\theta)$的值。这里的x轴是表示梯度下降算法的迭代步数：

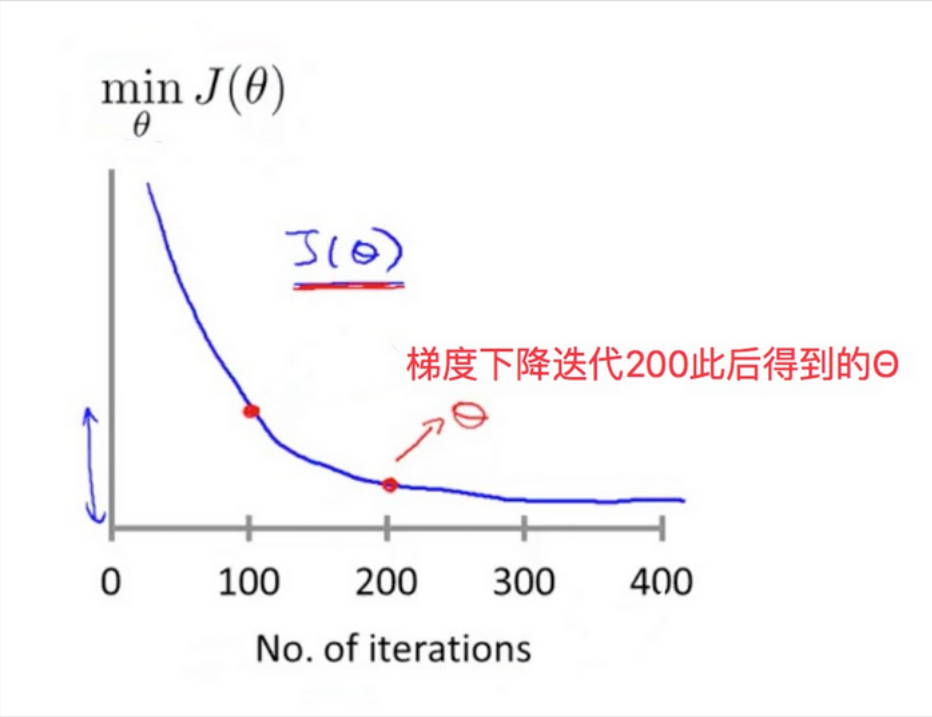


> 注意:这里的x轴是迭代步数，在我们以前看到的$J(\theta)$曲线中x轴，曾经用来表示参数θ但这里不是。

具体来说，这一点的含义是这样的：比如，当我运行完100步的梯度下降迭代之后，我将得到一个θ值，根据这个θ值，我将算出代价函数$J(\theta)$的值。而这个点的垂直高度就代表梯度下降算法100步迭代之后得到的θ算出的$J(\theta)$值：



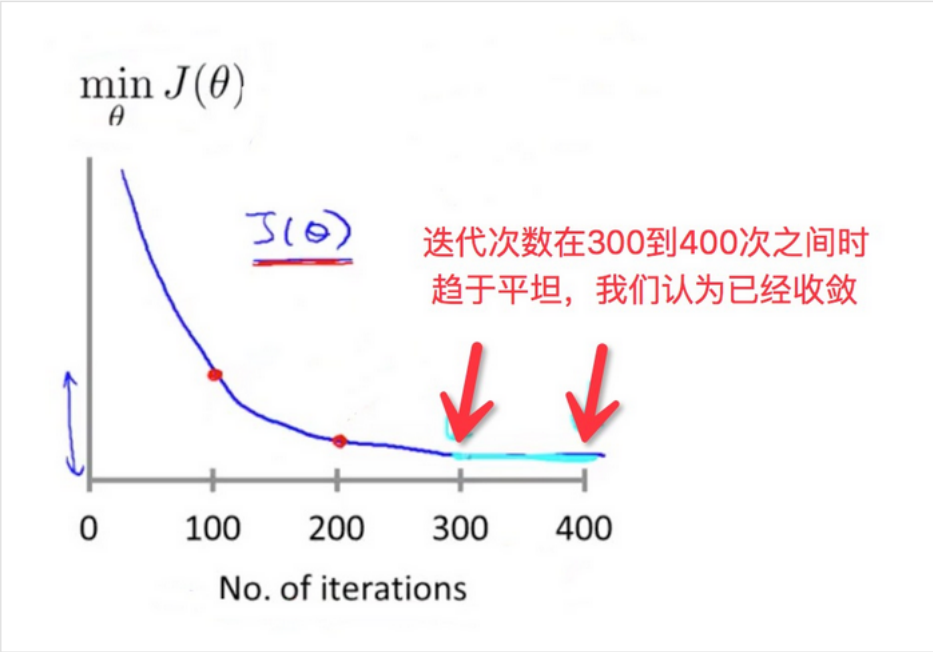
而这个点则是梯度下降算法迭代200次之后得到的θ算出的$J(\theta)$值：



