Learning from data: Logistic Regression

Christian Forssén[1]

Morten Hjorth-Jensen[2,3]

[1]Department of Physics, Chalmers University of Technology, Sweden
[2]Department of Physics, University of Oslo
[3]Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State University

Oct 11, 2020

1 Logistic Regression

In linear regression our main interest was centered on learning the coefficients of a functional fit (say a polynomial) in order to be able to predict the response of a continuous variable on some unseen data. The fit to the continuous variable $y^{(i)}$ is based on some independent variables $\boldsymbol{x}^{(i)}$. Linear regression resulted in analytical expressions for standard ordinary Least Squares or Ridge regression (in terms of matrices to invert) for several quantities, ranging from the variance and thereby the confidence intervals of the parameters $\boldsymbol{w}$ to the mean squared error. If we can invert the product of the design matrices, linear regression gives then a simple recipe for fitting our data.

Classification problems, however, are concerned with outcomes taking the form of discrete variables (i.e. categories). We may for example, on the basis of DNA sequencing for a number of patients, like to find out which mutations are important for a certain disease; or based on scans of various patients' brains, figure out if there is a tumor or not; or given a specific physical system, we'd like to identify its state, say whether it is an ordered or disordered system (typical situation in solid state physics); or classify the status of a patient, whether she/he has a stroke or not and many other similar situations.

The most common situation we encounter when we apply logistic regression is that of two possible outcomes, normally denoted as a binary outcome, true or false, positive or negative, success or failure etc.

1.1 Optimization and Deep learning

Logistic regression will also serve as our stepping stone towards neural network algorithms and supervised deep learning. For logistic learning, the minimization of the cost function leads to a non-linear equation in the parameters $\boldsymbol{w}$. The

optimization of the problem calls therefore for minimization algorithms. This forms the bottle neck of all machine learning algorithms, namely how to find reliable minima of a multi-variable function. This leads us to the family of gradient descent methods. The latter are the working horses of basically all modern machine learning algorithms.

We note also that many of the topics discussed here on logistic regression are also commonly used in modern supervised Deep Learning models, as we will see later.

1.2 Basics and notation

We consider the case where the dependent variables (also called the responses, targets, or outcomes) are discrete and only take values from $k = 0, \ldots, K-1$ (i.e. K classes).

The goal is to predict the output classes from the design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ made of n samples, each of which carries p features or predictors. The primary goal is to identify the classes to which new unseen samples belong.

Notice

We will use the following notation:

- $\boldsymbol{x}$: independent (input) variables, typically a vector of length p. A matrix of n instances of input vectors is denoted $\boldsymbol{X}$, and is also known as the *design matrix*.
- t: dependent, response variable, also known as the target. For binary classification the target $t^{(i)} \in \{0, 1\}$. For K different classes we would have $t^{(i)} \in \{1, 2, \ldots, K\}$. A vector of n targets from n instances of data is denoted $\boldsymbol{t}$.
- $\mathcal{D}$: is the data, where $\mathcal{D}^{(i)} = \{(\boldsymbol{x}^{(i)}, t^{(i)})\}$.
- $\boldsymbol{y}$: is the output of our classifier that will be used to quantify probabilities $p_{t=C}$ that the target belongs to class C.
- $\boldsymbol{w}$: will be the parameters (weights) of our classification model.

2 Binary classification

Let us specialize to the case of two classes only, with outputs $t^{(i)} \in \{0, 1\}$. That is

$$t^{(i)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \text{no} \\ \text{yes} \end{bmatrix}.$$

2.1 Linear classifier

Before moving to the logistic model, let us try to use our linear regression model to classify these two outcomes. We could for example fit a linear model to the default case if $y^{(i)} > 0.5$ and the no default case $y^{(i)} \leq 0.5$.

We would then have our weighted linear combination, namely

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}^T \boldsymbol{w} + \boldsymbol{\epsilon}, \tag{1}$$

where $\boldsymbol{y}$ is a vector representing the possible outcomes, $\boldsymbol{X}$ is our $n \times p$ design matrix and $\boldsymbol{w}$ represents our estimators/predictors.

2.2 Some selected properties

The main problem with our function is that it takes values on the entire real axis. In the case of logistic regression, however, the labels $t^{(i)}$ are discrete variables.

