Logistic Regression Interpretation

The idea behind creating this file was inspired by my custom-built function (logreg_interpretation) that calculates the odds ratio and the normalized odds. I wanted to ensure that my code was correct, and during this effort, I needed to deeply engage with basic logistic regression concepts. The purpose of this function is to reveal the correct information from the logistic regression model, enhancing interpretation.

Starting with Logistic Regression Understanding

Here is the logreg probability formula:

$$Log\ Reg\ Prob(p) = \frac{1}{1+e^{z}} \qquad (1)$$

Where z is the linear combination of features. The term e^{z} is the **odds**. Therefore, the natural logarithm of the odds, $\ln(e^{z})$, provides z. This is the reason z is called **log odds**. As z is a linear combination of features, it holds:

$$z = b_0 + b_1 x_1 + \cdots + b_n x_n \quad (2)$$

So, log-odds in a logreg model are essentially the same as a multiple linear regression equation.

1) Defining why e^{z} is the Odds

Statistically speaking, odds of an event occurring equal the probability of the event happening divided by the probability the event not happening:

$$odds = \frac{p}{1-p} \quad (3)$$

Now, let's substitute equation (1) into equation (3) (without going into detailed calculations):

$$odds = \frac{\frac{1}{1+e^{z}}}{1-\frac{1}{1+e^{z}}} = \cdots = e^{z}$$

Combining these, we get:

$$e^z = \frac{p}{1-p} \qquad (4)$$

Finally, we confirm that e^z represents the odds.

From equation (4) it can be derived that for log odds:

$$z = \ln\left(\frac{p}{1-p}\right) \quad (5)$$

2) Calculating and Interpreting the Odds

After running a logreg model we can use the following code to get the coefficients:

logreg_model.coef_.flatten() *# .flatten() converts 3d array to 2d array, but this doesn't impact the interpretation*

This code provides the weights of each feature in a 2d array. That is, b_1, b_2, …, b_n. We get the coefficients but still can't interpret a logreg model.

But why the coefficients above can't be directly interpreted in Logistic Regression?

They could, if we would deal with a continuous target variable and hence with multiple linear regression (MLR). In an MLR model, the linear combination of features is directly used to predict a continuous value. But in a logreg model, we use the same linear combination of features, but the result is passed through the sigmoid function to convert it into a probability (ranging from 0 to 1), as shown in equation (1).

Thus, revealing the coefficients in a logreg model provide the log odds, $\ln\left(\frac{p}{1-p}\right)$, which may be difficult to interpret directly.

Interpreting the Odds:

The solution to logreg interpretation lies in the term e^z, which represent the odds $\frac{p}{1-p}$. This is the value we calculate to interpret how the features affect the logreg probability. Combining equation (2) with equation (4), excluding the intercept term and applying the exponential property, we can calculate the odds:

$$e^{b_1} \cdot e^{b_2} \cdot \ldots \cdot e^{b_n} = \frac{p}{1-p} \qquad (6)$$

This means the odds are the multiplicative combination of the unique odds for each feature. Odds can range from 0 to infinity:

- An odds ratio less than 1 will decrease the total odds.
- An odds ratio greater than 1 will increase the total odds.
- An odds ratio close to 1 will have little effect on the total odds.

Total Odds Interpretation Example:

Let's assume the model for hotel booking cancellations includes 4 features: adr, deposit_type_Non Refund, lead_time, and total_kids. The numbers are randomly chosen:

$$0.9 \cdot 10 \cdot 1.2 \cdot 0.2 = 2.16$$

This means that the odds of a booking being canceled are 2.16 times higher compared to the baseline (when all feature values are zero or at their reference levels after one-hot encoding). In other words, the presence and values of these features together increase the odds of cancellation by 216%.

