Week 2 笔记

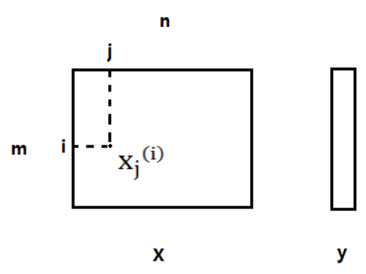
by 骆剑 2017/5/23

1.Linear Regression with multiple variables （多变量线性回归）

Basic Concept（基本概念）：

0. Notations:

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



m 表示样本数量

n 表示特征维度

$x_j^{(i)}$表示第 **i** 个样本的第 **j** 个特征

$x^{(i)}$表示第 **i** 个样本的全部 **n** 个特征

1.Hypothesis:

$$h_\theta(x) = \theta_0 \cdot 1 + \theta_1 x_1 + \cdots + \theta_j x_j + \theta_n x_n$$

为了简化表示，记

向量 $\theta = [\theta_0, \theta_1, \ldots, \theta_j, \ldots, \theta_n] \ \epsilon \mathbb{R}^{n+1}$;

$x_0 \equiv 1$

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

2.Parameters: θ_0 ; θ_1 ; … ; θ_j ; … ; θ_n

3.Cost Function: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

4.Goal: minimize $J(\theta)$

Gradient descent for multiple variables（多变量梯度下降算法）

Repeat until convergence

{ //依旧是同步更新

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

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

(for j from 0 to n)

$$\theta_j := tmp\_j$$

(for j from 0 to n)

}

偏导推导过程

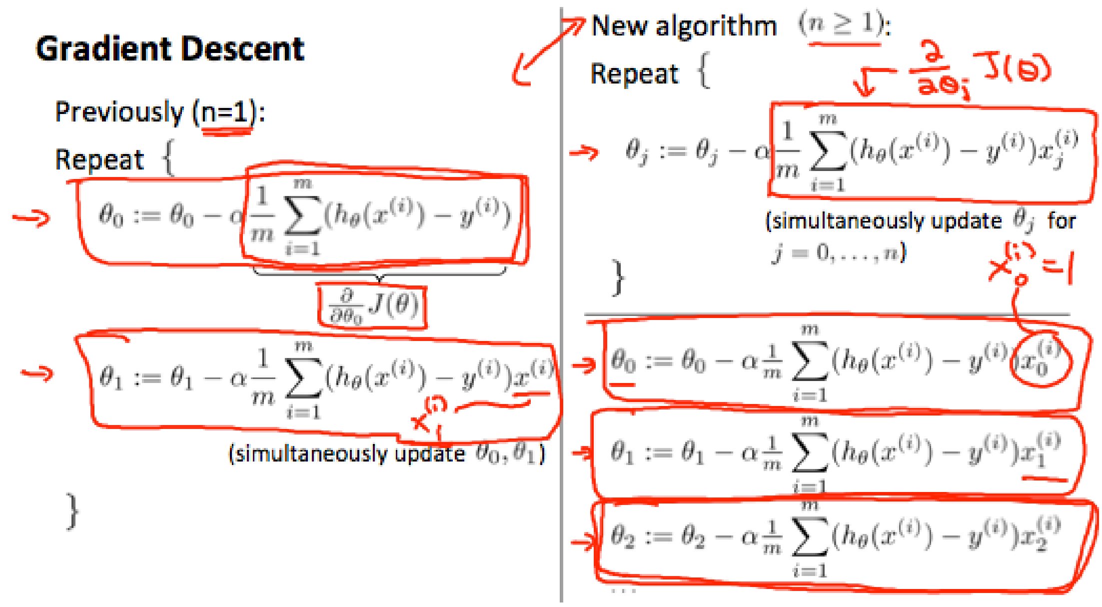
$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \{\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\}$$

$$= \frac{\partial}{\partial \theta_j} \{\frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 x_1{}^{(i)} + \ldots + \theta_j x_j{}^{(i)} + \cdots + \theta_n x_n{}^{(i)} - y^{(i)}\right)^2\}$$

$$= \frac{1}{m} \sum_{i=1}^{m} [\left(\theta_0 + \theta_1 x_1{}^{(i)} + \ldots + \theta_j x_j{}^{(i)} + \cdots + \theta_n x_n{}^{(i)} - y^{(i)}\right) \cdot x_j{}^{(i)}]$$

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

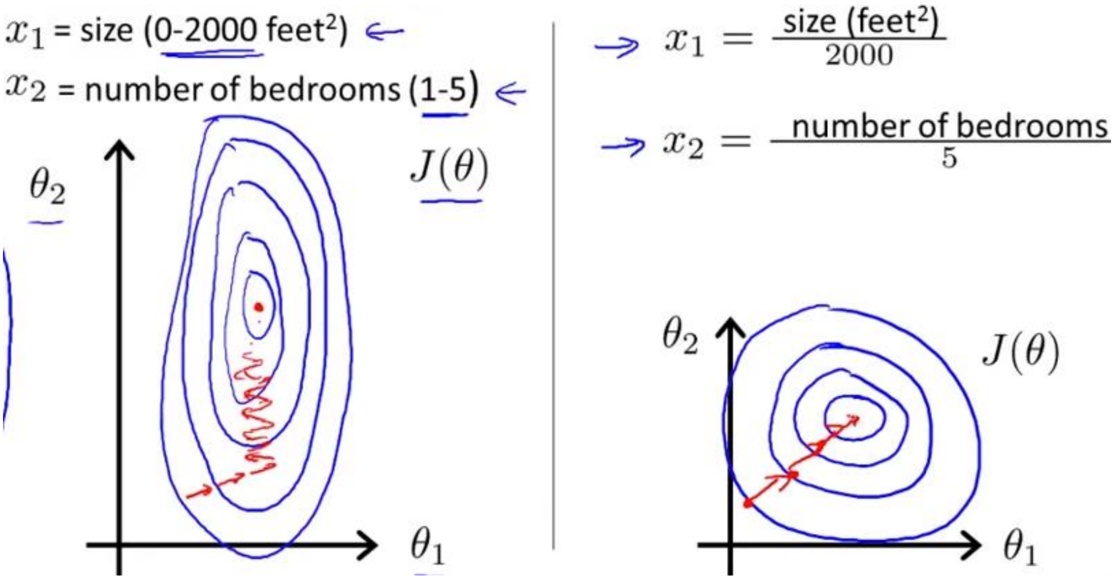
当 j=0 时，$x_0^{(i)} \equiv 1$，故与单变量表达一致



2.Gradient descent 优化

2.1 策略之一 数据预处理

***为什么要做 feature scaling？**



若多个特征的取值范围差距较大，则所形成的等高线非常尖，因为梯度下降是**沿着等高线的法线方向（垂直等高线走）**，故会走 Z 字形，收敛速度慢甚至不能收敛。

参考博文 http://blog.csdn.net/code_lr/article/details/51438649

feature scaling 特征缩放

原数据 $x_j^{(0)}$，$x_j^{(1)}$， … ， $x_j^{(m)}$

原数据的最大值 max

新数据 $\frac{x_j^{(0)}}{max}$， $\frac{x_j^{(1)}}{max}$， … ， $\frac{x_j^{(m)}}{max}$

