Table of Contents

1. Linear Regression

Linear Regression is a supervised learning algorithm in machine learning that models the relationship between a dependent variable (target) and one or more independent variables (features) as a linear equation. The goal of linear regression is to find the best-fit line that describes the relationship between the input features and the target variable.

There are two types of Linear Regression algorithms based on the number of independent variables. In a simple linear regression model, there is only one independent variable, while in a multiple linear regression model, there are multiple independent variables.

The equation for a simple linear regression line is:

$$y = B_0 + B_1 x$$

Where:

- y is the dependent variable (the value we want to predict)
- x is the independent variable (the value used to make the prediction)
- B_0 is the y-intercept (value of y when $x = 0$)
- B_1 is the slope (rate of change of y with respect to x)

For multiple linear regression the equation extends to:

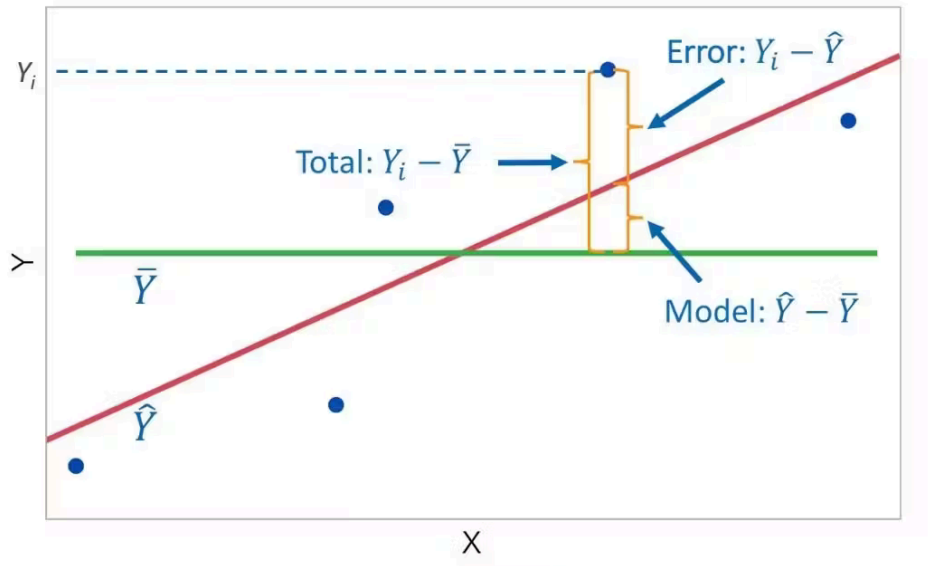
$$y = B_0 + B_1 x_1 + B_2 x_2 + \ldots + B_p x_p$$

Where x_1, x_2, ..., x_p are independent variables.

In order to find the "line of best fit" we can use two approaches:

1.1. Ordinary Least Squares Approach (Closed-Form Solution)

In this approach, we use a technique called the least squares method. This method calculates the values for the slope (B_1) and the y-intercept (B_0) that minimize the sum of the squared errors (or residuals). A residual is the vertical distance between a data point and the regression line. By squaring these differences, the method ensures that both positive and negative errors are treated equally and that larger errors have a greater penalty.



The formula for the mean squared error (MSE) is:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Where:

- y_i is the actual value of the dependent variable for data point i
- $\hat{y}_i$ is the predicted value of the dependent variable for data point i
- n is the number of observations

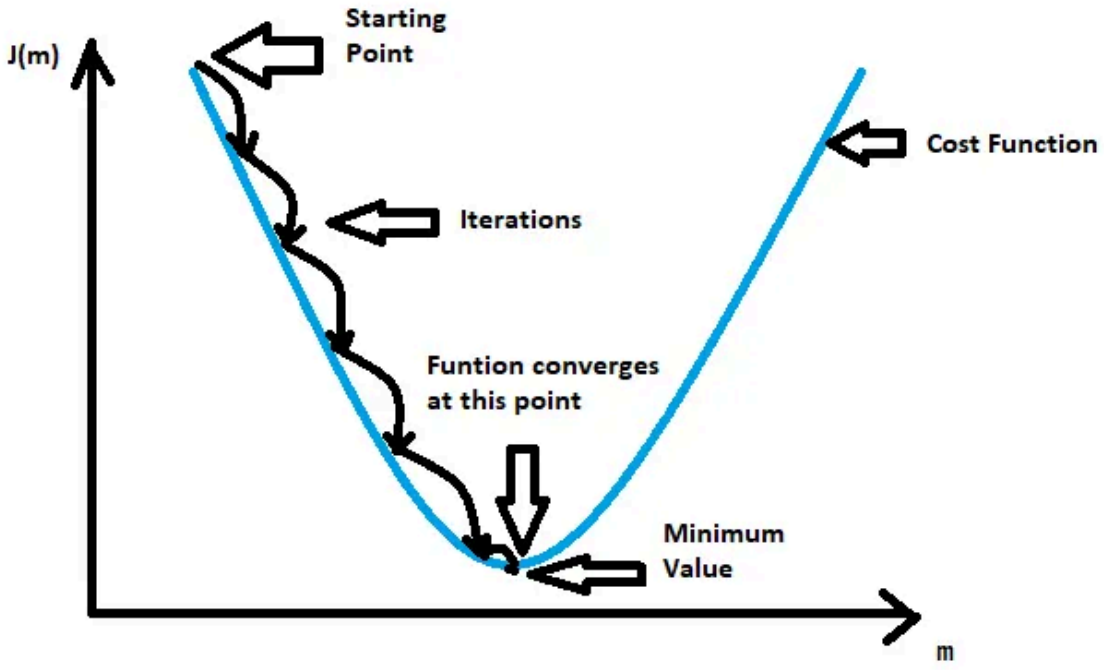
To minimize this function, we use calculus. We take the partial derivatives of the MSE with respect to B_0 and B_1 and set them to zero. This gives us a system of two equations that we can solve for the optimal values of B_0 and B_1. The resulting formulas for the slope and intercept are:

