Linear Regression

Mohammed Luqmaan Akhtar

October 2, 2025

Suppose that we are given the weights and sizes of several mice. Our task is to predict the size of a mouse (whose data was not provided) based on its weight. We achieve this using linear regression.

Fitting line to data

Our objective in linear regression is to find the line that best fits the data, i.e., the sum of the squared residuals from the line should be minimized. A residual is the difference between an observed (actual) data point and the value predicted by a statistical model. Suppose that the line is $y = wx + b$, we need to find the optimal values of w and b for which the sum of the squared residuals is minimum. Now, how can this be achieved? Conceptually, we can think of it like this: we plot the value of the sum of squared residuals for different lines and choose the line for which the value is minimum or for which the slope of the graph of sum of squared residuals vs line is zero. Even if the value of y depends on multiple parameters, we apply the same concept.

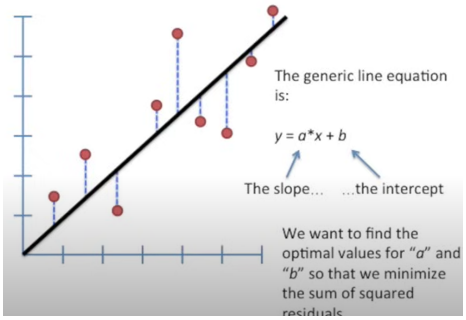

Figure 1:

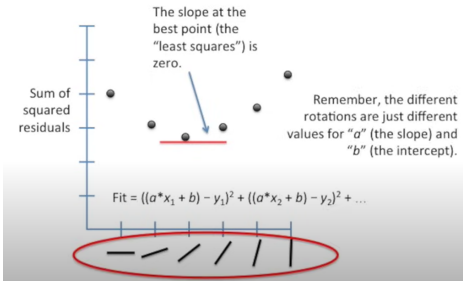

Figure 2:

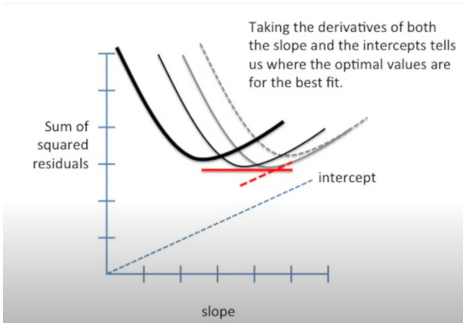

Figure 3:

Process of reaching final logic

It took a lot of experimentation and trial and error to finally reach the working version of my code. I will try to explain the ideas as thoroughly as possible.

- After learning about the basics of linear regression and the sum of squared residuals, the first idea that came to mind was to use nested loops to try different values of w & b and choose the one that gave the least sum of squared residuals. This idea was quickly discarded as I realized that the actual least value might not even be achieved from the values in the loop range and that it was practically impossible when performing multiple regression.
- After realizing my mistake, I learned that the least value has to be found using differentiation. I thought the numpy and pandas libraries would allow me to differentiate, so I started watching a tutorial on it. As I neared the end of the tutorial I realized differentiation cannot be done with these libraries. I tried to look for other libraries but understood that this was not the correct method to go about writing the code.
- I started reading articles on linear regression in the context of machine learning and came across **gradient descent**. Gradient descent was mentioned in the task as well but I had treated it as a topic independent of linear regression until now. I read an article on gradient descent and finally understood the method of writing the code.

Mathematics behind the code

- The data is in the form of a table. One column contains the value which needs to be predicted and the other columns contain the parameters the predicted value depends on.
- Let there be m different parameters. Let the value to be predicted be y. Therefore, the equation for linear regression can be written as

$$y_{pred} = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_mx_m + b \tag{1}$$

  where $x_1, x_2, \ldots, x_m$ are the parameters y depends upon, $w_1, w_2, \ldots, w_m$ are the weights and b is the bias.
- Mean Square Error(MSE) is the mean of the sum of the squared residuals or the mean of the sum of error squared. MSE is the value that needs to be minimized so as to get the optimized values of the weights and bias.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_{pred_i} - y_i)^2 \tag{2}$$

  where N is the total number of data points.