i.e.一组特征数值为 1，2，3，4，5，max=5

得到特征的新数值为 0.2，0.4，0.6，0.8 , 1

mean normalization 均值归一化

原数据 $x_j^{(0)}$，$x_j^{(1)}$， … ， $x_j^{(m)}$

原数据的均值 $mean = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$

新数据 $x_j^{(0)} - mean$ ， $x_j^{(1)} - mean$ ， … ， $x_j^{(m)} - mean$

新数据的均值 $new\_mean = \sum_{i=1}^{m} x_j^{(i)} - m * mean \equiv 0$

i.e.一组特征数值为 1，2，3，4，5，故均值为 3，各减 3，

得到特征的新数值为-2，-1，0，1，2，均值为 0

综上 数据预处理的方式可以整合为

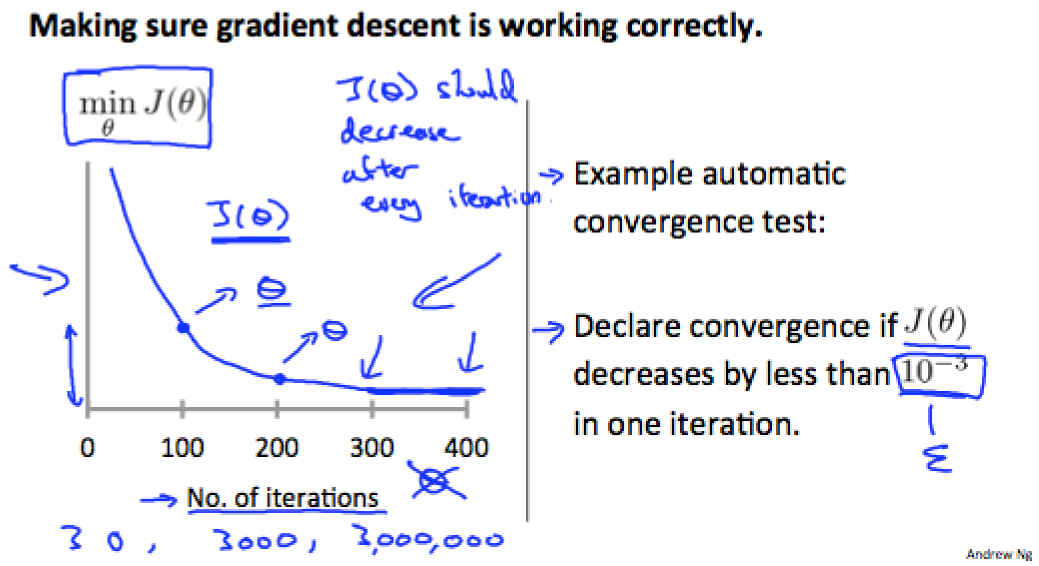
$$x_j = \frac{x_j - u_j}{S_j}$$

[注]: u_j 是 特征的均值

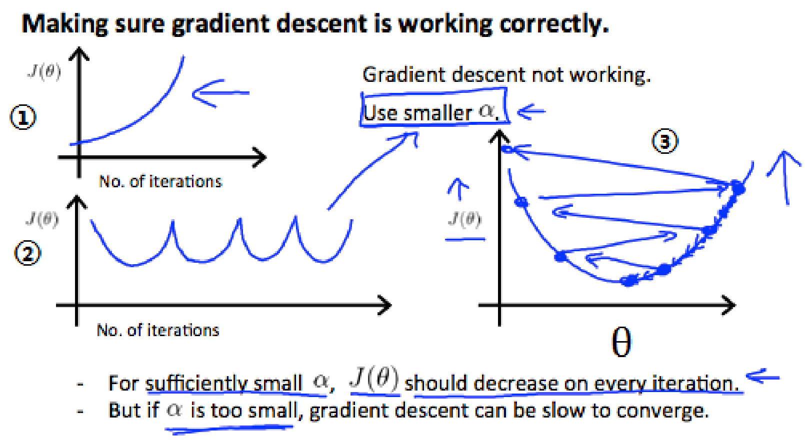
S_j 是 特征的最大最小值之差 或者 特征的标准差

Gradient descent 优化

2.2 策略之二 学习率α的调整



通常绘制J(θ)随 iterations 变化的趋势图来观察学习率是否设置得恰当。如上图所示的趋势图表示正常的梯度下降过程，在 300-400 的 iteration J(θ)趋于收敛。



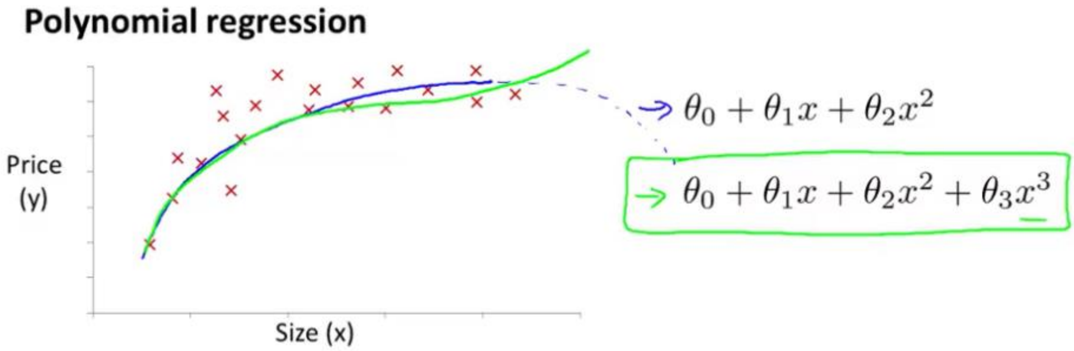
这①②两种情况表示非正常的梯度下降过程，建议采用更小的学习率

③解释了为什么会产生①②这两种情况

总结：学习率过小:收敛慢；学习率过大：可能不收敛，建议降低学习率

3.Polynomial Regression with one variable

（单变量多项式回归=>多变量线性回归）



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

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

以单变量为例，有很多时候，线性回归不能很好地拟合，有时需要加入二次项，三次项。如上所示。横轴是房屋的大小，纵轴是房屋的价格，明显一次项（线性）无法进行很好地拟合，二次项也不行，因为会二次项会下降，显然不符合实际情况。所以，在这里考虑到三次项。我们只需要自行添加特征：

