Machine Learning

卢婧宇

2017 年 9 月 25 日

目录

1 Linear Regression with Multiple Varibles 多变量线性回归

1.1 Multiple Features 多维特征

之前学习的中，关于线性回过问题，只有一个变量，是单变量。现在我们对房价模型增加更多的特征，例如房间数目、楼层数等等。构成一个含有多个变量的模型，模型中的特征为（$x_1, x_2, ..., x_n$）。如下图 1 所示。

Size (feet2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

图 1: 多特征房屋交易数据

Notation:

- n = number of features
- $x^{(i)}$ = input (features) of $i^{(th)}$ training example. 表示第 i 个样本
- $x_j^{(i)}$ = value of feature j in i^{th} training example. 表示第 i 个样本中第 j 个特整。

有多个变量的 Hypotheis:
Previously:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

For convenience of notation,define x_0=1.
用向量表示：

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix} \qquad \theta^T = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & ... & \theta_n \end{bmatrix}$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = \theta^T x$$

1.2 Graadient Descent for Multiple Varibles 多变量梯度下降

之前提到的一种线性回归的假设形式，这是一种有多特征或者多变量的形式。在本节中将找到满足这一假设的参数，用梯度下降法解决多特征的线性回归问题。

Hypothesis:

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

Parameters:

$$\theta_0, \theta_1, ..., \theta_n$$

不要把 θ_n 想成是 n+1 个单独的参数, 把这 n+1 个 θ 参数想象成一个 n+1 维的向量 θ。

Cost function:

$$J(\theta_0, \theta_1, ..., \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}\Big(h_\theta(x^{(i)}) - y^{(i)}\Big)^2 = \frac{1}{2m}\sum_{i=1}^{m}\Big((\sum_{j=0}^{n}\theta_j x_j^{(i)}) - y^{(i)}\Big)^2$$

同样不要把函数 J 想成是一个关于 n+1 个自变量的函数，而是看成带有一个 n+1 维向量的函数。

Gradient descent:

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

simultaneously update for every j=0,...,n

Gradient Descent:

Previously(n=1),Repeat:

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

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

simultaneously update θ_0, θ_1

New algorithm(n>1),Repeat:

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

simultaneously update θ_j for j=0,...,n

其实多参数和两个参数的形式一样只不过 x 的下标有变化，对于 θ_0，$x_0^{(i)} = 1$。

1.3 Gradient descent in pratice 1:Feature Scaling 梯度下降法实践 1-特征缩放

本节课是关于梯度下降运算中的实用技巧，以及一种方法叫特征缩放 (feature scaling)。在面对多维特征问题时，我们要保证这些特征都具有相同的尺度，这将帮助下降算法更快的收敛。

以房价问题为例, 有两个特征, 房屋的尺寸和房间的数量, 尺寸的值为 0-2000 平方英尺, 而房间数量的值是 0-5, 以两个参数分别为横纵坐标, 绘制代价函数的等高线图, 可以看出图像很扁, 梯度下降算法需要非常多次的迭代才能收敛。解决方法是尝试将所有特征的尺度都尽量缩放到-1 到 1 之间。如图 2 所示。

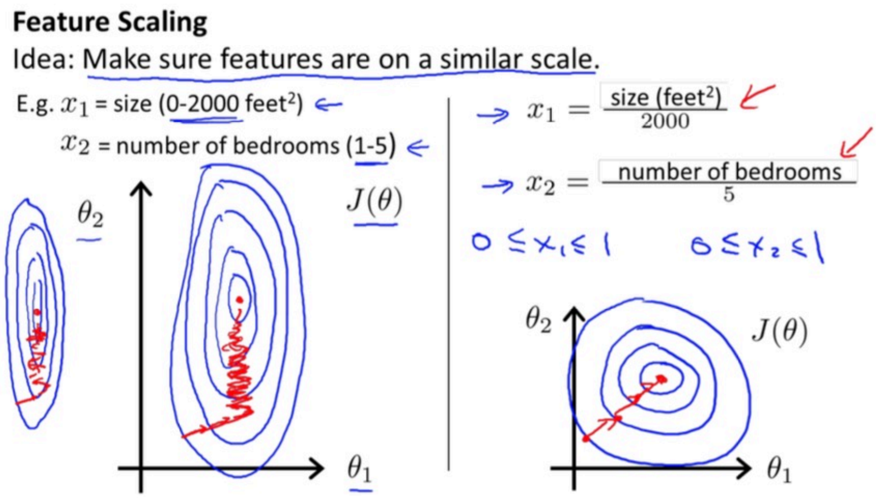


图 2: 等高线图

从上图可以看出有效的方法就是进行特征缩放，这使得等高线图变得圆一些，得到的梯度下降算法可以更快的收敛。更一般的，我们执行特征缩放时，将特征的取值约束到-1 到 +1 的范围内。这个范围在这附近都可以只要不差太大（大太多或者小太多）。

Mean normalization:

Replace x_i with $x_i - \mu_i$ to maje features have approximately zero mean(Do not apply to $x_0 = 1$).