所以这条曲线显示的是梯度下降算法迭代过程中代价函数$J(\theta)$的值。

如果梯度下降算法正常工作，那么每一步迭代之后$J(\theta)$都应该下降。

这条曲线中，当迭代达到300步到400步之间时（如下图所示），看起来$J(\theta)$并没有下降多少。也就是说在400步迭代的时候，梯度下降算法基本上已经收敛了，因为代价函数并没有继续下降：



所以说，看这条曲线可以帮助你判断梯度下降算法是否已经收敛。

> 对于每一个特定的问题，梯度下降算法所需的迭代次数可以相差很大。也许对于某一个问题，梯度下降算法只需要30步迭代就可以收敛，然而换一个问题，也许梯度下降算法就需要3000步迭代，对于另一个机器学习问题则可能需要三百万步迭代。

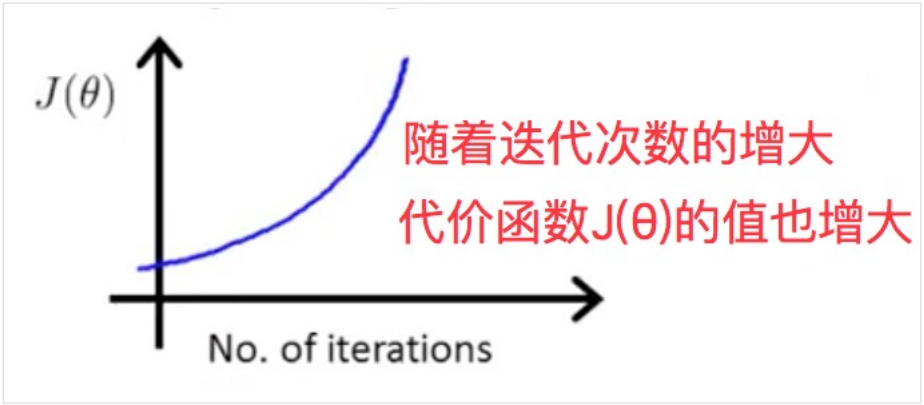
实际上，我们很难提前判断梯度下降算法需要多少步迭代才能收敛。通常我们需要画出这类**代价函数随迭代步数数增加的变化曲线**，通常我会通过看这种曲线来试着判断梯度下降算法是否已经收敛。另外也可以进行一些自动的**收敛测试**（也就是说用一种算法来告诉你梯度下降算法是否已经收敛）。

自动收敛测试一个非常典型的例子是：如果代价函数$J(\theta)$的下降小于一个很小的值ε那么就认为已经收敛。比如可以选择10^{-3}作为阈值ε。但通常要选择一个合适的阈值ε是相当困难的。

