

Using neural networks for classification and regression problems

Eina B. Jørgensen, Anna Lina P. Sjur and Jan-Adrian H. Kallmyr

November 5, 2019

Abstract

1 Introduction

2 Theory

2.1 Logistic Regression

Classification problems aim to predict the behaviour of a given object, and look for patterns based on discrete variables (i.e categories). Logistic regression can be used to solve such problems, commonly by the use of variables with binary outcomes such as true/false, positive/negative, success/failure etc., or in the specific credit card case: *risky/non-risky*

As opposed to linear regression, the equation one gets as a result of minimization of the cost function by $\hat{\beta}$ using logistic regression, is non-linear, and is solved using minimization algorithms called *gradient descent methods*.

When predicting the the output classes in which an object belongs, the prediction is based on the design matrix $\hat{\mathbf{X}} \in \mathbb{R}^{n \times p}$ that contain n samples that each carry p features.

A distinction is made between *hard classification* - deterministically determine the variable to a category, and *soft classification* - determines the probability that a given variable belongs in a certain category. The latter is favorable in many cases, and logistic regression is the most used example of this type of classifier.

When using logistic regression, the probability that a given data point x_i belongs in a category y_i is given by the Sigmoid-function (or logistic function):

$$p(t) = \frac{1}{1 + e^{-t}} = \frac{e^t}{1 + e^t} \quad (1)$$
$$1 - p(t) = p(-t)$$

Assuming a binary classification problem, i.e. y_i can be either 0 or 1, and a set of predictors $\hat{\beta}$ the Sigmoid function (1) gives the probabilities with relation:

$$p(y_i = 0|x_i, \hat{\beta}) = 1 - p(y_i = 1|x_i, \hat{\beta})$$

The total likelihood for all possible outcomes $\mathcal{D} = \{(y_i, x_i)\}$ is used in the Maximum Likelihood Estimation (MLE), aiming at maximizing the log/likelihood function (2). The likelihood function can be expressed with \mathcal{D} :

$$P(\mathcal{D}|\hat{\beta}) = \prod_{i=1}^n \left[p(y_i = 1|x_i, \hat{\beta}) \right]^{y_i} \left[1 - p(y_i = 0|x_i, \hat{\beta}) \right]^{1-y_i}$$

And the log/likelihood function is then:

$$P_{\log}(\hat{\beta}) = \sum_{i=1}^n \left(y_i \log \left[p(y_i = 1|x_i, \hat{\beta}) \right] + (1 - y_i) \log \left[1 - p(y_i = 0|x_i, \hat{\beta}) \right] \right) \quad (2)$$

The cost/error-function \mathcal{C} (also called cross-entropy in statistics) is the negative of the log/likelihood. Maximizing P_{\log} is thus the same as minimizing the cost function. The cost function is:

$$\begin{aligned} \mathcal{C}(\hat{\beta}) = -P_{\log}(\hat{\beta}) = \\ - \sum_{i=1}^n \left(y_i \log [p(y_i = 1|x_i, \hat{\beta})] \right. \\ \left. + (1 - y_i) \log [1 - p(y_i = 0|x_i, \hat{\beta})] \right) \end{aligned} \quad (3)$$

Finding the parameters $\hat{\beta}$ that minimize the cost function is then done through derivation. Defining the vector \hat{y} containing n elements y_i , the $n \times p$ matrix \hat{X} containing the x_i elements, and the vector \hat{p} that is the fitted probabilities $p(y_i|x_i, \hat{\beta})$, the first derivative of \mathcal{C} is

$$\frac{\partial \mathcal{C}(\hat{\beta})}{\partial \hat{\beta}} = -\hat{X}^T(\hat{y} - \hat{p}) \quad (4)$$

This gives rise to set of linear equations, where the aim is to solve the system for $\hat{\beta}$. By introduction of a diagonal matrix \hat{W} with diagonal elements $p(y_i|x_i, \hat{\beta}) \cdot (1 - p(y_i|x_i, \hat{\beta}))$ the second derivative is:

$$\frac{\partial^2 \mathcal{C}(\hat{\beta})}{\partial \hat{\beta} \partial \hat{\beta}^T} = \hat{X}^T \hat{W} \hat{X} \quad (5)$$

