Supervised learning
Logistic Regression

Séverine Affeldt

Université Paris Cité, Centre Borelli UMR 9010
UFR Sciences Fondamentales et Biomédicales

2023 – 2024

Usage

The *logistic regression* (or *binomial regression*) was the first method used to fit a model for a binomial binary variable (ie., number of success for n_i trials) or a Bernouilli variable ($n_i = 1$). The dominant fields were marketing and epidemiology.

Examples of application:

- patient survival
- disease diagnosis
- product ownership
- good/faithfull customer

For one variable

Let Y be a categorical variable with J levels. The *odd* to get the j^{th} level rather than the k^{th} is given by,

$$\Omega_{jk} = \frac{\pi_j}{\pi_k} = \frac{n_j}{n_k}$$

where π_j is the probability of level j. If Y is a binary variable and follows a Bernouilli law, the odd is $\pi/(1-\pi)$, which expresses a chance of gain. Example: if $\pi = 0.8$, the success odd is $0.8/0.2 = 4$, and the failure odd is $0.2/0.8 = 0.25$

For a contingency table

We consider two categorical variables X^1 and X^2, with $\pi_{ij} = P(X^1 = i \& X^2 = j)$ the occurrence probability for each combinaison. The corresponding confusion matrix is

$$\begin{bmatrix} \pi_{11} & \pi_{12} \\ \pi_{21} & \pi_{22} \end{bmatrix}$$

The odd for column 1 to happen is $\Omega_1 = \pi_{11}/\pi_{12}$ at 1^{st} row and $\Omega_2 = \pi_{21}/\pi_{22}$ at 2^{nd} row. The *odds ratio* is,

$$\Theta = \frac{\Omega_1}{\Omega_2} = \frac{\pi_{11}\pi_{22}}{\pi_{12}\pi_{21}}$$

For a contingency table

The *odds ratio* is,

$$\Theta = \frac{\Omega_1}{\Omega_2} = \frac{\pi_{11}\pi_{22}}{\pi_{12}\pi_{21}}$$

- $\Theta = 1 \leftrightarrow X^1 \& X^2 indep.$
- $\Theta > 1 \leftarrow$ individuals of the 1st row are more expected to follow the 1st column than the individuals of the 2nd column.
- $\Theta < 1 \leftarrow$ individuals of the 2nd row are more expected to follow the 1st column than the individuals of the 1st row.

Example

$$\mathbf{C}_\Theta = \begin{array}{cc} \text{Passed} & \text{Failed} & \\ \left[\begin{array}{c} 0.7 \\ 0.4 \end{array}\right. & \left.\begin{array}{c} 0.3 \\ 0.6 \end{array}\right] & \begin{array}{l} \text{Boy} \\ \text{Girl} \end{array} \end{array}$$

- $\Omega_{Boy} = 0.7/0.3 = 2.33$
- $\Omega_{Girl} = 0.4/0.6 = 0.67$
- $\Theta = \Omega_{Boy}/\Omega_{Girl} = 3.5$

The chance for a boy to pass the exam 3.5 greater than for the a girl.

Reminder: Our objective is to fit a model for a qualitative variable with 2 levels (pass or fail, disease or not, churner or not...).

Logit function

$$logit(\pi) = ln\frac{\pi}{1-\pi} \qquad with \quad g^{-1} = \frac{e^x}{1+e^x}$$

Reminder: The ratio $\pi/(1-\pi)$ is the odd.

$\leadsto$ The logistic regression searches for a linear model of the *log(odd)*. Caution: the predictor variables can be quantitative or qualitative.

Let's consider p observed predictor variables, $\{X^p\}_{p=1..q}$ that can be either quantitative or qualitative. These variables can take I combinaisons of values, $x_i^1, ..., x_i^q$, for which the variable Z is observed n_i times ($n = \sum_{i=1}^{I} n_i$). In particular, Z takes the values $y_1/n_1, ..., y_I/n_I$, where y_i is the number of successes for the n_i trials.

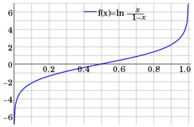
The observations are supposed to be independent and the probability π_i is constant for a given group I. Hence, the variable Y_i, given n_i, and with expectation $E(Y_i) = n_i\pi_i$ follows a *binomial law* $\mathcal{B}(n_i, \pi_i)$ with a density funcion

$$P(Y = y_i) = \binom{n_i}{y_i} \pi^{y_i} (1 - \pi_i)^{(n_i - y_i)}$$

The logit model

The hypothesis is that the vector of *logit* functions for the probabilities π_i belongs to the subspace holds by the predictor variables $\{X^1, ..., X^q\}$ such that, for $i = 1, ..., I$,

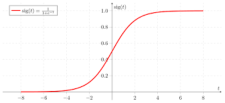
$$logit(\pi_i) = ln\left(\frac{\pi_i}{1-\pi_i}\right) = \mathbf{x}_i^\top \beta = b_0 + b_1 x_i^1 + ... + b_I x_i^I$$



$$\hat{y}_i = \begin{cases} 0 & \mathbf{x}_i^\top \mathbf{b} < 0 \\ 1 & \mathbf{x}_i^\top \mathbf{b} > 0 \end{cases}$$

