# Logistic regression

Paulo S. A. Sousa

2023-05-29

**Abstract**

Logistic regression is introduced, both theoretically and practically, in the field of machine learning.

Lasso regularization is also discussed.

## 1 Introduction

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). It is used to predict the probability of a certain event occurring based on the relationship between the independent variables and the outcome. Logistic regression is a type of generalized linear model (GLM) that uses the logistic function to model the probability of the binary outcome.

Logistic regression is best suited for situations where you want to model the probability of a binary outcome (yes/no, success/failure, 0/1, etc.) given a set of independent variables (predictors).

## 2 Model

Suppose we want to predict the probability of a binary outcome variable to be class 1 (instead of class 0), given a set of independent $k$ predictors. We could start by using the linear regression model:

$$p = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k,$$

where $p$ is the probability of the outcome variable to be class 1, $X_1, X_2, \cdots, X_k$ the predictors, and $\beta_0, \beta_1, \cdots, \beta_k$ the parameters to be estimated. However, that is not guaranteed that the predictions by that model will always be in the interval $[0, 1]$.

To overcome this problem, we can use the logistic function:

$$f(x) = \frac{1}{1+e^{-x}}.$$

The logistic function returns always a number between 0 and 1. Thus, instead of the linear model, we can the following, where $p$ is the probability of the outcome variable being class 1:

$$p = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k)}}.$$

We can manipulate the expression:

$$\begin{aligned}
p = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k)}} &\iff 1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k)} = \frac{1}{p} \\
&\iff e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k)} = \frac{1-p}{p} \\
&\iff e^{\beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k} = \frac{p}{1-p} \\
&\implies \ln\left(\frac{p}{1-p}\right) = \beta_0+\beta_1 X_1+\beta_2 X_2+\cdots+\beta_k X_k.
\end{aligned}$$

We can now estimate the model having the guarantee that the predicted $p$ will always be a valid probability.

## 3 Implementation in Python

To use logistic regression in Python, we need to import `LogisticRegression` from `sklearn`:

```
from sklearn.linear_model import LogisticRegression
```

For imbalanced classification problems, we can use the parameter `class_weight='balanced'`, likewise as we did in the case of classification trees.

### 3.1 Lasso regularization

To use Lasso with logistic regression in `sklearn`, we can adjust the penalty parameter of the `LogisticRegression` function to `'l1'`, and you will also need to select a solver that supports Lasso, like `'saga'`.

The `C` parameter in the `LogisticRegression` function in `sklearn` is the inverse of the regularization strength, which can assume any positive number:

- If `C` is small, it means the regularization strength is high, which increases the penalty for large values in the coefficient vector. This leads to a model with smaller coefficients, which can help to prevent overfitting, but might underfit the data if the regularization is too strong.
- If `C` is large, it means the regularization strength is low, which decreases the penalty for large values in the coefficient vector. This leads to a model that fits the training data more closely but might overfit the data if the regularization is too weak.

Likewise in the linear regression model, Lasso can drive some of the model's coefficients to exactly zero. This means that Lasso can effectively perform predictors selection by determining which predictors are not contributing to the model and removing them by setting their coefficients to zero.

It is worth mentioning that the optimal value of `C` is data-dependent, and it is usually not known in advance. Therefore, it is common to perform hyperparameter tuning, for example, using grid search and cross-validation, to find the best `C` value for a given dataset and problem.