就是用特征减去平均值，再除以范围，来替换原来的特征，范围的意思是最大值减最小值。不管怎样这些取值都是非常近似的，只要将特征转换为相近的范围就都是可以的。特征缩放其实并不需要太精确，只是为了让梯度下降能够更快一点，循环次数少。

1.4 Gradient descent in pratice 2:Lreaning Rate 梯度下降法实践 2-学习率

本小结将围绕学习率 α 展开，这是梯度下降算法的更新规则（Debugging），也就是如何确定梯度下降是正常工作的，并且如何选择 α。如图 3 所示。

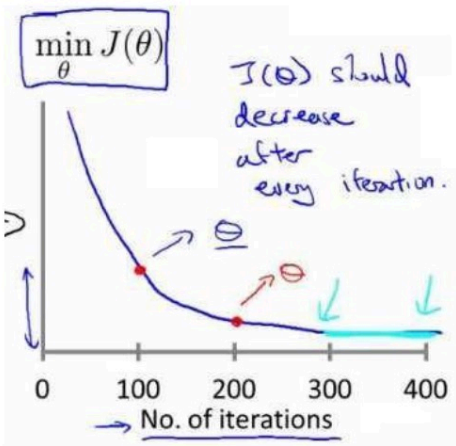


图 3: 迭代结果

从图中可以看出横轴代表迭代次数（No.of iterations），纵轴代表每次迭代后得到 θ 得到 $J(\theta)$ 的值。

另外，也可以进行一些自动收敛测试，也就是用一种算法测试梯度下降算法是否已经收敛。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}, 但通常要选择一个合适的阈值 ε 是相当困难的。所以通常是观察曲线图。

梯度下降算法的每次迭代受到学习率的影响，如果学习率 α 过小，则达到收敛所需要的迭代次数会非常高；如果学习率 α 过大，每次迭代可能不会减小代价函数，可能会越过局部最小值导致无法收敛。

通常可以考虑尝试的一些学习率：α=0.01，0.03，0.1，0.3，1，3，10

1.5 Features and polynomial regrssion 特征和多项式回归

多项式回归可以使得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。以预测房价为例如图 4 所示。

图 4: 房屋图形

假设有两个特征，临街宽度和房屋深度：

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

x_1=frontage(临街宽度)，x_2=depth(纵向深度)，x=frontage∗depth=area(面积)，得到：

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

通过定义新的特征，确实会得到一个更好的模型，与选择的想法密切相关的一个概念被称为多项式回归（polynaoial regression）。线性回归并不适用于所有数据，有时候我们需要用二次模型去拟合数据：

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$$

或者三次方模型：

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$$

如图 5 所示，为房价的数据，我们需要选择合适的模型拟合数据。

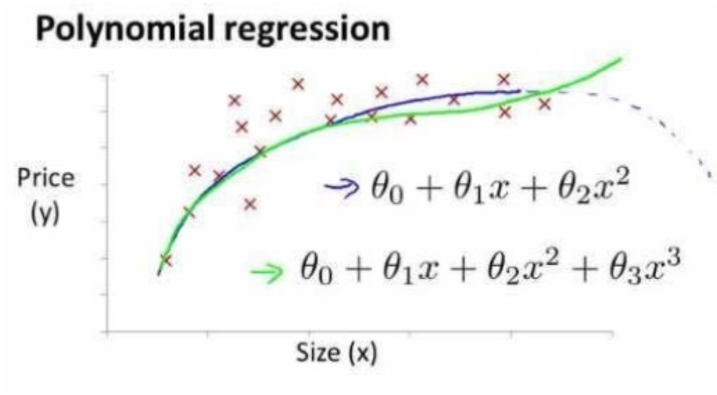


图 5: 房屋数据

通常需要先观察数据再决定准备尝试怎样的模型：

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

其中:$x_1 = (size), x_2 = (size)^2, x_3 = (size)^3$。

除了三次方模型还有其他模型：

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

注意一点非常重要！如果我们采用多项式回归模型，在运行梯度下降算法前，特征缩放非常有必要，将缩放后的值带入再计算。

1.6 Normal Equation 正规方程

对于某些线性回归问题，用正规方程法求解参数 θ 的最优值更好，与其使用迭代算法，可以一次性求解 θ 的最优值。如：

Intuition: If 1D ($\theta \in \mathbb{R}$)

$$J(\theta) = a\theta^2 + b\theta + c$$

正规方程是通过求解下面的方程来找出使得代价函数最小的参数的：

$$\frac{\partial}{\partial \theta_j} J(\theta_j) = 0$$

假设训练集特征矩阵为 X（包含了 X_0）并且训练集结果为向量 y，则利用正规方程解出向量：

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

上标 T 代表矩阵转置，上标-1 代表矩阵的逆。设矩阵 A=$X^T X$, 则：$(X^T X)^{-1} = A^{-1}$

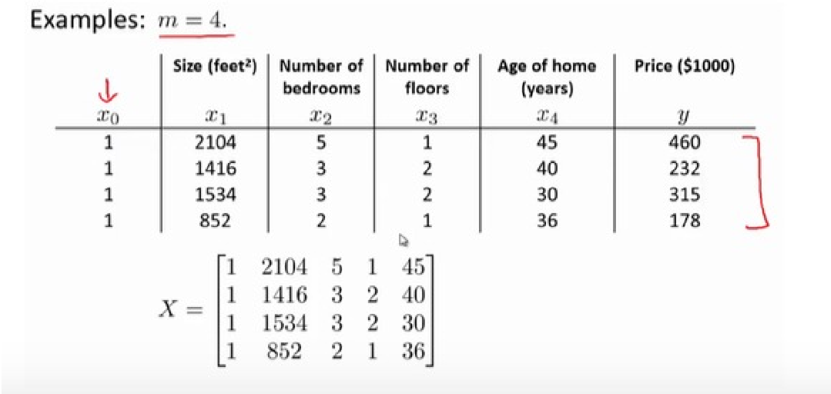
以下表示数据为例，如图 6所示：

在 Octave 中，正规方程写作：

$$pinv(X^{'} * X) * X^{'} * y$$

不可逆矩阵正规方程方法不可用。

梯度下降法与正规方程方法的比较：



Examples: $m = 4.$

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

图 6: 数据列表

- 梯度下降需要选择学习率 α，正规方程不需要。
- 梯度下降需要多次迭代，正规方程一次运算得出。
- 梯度下降当特征数量 n 大时也能较好适用，正规方程需要计算 $(X^TX)^{-1}$，如果特征数量 n 较大则运算代价大，因为矩阵逆的计算时间复杂度为 O(n^3)，通常来说当 n 小于 10000 时还是可以接受的。
- 梯度下降适用于各种类型的模型，正规方程只适用于线性模型，不适合逻辑回归模型等其他模型。

总结一下，只要特征变量的数目并不大，标准方程是一个很好的计算参数 θ 的替代方法。具体地说，只要特征变量数量小于一万，我通常使用标准方程法，而不使用梯度下降法。

对于那些更复杂的学习算法，我们将不得不仍然使用梯度下降法。因此，梯度下降法是一个非常有用的算法，可以用在有大量特征变量的线性回归问题。

1.7 Normal Equation Noninvertibility(Optional) 正规方程及不可逆性

当计算 $\theta = (X^TX)^{-1}X^Ty{=}inv(X'X)X'y$, 那些对于矩阵 $X^{-1}X$ 的结果是不可逆的情况该怎么办?

