Kharagpur Data Analytics Group

Associate Selection Task

Logistic Regression

An Overview

Ankit Meda
22MF3IM11

Introduction

Logistic regression is a popular machine learning algorithm for binary classification problems. It aims to model the relationship between the input features (independent variables) and a binary output (dependent variable) to predict the probability of the binary outcome.

In binary classification, our output predictions can only take one of the two possible outcomes, i.e. true/false, 0/1 and more.

Some real-world applications of the algorithm are as follows

1. **Spam email detection:** Logistic regression helps detect whether the incoming email is spam based on the email's content, sender information, and other features.

2. **Image classification:** Logistic regressions can be applied in cases with only two distinguishing classes, like cats and dogs.

3. **Medical diagnosis:** Logistic regression can predict where a person has a disease based on the medical test results, symptoms and medical history.

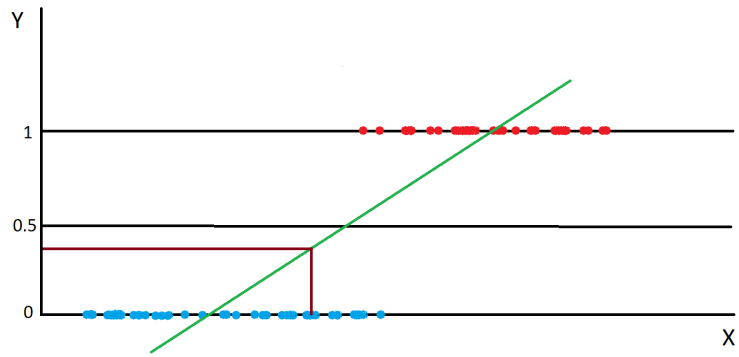

Figure 1: An example of Linear Regression over a hypothetical Binary Classification dataset

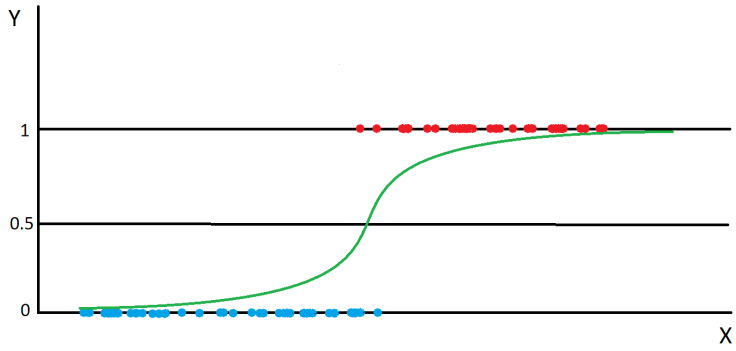

Figure 2: An example of Logistic Regression over a hypothetical Binary Classification dataset

Assumptions

Before we start learning about Logistic Regression, we first need to look at some assumptions we make when we make this model.

1. The relationship between the dependent and independent variables is linear.
2. Logistic Regression is specifically designed for binary classification, i.e. the dependent variable can only have two possible values (0 or 1).
3. Logistic regressions tend to perform well with a large sample size.

Sigmoid Curve

As shown in Figure 1, the straight line can help predict whether the input data gives a true or false (0 or 1). As a convention, by passing through the line equation, we classify an input that produces an output of 0.5 or greater as "true" and anything less than 0.5 as "false" .

However, the problem with this straight line is that it cannot fit properly into the dataset. Hence, it leads the model to give wrong outcomes.

One way to tackle this issue is to construct a curve that best fits the dataset, as in Figure 2.

The curve is called a **sigmoid curve**. The curve gives an output of 0.5 at $x = 0$, and when x approaches positive infinity (i.e., becomes extremely large), the output y tends to 1. Similarly, as x approaches negative infinity (i.e., becomes extremely negative), the output y tends to 0.

