Stanford Machine Learning - Week II

Eric N Johnson

July 19, 2016

1 Multi Variate Linear Regression

We can use multiple features to create a linear model as well.

1.1 Notation

- We will use $x_1, x_2...$ to denote *features*, and
- y to denote *response*.
- We additionally use n to denote the number of features, $x^{(i)}$ for the input features of the i^{th} training example and
- $x_j^{(i)}$ to denote the value of feature j in the i^{th} training example.

For example,

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

is the 2nd training example. Here $x_3^{(2)}$ will have the value 2.

1.2 Hypothesis

We will write our hypothesis like this:

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

For example:

$$h_\theta(x) = 80 + 0.1x_1 + 2x_2 + 3x_4 - x_5$$

We may simplify these expressions by setting $x_0^{(i)} = 1$, and we have

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{nx1}$$

and

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{nx1}$$

which makes writing our hypothesis much easier:

$$\begin{aligned} h_\theta(x) &= \theta_0 x_0 + \theta_1 x_1 + ... + \theta_n x_n \\ &= \theta_0(1) + \theta_1 x_1 + ... + \theta_n x_n \\ &= \theta^T x \end{aligned}$$

I.e., this is the same as

$$\begin{bmatrix}\theta_0 & \theta_1 & \cdots & \theta_n\end{bmatrix}\begin{bmatrix}x_0\\x_1\\\vdots\\x_n\end{bmatrix}$$

2 Gradient Descent for Multiple Variables

We have our hypothesis

$$h_\theta(x) = \theta^T x$$

with parameters

$$\theta = \begin{bmatrix}\theta_0\\\theta_1\\\vdots\\\theta_n\end{bmatrix} \in \mathbb{R}^{nx1}$$

and cost function

$$\begin{aligned} J(\theta) &= J(\theta_0, \theta_1, ..., \theta_n) \\ &= \frac{1}{2m}\sum_{i\leq m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 \end{aligned}$$

which may be written as

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(\left(\sum_{j=0}^{n}\theta_j x_j^{(i)}\right) - \left(\sum_{j=0}^{n} y_j^{(i)}\right)\right)^2$$

which is, in my opinion, much harder to read and interpret.

2.1 Gradient Descent Algorithm

We have our GD algorithm from before:

$$\begin{aligned} &\text{Repeat}\{ \\ &\quad \theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0, \theta_1, ...) \\ &\quad\quad := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta) \\ &\} \end{aligned}$$

and we simultaneously update all $j = 0, 1, ..., n$.

If we have $n \geq 1$, we need to write this a little differently:

$$\begin{aligned} &\text{Repeat}\{ \\ &\quad \theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} \\ &\quad\quad := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta) \\ &\} \end{aligned}$$

since we have to denote that very last term as the j^{th} element in the vector $x^{(i)}$. Again we simultaneously update θ_j for all $j = 0, 1, ..., n$.

Here are the first few terms of gradient descent for $j = 3$:

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

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

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

where we are still holding $x_0^{(i)} = 1$.

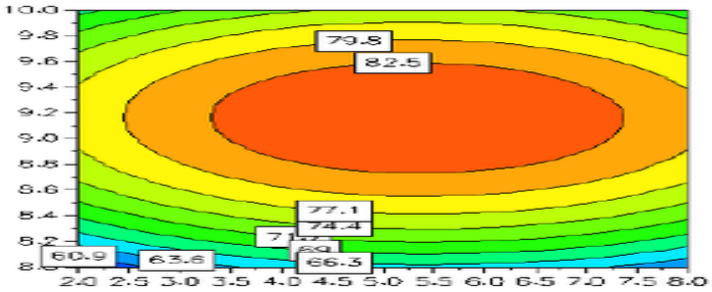
3 Gradient Descent in Practice I: Feature Scaling

Idea: Make sure features are on a similar scale. This causes Gradient Descent to converge more quickly.

Example:

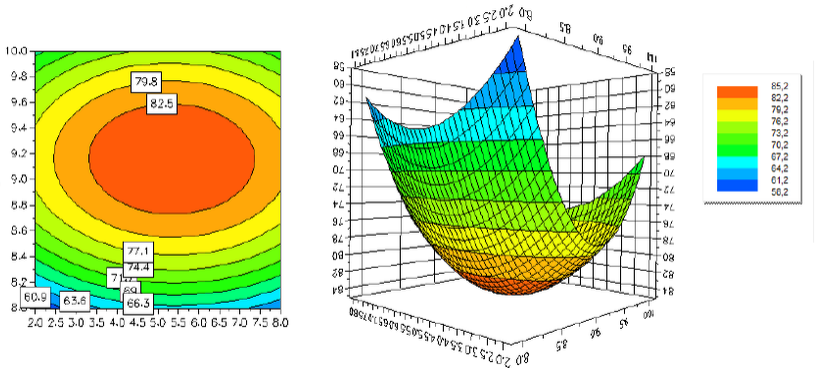
- Let x_1 = size (0 - 2000 ft^2)
- Let x_2 = number of bedrooms (say 1 - 5)...