$$B1 = \frac{\sum_{i=1}^{n}((x_i - mean(x)) \times (y_i - mean(y)))}{\sum_{i=1}^{n}(x_i - mean(x))^2}$$
$$B0 = mean(y) - B1 \times mean(x)$$

Once B_0 and B_1 are calculated using the least squares method, the resulting line represents the optimal fit for the data, ready to be used for predictions. But this becomes complex when the data is multidimensional.

1.2. Gradient Descent Approach (Iterative Solution)

Another way to find the “line of best fit” is to use gradient descent. Gradient descent is useful for larger multidimensional data and it is also less complex and cheaper solution. Gradient Descent runs in iterations, till the minimum value of the cost function is reached. To understand more, let us consider the figure below.



J(m) is our cost function and we can see that we want to reach a point where the function converges (reaches minimum value). In other words we want to reach a point where the value of our function is minimum. So we do this in steps, also called iterations.

Learning rate is the rate at which we move in the direction of minimization. Larger the learning rate, the higher is the chance that we miss the minimum value point. Hence it is wise to keep the learning rate small.

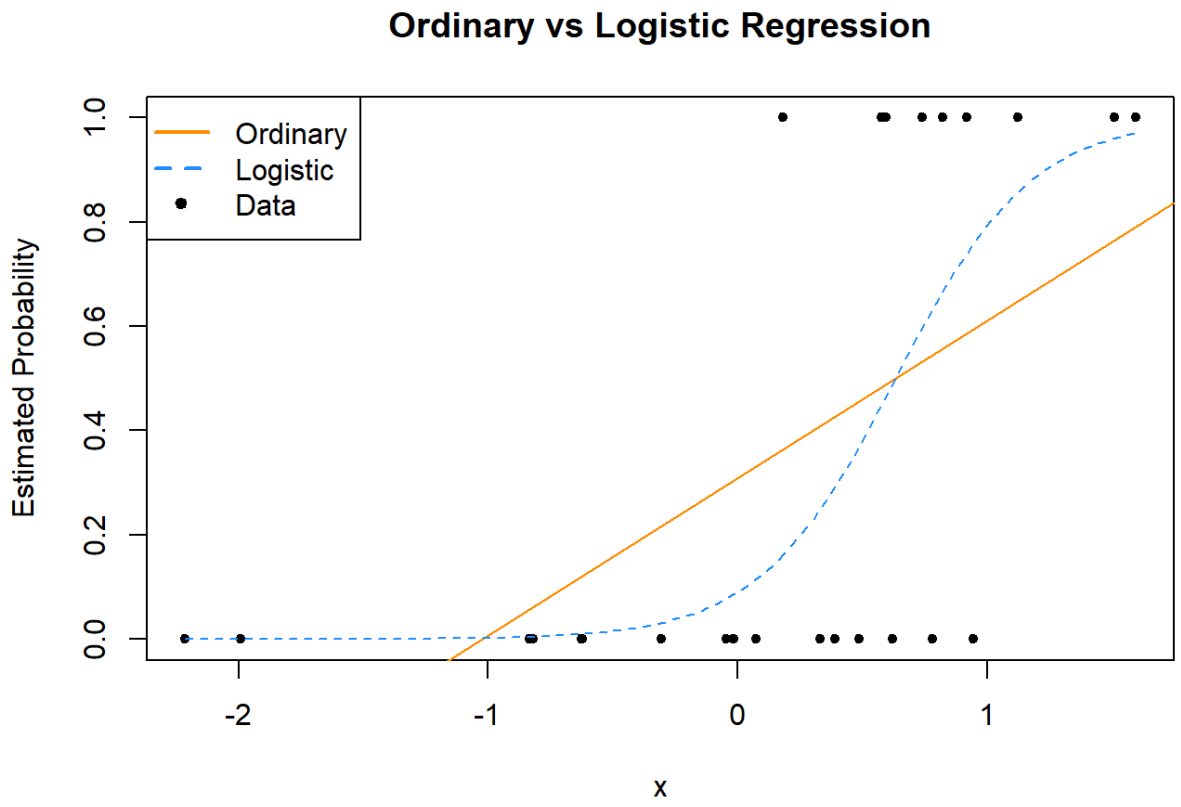
$$m_1 = m_0 - (learning\ rate).\ (dJ/dm)$$

