Multiple Features

Linear regression with multiple variables is also known as "multivariate linear regression". We now introduce notation for equations where we can have any number of input variables:

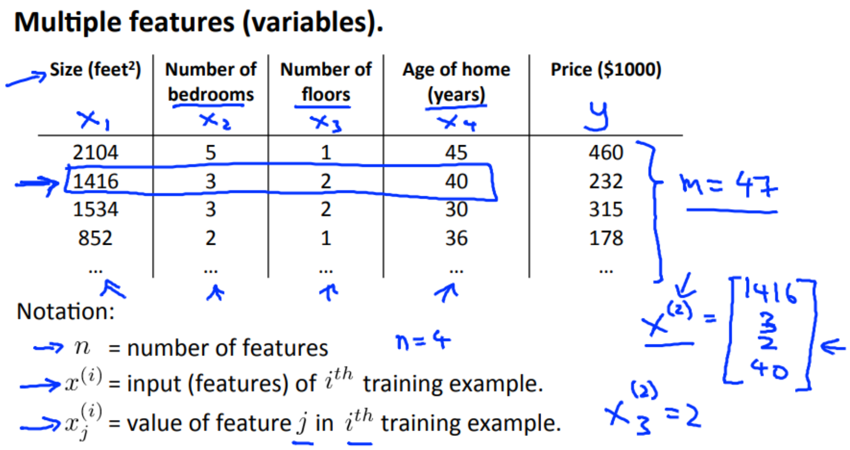
$$
\begin{aligned}
x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\
x^{(i)} &= \text{the input (features) of the } i^{th} \text{ training example} \\
m &= \text{the number of training examples} \\
n &= \text{the number of features}
\end{aligned}
$$

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

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

In order to develop intuition about this function, we can think about:

- $\theta 0$ as the basic price of a house, $\theta 1$ as the price per square meter, $\theta 2$ as the price per floor
- x1 will be the number of square meters in the house, x2 the number of floors



Multiple features (variables).

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

$m = 47$

Notation:

n = number of features $n = 4$

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$

$x_3^{(2)} = 2$

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

Gradient Descent For Multiple Variables

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

$$
\begin{aligned}
&\text{repeat until convergence: } \{ \\
&\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)} \\
&\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)} \\
&\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)} \\
&\cdots \\
&\}
\end{aligned}
$$

Feature Scaling

We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ideally:

$$-1 \le x(i) \le 1$$

or

$$-0.5 \le x(i) \le 0.5$$

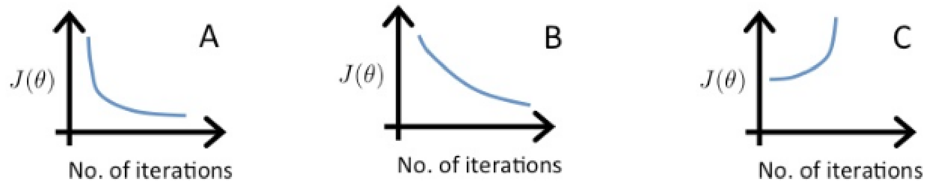
Two techniques to help with this are:

- Feature scaling involves dividing the input values by the range-S of the input variable, resulting in a new range of just -1 < x < 1.
- Mean normalization involves subtracting the average value-μ for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero. -0.5 < x < 0.5

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

Learning rate

Suppose a friend ran gradient descent three times, with $\alpha = 0.01$, $\alpha = 0.1$, and $\alpha = 1$, and got the following three plots (labeled A, B, and C):



Which plots corresponds to which values of α?

- ○ A is $\alpha = 0.01$, B is $\alpha = 0.1$, C is $\alpha = 1$.
- ◉ A is $\alpha = 0.1$, B is $\alpha = 0.01$, C is $\alpha = 1$.

Correct

In graph C, the cost function is increasing, so the learning rate is set too high. Both graphs A and B converge to an optimum of the cost function, but graph B does so very slowly, so its learning rate is set too low. Graph A lies between the two.

Features and Polynomial Regression

We can improve our features and the form of our hypothesis function in a couple different ways:

- We can combine multiple features into one. For example, we can combine x1 and x2 into a new feature x3 by taking x1*x2.

Polynomial Regression

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 hθ(x)=θ0+θ1x1 then we can create additional features based on x1, to get the quadratic function hθ(x)=θ0+θ1x1+θ2x1^2 or the cubic function hθ(x)=θ0+θ1x1+θ2x1^2+θ3x1^3

In the cubic version, we have created new features x2 and x3 where x2=x1^2 and x3=x1^3. To make it a square root function, we could do: hθ(x)=θ0+θ1x1+θ2√x1.

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important. If x1 has range 1 - 1000 then range of x21 becomes 1 - 1000000 and that of x1^3 becomes 1 - 1000000000

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, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equation formula is given below:

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

$$X = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad y = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

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

There is no need to do feature scaling with the normal equation. 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^TX
Works well when n is large	Slow if n is very large

With the normal equation, computing the inversion has complexity O(n^3). So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.