在数学中我们称不可逆的矩阵为奇异矩阵或退化矩阵。问题的重点在于 $X'X$ 不可逆的问题很少发生。在 Octave 里，如果你用它来实现 θ 的计算，你将会得到一个正常的解。在 Octave 里，有两个函数可以求解矩阵的逆，一个被称为 pinv()，另一个是 inv()，这两者之间的差异是些许计算过程上的，一个是所谓的伪逆，但算法执行的流程是正确的，另一个被称为逆。使用 pinv() 函数可以展现数学上的过程，即便矩阵 XX 是不可逆的。在 pinv() 和 inv() 之间，又有哪些具体区别?

What if X^TX is non-invertible?

- Redundant features (linearly dependent).

  E.g. x_1 = size in $feet^2$

  x_2 = size in m^2

- Too many features(e.g. m $\leq$ n).

  Delete some features,or use regularization.

Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

inv() 引入了先进的数值计算的概念。例如，在预测住房价格时，如果 x_1 是以英尺为尺寸计算的房子，x_2 是以平方米为尺寸规格计算的。1 米等于 3.28 英尺（四舍五入到两位小数），这样两个特征值满足约束：$x_1 = x_2 * (3.28)^2$。

实际上，你可以用这样的一个线性方程，来展示那两个相关联的特征值，矩阵 $X'X$ 将是不可逆的。

第二个原因是有大量的特征值时间学习算法，可能会导致矩阵 $X'X$ 的结果是不可逆的。例如，当 m≤n 时，m 等于 10 个训练样本 n 等于 100 个特征数量。要找到合适的（n+1）维参数矢量 θ 这是第 n+1 维。尝试从 10 个训练样本中找到满足 101 个参数的值，工作量巨大。

如何使用小数据样本以得到 100 或 101 个参数，通常使用一种叫做正则化的线性代数方法，通过删除某些特征或是使用某些技术，来解决当 m 比 n 小的时候的问题。即使你有一个相对较小的训练集，也可以使用很多的特征来找到很多合适的参数。

总之，当发现矩阵 $X'X$ 的结果是奇异矩阵，或者找到的其它矩阵是不可逆的。

首先，看特征值里是否有一些多余的特征，像 $x_1 x_2$ 是线性相关的，互为线性函数。同时，当有一些多余的特征是，可以删除这两个重复特征里的其中一个，无需两个特征同时保留，将解决不可逆问题。如果特征数量实在太多删除些用较少的特征来反映尽可能多内容，否则我会考虑使用正规化方法。

2 Octave Tutorial Octave 教程

Working on and submitting programming exercises.

2.1 Basic Operation 基本操作

可以做基本的算术运算。逻辑运算： 、 不等于、&& 与、*or* 或、*xor* 异或。在终端执行 Octave 时左边会有其版本的提示，如果不想看到那个提示，隐藏命名为：PS1('» '); 但 MATLAB 中没有。

谈到变量。如果想分配一个 a 赋值为 3，并按下回车键，显示变量 a 等于 3。如果你想分配一个变量但不希望屏幕上显示结果，在命令后加分号。如图 7所示

```
>> a = 3

a =

     3

>> a =3;
```

图 7: 变量赋值

如果你想打印出变量，或显示一个变量输入变量名，或者只要输出值用 disp(变量名)，对于更复杂的屏幕输出也可以用 DISP 命令显示。也可以用该命令来显示字符串输入 disp sprintf 。如图 8所示。

```
>> a=pi;
>> a

a =

    3.1416

>> disp(a);
    3.1416

>> disp(sprintf('2 decimals:%0.2f',a))
2 decimals:3.14
```

图 8: 变量打印

这是一种，就风格的 C 语言语法。也有一些控制输出长短格式的快捷键命令，如图 9所示

```
>> format long
>> a

a =

   3.141592653589793

>> format short
>> a

a =

    3.1416
```

图 9: 变量打印

向量和矩阵。建立一个矩阵 A，对矩阵 A 进行赋值，建立行向量列向量。如图 10所示。

```
>> A = [1 2;3 4; 5 6]

A =

     1     2
     3     4
     5     6

>> A=[1 2;
3 4;
5 6]

A =

     1     2
     3     4
     5     6

>> A = [1 2 3]

A =

     1     2     3
```

图 10: 向量和矩阵

下面是一些更为有用的符号，如 V=1：0.1：2。V 是一组值，从数值 1 开始，增量或说步长为 0.1，直到增加到 2，按照这样的方法对向量 V 操作，可以得到一个行向量，这是一个 1 行 11 列的矩阵的矩阵，其矩阵的元素是 1 1.1 1.2 1.3 以此类推，直到 2。也可以建立一个集合 V 并用命令 “1:6” 进行赋值，这样 V 就被赋值了 1 至 6 的六个整数。如图 11所示。

```
>> V=1:0.1:2

V =

  Columns 1 through 7

    1.0000    1.1000    1.2000    1.3000    1.4000    1.5000    1.6000

  Columns 8 through 11

    1.7000    1.8000    1.9000    2.0000

>> v=1:6

v =

     1     2     3     4     5     6
```