This creates a skewered contour plot that slows down the algorithm because it tends to zig zag down to the minimum.



But we can rescale our data to make θ_1 and θ_2 more similar in scale.

Define $size = \frac{size(ft^2)}{2000}$. The algorithm will converge more quickly.



3.1 Feature Scaling Ranges

It is often good practice to scale so that we get approximately a

$$-1 \leq x_i \leq 1$$

range. It doesn't have to exactly this - but this does tend to work well in practice. One good way to scale is to divide by either the largest or smallest value in the data. A good rule of thumb is to be within $(-3, 3)$.

3.2 Mean normalization

Another strategy is to set

$$x_i \leftarrow x_i - \mu_i$$

so that features have mean ≈ 0. We don't want to apply this to our $x_0 = 1$ set.

For example, set

$$x_1 = \frac{\text{size} - 1000}{2000}$$

$$x_2 = \frac{\text{number of bedrooms} - 2}{5}$$

This will obtain something with mean zero that is nicely scaled:

$$-0.5 \leq x_1 \leq 0.5$$
$$-0.5 \leq x_2 \leq 0.5$$

We may also set

$$x_i \leftarrow \frac{x_i - \mu_i}{\sigma}$$

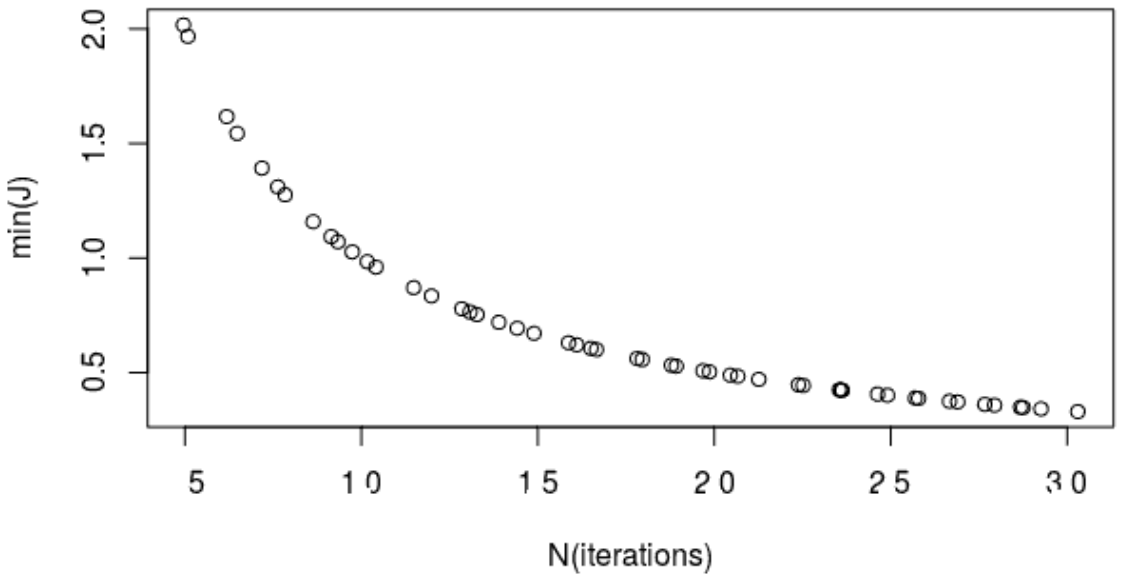
where σ = standard deviation of that particular feature.

4 Gradient Descent in Practive II: Verifying that it is working

A good way to verify that Gradient Descent is working correctly is to plot the cost function $J(\theta)$ vs. the number of iterations.

