Linear Regression with One Variable

Model Representation

Recall that in *regression problems*, we are taking input variables and trying to map the output onto a *continuous* expected result function.

Linear regression with one variable is also known as "univariate linear regression."

Univariate linear regression is used when you want to predict a **single output** value from a **single input** value. We're doing **supervised learning** here, so that means we already have an idea what the input/output cause and effect should be.

The Hypothesis Function

Our hypothesis function has the general form:

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

We give to hθ values for θ0 and θ1 to get our output 'y'. In other words, we are trying to create a function called hθ that is able to reliably map our input data (the x's) to our output data (the y's).

Example:

x (input)	y (output)
0	4
1	7
2	7
3	8

Now we can make a random guess about our h_θ function: $\theta_0 = 2$ and $\theta_1 = 2$. The hypothesis function becomes $h_\theta(x) = 2 + 2x$.

So for input of 1 to our hypothesis, y will be 4. This is off by 3.

Cost Function

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's compared to the actual output y's.

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

To break it apart, it is $\frac{1}{2}\,\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x^{(i)}) - y^{(i)}$, or the difference between the predicted value and the actual value.

This function is otherwise called the "Squared error function", or Mean squared error (https://eventing.coursera.org/api/redirectStrict/MVjwzk5RpNaUcf1rW5-_6NY1pjlm_UJznzHevNCMgC1Jr5Uvjh-O2fgxuczU-PyXOxiQJ7i5OCwpbNHQw_PdJA.t-2TIBWqOU_j7GK-dfR2xg.NZ16TXMdVf41ahQiiqgrx2yAiKZk5MvTkxC-fsUy_XFts707s9IPlXLf_rjC78UyLKQf5fAkdesJgEZnpzeCaC31uIp7A4yUrb6Se418fBLi6albYD62XrIlqBEC_5g3mV2QR6B9q0ZH_EOhWOajKmePWgOmSrwGYR7vl4A2-C4a8DRRNL2gZym3yW6RvWOvbqWGN9-8h4-ba2t4Nk1zWL6IzqN-ElKCTm67e6qPsg5KYJwe9G7zXyLeeEbsHM0MEXDytPaMrVglCP2kO5YEHmI7sLoXREe-yhAhX9FZ470hWozBve2jl-3j3oFbpggNCJadZUciWjya9Er7hNlMjCZ4M1zIEWT0Ygi629PDDHMsBadOjYWdUCeD0l5NXffDbq5Yu7jdxNezFzIgqHacDw). The mean is halved ($\frac{1}{2m}$) as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $\frac{1}{2}$ term.

Now we are able to concretely measure the accuracy of our predictor function against the correct results we have so that we can predict new results we don't have.

Gradient Descent

So we have our hypothesis function and we have a way of measuring how accurate it is. Now what we need is a way to automatically improve our hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields θ_0 and θ_1 (actually we are graphing the cost function for the combinations of parameters). This can be kind of confusing; we are moving up to a higher level of abstraction. We are not graphing x and y itself, but the guesses of our hypothesis function.

We put θ_0 on the x axis and θ_1 on the z axis, with the cost function on the vertical y axis. The points on our graph will be the result of the **cost function** using our hypothesis with those specific theta parameters.

We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.

The way we do this is by taking the **derivative** (the line tangent to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down that derivative by the parameter α, called the learning rate.

The gradient descent equation is:

repeat until convergence:

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$

for j=0 and j=1

Intuitively, this could be thought of as:

repeat until convergence:

$\theta_j := \theta_j - \alpha [\text{Slope of tangent aka derivative}]$

Gradient Descent for Linear Regression

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to (the derivation of the formulas are out of the scope of this course, but a really great one can be found here (https://eventing.coursera.org/api/redirectStrict/lUmrIluAfLls_l_UHMQBfhN10K0BhqDfOulXt-Z6A6vjWCJT7ZRqHAQXgGs9iORMm_TLDzA6irhG6tSornPq_A.cwn0kjGdX4-mbf-4XU-Q0Q._zvJRq0YMyAg-N_K33RyqxXdukEppmy6VPC_ht8PY7aM1pC8Tm7XJs5ZQFJeAr_xouo0tS6rutjBeSlWfaAgCqw_sUC5pdxMOgLqpGh38dF0yPsLIp9xC73jBovWmujtSxuVlqib-rSyrKOuSWiVkk8TgjKcphf1kjdvYDl5OTmAApFMvNLzx_BODmRkiejLXS-fueWx6-CXpA64jkl7MaoJiPsjbzvuHDEcPXTyJxlfxe07SStZ7ewCBC-2yXfKvE0Y0P5emBXSiWzN6RNY-KhHX3DKa-fam_tz00DMx5YsCFcLmfmDjn2AAYvmYxVsOxKNgldgwpRDw3mGLodaWu-mRTBQsUIg7w7Uh_AMKfXwMZm7BwbY176QC7eiUDTC39QiTxDlEEfzz9CpdPGTJitohJly_PE5F-q4y6lVIFvykr9bvQgH2ZJHZDnFJBlie3MTq4c_8pIGQlkjv9yTbPlHKvRMxUFsuJCTOUcQhp7_ZThKEsKV-06bvd2qW-DJ2xjE_sY-lKYoI7OMbgnxFL6o8NHkx911NpTqTAHMdYQAme4egRqfRAKqLiAINtbZCZLmHqzt4DEdfNQoBRlPEA):

$$\text{repeat until convergence: } \{$$

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

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

$$\}$$

where m is the size of the training set, θ_0 a constant that will be changing simultaneously with θ_1 and $x^{(i)}, y^{(i)}$ are values of the given training set (data).

Note that we have separated out the two cases for θ_j and that for θ_1 we are multiplying $x^{(i)}$ at the end due to the derivative.

The point of all this is that if we start with a guess for our hypothesis and then repeatedly

apply these gradient descent equations, our hypothesis will become more and more accurate.

What's Next

Instead of using linear regression on just one input variable, we'll generalize and expand our concepts so that we can predict data with multiple input variables. Also, we'll solve for θ_0 and θ_1 exactly without needing an iterative function like gradient descent.

‹ (/learn/machine-learning/supplement/X64SM/introduction) | › (/learn/machine-learning/supplement/NMXXL/linear-algebra-review)

(https://accounts.coursera.org/i/zendesk/courserahelp?return_to=https://learner.coursera.help/hc)