Linear Regression

Boyan Dafov

April 5, 2020

I Solving the 1-D problem

First we are going to define and solve the linear regression problem for one independant and one dependant variable. We are looking for the "best fitting line" for random 2D data points. The equation we are looking to build looks like that:

$$\hat{y} = a * x + b$$

Where "y" and "x" are given for every data point. Lets the fine the "Error" function, assuming that we have "n" data points given :

$$E = \sum_{i=1}^{n} (\hat{y_i} - y_i)^2, \text{ where } \hat{y_i} \text{ is the prediction for } x_i$$

Now we are looking for the model, that gives us the minimum "Error function" result. First we are going to use that:

$$\hat{y_i} = a * x_i + b$$

Substitute it in the "Error function" above and get :

$$E = \sum_{i=1}^{n} (a * x_i + b - y_i)^2, \text{ where we are given } y_i \text{ and } x_i \text{ for every i}$$

To find the values for "a" and "b", we are going to take the derivatives of the function with respect first to "a" and than to "b":

$$\frac{\partial E}{\partial a} = \sum_{i=1}^{n} 2 * (y_i - (a * x_i + b)) * (-x_i)$$

$$\frac{\partial E}{\partial b} = \sum_{i=1}^{n} 2 * (y_i - (a * x_i + b)) * (-1)$$

Now we are going to simplify and set them to 0 and get the following system with two equations and two unknowns :

$$a * \sum_{i=1}^{n} x_i^2 + b * \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i * x_i$$

$$a * \sum_{i=1}^{n} x_i + b * n = \sum_{i=1}^{n} y_i$$

Let $\sum_{n=1}^{n} x_i^2 = c$, $\sum_{n=1}^{n} x_i = d$, $\sum_{n=1}^{n} x_i * y_i = e$, $\sum_{n=1}^{n} y_i = f$ and get the following system :

$$\begin{cases} a * c + b * d = e \\ a * d + b * n = f \end{cases}$$

There are many ways to solve it for both "a" and "b", but as a result we get $b = \frac{e*d - f*c}{d^2 - n*c}$ and $a = \frac{e*n - f*d}{n*c - d^2}$

II Solving the multidimensional problem

We are given set of pairs - vector of independant variables and dependant (prediction) variable $\{(x_1, y_1), (x_2, y_2), \dots\}$. Lets define the letter d - dimensionality. Our model for the multidimensional solution should look like that :

$$\hat{y} = w^T * x + b$$

Keep in mind that "w" should be "d" - dimensional vector, x is "DxN" matrix, where N - number of samples, D - number of features (dimensionality). We make the following transformations :

$\hat{y} = w^T * x + b \Leftrightarrow \hat{y} = b + w_1 * x_1 + \dots + w_d * x_d$, let $b = w_0$ and $x_0 = 1$ and get :

$$\hat{y} = w_0 * x_0 + w_1 * x_1 + \dots + w_d * x_d \implies \hat{y} = \hat{w}^T * \hat{x}$$

We define the problem like that, because it makes the calculation of multiple predictions clearer (and more optimal when it comes to computation, especially if you use library that supports matrix arithmetics like python's numpy).
Now we are going to define the "Error" function :

$$E = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 = \sum_{i=1}^{n} (y_i - \hat{w}^T * x_i)^2$$

We still can take the derivative with respect to any component of "w" - $\hat{w_j}$, where $j = 1, \dots, d$. Let's see how does the derivative $\frac{\partial E}{\partial w_j}$ look like:

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{n} 2 * (y_i - w^T * w_j) * (-\frac{\partial w^T * x_i}{\partial w_j}) = \sum_{i=1}^{n} 2 * (y_i - w^T * x_i) * (-x_{ij}) = \sum_{i=1}^{n} y_i * (-x_{ij}) - \sum_{i=1}^{n} w^T * x_i * (-x_{ij})$$

Since $j = 1, \dots, d$, we set all derivatives to 0 and get "d" equations with "d" unknowns

$$\begin{aligned} \sum_{i=1}^{n} y_i * (-x_{i1}) &= \sum_{i=1}^{n} w^T * x_i * (-x_{i1}) \\ &\dots \\ &\dots \\ &\dots \\ \sum_{i=1}^{n} y_i * (-x_{id}) &= \sum_{i=1}^{n} w^T * x_i * (-x_{id}) \end{aligned}$$

We have that $a^T * b = \sum_{i=1}^{n} a_i * b_i \implies w^T * (X^T * X) = y^T * X$, we transpose the equation :

$[w^T (X^T * X)]^T = [y^T * X]^T \implies (X^T * X) * w = X^T * y \implies w = (X^T * X)^(- 1) * X^T * y$

III R-squared value

We are going to define the R-squared value, which is used to determine how good our model is :

$$R^2 = 1 - \frac{SSres}{SStot}, \text{ where}$$

$$SSres = \sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

$$SStot = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

Where $\bar{y}$ is the mean value of all y_i given. When R^2 is closer to 1, our SSres is closer to 0, which means that our model is good. When R^2 is closer to 0, it means that our model predicted just the mean value of all y_i.

IV L1 and L2 Regularization

I am not going to go into the mathematical details about that, but there is something very common in ML, called "overfitting". In general overfitting is when your model matches the datapoints so well, that it looses the ability of making good predictions on new datapoints. That's why we want noise data to influence the output. The are two very common ways of approaching this problem in linear regression and they are L1 and L2 regularizations. Both of them are updating the "Error function" and adding an extra penalty to it. We will look at both of them and than we are going to see what are the differences between them.

IV.1 L1 Regularization (Lasso Regression)

In L1 Regularization, the "Error function" looks like that :

$$E = \sum_{i=1}^{n}(y_i - \hat{y_i})^2 + \lambda * \sum_{i=1}^{p}|w_i|$$

You can try to find the optimal weights by taking the derivatives but you will find out that it is not possible (i won't cover this here, but you can try it yourself). That's why you have to apply gradient descent for the solution. The idea of L1 Regularization is to select a small number of important features, that will be used for the predictions calculation. After you apply L1 to your model you will get few non-zero weights and the others are going to be equal to zero. This comes from the fact that the derivative of the penalty term in L1 is absolute function (+1, -1, 0). After performing gradient descent we will se that after the derivative gets 0, it will stop changing the weights "w".

IV.2 L2 Regularization (Ridge Regression)

In L2 Regularization, the "Error function" looks like that :

$$E = \sum_{i=1}^{n} (y_i - \hat{y_i})^2 + \lambda * \sum_{i=1}^{p} w_i^2$$

The L2 Regularization penaltizes very large and disproportional weights. Let's take the derivative of L2 in order to perform gradient descent: