Week 2 Lecture Notes

Monday, May 3, 2021 9:36 AM

ML:Linear Regression with Multiple Variables

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.

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

Now define the multivariable form of the hypothesis function as follows, accommodating these multiple features:

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

In order to develop intuition about this function, we can think about θ_0 as the basic price of a house, θ_1 as the price per square meter, θ_2 as the price per floor, etc. x_1 will be the number of square meters in the house, x_2 the number of floors, etc.

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

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

This is a vectorization of our hypothesis function for one training example; see the lessons on vectorization to learn more.

Remark: Note that for convenience reasons in this course Mr. Ng assumes $x_0^{(i)} = 1 \text{ for } (i \in 1, \dots, m)$

[**Note**: So that we can do matrix operations with theta and x, we will set $x_0^{(i)}$ = 1, for all values of i. This makes the two vectors 'theta' and $x_{(i)}$ match each other element-wise (that is, have the same number of elements: n+1).]

The training examples are stored in X row-wise, like such:

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

You can calculate the hypothesis as a column vector of size (m x 1) with:

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

For the rest of these notes, and other lecture notes, X will represent a matrix of training examples $x_{(i)}$ stored row-wise.

Cost function

For the parameter vector θ (of type $\mathbb{R}^{n+1}$ or in $\mathbb{R}^{(n+1)\times 1}$, the cost function is:

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

The vectorized version is:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

Where $\vec{y}$ denotes the vector of all y values.

Gradient Descent for Multiple Variables

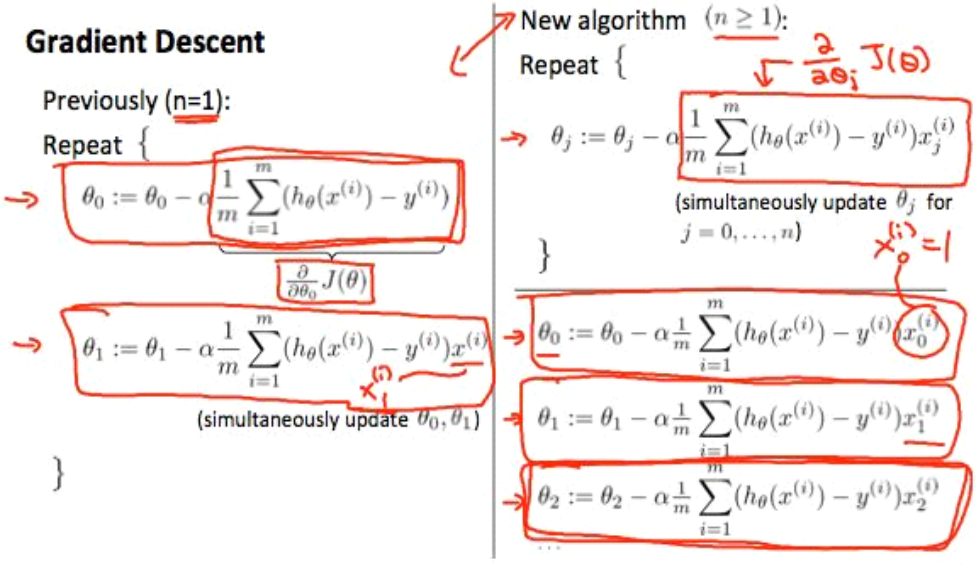
The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

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

In other words:

$$\begin{aligned}
&\textbf{repeat until convergence: } \{\\
&\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\cdot x_j^{(i)} \qquad \textbf{for j} := \textbf{0..n}\\
&\}
\end{aligned}$$

The following image compares gradient descent with one variable to gradient descent with multiple variables:



Feature Normalization

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:

$-1 \leq x_{(i)} \leq 1$

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 s_i is the standard deviation.

Note that dividing by the range, or dividing by the standard deviation, give different results. The quizzes in this course use range - the programming exercises use standard deviation.

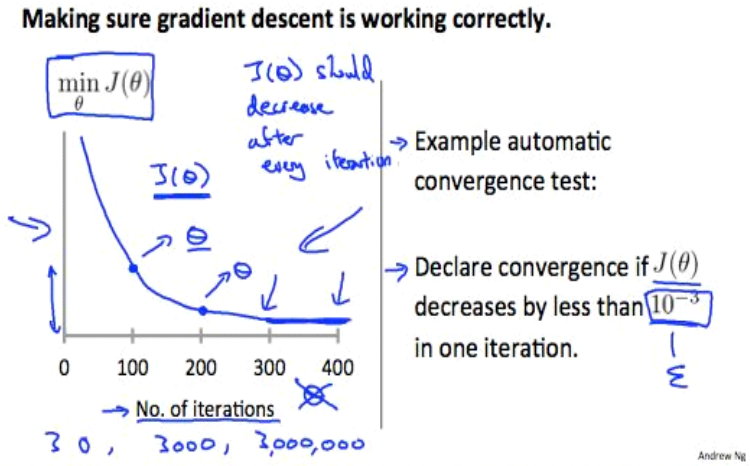
Example: x_i is housing prices with range of 100 to 2000, with a mean value of 1000. Then, $x_i := \frac{price - 1000}{1900}$.