图 11: 向量和矩阵

还有一些生成矩阵的方法。如图 12所示如图 12所示。

```
>> ones(2,3)

ans =

     1     1     1
     1     1     1

>> c= 2*ones(2,3)

c =

     2     2     2
     2     2     2

>> w=zeros(1,3)

w =

     0     0     0
```

图 12: 向量和矩阵

用 rand 命令建立一个一行三列的矩阵，元素均为随机值：rand(1,3)。数值介于 0 和 1 之间，我们可以得到数值均匀介于 0 和 1 之间的元素。

如果知道什么是高斯随机变量什么是正态分布的随机变量。标准正态分布期望值为 0，标准差为 1。randn 是均值为 0 方差为 1 的正态分布。randn(n) 或 rand(n) 生成 n∗n 的随机数矩阵。rand(n,m) 或 randn(m,n) 生成 m*n 的随机数矩阵。

产生一个随机分布的指定均值和方差的矩阵：将 randn 产生的结果乘以标准差，然后加上期望均值即可。并用 hist 命令绘制直方图。如图 13所示

```
>> w=-6+sqrt(10)*(randn(1,10000));
>> hist(w)
>> hist(w,50)
```

图 13: 向量和矩阵

绘制单位矩阵：I=eye(6)

如果对命令不清楚，建议用 help 命令：help eye 、help rand 、help help。

2.2 Moving Data Around 移动数据

size() 命令返回矩阵的尺寸。输出的是一行两列的矩阵，第一个元素是所求矩阵的行数，第二个元素的所求矩阵的列数。可以用 sz 来存放。输入 size(sz) 看看 sz 的尺寸，返回 1 2。

输入 size(A,1)，将返回 3，这个命令会返回 A 矩阵的第一个维度的尺寸，也就是 A 矩阵的行数。同样，命令 size(A,2)，将返回 2，也就是 A 矩阵的列数。

如果你有一个向量 v，假如 v=[1 2 3 4]，然后输入 length(v)，这个命令将返回最大维度的大小，返回 4。通常还对向量使用 length 命令，而不是对矩阵使用 length 命令，比如 length([1;2;3;4;5])，返回 5。

当打开 Octave 时，我们通常已经在一个默认路径中，这个路径是 Octave 的安装位置，pwd 命令可以显示出 Octave 当前所处路径。cd 命令，意思是改变路径: cd ‘~ Desktop’。ls 查看路径下文件。

如何将文件中的数据读入 Octave。只需要键入 featuresX.dat(load features)，这样就加载了 featuresX 文件。同样可以加载 priceY.dat。其实有好多种办法可以完成，如果你把命令写成了一个字符串的形式：load(‘featureX.dat’)，也是可以的，只不过把文件名写成了一个字符串的形式，现在文件名被存在一个字符串中。Octave 中使用引号来表示字符串。

另外 who 命令，能显示出在我的 Octave 工作空间中的所有变量。同样还有一个 whos 命令，能更详细地进行查看。

如果你想删除某个变量，你可以使用 clear 命令，我们键入 clear featuresX，然后再输入 whos 命令，featuresX 消失了。

如何存储数据？v=princeY(1:10) 表示将向量 Y 的前 10 个元素存入 v 中。假如我们想把它存入硬盘，那么用 save hello.mat v 命令，这个命令会将变量 v 存成一个叫 hello.mat 的文件，回车后桌面就出现了一个新文件，名为 hello.mat。

直接键入 clear，这样将删除工作空间中的所有变量，所以现在工作空间中啥都没了。

但如果载入 hello.mat 文件，又重新读取了变量 v，因为我之前把变量 v 存入了 hello.mat 文件中，所以 save 命令就是把数据按照二进制形式存储，或者说更压缩的二进制形式，因此，如果 v 是很大的数据，那么压缩幅度也很大，占用空间也更小。如果你想把数据存成一个人能看懂的形式，那么可以键入：

save hello.txt v ascii

这样就会把数据存成一个文本文档，或者将数据的 ascii 码存成文本文档。桌面会有 hello.txt 文件。如果打开它，我们可以发现这个文本文档存放着我们的数据。这就是读取和存储数据的方法。

操作数据的方法：假设 A 还是之前的矩阵。键入 A(3,2)，将索引第三行第二列的元素。也可以键入 A(2,:) 来返回 A 矩阵第二行的所有元素，A(:,2) 来返回 A 矩阵第二列的所有元素。如图 14所示。

```
>> A

A =

     1     2
     3     4
     5     6

>> A(3,2)

ans =

     6

>> A(2,:)

ans =

     3     4
```

图 14: 向量和矩阵

你也可以在运算中使用这些较为复杂的索引。A([1 3],:)，这个命令的意思是取 A 矩阵的第一行和第三行的每一列，冒号表示的是取这两行的每一列元素。A(:,2)=[10;11;12]，把 A 的第二列元素重新赋值。A=[A,[100;101;102]]，重新添加一列。如图 15所示。

```
>> A([1 3],:)

ans =

     1     2
     5     6

>> A(:,2)=[10;11;12]

A =

     1    10
     3    11
     5    12

>> A=[A,[100;101;102]]

A =

     1    10   100
     3    11   101
     5    12   102
```

图 15: 向量和矩阵

最后还有一个小技巧，如果你就输入 A(:)，这是一个特别的语法结构，意思是把 A 中的所有元素放入一个单独的列向量，这样我们就得到了一个 $9{\times}1$ 的向量，这些元素都是 A 中的元素排列起来的。

矩阵的合并，如图 16所示。

```
>> C=[A B]

C =

     1     2    11    12
     3     4    13    14
     5     6    15    16

>> C=[A,B]

C =

     1     2    11    12
     3     4    13    14
     5     6    15    16
```