Interpreting Specific Feature Odds:

To calculate the odds of each feature separately, i.e., the odds of i^{th} feature we need to calculate e^{b_i}. We can do this programmatically with:

'np.exp(logreg_model.coef_.flatten())'

Assuming that the coefficient for total_kids is 0.693, this means that odds are $e^{0.693} = 2$. Hence, each additional kid (other than the baseline) multiplies the total odds of a booking being canceled by 2. For instance:

- 0 kids (baseline): Odds=1
- 1 kis: Odds=2
- 2 kids: Odds=4

- 3 kids: Odds=8

Logistic Regression Assumption:

The above exponential growth results directly from the linear relationship assumed by logistic regression on the log-odds scale, $log\_odds = intercept + \beta \times total\_kids$. There is no internal adjustment for potential non-linearity. The assumption is that the relationship between total_kids and cancellation likelihood is linear. If this assumption holds, logistic regression is appropriate and provides meaningful interpretations.

However, in real data, the relationship between number of kids and cancellation likelihood might not be linear. For example, going from 0 to 1 kid increases odds (e.g., harder to travel), but having 4 or 5 kids actually reduces cancellation because it's a family vacation they've planned well.

So, in cases where logistic regression seems to be a suitable model, but some features appear to have a non-linear relationship with the target, the following methods can help reduce the risk of misleading results while still using logistic regression:

- One-hot encoding

We can treat total_kids as a categorical feature. Create dummy variables like is_1_kid, is_2_kids etc., so that each value is treated independently.

- Polynomial features

We can include squared or higher-order terms of total_kids to capture non-linear relationships using sklearn.preprocessing.PolynomialFeatures.

- Logarithmic or other transformations

We can transform the feature (e.g., log, square root) so that the new relationship becomes more linear and better fits the assumptions of logistic regression.

Interpreting Continuous Feature Odds:

It is generally recommended all continuous features to be scaled properly. If not, an odds ratio of 2 can lead to extreme odds when feature values are large (because 1 unit of change doubles the previous odds), which may not reflect reality. Scaled continuous features keep the odds within a realistic range. Supposing 4 different adr values (30, 50, 100, 200€), we apply the formula of standardization:

$$scaled\ adr\ values = \frac{x-\mu}{\sigma} \quad (7)$$

Applying Equation (7) with assumed mean (μ) of 80 and standard deviation (σ) of 30, we get scaled values of -1.67, -1, 0.67, and 4 respectively.

Assuming an odds ratio of 2 for standardized values (the unit is standard deviation here), the interpretation of the continuous feature is as follows:

Price (€)	**Scaled adr**	**Odds (base × 2^scaled)**
30	-1.67	$1 \times 2^{-1\,.\,67} \approx \mathbf{0.31}$
50	-1.00	$1 \times 2^{-1} = \mathbf{0.50}$
100	+0.67	$1 \times 2^{0\,.\,67} \approx \mathbf{1.59}$
200	+4.00	$1 \times 2^{4} = \mathbf{16.00}$

3) Calculating the Normalized Odds

After defining the odds of each feature, a useful and more intuitive metric is the normalized odds. This helps us interpret how much each feature contributes relative to others. There are two main ways to normalize the odds, normalized odds as a proportion of the total odds and normalized odds relative to the maximum.

Normalized Odds as a Proportion of the Total Odds:

This method shows the relative contribution of each feature's odds to the sum of all odds. The result is a distribution that sums to 1, making it easy to interpret the relative strength of each feature.

To calculate the normalized odds of i^{th} feature:

$$normalized\ odds_i = \frac{e^{b_i}}{\sum_{k=1}^{k=n} e^{b_k}}$$

We can do this programmatically with:

‘np.exp(logreg_model.coef_.flatten()) / np.exp(logreg_model.coef_.flatten()).sum() ’

Normalized Odds relative to the Maximum:

This approach scales all feature odds between 0 and 1 by dividing each by the maximum feature odds. It answers the question: “How important is each feature compared to the most influential one?”

$$normalized\ odds_i = \frac{e^{b_i}}{\max\left(e^{b_k}\right)}$$

And here is it’s programmatic version:

‘np.exp(logreg_model.coef_.flatten()) / np.exp(logreg_model.coef_.flatten()).max() ’

Personally, I prefer to calculate as many interpretation metrics as possible because I enjoy interpretation, and they may prove useful in a potential comparison with more advanced models, especially since logistic regression is very suitable for baseline modeling. So, I include both of them in my custom logreg_interpretation function.