2. Multivariate Linear Regression

In this section we begin to talk about a more powerful version of *linear regression* - one that works with multiple variables or with multiple features. Linear regression with multiple variables is also known as "multivariate linear regression"

In the original version of linear regression that we developed, we had a single feature x, *the size of the house*, and we wanted to use that to predict y, *the price of the house*.

Week 2: 5/20/17

Size (ft^2) (x)	Price ($\$\times 1000$)($y$)
2104	460
1416	232
1534	315
852	178
...	...

The form of our hypothesis is:

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

But now imagine if we not only had the size of the house, but we also knew the number of bedrooms, and the age of the home. This would give us a lot more information with which to predict the price.

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

Notation:

n = Number of features

m = Number of training examples

$x^{(i)}$ = Input (features) of the i^{th} training example

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

y = 'Output' variable / *target* variable

Now our hypothesis looks like this:

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

Or, more generally:

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

Now, for convenience, define $x_0 = 1$. Then, this can be written in vector form as:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}, \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1},$$

Then, we can write the *hypothesis function* in compact form:

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\ &= \theta^\intercal x \end{aligned}$$

2.1. **Gradient Descent: Multiple Variables.**

Now, extended this formulation to the gradient descent method, we recall our cost function:

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

Then, the gradient descent algorithm is as follows:

Gradient Descent:

Repeat until convergence: {

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

}

In which we must simultaneously update every $j = 0, \ldots, n$. Expanding on this algorithm:

Gradient Descent

(n = 1)

Repeat{

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

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

}

(n ≥ 1)

Repeat{

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

}

2.1.1. ***Feature Scaling***.

We can speed up the gradient descent method 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, which can 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: $-1 \leq x^{(i)} \leq 1$ or $-0.5 \leq x^{(i)} \leq 0.5$.

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.

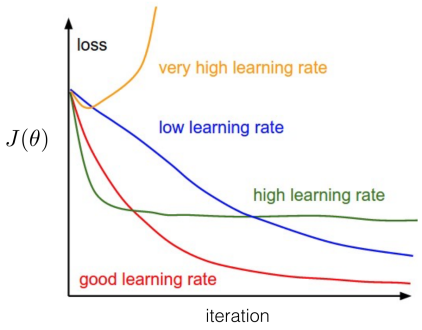
Two techniques to help with this are ***feature scaling*** and ***mean normalization***. Feature scaling involves dividing the input values by the range (*i.e. the maximum value minus the minimum value*) of the input variable, resulting in a new range of just 1. Mean normalization involves subtracting 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:

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

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

2.1.2. *Learning Rate.*

One of the things that has not been discussed yet is the *learning rate*, α - this term can make a big difference, particularly when debugging. To see the effect, it is best to plot the cost function vs. iterations. If $J(\theta)$ ever increases, then you probably need to decrease α.



For this problem, it can be shown that:

- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

2.1.3. *Features.*

Choosing the appropriate features can be a critical part of the solution. More specifically, we can combine multiple features into one. For example, we can combine x_1 and x_2 into a new feature x_3 by taking $x_1 \cdot x_2$.

Example 2.1 : *Features:* *Housing prices*

From the housing prices example, we have the following prices prediction:

$$h_\theta(x) = \theta_0 + \theta_1 \times \underbrace{frontage}_{x_1} + \theta_2 \times \underbrace{depth}_{x_2}$$

When applying linear regression we can create our own variables (features):

Area: $x = frontage \times depth$

Then, we have:

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

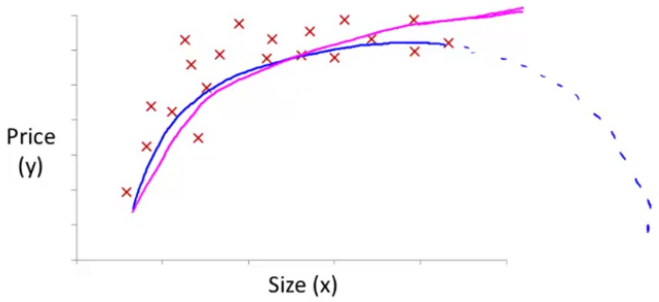
In which x is now the 'land area.'

2.2. **Polynomial Regression.**

Our hypothesis function need not be linear (a straight line) if that does not fit the data well. We can choose several types of functions.

Example 2.2 : *Polynomial Regression*

Given a dataset, 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):



- ***Linear:*** $h_\theta(x) = \theta_0 + \theta_1 x$
- ***Quad:*** $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
- ***Cubic:*** $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- ***Root:*** $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x_1}$

The generic form of our hypothesis is:

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

in which,

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

While this can be a very effective approach, one important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

2.3. **Normal Equations.**

which for some linear regression problems, will give us a much better way to solve for the optimal value of the parameters, θ.