Where:

- m_1 is the next point given by the iteration
- m_0 is the starting point of current iteration
- We know that our cost function is MSE

2. Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for classification problems. Unlike linear regression which predicts continuous values it predicts the probability that an input belongs to a specific class. It is used for binary classification where the output can be one of two possible categories such as Yes/No, True/False or 0/1. It uses sigmoid function to convert inputs into a probability value between 0 and 1.

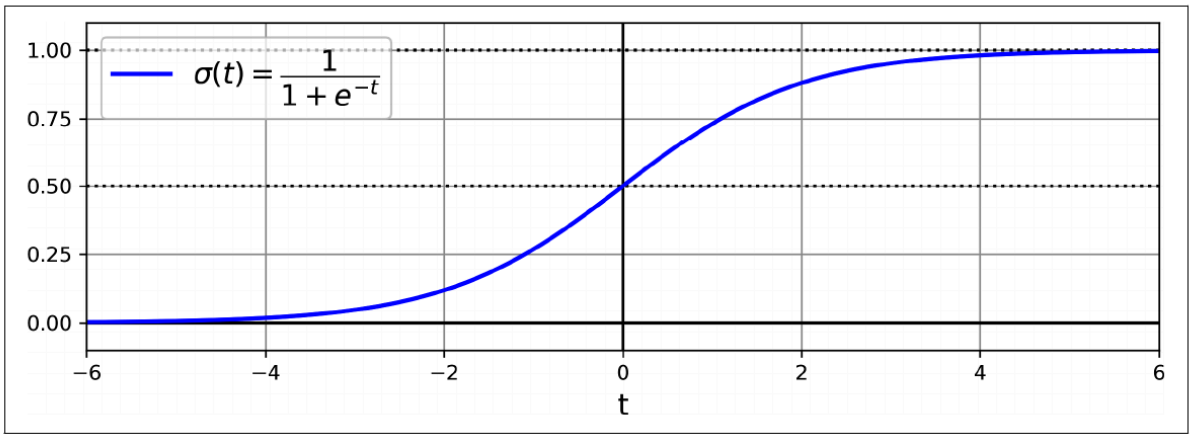


Just like a Linear Regression model, a Logistic Regression model computes a weighted sum of the input features (plus a bias term), but instead of outputting the result directly like the Linear Regression model does, it outputs the logistic of this result:

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x})$$

The logistic—noted σ(·)—is a sigmoid function (i.e., S-shaped) that outputs a number between 0 and 1. It is defined as shown below:

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$



Once the logistic regression model has estimated the probability $\hat{p} = h_{\theta}(x)$ that an instance x belongs to the positive class, it can make its prediction ŷ easily.

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

So a logistic regression model using the default threshold of 50% probability predicts 1 if $\theta^{T} x$ is positive and 0 if it is negative.

The objective of training is to set the parameter vector θ so that the model estimates high probabilities for positive instances ($y = 1$) and low probabilities for negative instances ($y = 0$). This idea is captured by the cost function shown in equation below for a single training instance x.

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$

This cost function makes sense because $-\ log(t)$ grows very large when t approaches 0, so the cost will be large if the model estimates a probability close to 0 for a positive instance, and it will also be large if the model estimates a probability close to 1 for a negative instance. On the other hand, $-\ log(t)$ is close to 0 when t is close to 1, so the cost will be close to 0 if the estimated probability is close to 0 for a negative instance or close to 1 for a positive instance, which is precisely what we want.

The cost function over the whole training set is the average cost over all training instances. It can be written in a single expression called the log loss, shown in equation below:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\Sigma_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)log\left(1 - \hat{p}^{(i)}\right)\right]$$

There is no known closed-form solution to find the value of θ that minimizes the cost function. So, we need to use gradient descent to compute the value of θ. The partial derivatives of the cost function with regard to the j^{th} model parameter θ_j are given by equation below.

$$\frac{\partial}{\partial \theta_j} \mathrm{J}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\boldsymbol{\theta}^{\mathsf{T}} \mathbf{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

So, once you have the gradient vector containing all the partial derivatives, you can use it in the batch gradient descent algorithm. The algorithm continues to update the coefficients until the cost function converges to a minimum, at which point we have the optimal parameters for our logistic regression model.