Abstract:

Linear regression is an important technique. Its basis is illustrated here, and various derived values such as the standard deviation from regression and the slope of the relationship between two variables are shown. The way to study residuals is given, as well as information to evaluate auto-correlation.

Introduction:

In statistics, linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called simple linear regression; for more than one, the process is called multiple linear regression.

Explanation:

MATHS BEHIND IT(normal method):

In linear regression, we obtain an estimate of the unknown variable (denoted by y; the output of our model) by computing a weighted sum of our known variables (denoted by xi ; the inputs) to which we add a bias term.

$$y = \sum_{i=1}^{n} x_i.w_i + b$$

where n is the number of data points. Adding a bias is the same thing as imagining we have an extra input variable that's always 1 and using only the weights. We will consider this case to make the math notation a little easier.

$$y = \sum_{i=1}^{n} x_i.w_i$$

Where x_0 is always 1, and x_0 is our previous b.

For making it easier, we will transform it in matrix equation.

$$y = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Now, the above equation is just for one data point. If we want to compute the outputs of more data points at once, we can concatenate the input rows into one matrix which we will denote by X. The weights vector will remain the same for all those different input rows and we will denote it by w. Now y will be used to denote a column-vector with all the outputs instead of just a single value.

$$y = Xw$$

To learn the weights, we need a dataset in which we know both X and y values, and based on those we will find the weights vector.

For obtaining weights the equation can be:

$$w = X^{-1}y$$

Most of the time the requirement for the solution above will not hold. Most of the time, our data points will not perfectly fit a line. There will be some random noise around our regression line, and we will not be able to obtain an exact solution for w However, we will try to obtain the best possible solution for w so that the error is minimal.

This defines that y doesn't belong to the column space of X. So, instead of y, we will use the projection of y onto the column space of X. This is the closest vector to y that also belongs to the column space of X. If we multiply both sides by the transpose of X, we will get an equation in which this projection is considered.

$$y = Xw$$
$$X^T.y = X^T.Xw$$
$$(X^T.X)^{-1}(X^T.y) = (X^T.X)^{-1}(X^T.X)w$$
$$(X^T.X)^{-1}(X^T.y) = w$$

i.e we get, $(X^T.X)^{-1}(X^T.y) = w$ But this method becomes infeasible if X has huge dimensions and n is also large.

MATHS BEHIND IT(gradient descent method):

The equation of our line of best fit is given by this equation. We call this our hypothesis, represented by the function $h(x)$.

$$h(x) = \sum_{i=1}^{n} \theta_i.x_i + \theta_0$$

Thus, in multivariate (dealing with multiple variable, x) linear regression, we must find a way to tweak these thetas to get a hyper plane that best represents the hyper plane of best fit. Here, θ_0 is the bias, in that it shifts the entire line by default. $\theta_1, \theta_2, \theta_3, ..., \theta_n$ represents the weight, or how much does the input x impact the output y.

The mean squared error (cost function) finds the average squared difference between our predictions, and the actual y values in our training set. Our goal is to minimize the total cost.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i) - y_i)^2$$

Summarising everything:
Hypothesis: $y = h(x) = \sum_{i=1}^{n} \theta_i.x_i + \theta_0$
Parameters: θ
Cost Function: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i) - y_i)^2$
Goal: $min_\theta J(\theta$

Further Intuition:

Choosing ordinary least squares (OLS) as our cost function comes at a cost. There are certain assumptions one has to deal with in order to finalize OLS as our cost function. The below derivation will make things more clear. $y_i = \theta x_i + \epsilon_i$. The term ϵ_i is introduced to capture the unmodelled effects and errors caused by a given θ. In other words, it can be called as an error term or residual.