One simple way to get a discrete output is to have sign functions that map the output of a linear regressor to values $y^{(i)} \in \{0, 1\}$, $y^{(i)} = f(\tilde{y}^{(i)}) = \frac{\text{sign}(\tilde{y}^{(i)})+1}{2}$, which will map to one if $\tilde{y}^{(i)} \geq 0$ and zero otherwise. We will encounter this model in our first demonstration of neural networks. Historically it is called the *perceptron* model in the machine learning literature. This model is extremely simple. However, in many cases it is more favorable to use a *soft* classifier that outputs the probability of a given category. This leads us to the logistic function.

2.3 The logistic function

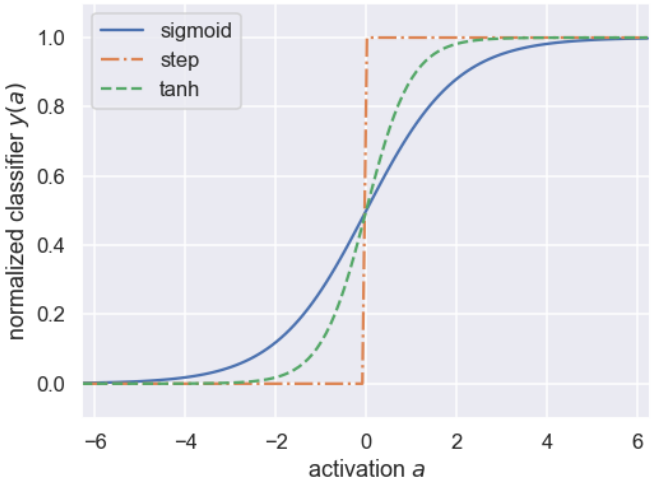
The perceptron is an example of a "hard classification" model. We will encounter this model when we discuss neural networks as well. Each datapoint is deterministically assigned to a category (i.e $y^{(i)} = 0$ or $y^{(i)} = 1$). In many cases, it is favorable to have a "soft" classifier that outputs the probability of a given category rather than a single value. For example, given $\boldsymbol{x}^{(i)}$, the classifier outputs the probability of being in a category k. Logistic regression is the most common example of such a soft classifier. In logistic regression, the probability that a data point $\boldsymbol{x}^{(i)}$ belongs to a category $t^{(i)} \in \{0, 1\}$ is given by the so-called *logit* function (an example of a S-shape or *Sigmoid* function) which is meant to represent the likelihood for a given event,

$$y(\boldsymbol{x}; \boldsymbol{w}) = y(a) = \frac{1}{1 + e^{-a}} = \frac{e^a}{1 + e^a},$$

where the so called *activation* $a = a(\boldsymbol{x}; \boldsymbol{w})$.

- Most frequently one uses $a = a(\boldsymbol{x}, \boldsymbol{w}) \equiv \boldsymbol{x} \cdot \boldsymbol{w}$.
- Note that $1 - y(a) = y(-a)$.
- The sigmoid function can be motivated in several different ways. E.g. in information theory this function represents the probability of a signal $s = 1$ rather than $s = 0$ when transmission occurs over a noisy channel.

2.4 Standard activation functions



Figur 1: The sigmoid, step,and (normalized) tanh functions; three common classifier functions used in classification and neural networks.

2.5 A binary classifier with two parameters

We assume now that we have two classes with $t^{(i)}$ being either 0 or 1. Furthermore we assume also that we have only two parameters w_0, w_1 and the predictors $\boldsymbol{x}^{(i)} = \{1, x^{(i)}\}$ defining the Sigmoid function. I.e., there is a single independent (input) variable x. We can produce probabilities from the classifier output $y^{(i)}$

$$p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w}) = y(a^{(i)}) = \frac{\exp\left(w_0 + w_1 x^{(i)}\right)}{1 + \exp\left(w_0 + w_1 x^{(i)}\right)},$$
$$p(t^{(i)} = 0|x^{(i)}, \boldsymbol{w}) = 1 - p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w}) = \frac{1}{1 + \exp\left(w_0 + w_1 x^{(i)}\right)},$$

where $\boldsymbol{w} = (w_0, w_1)$ are the weights we wish to extract from training data.

Note that $[p(t^{(i)} = 0), p(t^{(i)} = 1)]$ is a discrete set of probabilities that we will still refer to as a probability distribution.

2.6 Determination of weights

Among ML practitioners, the prevalent approach to determine the weights in the activation function(s) is by minimizing some kind of cost function using some version of gradient descent. As we will see this usually corresponds to maximizing a likelihood function with or without a regularizer.

In this course we will obviously also advocate (or at least make aware of) the more probabilistic approach to learning about these parameters.

