Multiple Features

Linear regression with multiple variables is also known as "multivariate linear regression".

$x_j^{(i)}$ = value of feature j in the i^{th} training example
$x^{(i)}$ = the column vector of all the feature inputs of the i^{th} training example
m = the number of training examples
$n = \left|x^{(i)}\right|$; (the number of features)

The multivariable form of the hypothesis function accommodating these multiple features is as follows:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

$$h_\theta(x) = \begin{bmatrix}\theta_0 & \theta_1 & \dots & \theta_n\end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

The training examples are stored in X row-wise. The following example shows us the reason behind setting $x_0^{(i)} = 1$:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

As a result, you can calculate the hypothesis as a column vector of size (m x 1) with:

$$h_\theta(X) = X\theta$$

多元线性回归的梯度下降

The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

repeat until convergence: {

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

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

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

...

}

In other words:

repeat until convergence: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j := 0...n}$$

}

Two techniques to help with this are **feature scaling** and **mean normalization**. Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero. To implement both of these techniques, adjust your input values as shown in this formula:

$$x_i := \frac{x_i - \mu_i}{s_i}$$

调节各项值范围的方法：每一项减去平均值，除以范围（最大值和最小值的差）

注意是以给定的样本为基础来进行计算

Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

Polynomial Regression（多项式回归）

Our hypothesis function need not be linear (a straight line) if that does not fit the data well.

We can **change the behavior or curve** of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

For example, if our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1, to get the quadratic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

二次方程是抛物线，最终会下降，因此三次方程或根式更为理想

In the cubic version, we have created new features x_2 and x_3 where $x_2 = x_1^2$ and $x_3 = x_1^3$.

To make it a square root function, we could do: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

使用多项式回归时必须注意调节各项数值的大小

Normal Equation（正规方程）

Gradient descent gives one way of minimizing J. Let’s discuss a second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm. In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj ’s, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

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

Octave：pinv(X' * X) * X' * y

The following is a comparison of gradient descent and the normal equation:

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large

正规方程不适用于大量特征计算以及其他更为复杂的学习算法

番外篇：可能导致XTX不可逆的两种情况
①特征之间呈线性相关（可以通过删除多余特征解决）

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)

②样本数量 ≦ 特征数量 （可以通过regularization正规化解决）

- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization" (to be explained in a later lesson).

事实是不可逆的情况不会经常出现，因此没有必要过于关注
并且在Octave中使用pinv函数可以无视不可逆的情况得出结果

Octave教学篇
使用Octave是因为它更快更牛逼，可以先用Octave编写并测试程序，再改写成Java或者Python等等
不等于： ～=
隐藏命令行：PS1('>>');
以二进制形式把v的数据存储为名为hello的文件：save hello.mat v
以人能看懂的形式存储：save hello.txt v -ascii
索引：
A(3, 2) = 矩阵A的第3行第2列
A(2, :) = 矩阵A的第2行所有元素
A([1 3], :) = 矩阵A的第一个索引值为1或3的所有元素（此处为第1行和第3行的所有元素）
可以通过A(: , 2) = [10; 11; 12]的形式给矩阵A的第二列进行赋值
A = [A, [100; 101; 102]]; 在原矩阵A的右边加一列[100; 101; 102]
A(:) = 把矩阵A中所有的元素变为一个单列的向量

A .* B = 将矩阵A中每一个元素乘以矩阵B中对应位置的元素
A .^ B = 将矩阵A中每一元素B方
1 ./ A = 求矩阵A中每一个元素的倒数
log(A) = 求矩阵A中每一个元素的对数
exp(A) = 以矩阵A中每一个元素进行自然数e的幂运算
abs(v) = 求矩阵A中每一元素的绝对值
-A = 求矩阵A中每一个元素的负数
A' = 矩阵A的转置
max(A) = 求向量A的最大值，当A为矩阵时对列求最大值
A < 3 = 对矩阵A的每一个元素进行逻辑运算
find(A < 3) = 求矩阵A中满足<3的元素的位置
magic(3) = 形成一个每一行和每一列以及对角线的和都相等的矩阵（没有实际作用）
[r, c] = find(A >= 7) 返回满足>7的元素所在的行和列
sum(A) = 矩阵A中所有元素的和
sum(A, 1) = 矩阵A中每一列的和
size(A) = 返回矩阵A的行和列的数量
size(A, 2) = 返回矩阵A的列数
prod(A) = 矩阵A中所有元素的积
floor(A) = 矩阵A中所有元素向下取整
ceil(A) = 矩阵A中所有元素向上取整
rand(3) = 生成一个3x3的随机矩阵
max(rand(3), rand(3)) = 取两个3x3矩阵中的最大值，生成一个3x3的随机矩阵
max(A, [], 1) = 求矩阵A中每一列的最大值（1表示取第一个维度的最大值，即列的最大值）
max(A, [], 2) = 求矩阵A中每一行的最大值
max(A)默认求列的最大值，可以通过max(max(A))或max(A(:))求整个矩阵A的最大值
sum(sum(A .* eye(9))) = 求矩阵A(9x9)的对角线和
flipud = 向上或向下翻转
sum(sum(A .* flipud(eye(9)))) = 求矩阵A(9x9)的副对角线和
pinv(A) = 求矩阵A的逆
[0：0.01: 1] = 0到1之间以0.01为间隔形成矩阵