图 16: 矩阵合并

还可以 C=[A;B] 将 A 和 B 矩阵上下放在一起。

2.3 Computing on Data 计算数据

首先初始化一些变量，设置 A 为一个 3×2 的矩阵，设置 B 为一个 3×2 矩阵，设置 C 为 2×2 矩阵。

计算两个矩阵的乘积。键入 A×C，得到一个 3×2 矩阵。

对每个元素做运算。方法是做点乘运算 A. B，将 A 中的每一个元素与矩阵 B 中的对应元素相乘，就是两个矩阵元素位运算。Octave 中点号一般用来便是元素位运算。如图 17所示。输入 A.^2 将对每个元素做平方。

```
A =

     1     2
     3     4
     5     6

>> B

B =

    11    12
    13    14
    15    16

>> A.*B

ans =

    11    24
    39    56
    75    96
```

图 17: 矩阵元素位运算

设一个列向量 V=[1;2;3]，键入 1./V，得到每一个元素的倒数，所以会分别算出 1/1 1/2 1/3。矩阵也可以这样操作，1./A 得到 A 中每一个元素的倒数。如图 18所示。

```
>> V=[1;2;3];
>> 1./V

ans =

    1.0000
    0.5000
    0.3333

>> 1./A

ans =

    1.0000    0.5000
    0.3333    0.2500
    0.2000    0.1667
```

图 18: 求倒数

对数运算，对每个元素进行求对数运算 log(V)。还有自然数 e 的幂次运算，以 e 为底，这些元素为幂的运算 exp(V)。用 abs 来对 V 的每一个元素求绝对值 abs(V)。$-V$ 等价于 -1 乘以 V，一般就直接用 $-V$。

对每个元素都加 1。首先构造一个 3 行 1 列的 1 向量，然后把这个 1 向量跟原来的向量相加。用 length(V) 命令，ones(length(V),1) 就相当于 ones(3,1)。另一种更简单的方法直接用 $V+1$。如图 19所示。

```
>> V+ones(length(V),1)

ans =

     2
     3
     4

>> V+1

ans =

     2
     3
     4
```

图 19: 加一

矩阵的转置，A’，转置两次 (A’)’，又重新得到 A 矩阵。

还有一些有用的函数，比如：a=[1 15 2 0.5]，这是一个 1 行 4 列矩阵，val=max(a)，这将返回 A 矩阵中的最大值 15。[val, ind]=max(a)，这将返回 a 矩阵中的最大值存入 val，以及该值对应的索引，元素 15 对应的索引值为 2 存入 ind，所以 ind 等于 2。如图 20所示。注意用命令 max(A)，A 是一个矩阵的话，这样做就是对每一列求最大值。

```
>> a=[1 15 2 0.5]

a =

    1.0000   15.0000    2.0000    0.5000

>> val=max(a)

val =

    15

>> [val,ind]=max(a)

val =

    15


ind =

     2
```

图 20: 求最大值

输入 $a < 3$，这将进行逐元素的运算，所有元素小于 3 的返回 1，否则返回 0。如果输入 find(a<3)，输出小于 3 的元素的索引（下标从 1 开始)。如图 21所示。

```
>> a<3

ans =

     1     0     1     1

>> find(a<3)

ans =

     1     3     4
```

图 21: 比较值

设 A=magic(3)，magic 函数将返回一个矩阵，称为魔方阵或幻方（magic squares)，数学性质：他们所有的行和列和对角线加起来都等于相同的值。这在机器学习基本用不上，但是可以用这个方法很方便地生成一个 3 行 3 列的矩阵。

输入 [r,c]=find(A>=7)，这将找出 A 中大于等于 7 的所有元素的索引，因此，r 和 c 分别表示行和列。如图 22所示。

```
>> [r,c]=find(A>=7)

r =

     1
     3
     2


c =

     1
     2
     3
```

图 22: 比较值

求和函数。sum(a)，就是把 a 中的所有元素加起来了。乘积函数。键入 prod(a)，prod 意思是乘积，它将返回这四个元素的乘积。四舍五入函数。floor(a) 向下四舍五入，所以 a 中的 0.5 将被下舍变成 0。ceil(a)，表示向上四舍五入，所以 0.5 将上舍入变为 1。如图 23所示。

```
>> sum(a)

ans =

   18.5000

>> prod(a)

ans =

    15

>> floor(a)

ans =

     1    15     2     0

>> ceil(a)

ans =

     1    15     2     1
```

图 23: 其它函数

键入 type(3)(MATLAB 中不是此用法)，得到一个 3×3 的矩阵，键入 max[rand(3),rand(3)]，这样做的结果是返回两个 3×3 的随机矩阵，并且逐元素比较取最大值。输入 max(A,[],1)，得到每一列的最大值，1 表示取 A 矩阵第一个维度的最大值。相对地，如果键入 max(A,[],2)，这将得到每一行的最大值。如图 24所示。

```
>> max(rand(3),rand(3))

ans =

    0.7922    0.7060    0.6787
    0.9595    0.8491    0.7577
    0.6557    0.9340    0.8235

>> max(A,[],1)

ans =

     8     9     7

>> max(A,[],2)

ans =

     8
     7
     9
```

图 24: 最大值

所以可以用这个方法求每一行或每一列的最值，另外，在默认情况下 max(A) 返回的是每一列的最大值，如果要找出整个矩阵 A 的最大值，可以输入 max(max(A))，或者将 A 矩阵转换为一个列向量，然后键入 max(A(:))。

最后，A 为 9 行 9 列的魔方阵，输入 sum(A,1)，得到每一列的和。sum(A,2)，得到每一行的和。算对角线元素的和。首先构造一个 9×9 的单位阵，键入 eye(9) ，然后用 A 逐点乘以这个单位矩阵，除了对角线元素外，其他元素都会得到 0。键入 sum(sum(A.*eye(9)))。如图 25所示。

```
   369   369   369   369   369   369   369   369   369

>> sum(A,2)

ans =

   369
   369
   369
   369
   369
   369
   369
   369
   369

>> sum(sum(A.*eye(9)))

ans =

   369
```

