Using Neural Networks and Logistic Regression for Classification and Regression Problems

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

November 7, 2019

Abstract

1 Introduction

2 Theory and methods

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/failiure 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 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 cathegory, and *soft classification* - determines the probability that a given variable belongs in a certain cathegory. The latter is favorable in many cases, and logistic regression is the most used example og this type of classifier.

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

$$\begin{aligned} p(t) &= \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t} \\ 1 - p(t) &= p(-t) \end{aligned} \tag{1}$$

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 funciton (2). The likelihood function can be expressed with $\mathcal{D}$:

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

And the log/likelihood function is then:

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

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 funciton is:

$$\mathcal{C}(\hat{\beta}) = -P_{\log}(\hat{\beta}) = -\sum_{i=1}^{n} \Big(y_i \log \Big[p(y_i = 1 | x_i, \hat{\beta}) \Big] + (1 - y_i) \log[1 - p(y_i = 0 | x_i, \hat{\beta})] \Big) \tag{3}$$

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 fittet probabilities $p(y_i | x_i, \hat{\beta})$, the first derivative of $\mathcal{C}$ is

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

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:

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

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

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

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

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

2.2 Gradient Descent Methods

The General Idea

With the gradient of $\mathcal{C}$ defined as in (4), we use this to find the minimum of the cost function. The basic idea is that by moving in the direction of the negative gradient of a function, we can move towards the value (in this case the β) that minimizes the function (in this case $\mathcal{C}(\beta)$)

This is done by repeating the algorithm

$$\beta_{j+1} = \beta_j - \gamma \nabla_\beta \mathcal{C}(\beta) \quad j = 0, 1, 2, ... \tag{8}$$

When a minimum is approached, $\nabla_\beta \mathcal{C}(\beta) \to 0$, and thus we can set a limit when $\beta_{k+1} \approx \beta_k$ given a certain tolerance, and the β which minimizes the cost funciton is found. γ is in this case called the *learning rate*, and is a parameter that must be tuned to each specific case in order to optimize the regression.

Stochastic Gradient Descent

In this project we use a stochastic version of gradient descent, which is an improvement upon the regular gradient descent (HOW??). This is done by expression the cost function (and thus also its gradient) as a sum

$$\nabla_\beta \mathcal{C}(\beta) = \sum_{i}^{n} \nabla_\beta c_i(\boldsymbol{x_i}, \beta), \tag{9}$$

and by only taking calculating the gradient of a subset of the data at the time. These subsets, called *minibatches* are of size $\boldsymbol{M}$, and the total amount is $\frac{n}{\boldsymbol{M}}$ where n is the amount of data points. The minibatches are denoted $\boldsymbol{B}_k$, with k = 1,2,...,$\frac{n}{\boldsymbol{M}}$.

Instead of a sum over all the the data points $i \in [1, n]$ we now in each step, sum over all the data points in the given minibatch $i \in \boldsymbol{B}_k$ where k is picked randomly with uniform proabaility from $[1, \frac{n}{\boldsymbol{M}}]$.

The stochastic and final version of (8) is therefore given by the algorithm

$$\beta_{j+1} = \beta_j - \gamma_j \sum_{i \in \boldsymbol{B}_k} \nabla_\beta c_i(\boldsymbol{x_i}, \beta) \tag{10}$$

An iteration over the total number of minibatches is commonly refered to as an *epoch.*

By using the stochastic gradient descent method (10) to minimize the cost function (3) we can this find the β values that give the most accurate classification, by doing *logistic regression.*

2.3 Neural networks

In this section, the equations used are based off the book by Nielsen (2015).

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.