Maximum likelihood. In order to define the total likelihood for all possible outcomes from a dataset $\mathcal{D} = \{(x^{(i)}, t^{(i)},)\}$, with the binary labels $t^{(i)} \in \{0, 1\}$ and where the data points are drawn independently, we use the binary version of the Maximum Likelihood Estimation (MLE) principle. We express the likelihood in terms of the product of the individual probabilities of a specific outcome $t^{(i)}$, that is

$$\mathcal{L} = P(\mathcal{D}|\boldsymbol{w}) = \prod_{i=1}^{n} \left[p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w})\right]^{t^{(i)}} \left[1 - p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w}))\right]^{1-t^{(i)}}$$

from which we obtain the log-likelihood

$$L \equiv \log(\mathcal{L}) = \sum_{i=1}^{n} \left(t^{(i)} \log p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w}) + (1 - t^{(i)}) \log \left[1 - p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w}))\right]\right).$$

The **cost/loss** function is then defined as the negative log-likelihood

$$\mathcal{C}(\boldsymbol{w}) \equiv -L = -\sum_{i=1}^{n} \left(t^{(i)} \log p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w}) + (1 - t^{(i)}) \log \left[1 - p(t^{(i)} = 1|x^{(i)}, \boldsymbol{w}))\right]\right).$$

The cost function rewritten as cross entropy. Using the definitions of the probabilities we can rewrite the **cost/loss** function as

$$\mathcal{C}(\boldsymbol{w}) = -\sum_{i=1}^{n} \left(t^{(i)} \log y(x^{(i)}, \boldsymbol{w}) + (1 - t^{(i)}) \log \left[1 - y(x^{(i)}, \boldsymbol{w})\right]\right),$$

which can be recognised as the relative entropy between the empirical probability distribution $(t^{(i)}, 1 - t^{(i)})$ and the probability distribution predicted by the classifier $(y^{(i)}, 1 - y^{(i)})$. Therefore, this cost function is known in statistics as the **cross entropy**.

Using specifically the logistic sigmoid activation function with two weights, and reordering the logarithms, we can rewrite the log-likelihood as

$$L(\boldsymbol{w}) = \sum_{i=1}^{n} \left[t^{(i)}(w_0 + w_1 x^{(i)}) - \log\left(1 + \exp\left(w_0 + w_1 x^{(i)}\right)\right)\right].$$

The maximum likelihood estimator is defined as the set of parameters (weights) that maximizes the log-likelihood (where we maximize with respect to w).

Since the cost (error) function is here defined as the negative log-likelihood, for logistic regression, we have that

$$\mathcal{C}(\boldsymbol{w}) = -\sum_{i=1}^{n} \left[t^{(i)}(w_0 + w_1 x^{(i)}) - \log\left(1 + \exp\left(w_0 + w_1 x^{(i)}\right)\right)\right].$$

Regularization. In practice, just as for linear regression, one often supplements the cross-entropy cost function with additional regularization terms, usually L_1 and L_2 regularization. This introduces hyperparameters into the classifier.

In particular, Lasso regularization is obtained by defining another cost function

$$\mathcal{C}_W(\boldsymbol{w}) \equiv \mathcal{C}(\boldsymbol{w}) + \alpha E_W(\boldsymbol{w})$$

where $E_W(\boldsymbol{w}) = \frac{1}{2}\sum_j w_j^2$ and α is known as the *weight decay*.

Question

Can you motivate why α is known as the weight decay? *Hint*: Recall the origin of this regularizer from a Bayesian perspective.

Minimizing the cross entropy. The cross entropy is a convex function of the weights $\boldsymbol{w}$ and, therefore, any local minimizer is a global minimizer.

Minimizing this cost function (here without regularization term) with respect to the two parameters w_0 and w_1 we obtain

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial w_0} = -\sum_{i=1}^{n}\left(t^{(i)} - \frac{\exp\left(w_0 + w_1 x^{(i)}\right)}{1+\exp\left(w_0 + w_1 x^{(i)}\right)}\right) = -\sum_{i=1}^{n}\left(t^{(i)} - y^{(i)}\right),$$

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial w_1} = -\sum_{i=1}^{n}\left(t^{(i)}x^{(i)} - x^{(i)}\frac{\exp\left(w_0 + w_1 x^{(i)}\right)}{1+\exp\left(w_0 + w_1 x^{(i)}\right)}\right) = -\sum_{i=1}^{n} x^{(i)}\left(t^{(i)} - y^{(i)}\right).$$