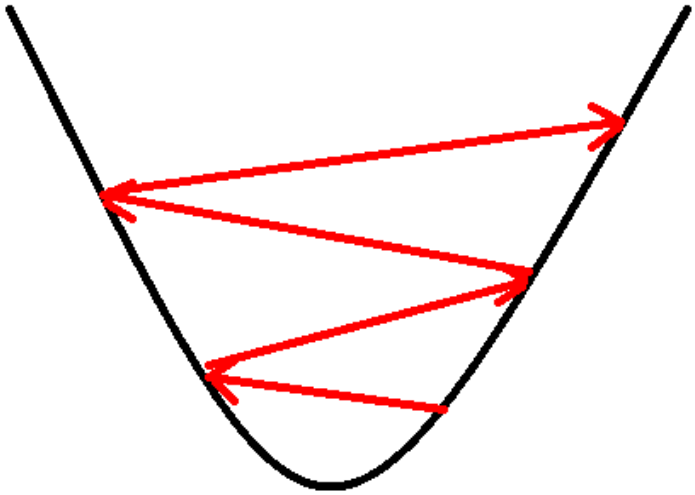
5 Gradient Descent in Practice III: Learning Rate α

- We should see the cost decrease monotonically.



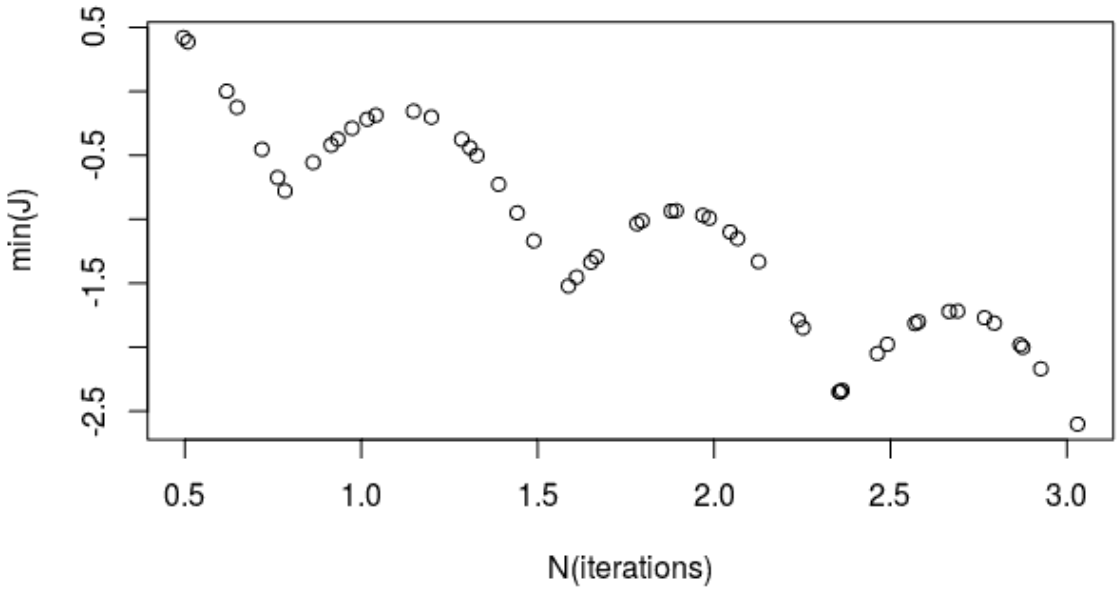
In general it is very difficult to see how many iterations will be required for convergence.

- If, however, $J(\theta)$ is increasing, Gradient Descent is not working.



Try a smaller α. The algorithm is probably overshooting.

- If $J(\theta)$ is oscillating like this:



Again, try a smaller α.

- If $J(\theta)$ is not doing anything, try a larger α.

5.1 Summary of Learning Rate

- It is good to plot $J(\theta)$ with different values of α vs. the number of iterations. Say $\alpha = 0.001, 0.01, 0.1, \ldots$ and look at how fast the cost function decreases.
- If α is too small, we get *slow convergence*.
- If α is too large, the cost function *may not decrease* on every iteration or *may not converge* at all.

6 Polynomial Regression

Note that we may choose to make new variables from features and do regression on those. For instance, if we have "frontage" and "depth", we may just choose to let x = frontage * depth.

We can create models that are quadratic:

$$h(\theta) = \theta_0 + \theta_1 x + \theta_2 x^2$$

or third degree (cubic):

$$h(\theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

or even make up something:

$$h(\theta) = \theta_0 + \theta_1 x + \theta_2 \ln(x)$$

To do so, we just define our features like this:

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

If we do, feature scaling becomes very important!