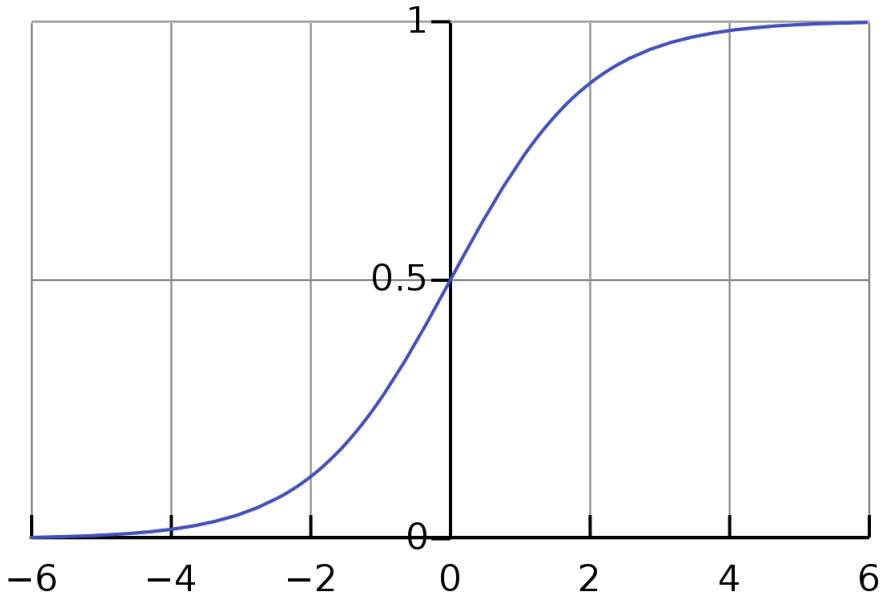

Figure 3: A pictorial representation of sigmoid curve

The equation governing the sigmoid curve is given by

$$y = \sigma(x) = \frac{1}{1 + e^{-x}}$$

So, to draw the sigmoid curve to fit our dataset, we can pass the equation as an input to the sigmoid function.

$$y = mx + b$$
$$\boxed{\widehat{y} = \sigma(y) = \sigma(mx + b)}$$

If our dataset had n **features** instead of one, our best-fit line equation would be as follows (NOTE: Features and samples/entries in a dataset are not the same. The following equation is for a dataset with only one entry).

$$y = w_1x_1 + w_2x_2 + w_3x_3 + ... + w_nx_n + b$$

where w are called **weights**. The weights in logistic regression control the strength of the relationship between the input features and the predicted probabilities. The model finds accurate predictions for binary classification tasks by adjusting these weights during training.

To make calculations more manageable, we will define a row weight vector and column input vector as follows.

$$W = \begin{pmatrix} w_1 & w_2 & w_3 & ... & w_n \end{pmatrix}_{1\times n}, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ .. \\ .. \\ x_n \end{pmatrix}_{n\times 1}$$

Hence, our line equation and sigmoid curve equation become as

$$y = Z = WX + b$$

$$\boxed{\widehat{y} = \sigma(y) = \sigma(Z) = \sigma(WX + b)}$$

We predict the model as follows: We compute $WX+b$, then pass it through the sigmoid function, and we get $\widehat{y}$, which represents the probability (e.g. 0.23. Then if $\widehat{y} \geq 0.5$, then it is classified as one, otherwise zero.

Now, if our dataset had M entries, that means it has m different n features in the dataset. Then our equation turns out as follows.

$$X^{[1]} = \begin{pmatrix} x_1^{[1]} \\ x_2^{[1]} \\ .. \\ .. \\ x_n^{[1]} \end{pmatrix}_{n \times 1}$$

$$Z_1 = WX^{[1]} + b_1$$

This equation explains that there is only one predicted outcome for one sample ([.] represents the sample number). Then, our curve equation becomes as

$$\widehat{y}_1 = \sigma(Z_1) = \sigma(WX^{[1]} + b_1)$$

Hence, in the general form, we get

$$\boxed{\widehat{y}_k = \sigma(Z_k) = \sigma(WX^{[k]} + b_k)}$$

where $1 \leq k \leq m$.

Ok, but how do we decide on the weights? How can we change them so that they best fit our curve?

Cost Function

The difference between the output that is provided and the appropriate label for it is determined using the cost function or error function. By minimising the error function, we can train the model to adjust the weights to make predictions that are more accurate.

The cost function for logistic regression is given as follows.

$$C_{1*} = -y_1 \cdot log(\widehat{y}_1) - (1 - y_1) \cdot log(1 - \widehat{y}_1)$$

where y represents the actual/desired outcome and $\widehat{y}$ is the predicted outcome.

The equation above represents the "cost" or error produced by one sample from the dataset of M samples. Hence, by averaging all the M samples, we can get the overall cost of the model.

$$\boxed{C = \frac{-1}{m} \sum_{i=1}^{m} y_i \cdot \log(\widehat{y}_i) - (1 - y_i) \cdot log(1 - \widehat{y}_i)}$$

Hence, we need to minimise the error function, and through that, we need to update the weights and the intercept.