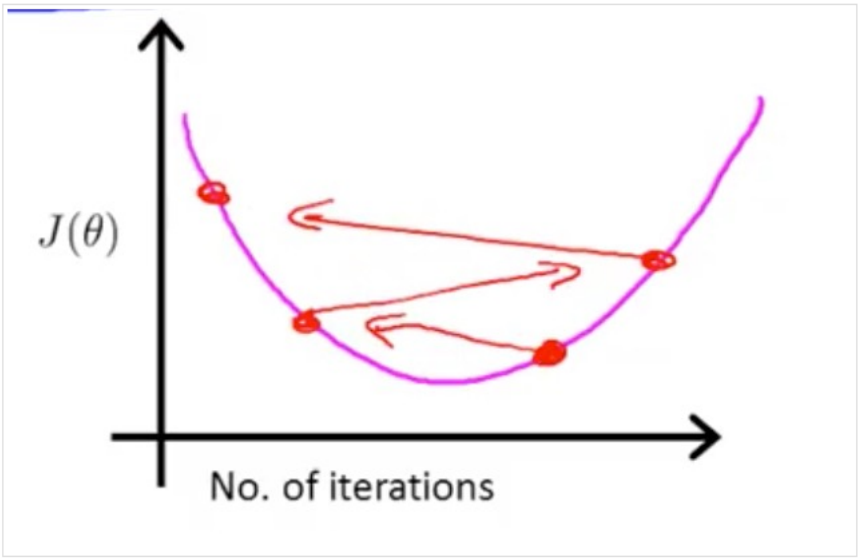
因此，为了检查梯度下降算法是否收敛,我实际上还是通过看上边的这条曲线图，而不是依靠自动收敛测试。

判断梯度下降算法是否正常工作

此外这种曲线图也可以在算法没有正常工作时提前警告你。具体地说如果代价函数$J(\theta)$随迭代步数的变化曲线是这个样子:

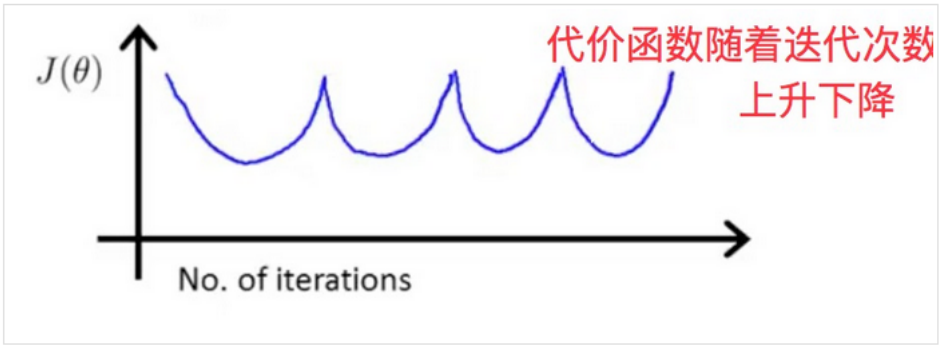


很明确的表示梯度下降算法没有正常工作，而这样的曲线图通常意味着你应该使用较小的学习率α。如果$J(\theta)$在上升，那么最常见的原因是你在最小化一个函数时，如果你的学习率太大，梯度下降算法可能将冲过最小值达到最小值另一侧的更大的一个点。下次迭代时，也可能再次冲过最小值，达到更大的一个点，然后一直这样下去。



所以，如果你看到这样一个曲线图，通常的解决方法是使用较小的α值，当然也要确保，你的代码中没有错误。

同样的，有时你可能看到这种形状的$J(\theta)$曲线:



它先下降，然后上升，接着又下降，然后又上升，然后再次下降，再次上升，如此往复。而解决这种情况的方法通常同样是选择较小α值。

我不打算证明这一点，但对于我们讨论的线性回归可以很容易从数学上证明：只要学习率足够小，那么每次迭代之后，代价函数$J(\theta)$都会下降。因此如果代价函数没有下降，那可能意味着学习率过大，这时你就应该尝试一个较小的学习率。当然，你也不希望学习度太小，因为如果这样，那么梯度下降算法可能收敛得很慢，你需要迭代很多次才能到达最低点。因此如果学习率α太小，梯度下降算法的收敛将会很缓慢。

总结：

- **如果学习率α太小，你会遇到收敛速度慢的问题。**
- **如果学习率α太大，代价函数$J(\theta)$可能不会在每次迭代都下降，甚至可能不收敛。**

> 在某些情况下，如果学习率α过大，也可能出现收敛缓慢的问题。

如何选择α

为了调试，当我运行梯度下降算法时，我通常会尝试一系列α值，比如0.001， 0.01， 0.1， 1 ... 这里每隔10倍取一个值，然后对于这些不同的α值绘制$J(\theta)$随迭代步数变化的曲线，然后选择看上去使得$J(\theta)$快速下降的一个α值。

事实上在为梯度下降算法选择合适的学习率时，我大致是按3的倍数来取值的。例如：0.001，0.003， 0、01，0.03，0.1，0.3，1...

