share.coursera.org

ML:Linear Regression with One Variable

Recall that in *regression problems*, we are taking input variables and trying to fit 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 y from a **single input** value x. We're doing **supervised learning** here, so that means we already have an idea about what the input/output cause and effect should be.

Our hypothesis function has the general form:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

Note that this is like the equation of a straight line. We give to $h_\theta(x)$ values for θ_0 and θ_1 to get our estimated output $\hat{y}$. In other words, we are trying to create a function called h_θ that is trying to map our input data (the x's) to our output data (the y's).

Example:

Suppose we have the following set of training data:

input	output
x	**y**
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. Note that we will be trying out various values of θ_0 and θ_1 to try to find values which provide the best possible "fit" or the most representative "straight line" through the data points mapped on the x-y plane.

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.

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

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". The mean is halved $\left(\frac{1}{2m}\right)$ 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.

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make straight line (defined by $h_\theta(x)$) which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all the points of our training data set. In such a case the value of $J(\theta_0, \theta_1)$ will be 0.

Q: Why is the cost function about the sum of

squares, rather than the sum of cubes (or just the $(h(x) - y)$ or $abs(h(x) - y)$) ?

A: It might be easier to think of this as measuring the distance of two points. In this case, we are measuring the distance of two multi-dimensional values (i.e. the observed output value y_i and the estimated output value $\hat{y}_i$). We all know how to measure the distance of two points (x_1, y_1) and (x_2, y_2), which is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. If we have n-dimension then we want the positive square root of $\sum_{i=1}^{n} (x_i - y_i)^2$. That's where the sum of squares comes from. (see also Euclidean distance)

The sum of squares isn’t the only possible cost function, but it has many nice properties. Squaring the error means that an overestimate is "punished" just the same as an underestimate: an error of -1 is treated just like +1, and the two equal but opposite errors can’t cancel each other. If we cube the error (or just use the difference), we lose this property. Also in the case of cubing, big errors are punished more than small ones, so an error of 2 becomes 8.

The squaring function is smooth (can be differentiated) and yields linear forms after differentiation, which is nice for optimization. It also has the property of being “convex”. A convex cost function guarantees there will be a global minimum, so our algorithms will converge.

If you throw in absolute value, then you get a non-differentiable function. If you try to take the derivative of abs(x) and set it equal to zero to find the minimum, you won't get any answers since it's undefined in 0.

Q: Why can’t I use 4th powers in the cost function? Don’t they have the nice properties of squares?

A: Imagine that you are throwing darts at a dartboard, or firing arrows at a target. If you use the sum of squares as the error (where the center of the bulls-eye is the origin of the coordinate system), the error is the distance from the center. Now rotate the coordinates by 30 degree, or 45 degrees, or anything. The distance, and hence the error, remains unchanged. 4th powers lack this property, which is known as “rotational invariance”.

Q: Why does 1/(2 * m) make the math easier?

A: When we differentiate the cost to calculate the gradient, we get a factor of 2 in the numerator, due to the exponent inside the sum. This '2' in the numerator cancels-out with the '2' in the denominator, saving us one math operation in the formula.

Next: Gradient Descent Back to Index: Main