Weight and intercept optimisation

In order to minimise the cost function, we can find the gradient of the cost function with respect to the weight and intercept. Now, by applying the chain rule to the first sample's cost, we get the following.

$$1)\ \frac{\partial C_{1*}}{\partial w_1} = \frac{\partial Z_1}{\partial w_1} \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial C_{1*}}{\partial A_1}$$

Let us now discuss each term in detail.

1. $$\frac{\partial C_{1*}}{\partial A_1}$$

   Here, $A_1 = \widehat{y}_1$ which is a parameter in C, the cost function. Hence the gradient of the cost function can be computed with respect to A_1 without any issues.

   $$\boxed{\frac{\partial C_{1*}}{\partial A_1} = \frac{-y_1}{A_1} + \frac{(1-y_1)}{1-A_1}}$$

2. $$\frac{\partial A_1}{\partial Z_1}$$

   From the equation, $A_1 = \sigma(z_1) = \frac{1}{1+e^{-z_1}}$, we can compute the gradient.

   $$\boxed{\frac{\partial A_1}{\partial Z_1} = A_1(1-A_1)}$$

3. $$\frac{\partial Z_1}{\partial w_1}$$

   From the equation $Z_1 = w_1 x_1^{[1]} + w_2 x_2^{[1]} + ... + w_n x_n^{[1]} + b_1$, we can calculate the the gradient.

   $$\boxed{\frac{\partial Z_1}{\partial w_1} = x_1^{[1]}}$$

Hence we get the required gradient as follows

$$\frac{\partial C_{1*}}{\partial w_1} = \frac{\partial Z_1}{\partial w_1} \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial C_{1*}}{\partial A_1}$$

$$\boxed{\frac{\partial C_{1*}}{\partial w_1} = (A_1 - y_1)x_1^{[1]}}$$

Now, by following the same steps for our intercept, we get the following.

$$2)\ \frac{\partial C_{1*}}{\partial b_1} = \frac{\partial Z_1}{\partial w_1} \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial C_{1*}}{\partial A_1}$$

Since two of the three terms are identical for the gradient with respect to the weight and bias term, we will only focus on the first term, which is yet to be defined.

1. $$\frac{\partial Z_1}{\partial b_1}$$

   From the equation $Z_1 = w_1 x_1^{[1]} + w_2 x_2^{[1]} + ... + w_n x_n^{[1]} + b_1$, we can calculate the gradient.

$$\boxed{\frac{\partial Z_1}{\partial b_1} = 1}$$

Hence we get the required gradient as follows

$$\frac{\partial C_{1*}}{\partial b_1} = \frac{\partial Z_1}{\partial b_1} \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial C_{1*}}{\partial A_1}$$

$$\boxed{\frac{\partial C_{1*}}{\partial b_1} = (A_1 - y_1)}$$

Now, if we can combine the cost gradient for all weights and intercepts into matrices, we get the following.

$$\boxed{\frac{\partial C_{1*}}{\partial W} = (A_1 - y_1)(X^{[1]})_T}$$

$$\boxed{\frac{\partial C_{1*}}{\partial B} = (A_1 - y_1)}$$

where $B = \begin{pmatrix} b_1 & b_2 & ... & b_m \end{pmatrix}^T$ is the intercept vector, and $(X)_T$ represents the transpose of the matrix.

Hence, the average gradient with respect to weights and intercepts for all the costs from the whole dataset is calculated as follows.

$$\boxed{\frac{\partial C}{\partial W} = \frac{1}{m} \sum_{i=1}^{M} \frac{\partial C_{i*}}{\partial W}}$$

$$\boxed{\frac{\partial C}{\partial W} = \frac{1}{m} \sum_{i=1}^{M} \frac{\partial C_{i*}}{\partial W}}$$

Now that we have found the required gradients, we can improve the weights and intercepts by changing them as follows.

$$W_{new} = W_{old} - \alpha(\frac{\partial C}{\partial W_{old}})$$

$$B_{new} = B_{old} - \alpha(\frac{\partial C}{\partial B_{old}})$$

The learning rate α is an example of a hyperparameter.

A hyperparameter is a setting that is determined before the training process and remains constant during training. Unlike model parameters (weights, intercepts) learned from the training data, hyperparameters are set manually.

Finding the appropriate hyperparameter values is essential as they impact the model's performance and generalisation.

We have completed the training. Now, the sigmoid curve with the fine-tuned weights and intercepts will best fit the dataset.