所以我会尝试一系列α值，直到我找到不能再小了的值，同时找到另一个不能再大的值，然后我尽量挑选其中最大的那个α值，或者一个比最大值略小一些的合理的值。当我做了以上工作时，我通常就可以得到一个不错的学习率。

特征的选择和多项式回归

视频地址

你现在了解了多变量的线性回归，在本节中我想告诉你一些用来选择特征的方法，以及如何得到不同的学习算法。当选择了合适的特征后，这些算法往往是非常有效的。另外，我也想给你们讲一讲多项式回归，它使得你们能够使用，线性回归的方法来拟合非常复杂的函数，甚至是非线性函数。

特征的选择

以预测房价为例，假设你有两个特征，分别是房子临街的宽度和垂直宽度。这就是我们想要卖出的房子的图片：

临街宽度其实就是拥有的土地的宽度，而纵向深度就是你的房子的深度。你可能会建立一个像这样的线性回归模型：

Housing prices prediction

$$h_\theta(x) = \Theta_0 + \Theta_1 \times frontage + \Theta_2 \times depth$$

其中临街宽度是你的第一个特征$frontage$，纵深是你的第二个特征$depth$。但当我们在运用线性回归时，你不一定非要直接用给出的$frontage$和$depth$作为特征，其实你可以自己创造新的特征。因此，如果我要预测房子的价格，我真正要需做的也许是确定真正能够决定我房子大小，或者说我土地大小的因素是什么。因此，我可能会创造一个新的特征，我称之为x。它是临街宽度与纵深的乘积：

$$x = frontage \times depth$$

$$h_\theta(x) = \Theta_0 + \Theta_1 x$$

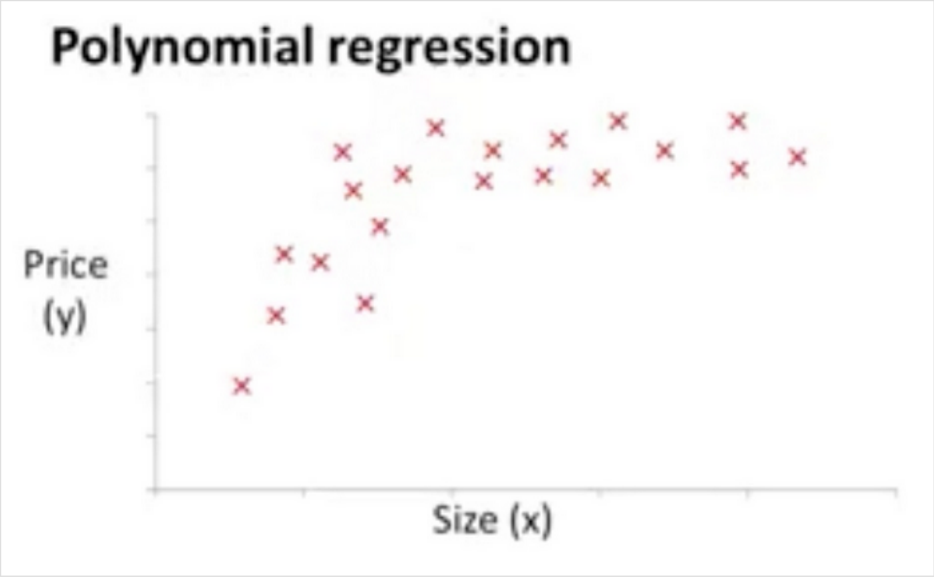
这得到的就是我拥有的土地的面积。然后我可以把假设选择为使其只使用一个特征，也就是我的土地的面积。

由于矩形面积的计算方法是矩形长和宽相乘，因此这取决于你从什么样的角度去审视一个特定的问题，而不是直接去使用临街宽度和纵深，这两个我们只是碰巧在开始时使用的特征。有时通过定义新的特征，你确实会得到一个更好的模型。