We assume that ϵ follows Gaussian distribution (with mean zero and some variance). The PDF is:

$$PDF(\epsilon_i) = \frac{1}{\sqrt{2\pi\sigma}}e^{-\frac{\epsilon_i^2}{2\sigma^2}}$$

It can also be written as,

$$P(y_i|x_i, \theta) = \frac{1}{\sqrt{2\pi\sigma}}e^{-\frac{(y_i-\theta x_i)^2}{2\sigma^2}}$$

Our goal is to get the highest probability of y_i's given our data i.e., we need to maximize the likelihood of pdf at each point.

$$L(\theta) = \prod_{i=1}^{n} P(y_i|x_i, \theta)$$

$$L(\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}}e^{-\frac{(y_i-\theta x_i)^2}{2\sigma^2}}$$

We know that logarithm is a monotonic function. So, $L(\theta)$ is maximum when $\log(L(\theta))$ is maximum. i.e.

$$l(\theta) = n\log\frac{1}{\sqrt{2\pi\sigma}} - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \theta x_i)^2$$

This expression is nothing but our OLS(Ordinary Least Squares) cost function.

Note: This is one approach that justifies choosing OLS as a cost function. There can be other approaches that might infer the same end result.

Gradient Descent Algorithm:

Gradient descent is an iterative algorithm that randomly initializes somewhere on this plane. It then adjusts the values of θ_0 and θ_1 until it gets to a point on the plane where the cost function is low. Intuitively, what gradient descent does is upon each iteration (each star on the graph) it 'looks around a full 360 degrees' and chooses the direction of steepest (most efficient) descent. It then takes a baby step in that direction, and reiterates again, until it converges — when it gets to a point where the cost can't go any lower. Below are the mathematical formulas that describe how these steps are taken (note that the formulas on the right are directly equivalent, but have the derivative term, the fraction thing, calculated).

The alorithm can be defined as:

Algorithm 0.1 Gradient Descent

```
1: procedure GRADIENT DESCENT($\{(\mathbf{X}_i, Y_i)\}_{i\in\text{train}}, Ep, \eta, \mathbf{w}_0$)
2:     $\mathbf{w} \leftarrow \mathbf{w}_0$.
3:     for $i \in \{1, \dots Ep\}$ do
4:         $\mathbf{w} \leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}} J(c_{\mathbf{w}})$
5:     end for
6:     Return w.
7: end procedure
```

Mathematically,

$$\begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \vdots\\ \theta_n\end{bmatrix} = \begin{bmatrix}\theta_0\\ \theta_1\\ \theta_2\\ \vdots\\ \theta_n\end{bmatrix} - \begin{bmatrix}\frac{\partial J(\theta)}{\partial\theta_0}\\ \frac{\partial J(\theta)}{\partial\theta_1}\\ \frac{\partial J(\theta)}{\partial\theta_2}\\ \vdots\\ \frac{\partial J(\theta)}{\partial\theta_n}\end{bmatrix}\alpha$$

If we were to think of this as a vector operation, it could be adjusting a theta vector with θ by the negative gradient scaled by some learning rate α.

$$
\begin{gathered}
\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(x_i) - y_i)^2 \\
\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{2m}\frac{\partial}{\partial\theta}\sum_{i=1}^{m}(h(x_i) - y_i)^2 \\
\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta}(h(x_i) - y_i)^2 \\
\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}2(h(x_i) - y_i)\frac{\partial}{\partial\theta}(h(x_i) - y_i) \\
\frac{\partial}{\partial\theta}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}2(h(x_i) - y_i)\frac{\partial}{\partial\theta}(\theta_0 + \theta_1 X - y_i) \\
\frac{\partial}{\partial\theta_0}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}2(h(x_i) - y_i)\frac{\partial}{\partial\theta}(\theta_0) \\
\frac{\partial}{\partial\theta_0}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}2(h(x_i) - y_i) \\
\frac{\partial}{\partial\theta_1}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}2(h(x_i) - y_i)\frac{\partial}{\partial\theta}(\theta_0 + \theta_1 X - y_i) \\
\frac{\partial}{\partial\theta_1}J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}2(h(x_i) - y_i)x_i
\end{gathered}
\tag{1}
$$

The derivation is pretty simple year 1 calculus stuff, except for the brief use of multivariable calculus in determining the second product after using the chain rule. This technically only derives the partial derivative segment of the overall update rule, but the rest.

Conclusion:

Thus finding all the deriatives we can find all the weights and reach our minimum loss.