Gradient Descent in Practice II - Learning Rate

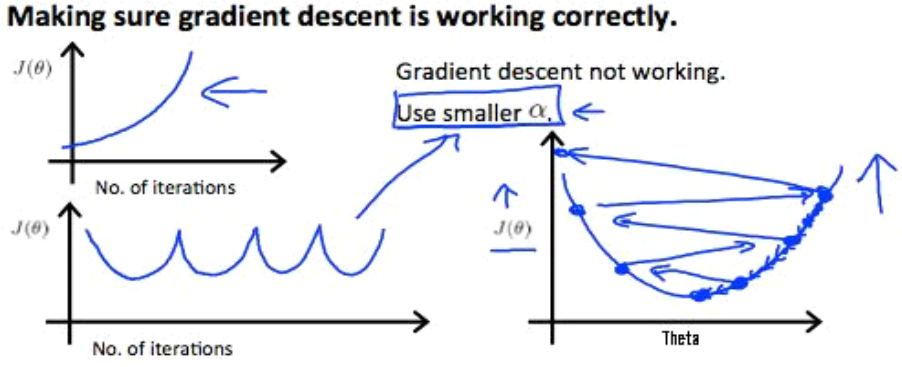
Note: [5:20 - the x -axis label in the right graph should be θ rather than No. of iterations]

Debugging gradient descent. Make a plot with *number of iterations* on the x-axis. Now plot the cost function, J(θ) over the number of iterations of gradient descent. If J(θ) ever increases, then you probably need to decrease α.

Automatic convergence test. Declare convergence if J(θ) decreases by less than E in one iteration, where E is some small value such as 10^{-3}. However in practice it's difficult to choose this threshold value.



It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.



To summarize:

If α is too small: slow convergence.

If α is too large: may not decrease on every iteration and thus may not converge.

Normal Equation

The "Normal Equation" is a method of finding the optimum theta **without iteration.**

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

There is **no need** to do feature scaling with the normal equation.

Mathematical proof of the Normal equation requires knowledge of linear algebra and is fairly involved, so you do not need to worry about the details.

Proofs are available at these links for those who are interested:

https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)

http://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression

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

With the normal equation, computing the inversion has complexity $\mathcal{O}(n^3)$. 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.

Derivation of the Normal Equation for linear regression

Tuesday, May 4, 2021 11:11 AM

First, some terminology. The following symbols are compatible with the machine learning course, not with the exposition of the normal equation on Wikipedia and other sites - semantically it's all the same, just the symbols are different.

Given the hypothesis function:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

We'd like to minimize the least-squares cost:

$$J(\theta_{0\ldots n}) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Where

$x^{(i)}$ is the `i`-th sample (from a set of `m` samples) and

$y^{(i)}$ is the `i`-th expected result.

To proceed, we'll represent the problem in matrix notation; this is natural, since we essentially have a system of linear equations here. The regression coefficients

θ we're looking for are the vector:

$$\begin{pmatrix} \theta_0 \\ \theta_1 \\ \cdots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}$$

Each of the `m` input samples is similarly a column vector with `n+1` rows,

x_0 being 1 for convenience. So we can now rewrite the hypothesis function as:

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

When this is summed over all samples, we can dip further into matrix notation. We'll define the "design matrix" `X` (uppercase X) as a matrix of `m` rows, in which each row is the `i`-th sample (the vector $x^{(i)}$). With this, we can rewrite the least-squares cost as following, replacing the explicit sum by matrix multiplication:

$$J(\theta) = \frac{1}{2m} (X\theta - y)^T (X\theta - y)$$

Now, using some matrix transpose identities, we can simplify this a bit. I'll throw the $\frac{1}{2m}$ part away since we're going to compare a derivative to zero anyway:

$$J(\theta) = ((X\theta)^T - y^T)(X\theta - y)$$

$$J(\theta) = (X\theta)^T X\theta - (X\theta)^T y - y^T (X\theta) + y^T y$$

Note that $X\theta$ is a vector, and so is `y`. So when we multiply one by another, it doesn't matter what the order is (as long as the dimensions work out). So we can further simplify:

$$J(\theta) = \theta^T X^T X\theta - 2(X\theta)^T y + y^T y$$

Recall that here θ is our unknown. To find where the above function has a minimum, we will derive by θ and compare to 0. Deriving by a vector may feel uncomfortable, but there's nothing to worry about. Recall that here we only use matrix notation to conveniently represent a system of linear formulae. So we derive by each component of the vector, and then combine the resulting derivatives into a vector again. The result is:

$$\frac{\partial J}{\partial \theta} = 2X^T X\theta - 2X^T y = 0$$

Or:

$$X^T X\theta = X^T y$$

Now, assuming that the matrix

$X^T X$

is invertible, we can multiply both sides by

$(X^T X)^{-1}$

and get:

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

Which is the normal equation.

Normal 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 $X^T X$ is not invertible.

If $X^T X$ 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.