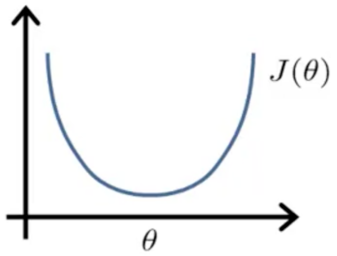
Thus far, the algorithm that we've been using for linear regression is *gradient descent* (GD) where in order to minimize the cost function $J(\theta)$, we use the iterative algorithm - which takes many steps and multiple iterations to converge to the global minimum.

In contrast, the *normal equation* gives us a method to solve for θ analytically, so that rather than needing to run this iterative algorithm, we can instead just solve for the optimal value for theta all at one go.

Example 2.3 : *Normal Equation*

Assume we have a quadratic function (1D, $\theta \in \mathbb{R}$), in which we wish to find the minimum:

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



Setting the derivative to zero:

$$\frac{d}{d\theta}J(\theta) = \cdots = 0.$$

$\rightarrow$ then we can solve for the minimum value.

However, in our case we are no longer in 1D, here $\theta \in \mathbb{R}^{n+1}$. Now we have:

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

Following the same procedure, we now find the partial derivative:

$$\frac{\partial}{\partial\theta_j} = \cdots = 0 \quad \text{(for every } j\text{)}$$

$\rightarrow$ then we solve for $\theta_0, \theta_1, \ldots, \theta_n$

The derivation ends up being somewhat involved and will not be covered here.

Example 2.4 : *Normal Equation:* *Apply to training set*

Given our running dataset:

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

We first add a row of ones, x_0, and place this into a matrix:

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}, \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Here, $X \in \mathbb{R}^{m\times(n+1)}$ and y is an m-dimensional vector.

Finally, we compute:

$$\theta = \left(X^\intercal X\right)^{-1} X^\intercal y$$

In the ***general case*** for m examples: $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(1m)})$; n features.

$$x = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}, \qquad \mathbb{X} = \begin{bmatrix} — & x^{(1)\intercal} & — \\ — & x^{(2)\intercal} & — \\ & \vdots & \\ — & x^{(n)\intercal} & — \end{bmatrix} \in \mathbb{R}^{n+1},$$

Example 2.5 : *One Feature:*

If we're given a feature vector of $x^{(i)}$:

$$x = \begin{bmatrix} 1 \\ x_1^{(2)} \end{bmatrix}$$

We can create a *design matrix* and y vector as follows:

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_2^{(2)} \\ 1 & \vdots \\ 1 & x_m^{(1)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

In which , $X \in \mathbb{R}^{m \times 2}$ and y is an m-dimensional vector. Finally, we compute:

$$\theta = \left(X^\intercal X\right)^{-1} X^\intercal y$$

2.3.1. ***Implementation Aspects.***

From the general equation:

$$\theta = \left(X^\intercal X\right)^{-1} X^\intercal y$$

If we focus on the first term: $\left(X^\intercal X\right)^{-1}$, we notice that this is the inverse of the matrix $X^\intercal X$. In practice, when computing these values, we:

$$\texttt{Set:} \quad A = X^\intercal X$$

$$\texttt{Then:} \quad A^{-1} = \left(X^\intercal X\right)^{-1}$$

In Matlab, this is computed as follows:

```
% compute inverse
pinv(X' * X)

% compute optimal theta
pinv(X' * X) * X' * y
```

Note: Using the normal equation method, *feature scaling* is generally not an issue.

2.4. **Summary.**

Now, what are the advantages and disadvantages of each method?

Gradient Descent vs. Normal Eqn.

Gradient Descent	**Normal Equation**
• Need to choose α	• No need to choose α
• Needs many iterations	• Don't need to iterate
• Works well even when n is large	• Need to compute $A = X^\intercal X \in \mathbb{R}^{n \times n} \mathcal{O}(n^3)$
	• Slow if n is very large

- The tipping point is around $n \approx 10^6$. Anything less, $n \leq 10^5$, the normal equations are useful.
- Also: Normal Eqns do not work with more sophisticated algorithms.

2.4.1. ***Non-invertability in Normal Eqn.***

Given:

$$\theta = \left(X^\intercal X\right)^{-1} X^\intercal y \qquad (\textit{Normal Equation})$$

Q: *What if $X^\intercal X$ is non-invertible? (singular / degenerate)*

This should happen pretty rarely, but we will cover it just in case. If implemented in Matlab, we use:

```
% compute theta
pinv(X' * X) * X' * y
```

This function `pinv` stands for *pseudo*-inverse and will compute θ, even if the matrix is not invertible. This is why `pinv` should be used, rather than `inv`.

If $X^\intercal X$ is non-invertible, the common causes might be having :

- ***Redundant features***, (*linearly dependent*) – where two features are very closely related.
  - *Ex:* $x_1 =$ size in feet^2, & $x_2 =$ size in m^2
- ***Too many features*** (*e.g.* $m \leq n$). In this case, 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.