With $\hat{x} = [1, x_1, x_2, \dots, x_p]$ and p predictors $\hat{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_p]$ the relation between likelihoods of outcome is:

$$\log \frac{p(\hat{\beta}\hat{x})}{1 - p(\hat{\beta}\hat{x})} = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \quad (6)$$

and $p(\hat{\beta}\hat{x})$ defined by:

$$p(\hat{\beta}\hat{x}) = \frac{e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}}{1 + e^{\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p}} \quad (7)$$

2.2 Gradient Descent Methods

2.3 Neural networks

The structure of a network

Neural networks, as the name suggests, are inspired by our understanding of how networks of neurons function in the brain. As can be seen in the example network in Figure 1, neurons are structured in layers. We always have a input and an output layer, in addition to a varying number of hidden layers. The input layer has as many neurons as there are input variables, while the output layer has one neuron for each output. How many neurons you have in the output layer depends on the specific problem. The number of neurons in each hidden layer, on the other hand, is not directly related to inputs or outputs, and must be decided in some other way.

As the diagram in Figure 1 suggests, the neurons in each layer are not connected with each other, but takes in inputs from the previous layer and passes on an output to the neurons in the next layer, as illustrated with arrows. This way, the inputs are fed through the network and processed, resulting in an output.

Forward feeding

Each neuron has one or multiple inputs, as illustrated with arrows in Figure 1. Each of these inputs has a weight associated with it. To clarify the notation used, let's take a look at the j th neuron in the l th layer. The weight associated with the input coming from the k th neuron in the previous layer is denoted as w_{jk}^l . In addition, each neuron has a bias associated with it, for the neuron in question denoted as b_j^l . Summing the weighted inputs and the bias, and feeding this to a function σ , gives the ac-

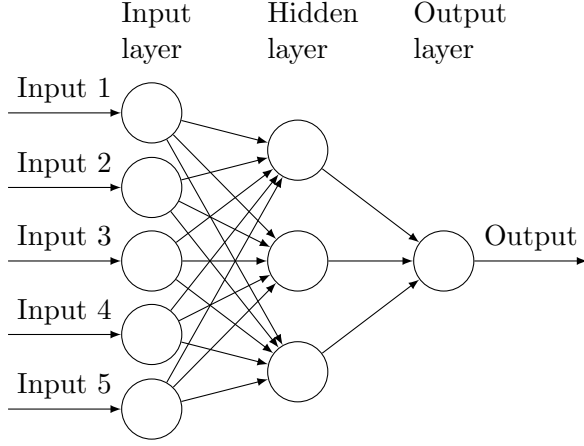


Figure 1: Schematic diagram of a neural network with five input neurons in the input layer, one hidden layer with three neurons and a single output neuron in the output layer.

tivation a_j^l :

$$a_j^l = \sigma \left(\left(\sum_k w_{jk}^l a_k^{l-1} \right) + b_j^l \right)$$

This activation is then fed forward as input to all the neuron in the next layer.

In matrix notation, the activation for the whole layer l can be written as

$$\mathbf{a}^l = \sigma \left(\mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l \right) \quad (8)$$

Here, \mathbf{a}^l and \mathbf{b}^l are vertical vectors containing the activations and biases of the l th layer, while \mathbf{w}^l is a matrix with elements w_{jk}^l , i.e. the j th row contains the weights of the inputs reaching the j th neuron.

Let's look at the activation function in Eq. (8) denoted with a σ . The use of σ as notation is not arbitrary, since the sigmoid function stated in Eq. (1) is often used.

The algorithm for forward feeding is given in Algorithm 1. Here L is the total number of layers.

```

Set activation  $\mathbf{a} = \text{input}$ ;
foreach  $l=1:L$  do
    | Compute  $\mathbf{z}^l = \mathbf{w}^l \mathbf{a} + \mathbf{b}^l$ ;
    | Compute  $\mathbf{a} = \sigma(\mathbf{z})$ ;
Return the activation  $\mathbf{a}$ ;

```

Algorithm 1: The forward feeding algorithm.

Training the network

Backpropagation

```

Compute  $\delta = \nabla_a C \odot \sigma'(z^L)$ ;
foreach  $l=L-1:2$  do
    | Compute  $\delta^l = ((\mathbf{w}^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ ;
Return ;

```

Algorithm 2: The backpropagation algorithm.

Implementation

3 Results

Figure 2

4 Discussion

5 Conclusion

References