Supervised Learning: Linear Regression

April 7, 2021

1 Motivation

Imagine you are trying to predict the prices of houses. The only data you have available to you are the prices of a bunch of houses, and the square footage of all the houses. If given another house, alongside the square footage of said house, can you predict the price of the house?

If we assume that increasing square footage is approximately proportional to the price by a constant (the data is approximately linear), we can plot each point $P_i = (x, y) = (sqft, \$\$)$ in a scatter plot and find a line of best fit to the data.

2 Fitting a Line

For any set of data *assuming the data is approximately linear*, we can use a linear model to predict the output of any new data that comes in. As a reminder, the linear model we are working with (for now) is:
$\hat{y} = \beta_1 x + \beta_2$
where
$\hat{y}$:= Predicted output of the linear model.
Note that if we increase β_1, we rotate the line CCW, if we decrease β_1, we rotate the line CW, and increasing/decreasing β_2 pulls up/down the line.

3 Moving a Line Towards a Point

Let's assume the line we have to work with is: $y(x) = \beta_1 x + \beta_2$ ($y(x)$ will be treated as y unless explicitly used as a function), where β_1, β_2 are arbitrary constants, like a really bad guess as to what the actual line would be. Place a point reasonably far away from the line, $P = (p, q)$. How do we, using the data that we have, move the line such that it gets closer to the point?

3.1 Absolute Trick

The Absolute Trick is as such: Every iteration, we change β_1 by the value p (the x coordinate), and change the value β_2 by the value 1 (we will get to cases where we increase/decrease in a second). So our model transforms to:
$y = \beta_1 x + \beta_2 \rightarrow y = (\beta_1 \pm p)x + (\beta_2 \pm 1)$, where each $\pm$ is independent of one another
If we continue under the assumption that the point is far away, we might get a good approximation. In all likelihood, however, we are likely to overshoot. So instead we iterate over this multiple times, taking steps towards the right solution. We can do this by multiplying p and 1 by α, the learning rate. So our formula becomes:
$y = \beta_1 x + \beta_2 \rightarrow y = (\beta_1 \pm p\alpha)x + (\beta_2 \pm \alpha)$
To determine whether to use + or -, we need to look at how we want the line to move. For β_1, we see which direction (CCW or CW) we should rotate the line to reach the point (smallest angle to sweep across so the line meets the point). For β_2, do the same thing, but this time your distance is up/down.

3.2 Square Trick

The above trick works to get near to the point, but we never factor in the y distance from the line (other than to determine the sign). What if we adjust β_1 and β_2 the same way as above, but also account for the vertical distance from the point to the line? We can do this by multiplying our $p\alpha$ and α term by $q - \hat{q}$, where $\hat{q} = y(p)$. So our formula for the square trick becomes:
$y = \beta_1 x + \beta_2 \rightarrow y = (\beta_1 + p\alpha(q - \hat{q}))x + (\beta_2 + \alpha(q - \hat{q}))$
Note that we don't need to use $\pm$ anymore because multiplying by $q - \hat{q}$ means that if our point is below our line, this difference will yield a negative number, and if the point is above our line, the difference will yield a positive number.

3.3 Why not just do $\frac{q}{p}x$ as our line?

Obviously these tricks are pretty bad standalone operations, but their power is in the fact that they aren't exact. When we do linear regression, we're modifying our line of best fit so it coincides with all the data, and, unless all of our data points are co-linear (HIGHLY unlikely!), we want our line of best fit to be close to all of our data points. Think of each point pulling towards a line, the farther away the point from the line, the harder it pulls the line towards it.

4 What is the Line of Best Fit

Now that we have an iterative approach to pulling a line closer to a point, we need to define what a line of best fit is. If we define error (roughly) as the difference between our predicted y for each x value in our data, $\hat{y} = y(x)$, and the actual y value from each data point in our value, then our line of best fit is such that $\sum f(|y - \hat{y}|)$, such that f is a monotonic function, is at a minimum.

5 Linear Least Squares

Turns out, finding the exact line of best fit for a set of points (x_i, y_i) is an already solved problem, with a nice, closed form solution. Let our linear model be $\beta_1 x + \beta_0 = y$. If we consider the linear system:
$X\beta = y$
Where:
$X :=$ the overdetermined matrix $\begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$
$y :=$ the vector of y values associated with each x in the data
$\beta :=$ the vector of beta constants in the linear system (from this model, it's β_1 and β_0)
We have an overdetermined system that does not have exact solutions for β_1, β_0. Since we need a square matrix were X is, so we can solve it, we can multiply both sides by X^T to get:
$X^T X\beta = X^T y$
Solving for β, we get:
$\beta = (X^T X)^{-1} y$, where β is guaranteed to be such that our model $\beta_1 x + \beta_0 = y$ minimizes the error.

Unfortunately, this linear least squares approach is rarely used. This is because the formula $\beta = (X^T X)^{-1} y$ means that we need to calculate the inverse of a matrix, which is an $O(n^3)$ operation.

Since we can't solve this exactly, we need to get an approximation of our answer via iterative methods.

6 Error Functions

Since our method must be iterative and will not give us the exact answer, it would be useful to have an idea of what our error for any given model is.

6.1 Mean Approximation Error (MAE)

MAE $= \frac{1}{m}\sum|y_i - \hat{y}(x_i)$
We use absolute value so we do not have negative differences accidentally canceling positive ones. It's also a metric.

6.2 Mean Squared Error (MSE)

MSE $= \frac{1}{2m}\sum(y_i - \hat{y}(x_i))^2$
The added $\frac{1}{2} is a constant added for convienience.$

7 Gradient Descent

Since our goal is to minimize error, why not just change β_0, β_1 in the direction of the greatest decrease of the error function, multiplied by a small number (so we don't overshoot)? That is, every iteration,
$\beta_i = \beta_i - \alpha\frac{\partial}{\partial\beta_i}Error = \beta_i - \alpha\frac{\partial Error}{\partial y_i}\frac{\partial y_i}{\partial x_i}$
If we apply this to our MAE and MSE square functions, we essentially get the Absolute and Square Trick, respectively

7.1 Types of Gradient Descent

Batch - compute the average change in β using all data points, update β, and iterate over that Stochastic - compute the change in β using one data point, updating β, then do another, etc. Mini-Batch - Split your data into small, even groups, and compute the change, and iterate over that.

Batch and Stochastic are pretty slow, so we do mini batch typically

Algorithm 1 Gradient Descent- Mini Batch with MSE

procedure STEP(X, β, y, b, α)

▷ X the matrix containing loaded input data, β the current values of our model's coefficients, b current value of the regression intercept (β_0), α the step size.

$\hat{y} = XW + b$ $error = y - \hat{y}$

$W+ = \alpha \times error \times X$ ▷ error * X is the $b+ = \alpha * \sum error_i$

end procedure

8 Generalizing to Higher Dimensions

For an input vector $\vec{x} = x_1, ..., x_n$, we get an $n-1$ dimensional hyperplane.

9 Warnings

10 Polynomial Regression

11 Regularization

12 Feature Scaling

13 External Resources