6.1 Feature Scaling Example

Suppose we wish to predict a house price using a function like this:

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

where the size is between 1 and 1000 ft^2. We will implement this by fitting a model

$$h(\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

where we scale using the following:

$$\begin{aligned} x_1 &= \frac{\text{size}}{1000} \\ x_2 &= \frac{\sqrt{\text{size}}}{\sqrt{1000}} \\ &= \sqrt{\frac{\text{size}}{1000}} \end{aligned}$$

so that $-1 \leq x_1 \leq 1$, and $-1 \leq x_2 \leq 1$.

7 Computing Parameters Analytically

We can solve for θ analytically using *normal equations*.

Here:

- 1 Dimensional: $\theta \in \mathbb{R}$

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

  We just need to minimize this quadratic by setting $J'(\theta) = 0$.

  $$\begin{aligned} J'(\theta) &= \frac{d}{d\theta}\left(a\theta^2 + b\theta + c\right) \\ &= 2a\theta + b \end{aligned}$$

  which allows us to solve for θ.

- $n+1$ Dimensional: $\theta \in \mathbb{R}^{n+1}$

  $$J(\theta_0, \theta_1, ..., \theta_m) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)} - y^{(i)}\right)^2$$

  We then set

  $$\frac{\partial}{\partial\theta_j}J(\theta) = 0 \quad \forall j$$

  and solve for $\theta_0, \theta_1, ..., \theta_m$.

- Example: Suppose we have the following four training examples.

Size (ft^2)	**Num Bedrooms**	**Num Floors**	**Age (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 let $x_0 = (1, 1, ..., 1)^T$,

	Size (ft^2)	**Num Bedrooms**	**Num Floors**	**Age (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

  which gives us this matrix X:

  $$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \in \mathbb{R}^{mx(n+1)}$$

  and vector y:

  $$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \in \mathbb{R}^{mx1}$$

  where m = the number of training examples, n is the number of features. To find the values of θ that minimize the cost function, solve

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

7.1 A General Procedure for Calculating the Minimum Cost Analytically

Suppose we have m examples $(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})$ with n features. We have a vector for each example

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

and we create a block matrix X (called a *Design Matrix*) made up of each $x^{(i)}$ as its rows:

$$X = \begin{bmatrix} - - (x^{(1)})^T - - \\ - - (x^{(2)})^T - - \\ \vdots \\ - - (x^{(m)})^T - - \end{bmatrix} \in \mathbb{R}^{mx(n+1)}$$

and create y based on the response variables in the training set. Then we find $\theta = (X^T X)^{-1} X^T y$.

For example, if $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$, we have

$$X = \begin{bmatrix} 1 & x_1^{(i)} \\ 1 & x_2^{(i)} \\ \vdots & \\ 1 & x_m^{(i)} \end{bmatrix} \in \mathbb{R}^{mx2}$$

and our y:

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^{mx1}$$

Finally to calculate the minimum, we use the *Normal Equations* and obtain

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

Another concrete example: Suppose we have the following training data.

Age x_1	**Height in cm** x_2	**Weight in kg** y
4	89	16
9	124	28
5	103	20

We then wish to predict y, based on this training data, with the model:

$$weight = \theta_0 + \theta_1 age + \theta_2 height$$

We use:

$$X = \begin{bmatrix} 1 & 4 & 89 \\ 1 & 9 & 124 \\ 1 & 5 & 103 \end{bmatrix}$$

$$\text{and } y = \begin{bmatrix} 16 \\ 28 \\ 20 \end{bmatrix}$$

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

7.2 More About the Normal Equation

We have

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

which can be completed in Octave using

```
pinv(X'*X)*X'*y
```

Recall that previously we talked about *Feature Scaling*. If we use the Normal Equations, we do not need to perform any scaling.

7.3 Gradient Descent vs. Normal Equation

Suppose we have m training examples and n features.

- Use Gradient Descent when...
  - Need to choose α
  - We need to perform many iterations
  - Works very well when we have a large number of features n
- Use the Normal Equation when...
  - We do not need to choose α
  - We do not need to iterate
  - We have to compute $(X^T X)^{-1}$ so this may be very slow if n is large. (Note the cost for this inversion is approximately $O(n^3)$)
  - This is slow if n is very large (as in 10,000 or more).
  - But if n is not very large, this method is very fast!

7.4 Normal Equation Noninvertibility

What if $X^T X$ is singular? This should be a rare occurance (*). In Octave, the command

```
pinv(X'*X)*X'*y
```

calls for a pseudo inverse. There is another Octave command called 'inv'. In most cases 'pinv' will work.

How do we get singular matrices out?

- If we have linearly dependent variables. For example, x_1 = square feet and x_2 = square meters.
- We have too many features. I.e., $m \leq n$. In this case we may delete some features or use *regularization*.