图 25: 魔方矩阵求和

flipup/flipud 表示向上/向下翻转。(matlab 无此功能) 求逆矩阵，pinv(A) 称为为逆矩阵，你就把它看成是矩阵 A 求逆，因此这就是 A 矩阵的逆矩阵。设 temp=pinv(A)，然后再用 temp 乘以 A，实际上得到的就是单位矩阵，对角线为 1，其他元素为 0。如图 26所示。

```
>> A=rand(3)

A =

    0.6948    0.0344    0.7655
    0.3171    0.4387    0.7952
    0.9502    0.3816    0.1869

>> pinv(A)*A

ans =

    1.0000   -0.0000    0.0000
    0.0000    1.0000   -0.0000
   -0.0000   -0.0000    1.0000
```

图 26: 求逆

2.4 Plotting Data 绘图数据

之前的视频中，谈到绘制成本函数 J(θ)，可以帮助确认梯度下降算法是否收敛。通常情况下，绘制数据或学习算法所有输出，会帮助改进学习算法。

首先快速生成一些数据用来绘图。

» t=[0:0.01:0.98]; » t » y1 = sin(2*pi*4*t); » plot(t,y1)

回车后得到图像。如图 27所示。

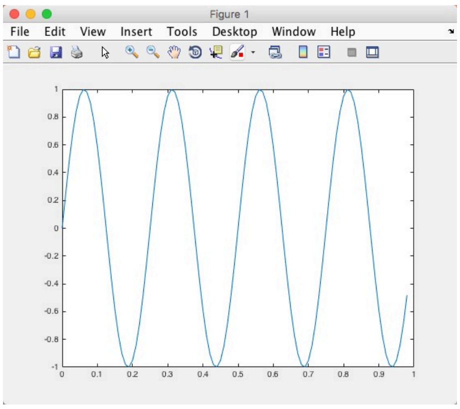


图 27: 绘制 sin 图像

横轴是变量 t，纵轴是 y1，也就是我们刚刚所输出的正弦函数。

设置 y2:

» y2 = cos(2*pi*4*t);

» plot(t,y2)

同时表示正弦和余弦曲线。

» plot(t,y1)

» hold on

» plot(t,y2)

» plot(t,y2,‘r’)

» xlabel(‘time’)

» ylabel('value')

» legend('sin','cos')

» title('myplot')

输入：plot(t,y1)，得到正弦函数，我使用函数 hold on，hold on 函数的功能是将新的图像绘制在旧的之上。再绘制 y2，输入：plot(t,y2)。要以不同的颜色绘制余弦函数，r 代表所使用的颜色。xlabel('time')，来标记 X 轴即水平轴，ylabel('value')，来标记垂直轴的值。legend('sin','cos') 标记两条函数曲线。最后输入 title('myplot')，在图像的顶部显示这幅图的标题。如图 28所示。

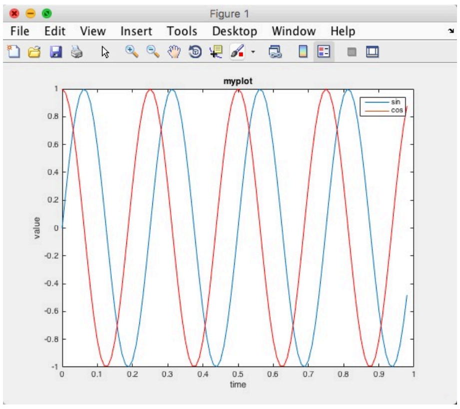


图 28: 绘制 sin、cos 图像

如果你想保存这幅图像，你输入 cd '地址' print -dpng'myplot.png'，png 是一个图像文件格式。

删掉图像 close 命令。Octave 可以让你为图像标号。你键入 figure;plot(t,y1); 将显示第一张图，绘制了变量 t y1。键入 figure(2);plot(t,y2); 将显示第一张图，绘制了变量 t y2。subplot 命令。subplot(1,2,1)，将图像分为一个 1*2 的格子，就是前两个参数，然后它使用第一个格子：plot(t y1)，也就是最后一个参数 1 的意思。如图 29所示。

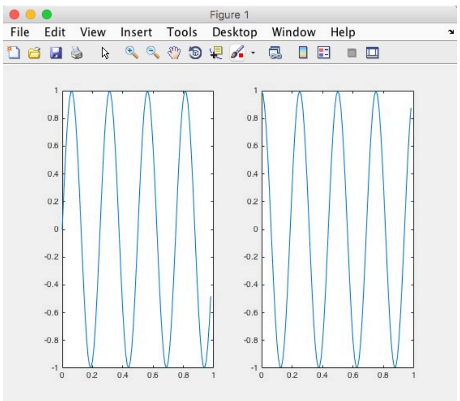


图 29: 绘制 sin、cos 图像

最后一个命令，你可以改变轴的刻度，比如改成 [0.5 1 -1 1]，输入命令：axis([0.5 1 -1 1])，就是设置了右边图的 x 轴和 y 轴的范围。

Clf(清除一幅图像)。

可视化矩阵，imagesc(A) 命令，他将绘制一个 $5*5$ 的矩阵，一个 $5*5$ 的彩色格图，不同颜色对应

矩阵中的不同值。使用 colorbar，命令 imagesc(A),colorbar,colormap gray; 。实际上是在同一时间运行三个命令：运行 imagesc，然后运行 colorbar，然后运行 colormap grap。生成了一个颜色图像，一个灰度分布图，并在右边也加入一个颜色条。所以这个颜色条显示不同深浅的颜色所对应的值。如图 30所示。