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

就可以将 单变量的多项式回归 转换成 多变量的线性回归

多变量的线性回归在上面已经讲解过了，就不再赘述

另外有一点需要注意，比如 size 的取值为[1,1000]，那么 size^2 的取值为[1,1000000]，size^3 同理，故需要对特征进行**预处理**。

若仅进行 **feature scaling**，那么

x1=size/1000

x2=(size)^2/1000000

x3=(size)^3/1000000000

在此，推荐一个比较不错的函数

Suppose you want to predict a house's price as a function of its size. Your model is

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

Suppose size ranges from 1 to 1000 (feet^2). You will implement this by fitting a model

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

Finally, suppose you want to use feature scaling (without mean normalization).

Which of the following choices for x_1 and x_2 should you use? (Note: $\sqrt{1000} \approx 32$.)

当然不能忘记预处理(假设仅进行 feature scaling)

$$x_1 = \frac{\text{size}}{1000},\ x_2 = \frac{\sqrt{(\text{size})}}{32}$$

4.Normal Equation （正规方程，不需要 feature scaling）

4.1 求解 θ 的两种方法：1. Gradient descent　　2. Normal Equation

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 语句 θ=pinv(x’*x)*x’*y

Gradient Descent **vs** 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^TX
Works well when n is large	Slow if n is very large

关于如何特征维度 n 的大小如何界定问题，Ng 的意见是

一般 n<=10000 算是小，可以用 Normal Equation

n>10000 算是大，用 Gradient Descent

4.2Normal Equation Noninvertibility

When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv.' The 'pinv' function will give you a value of θ even if X^TX is not invertible.

If X^TX is **noninvertible,** the common causes might be having :

- Redundant features, where two features are very closely related (i.e. they are **linearly dependent**)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "**regularization**" (to be explained in a later lesson).

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.

5. x = A^{-1}b 的梯度下降解法

原始解法：x = pinv(A)*b

梯度下降解法： Ax=b

定义损失函数 cost function 为

$$f(x) = ||Ax - b||^2 = \sum_i^n \left[(Ax)_i - b_i\right]^2$$

梯度下降 x 更新公式

$$x_{\text{new}} = x - \text{alpha} \cdot \frac{df(x)}{dx}$$

$$\frac{df(x)}{dx} = 2A(Ax - b)$$

Matlab 函数：收敛定义为 cost<10^(-6)

```
function A_inv_b = matrixInverseVector(A, b, x_init, alpha)
    % Your code here
    x = x_init
     cost = norm(A*x-b) ^ 2
     while cost >= 10^(-6)
          x = x - alpha*2*A*(A*x-b)
          cost = norm(A*x-b) ^ 2
     end
     A_inv_b = x
End
```

6. 线性回归大作业

（1）cost function 部分代码：

```
J = (X*theta-y)'*(X*theta-y)/(2*m);
```
（Ng 推荐写法）

```
J = sum((X*theta-y).^2)/(2*m);
```
（自己一开始的写法）

（2）Gradient Descent 部分代码

（简便写法）

```
theta = theta - alpha * 1/m * (X'*(X*theta-y));
```

（复杂写法）

```
for j = 1:n
        tmp(j) = theta(j) - alpha/m*sum((X*theta-y).*X(:,j));
end
theta = tmp;
```

（3）theta 的两种解法

解法一：Gradient Descent

由于 Gradient Descent 需要对数据做 normalization，并且预测集也要做同样的 normalization，所以要保存训练集 normalization 时的 mu（均值）和 sigma（标准差）。

解法二：Normal Equation

由于 Normal Equation 不需要对数据做 normalization，所以预测集也不做处理。

这两种解 theta 的区别也导致了 theta 值的不同，但是前者对预测集做预处理，后者不需要，故也解释了疑问。

（4）feature Normalization 部分的 bsxfun 函数的使用

```
mu = mean(X);
sigma = std(X);
X_norm = bsxfun(@minus,X,mu);
X_norm = bsxfun(@rdivide,X_norm,sigma);
```

X 是 m x (n+1)矩阵

mu 是 1 x (n+1)矩阵

若想让 X 的每一行按减去 mu，要使用 bsxfun 函数

除以 sigma 同理