多项式回归

与选择特征的想法密切相关的一个概念，被称为多项式回归(polynomial regression)。

比方说你有这样一个住房价格的数据集：

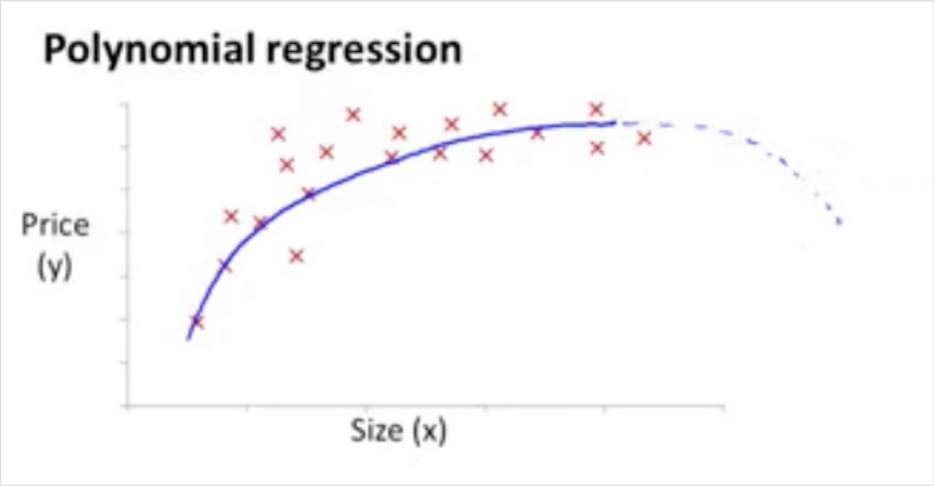


为了拟合它，可能会有多个不同的模型供选择。其中一个你可以选择的是像这样的二次模型：

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

因为直线似乎并不能很好地拟合这些数据，因此也许你会想到用这样的二次模型去拟合数据。

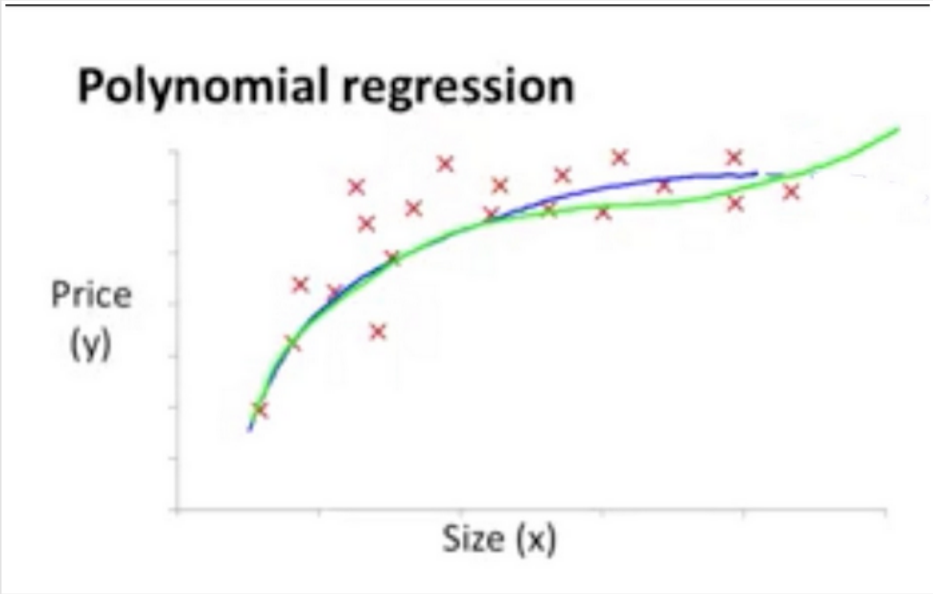
但是你可能会觉得二次函数的模型并不好用，因为一个二次函数最终会降回来。而我们并不认为房子的价格在高到一定程度后会下降回来：