A more compact expression. Let us now define a vector $\boldsymbol{t}$ with n elements $t^{(i)}$, an $n \times 2$ matrix $\boldsymbol{X}$ which contains the $(1, x^{(i)})$ predictor variables, and a vector $\boldsymbol{y}$ of the outputs $y^{(i)} = y(x^{(i)}, \boldsymbol{w})$. We can then express the first derivative of the cost function in matrix form

$$\frac{\partial \mathcal{C}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\boldsymbol{X}^T(\boldsymbol{t} - \boldsymbol{y}).$$

2.7 A learning algorithm

Notice

Having access to the first derivative we can define an *on-line learning rule* as follows:

- For each input i, compute the error $e^{(i)} = t^{(i)} - y^{(i)}$.
- Adjust the weights in a direction that would reduce this error: $\Delta w_j = \eta e^{(i)} x_j^{(i)}$.

- The parameter η is called the *learning rate*.

This learning algorithm is a variant of *stochastic learning*.

Alternatively, one can perform *batch learning* for which multiple instances are combined into a batch, and the weights are adjusted following the matrix expression stated above. One can then repeat the training multiple times where each iteration consists of a *forward pass* (computing the outputs $\boldsymbol{y}$ given a set of weights $\boldsymbol{w}$) and *back-propagation* in which the gradient is computed and the weights are adjusted. At the end, one hopes to have reached an optimal set of weights.

Extending to more predictors. Within a binary classification problem, we can easily expand our model to include multiple predictors. Our activation function is then (with p predictors)

$$a(\boldsymbol{x}^{(i)}, \boldsymbol{w}) = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_p x_p^{(i)}.$$

Defining $\boldsymbol{x}^{(i)} \equiv [1, x_1^{(i)}, x_2^{(i)}, \ldots, x_p^{(i)}]$ and $\boldsymbol{w} = [w_0, w_1, \ldots, w_p]$ we get

$$p(t^{(i)} = 1 | \boldsymbol{w}, \boldsymbol{x}^{(i)}) = \frac{\exp\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)}\right)}{1 + \exp\left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)}\right)}.$$

3 Including more classes

Until now we have mainly focused on two classes, the so-called binary system. Suppose we wish to extend to K classes. We will then need to have $K-1$ outputs $\boldsymbol{y}^{(i)} = \{y_1^{(i)}, y_2^{(i)}, \ldots, y_{K-1}^{(i)}\}$.

Question
Why do we need only $K-1$ outputs if there are K classes?

Let us for the sake of simplicity assume we have only one independent (inout) variable. The activation functions for the outputs are (suppressing the index i)

$$a_1 = w_{1,0} + w_{1,1} x_1,$$

$$a_2 = w_{2,0} + w_{2,1} x_1,$$

and so on until the class $C = K - 1$ class

$$a_{K-1} = w_{(K-1),0} + w_{(K-1),1} x_1,$$

and the model is specified in term of $K-1$ so-called log-odds or **logit** transformations $y_j^{(i)} = y(a_j^{(i)})$.

Class probabilities: The Softmax function. The transformation of the multiple outputs, as described above, to probabilities for belonging to any of K different classes can be achieved via the so-called *Softmax* function.

The Softmax function is used in various multiclass classification methods, such as multinomial logistic regression (also known as softmax regression), multiclass linear discriminant analysis, naive Bayes classifiers, and artificial neural networks. Specifically, the predicted probability for the k-th class given a sample vector $\boldsymbol{x}^{(i)}$ and a weighting vector $\boldsymbol{w}$ is (with one independent variable):

$$p(t^{(i)} = k|\boldsymbol{x}^{(i)}, \boldsymbol{w}) = \frac{\exp\left(w_{k,0} + w_{k,1}x_1^{(i)}\right)}{1 + \sum_{l=1}^{K-1} \exp\left(w_{l,0} + w_{l,1}x_1^{(i)}\right)}.$$

It is easy to extend to more predictors. The probability for the final class is

$$p(t^{(i)} = K|\boldsymbol{x}^{(i)}, \boldsymbol{w}) = \frac{1}{1 + \sum_{l=1}^{K-1} \exp\left(w_{l,0} + w_{l,1}x_1^{(i)}\right)},$$

which means that the discrete set of probabilities is properly normalized.

Our earlier discussions were all specialized to the case with two classes only. It is easy to see from the above that what we derived earlier is compatible with these equations.