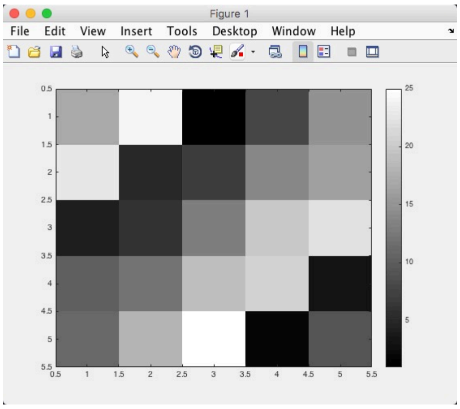


图 30: 绘制矩阵

不同的方格对应于一个不同的灰度。输入 imagesc(magic(15)),colorbar,color gray。这将会是一副 15∗15 的 magic 方阵值的图。如图 31所示。

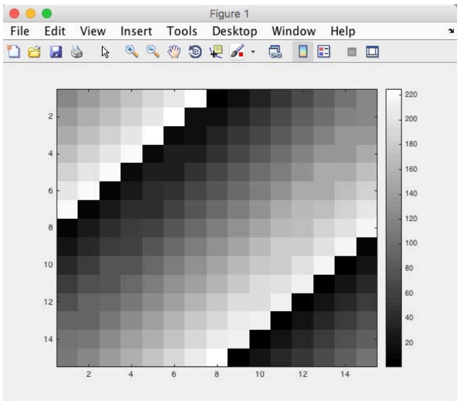


图 31: 绘制矩阵

总结，上几条命令是用逗号进行连接函数调用。如果我键入 a=1,b=2,c=3 然后一直按 Enter 键，其实这是将这三个命令同时执行，或者是将三个命令一个接一个执行，他将输出所有这三个结果。Octave 中分号连接不输出任何值，MATLAB 分号返回最后一个表达式。

2.5 For,while,if statements,and functions 控制语句：for,while,if 语句

控制语句的定义和使用。

首先，将 V 值设为一个 10 行 1 列的零向量:v=zeros(10,1)。接着用 for 循环，让 i 等于 1 到 10，写出来就是 i=1:10。设置 V(i) 的值等于 2 的 i 次方，循环最后写上 “end”。如图 32所示。

```
>> for i=1:10,
     v(i)=2^i;
end;
>> v

v =

      2
      4
      8
     16
     32
     64
    128
    256
    512
   1024
```

图 32: for 循环

通过索引 (indices) 等于 1 到 10: indices=1:10，这时 indices 就是一个从 1 到 10 的序列。i=indices, 等价于直接把 i 写到 1 到 10。这时 indices 就是一个从 1 到 10 的序列。如图 33所示。

```
>> for i=indices,
disp(i);
end;
 1

 2

 3

 4

 5

 6

 7

 8

 9

10
```

图 33: for 循环

Octave 里也有 “break” 和 “continue” 语句，你也可以在 Octave 环境李世勇那些循环语句。

while 循环语句。例子如图 34所示。

```
>> i=1;
>> while true,
 v(i)=999;
i=i+1;
if i==6;
break;
end;
end;
>> v

v =

        999
        999
        999
        999
        999
         64
        128
        256
        512
       1024
```

图 34: whlie 循环

ifelse 语句。例子如图 35所示。

```
>> v(1)=2;
>> if v(1)==1,
     disp('The value is one');
   elseif v(1)==2,
     disp('The value is two');
   else
     disp('The value is not one or two.');
   end;
The_value is two
```

图 35: ifelse 语句

如果需要退出 Octave，可以键入 exit 命令，然后回车就会退出，或者 quit 也可以。

如何定义和调用函数。

在桌面上存一个预先定义的文件名为 “SquareThisnNmber.m”(文件名一定要和函数名一致)，是在 Octave 环境下定义的函数。用写字板程序来打开文件。如何在 Octave 里定义函数，文件只有三行。

function y = SquareThisNumber(x)

y = x^2;

第一行意思是，告诉 Octave，我想返回一个值，并且返回值被存放在变量 y 里。另外，告诉 Octave 这个函数有一个参数 x，还有定义的函数体，也就是 y 等于 x 的平方。尝试直接调用这个函数 SquareThisNumber(5)，这是无法完成的。，Octave 说这个方程为被定义。这是因为 Octave 不知道不知道文件的路径。使用 pwd 查看路径，用 cd 将路径设为文件所在位置。再次调用函数，可以运行。如图 36所示。

```
>> SquareThisNumber(5)
Undefined function or variable 'SquareThisNumber'.

>> cd '~/Desktop/'
>> SquareThisNumber(5)

y =

    25


ans =

    25
```

图 36: 定义调用函数

一个更高级的功能，“search path(搜索路径)”，使用 addpath 命令添加路径，添加路径将该目录添加到 Octave 的搜索路径，这样即使跑到其他路径下，Octave 依然知道会在文件所在路径下寻找函数。如图 37所示。如图 37所示。

```
>> addpath '~/Desktop/'
>> cd ~
>> SquareThisNumber(5)

y =

    25


ans =

    25
```

图 37: 添加路径

Octave 还有一个其他许多编程语言都没有的概念，就是可以允许定义一个函数，使得返回值是多个值或多个参数。例子，定义一个函数 “SquareAndCubeThisNumber(x)”(x 的平方以及 x 的立方)。返回值时两个 y1 y2。

function [y1,y2]=SquareAndCubeThisNumber(x)

$y1 = x^2$;

$y2 = x^3$;

如果键入 [a,b]=SquareAndCubeThisNumber(5)，然后，a=25 b=125。

复杂例子。有一个数据集，数据点为 [1,1]，[2,2]，[3,3]。用 Octave 函数来计算代价函数 $J(\theta)$，就是计算不同的 θ 所对应的代价函数值 J。