因此也许我们会选择一个不同的多项式模型，并转而选择使用一个三次函数：

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

我们用这个三次函数进行拟合，我们可能得到这样的模型（绿线部分）：



也许这条绿色的线对这个数据集拟合得更好，因为它不会在最后下降回来。

那么，我们到底应该如何将模型与我们的数据进行拟合呢？使用多元线性回归的方法，我们可以通过将我们的算法做一个非常简单的修改来实现它：

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\ &= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3 \end{aligned}$$

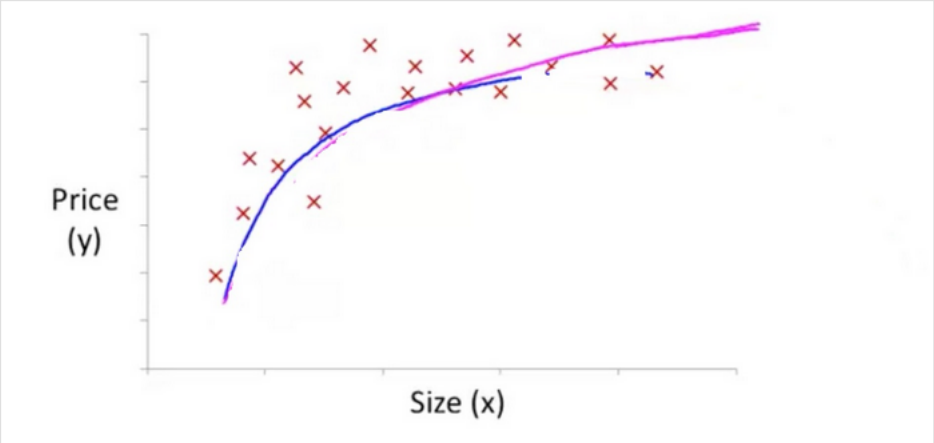
$$\begin{aligned} x_1 &= (size) \\ x_2 &= (size)^2 \\ x_3 &= (size)^3 \end{aligned}$$

我还想再说一件事，如果你像这样选择特征，那么特征的归一化就变得更重要了。因此，如果房子的大小范围在 1 到 1000 之间，那么房子面积的平方的范围就是 1 到 1000000（也就是1000^2），而你的第三个特征x_3它是房子面积的立方,范围会扩大到1到10^9。这三个特征的范围有很大的不同，因此，如果你使用梯度下降法，应用特征值的归一化是非常重要的，这样才能将他们的值的范围变得具有可比性。

最后一个例子，除了建立一个三次模型以外，你也许有其他的选择特征的方法。这里有很多可能的选项，但是给你另外一个合理的选择的例子：

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

这样的一种函数曲线看起来趋势是上升的，但慢慢变得平缓一些，而且永远不会下降回来：



参考自：
http://studyai.site/2016/07/29/%E6%96%AF%E5%9D%A6%E7%A6%8F%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0%E8%AF%BE%E7%A8%8B%20%E7%AC%AC%E4%BA%8C%E5%91%A8%20(2)%E5%A4%9A%E5%85%83%E7%BA%BF%E6%80%A7%E5%9B%9E%E5%BD%92%E5%88%86%E6%9E%90/

分类： Machine Learning

« 上一篇：多分类问题multicalss classification
» 下一篇：支持向量机SVM

posted @ 2016-12-15 17:50 nolonely 阅读(6561) 评论(0) 编辑 收藏

最新IT新闻:
· Netflix第三季度净利润4.03亿美元 同比增长210%
· 凌动智行高管欲发起股东大会 重组董事会踢走史文勇
· 石墨烯被启用 华为手机能否超越苹果
· 脸书数据失窃影响300万欧洲用户 最高可被罚16.3亿美元
· 贾跃亭近九成乐视网股份待解押 贾氏兄弟被动减持约4033万股
» 更多新闻...

最新知识库文章:
· 为什么说 Java 程序员必须掌握 Spring Boot ？
· 在学习中，有一个比掌握知识更重要的能力
· 如何招到一个靠谱的程序员
· 一个故事看懂“区块链”
· 被踢出去的用户
» 更多知识库文章...