图形化

```
t = [0: 0.01: 0.98];
y1 = sin(2*pi*4*t);
y2 = cos(2*pi*4*t);
plot(t, y1);
```

hold on; = 保留上一个图片

```
plot(t, y2, 'r'); >r表示使用的颜色
xlabel('time') = 将x轴标记为time，同理ylabel
legend('sin', 'cos') = 显示sin和cos的曲线的颜色
title('my plot') = 在图像顶部显示标题
cd 'C:/Users/Shawn/Desktop'; print -dpng 'myPlot.png' = 改变路径保存图像
close = 关闭图像
figure(1); plot(t, y1); = 定义figure可将图像保存在不同的figure
subplot(1, 2, 1); = 将plot分为1x2（前两个引数）的格子，并显示第1（第三个引数）个格子
axis([0.5 1 -1 1]) = 调整x轴和y轴的取值
clf; = 清除所有图像
imagesc(A); = 显示彩色的矩阵A
imagesc(A), colorbar, colormapgray; = colorbar显示不同颜色对应不同的数字
```

图形化实用例：

```
plot(X(negative, 1), X(negative, 2), 'ko', 'LineWidth', 2, 'MarkerSize', 5, 'MarkerFaceColor', 'y');
```

‘ko’：圆圈形标记
‘k+’：十字形标记
LineWidth：线宽
MarkerSize： 标记大小
MarkerFaceColor：标记颜色

在Octave中使用函数

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

i = 1;
while i <= 5,
   v(i) = 100;
   i = i+1;
end;

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

```
        break;
    end;
end;

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;
```

Octave Search Path

```
addpath('C:/Users/Shawn/Desktop')
```

定义函数
新建后缀名为.m的文件（告诉Octave这儿有函数）
文件内容例：

```
function y = squareThisNumber(x)
y = x^2;
```

Octave不同与其他语言的地方在于可以返回一个以上的结果
例如：

```
function [y1, y2] = squareAndCubeThisNumber(x);
y1 = x^2;
y2 = x^3;
```

输入：

```
[a, b] = squareAndCubeThisNumber(5);
```

将返回a和b的值

向量化（Vectorization）
用更简便的方法实现计算

Week2 Programming

Data Set:

```
data = load('ex1data1.txt');       % read comma separated data
X = data(:, 1); y = data(:, 2);
m = length(y);                     % number of training examples
```

ComputeCost:

```
J = 0;
J = (1/(2*m))*sum((X*theta - y).^2);
```

Gradient Descent:

```
J_history = zeros(num_iters, 1);
for iter = 1:num_iters
    a = theta(1);
    b = theta(2);
    a = a - alpha * (1/m) * sum((X (:, 1))' * (X * theta - y));
    b = b - alpha * (1/m) * sum((X (:, 2))' * (X * theta - y));
    theta = [a; b];
    % Save the cost J in every iteration
    J_history(iter) = computeCost(X, y, theta);
end
```

Gradient Descent For Multiply Feature

```
    hypo = (1/m) * sum(repmat((X * theta - y), 1, size(X,2)) .* X);
    theta = theta - (alpha * hypo)';
```

repmat(a, b ,c)：
a = 重复对象
b = 行的重复次数
c = 列的重复次数

repmat(a, b)：
a = 重复对象
b = 行和列的重复次数

repmat(a,[b c d])：
a = 重复对象
b = 行的重复次数
c = 列的重复次数
d = 生成矩阵的数量