- The value of MSE is minimized using the gradient descent method. Gradient Descent is an iterative optimization algorithm used to minimize a function by adjusting model parameters in the direction of the steepest descent of the function's gradient. In simple terms, it finds the optimal values of weights and biases by gradually reducing the error between predicted and actual outputs. The equation for gradient descent are:

$$w = w - \gamma \frac{\partial MSE}{\partial w} \tag{3}$$

where w is any weight and γ is the learning rate.
Calculating the gradient with respect to our model:

$$\begin{aligned}
\frac{\partial MSE}{\partial w_j} &= \frac{\partial \frac{1}{N}\sum_{i=1}^{N}(y_{pred_i} - y_i)^2}{\partial w_j} \\
&= \frac{1}{N}\sum_{i=1}^{N}\frac{\partial(w_1x_{i1} + w_2x_{i2} + w_3x_{i3} + \cdots + w_mx_m + b - y_i)^2}{\partial w_j} \\
&= \frac{2}{N}\sum_{i=1}^{N}(y_{pred_i} - y_i)\frac{\partial w_1x_{i1} + w_2x_{i2} + w_3x_{i3} + \cdots + w_mx_m + b - y_i}{\partial w_j} \\
&= \frac{2}{N}\sum_{i=1}^{N}(y_{pred_i} - y_i)x_{ij}
\end{aligned} \tag{4}$$

$$\frac{\partial MSE}{\partial b} = \frac{2}{N}\sum_{i=1}^{N}(y_{pred_i} - y_i) \tag{5}$$

Now, the values of w and b are updated as:

$$w_j = w_j - \gamma \frac{\partial MSE}{\partial w_j} \tag{6}$$

$$b = b - \gamma \frac{\partial MSE}{\partial b} \tag{7}$$

They are updated until the convergence condition of the regression is reached.

- The learning rate γ chosen by me is 0.001 and the value of the weights and parameters are updated until the change in MSE in the current and previous iteration is less than 10^-8. The learning rate ensures that no overstepping occurs and the convergence condition ensures that the parameters are optimized.

- The data, before processing, is also normalized and after processing the parameters are converted to what it should be for non-normalized data. The normalization is done by the following equation:

$$x_{norm,j} = \frac{x_j - mean(x_j)}{std(x_j)} \tag{8}$$

And the parameters are converted back in the following way:

$$\begin{aligned} y_{pred} &= w_{norm,1}x_{norm,1} + w_{norm,2}x_{norm,2} + \cdots + w_{norm,m}x_{norm,m} + b_{norm} \\ &= \sum_{j=1}^{m} w_{norm,j}x_{norm,j} + b_{norm} \\ &= \sum_{j=1}^{m} w_{norm,j}\frac{x_j - mean(x_j)}{std(x_j)} + b_{norm} \\ &= \sum_{j=1}^{m} \frac{w_{norm,j}x_j}{std(x_j)} + b_{norm} - \sum_{j=1}^{m} \frac{w_{norm,j}mean(x_j)}{std(x_j)} \end{aligned} \tag{9}$$

Comparing the coefficients of this equation with equation 1, we get:

$$w_j = \frac{w_{norm,j}}{std(x_j)} \tag{10}$$

$$b = b_{norm} - \sum_{j=1}^{m} \frac{w_{norm,j}mean(x_j)}{std(x_j)} \tag{11}$$

Comparing the results

Scratch Code Values	SKLearn Values	Difference
-0.097460	-0.097594	0.000134
0.048815	0.048905	-0.000090
0.029390	0.030379	-0.000989
2.773195	2.769378	0.003817
-17.948568	-17.969028	0.020460
4.284722	4.283252	0.001470
-0.013027	-0.012991	-0.000036
-1.458399	-1.458510	0.000111
0.284146	0.285866	-0.001720
-0.013045	-0.013146	0.000101
-0.914273	-0.914582	0.000309
0.009654	0.009656	-0.000002
-0.423643	-0.423661	0.000018
32.642134	32.680059	-0.037925

Table 1: Comparison

References

[1] *Linear Regression*

[2] *Numpy and Pandas*

[3] *Gradient Descent*