The parameters vector β is estimates with the maximisation of the log-likelihood. We use iterative numerical methods (eg. Newton Raphson). The optimisation provides an estimation of $\mathbf{b}$ for β. When can then easily obtain an estimation of the probabilities (and groups cardinality, as $\hat{y}_i = n_i \hat{\pi}_i$),

$$\hat{\pi} = \frac{e^{\mathbf{x}_i^\top \mathbf{b}}}{1 + e^{\mathbf{x}_i^\top \mathbf{b}}} = \frac{1}{1 + e^{-\mathbf{x}_i^\top \mathbf{b}}}$$



$$\hat{y}_i = \begin{cases} 0 & \pi_i < 0.5 \\ 1 & \pi_i \geq 0.5 \end{cases}$$

Objective: Set the parameter vector β, so that the model estimates high probabilities for positive instances ($y = 1$) and low probabilities for negative instances ($y = 0$).

Cost function of a single trianing instance

$$c(\beta) = \begin{cases} -\log(\hat{\pi}_i) & \text{if } y = 1 \\ -\log(1 - \hat{\pi}_i) & \text{if } y = 0 \end{cases}$$

- If $\hat{\pi}_i \to 0$ while $y_i = 1$ then $-\log(\hat{\pi}_i)$ is large
- If $\hat{\pi}_i \to 1$ while $y_i = 0$ then $-\log(1 - \hat{\pi}_i)$ is large

Logistic Regression cost function (whole train set)

$$J(\beta) = -\frac{1}{n} \sum_{k=1}^{n} [y_i \log(\hat{\pi}_i) + (1 - y_i) \log(1 - \hat{\pi}_i)]$$

The cost function is convex, so Gradient Descent (or any other optimization algorithm) is guaranteed to find the global minimum (if the learning rate is not too large and if you wait long enough!).

Let's try a simple logistic regression with the *iris* dataset $\rightsquigarrow$ `P_LogisticRegression_simple_Iris.ipynb`

We can **regularize** (ie. constrain) a model to **reduce overfitting**. For a linear model, it is archieve by constraining the weights of the model.

Ridge Regression

The cost function is $J(\beta) = MSE(\beta) + \alpha \frac{1}{2} \sum_{i=1}^{n} b_i^2$. The red part (half the square of the ℓ_2 norm) keeps the model weights as small as possible. The α hyperparameter set the regularization strength. Caution: data must be scaled! NB: Inceasing α leads to weights very closed to 0.

Lasso Regression (Least Absolute Shrinkage and Selection Operator)

The cost function is $J(\beta) = MSE(\beta) + \alpha \sum_{i=1}^{n} |b_i|$. The red part ($\ell_1$ norm) that helps to eliminate the weights of the least important features. Nice: Lasso Regression automatically performs feature selection and outputs a sparse model.

Elastic Net

It is a middle ground between between Ridge Regression and Lasso Regression. The cost function is $J(\beta) = MSE(\beta) + \frac{1-r}{2} \alpha \frac{1}{2} \sum_{i=1}^{n} b_i^2 + r\alpha \sum_{i=1}^{n} |b_i|$.

So what?

Ridge Regression is good by default. But if you suspect that few features are involved, Lasso Regression or Elastic Net should be prefered as they tend to reduce useless features'weights down to zero.

The Logistic Regression model can be generalized to support **multiple classes** directly.

The idea

Let's consider an instance $\mathbf{x}$. First, the Softmax Regression model computes a score $s_k(\mathbf{x})$ for each class k. Then, it estimates the probability of each class by applying the *softmax function* to the scores.

Softmax score for class k

For each class k, we have now a parameter vector $\beta^{(k)}$.

$$s_k(\mathbf{x}) = \mathbf{x}^\top \beta^{(k)}$$

Probability

The probability $\hat{\pi}_k$ that the instance $\mathbf{x}$ belongs to class k is based on the softmax function.

$$\hat{\pi}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{e^{s_k(\mathbf{x})}}{\sum_{j=1}^{K} e^{s_j(\mathbf{x})}}$$

- K is the number of classes
- $\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for $\mathbf{x}$

$\Rightarrow$ The classifier predicts the class with the highest estimated probability.

Training objective: have a model that estimates a high probability for the target class and a low probability for the other classes. Minimizing the *cross entropy* fullfills this objective.

Cross entropy cost function

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_k^{(i)} log(\hat{\pi}_k^{(i)})$$

where $y_k^{(i)} = 1$ if the target class for the i^{th} instance is k, otherwise 0.

Cross entropy gradient vector for class k

$$\nabla_{\beta^{(k)}} J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{\pi}_k^{(i)} - y_k^{(i)})\mathbf{x}^i$$

Now, we can compute the gradient vector for every class, then use Gradient Descent (or any other optimization algorithm) to find the parameter matrix that minimizes the cost function.

Let's try a simple softmax regression with the *iris* dataset $\rightsquigarrow$

`P_SoftmaxRegression_simple_Iris.ipynb`