首先把数据放到 Octave 里，X=[1 1;1 2;1 3]；如图 38所示。

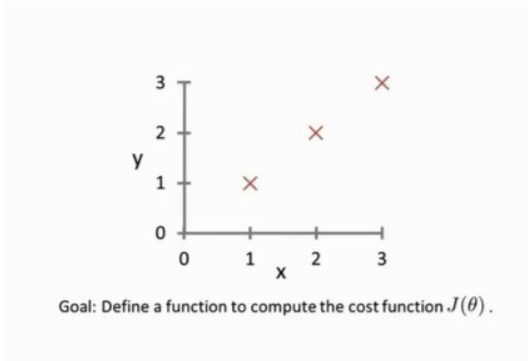


图 38: 数据

写一个函数 costFunctionJ.m。如图 39所示。

```
costFunctionJ.m

function J = costFunctionJ(X, y,theta)

% X is the ''design matrix'' containing our tarining examples.
% y is the class labels

m = size(X,1);      % number of training examples
predictions = X*theta;   %predictions of hypothesis on all m examples
sqrErrors = (predictions-y).^2;   %squared errors

J = 1/(2*m) * sum(sqrErrors);
```

图 39: 计算代价函数

在 Octave 里运行。如图 40所示。

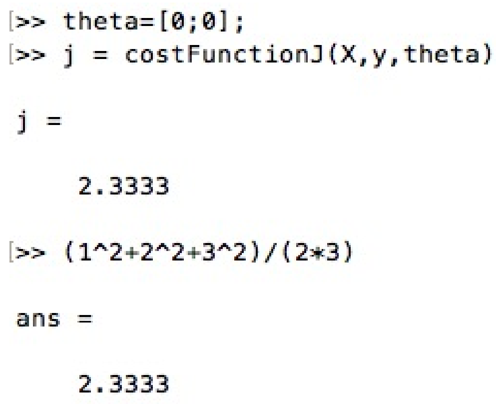
```
>> X=[1 1;1 2;1 3];
>> y=[1;2;3];
>> theta=[0;1];
>> j = costFunctionJ(X,y,theta)

j =

     0
```

图 40: 计算代价函数

设置 $\theta0$ 等于 0，$\theta1$ 等于 1，恰好 45 度的斜线，这条线是可以完美拟合数据集的。而相反地，如果设置 theta 等于 [0; 0]，那么这个假设就是 0 是所有的预测值，和刚才一样，设置 $\theta0 = 0$，$\theta1$ 也等于 0，然后我计算的代价函数，结果是 2.333。实际就等于 1 的平方，也就是第一个样本的平方误差，加上 2 的平方，加上 3 的平方，然后除以 2m，也就是训练样本数的两倍，这就是 2.33。如图 41所示。



```
>> theta=[0;0];
>> j = costFunctionJ(X,y,theta)

j =

    2.3333

>> (1^2+2^2+3^2)/(2*3)

ans =

    2.3333
```

图 41: 计算代价函数

确实是可以计算出正确的代价函数的。至少基于这里的 X 和 y 是成立的。也就是我们这几个简单的训练集，至少是成立的。

2.6 Vetorization 向量化

无论你是用 Octave，还是别的语言，比如 MATLAB 或者你正在用 Python、NumPy 或 Java C C++，所有这些语言都具有各种线性代数库，这些库文件都是内置的，容易阅读和获取，他们通常写得很好，已经经过高度优化，通常是数值计算方面的博士或者专业人士开发的。优点：首先，这样更有效，也就是说运行速度更快，并且更好地利用你的计算机里可能有的一些并行硬件系统等等（first, is more efficient, so you just run more quickly and take better advantage of any parallel hardware your computer may have and so on. ）；其次，这也意味着你可以用更少的代码来实现你需要的功能（it also means that you end up with less code that you need to write, so it's a simpler implementation that is therefore maybe also more likely to be by free.）。因此，实现的方式更简单，代码出现问题的有可能性也就越小。

Vectorization example. 这是一个常见的线性回归假设函数：

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

如果要计算 $h_\theta(x)$，把它看作 $\theta^T x$，可以写成两个向量的内积:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

Unvectorized implementation（变形）：

prediction = 0.0;

for j= 1:n+1,

prediction = prediction + theta(j) * x(j)

end;

计算 $h_\theta(x)$ 是未向量化的。注意：这里的向量用下标是 0，但因为 MATLB 的下标从 1 开始，在 MATLAB 中 θ_0 可能用 θ_1 来表示，从 1 开始。

Vectorized implementation:

prediction = theta' * x;

把 θ 和 x 看做向量，只需要令变量 prediction 等于 theta 转置乘以 x。这行代码就是利用 Octave 的高度优化的数值，线性代数算法来计算两个向量 θ 以及 x 的内积，这样向量化的实现更简单，它运行起来也将更加高效。

这就是 Octave 所做的而向量化的方法，在其他编程语言中同样可以实现。让我们来看一个 C++ 的例子，如图 42所示。

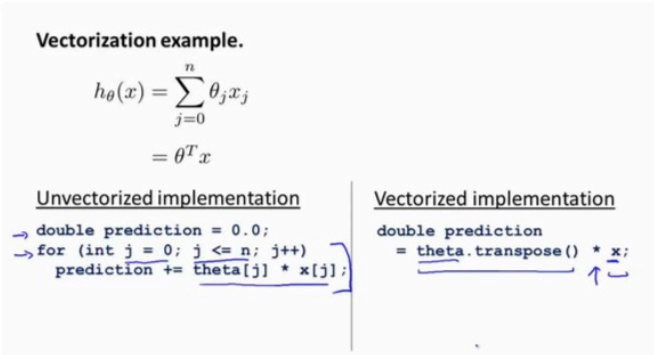


图 42: 向量化 vs 非向量化

使用较好的 C++ 数值线性代数库 library ，你可以写出像右边这样的代码。

一个更为复杂的例子，这是线性回归算法梯度下降（gradient descent of a linear ragression) 的更新规则:

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_2^{(i)}$$

可以想象实现这三个方程的方式之一，就是用一个 for 循环，就是让 j 等于 0、等于 1、等于 2，来更新 θ_j。

用向量化的方式来实现，把 θ 看做一个向量，然后我用 $\theta - \alpha$ 乘以某个别的向量 δ 来更新 θ 。

δ 等于:

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