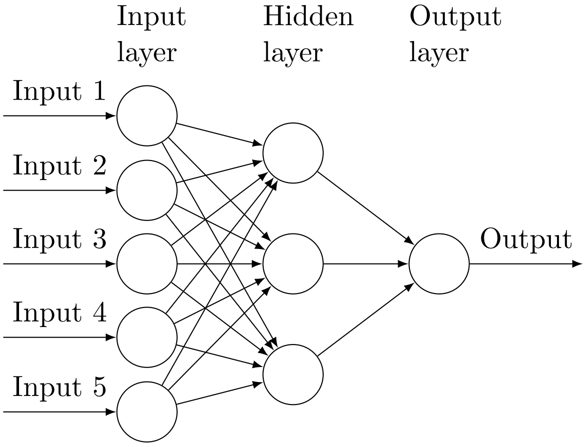

Figure 1: Schematic diagram of a neural network with five input neurons in the input layer, one hidden layer with tree neurons and a single output neuron in the output layer.

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 jth neuron in the lth layer. The weight associated with the input coming from the kth neuron in the previous layer is denoted as w^l_{jk}. In addition, each neuron has a bias associated with it, for the neuron in question denoted as b^l_j. Summing the weighted inputs and the bias, and feeding this to a function σ, gives the activation a^l_j:

$$a^l_j = \sigma\left(\left(\sum_k w^l_{jk} a^{l-1}_k\right) + b^l_j\right)$$

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

$$\boldsymbol{a}^l = \sigma\left(\boldsymbol{w}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) \tag{11}$$

Here, $\boldsymbol{a}^l$ and $\boldsymbol{b}^l$ are vertical vectors containing the activations and biases of the lth layer, while $\boldsymbol{w}^l$ is a matrix with elements w^l_{jk}, i.e. the jth column contains the weights of the inputs reaching the jth neuron.

Let's look at the activation function in Eq. (11) denoted with a σ. The use of σ as notation is not arbitrary, since the sigmoid function stated in Eq. (1) is often used. As we will see in the backpropagation algorithm, the sigmoid is a good choice for activation function, since a small change in the output can be propagated backwards, resulting in small changes in the weights and biases through the network.

With a basis in Eq. (11), the algorithm for forward feeding is given in Algorithm 1. Here L is the total number of layers.

```
Set $\boldsymbol{a}^1$ = input;
foreach $l$=2:L do
    Compute $\boldsymbol{a}^l$;
Set output to $\boldsymbol{a}^L$;
```

Algorithm 1: The forward feeding algorithm.

Note that the output will have values between 0 and 1, when the sigmoid function is used to compute the activations of all the layers. In a classification problem, this corresponds to the likelihood of an outcome. For example, in a classification problem with five classes, the network would have five output neurons, each representing a class. The final classification of an input would then be the class with the highest probability.

Backpropagation

When training the network, the goal is to find the weights and biases that minimize the cost function C. For a classification problem, the cost function is often given as

$$C = \frac{1}{2}\sum_i \left(a_i^L - y_i\right)^2 \tag{12}$$

To minimize Eq.(12), one can use Stochastic Gradient Decent, as described previously. But in order to use SGD, the derivatives of C must be computed, and it is here that backpropagation comes in. It can be shown that the derivatives are given as in Eq. (13). For a derivation of these expressions see APPENDIX?!?!?!.

$$\begin{aligned}
\delta^L &= \nabla_a C \odot \sigma'(z^L)\delta^l \\
\delta^l &= ((\boldsymbol{w}^{l+1})^T\delta^{l+1}) \odot \sigma'(\boldsymbol{z}^l) \\
\frac{\partial C}{\partial b^l_j} &= \delta^l_j \\
\frac{\partial C}{\partial w^l_{jk}} &= a^{l-1}_k \delta^l_j
\end{aligned} \tag{13}$$

These equation are the basis for the backpropagating algorithm, described in Algorithm 2.

```
Compute $\{\boldsymbol{a}^l\}_{l=1}^L$ with feed forward;
Compute $\delta^L$;
Set $\frac{\partial C}{\partial \boldsymbol{b}^L} = \delta^L$;
Compute $\frac{\partial C}{\partial \boldsymbol{w}^L} = \delta^L(\boldsymbol{a}^{L-1})^T$;
foreach $l$=L-1:2 do
    Compute $\delta^l$;
    Set $\frac{\partial C}{\partial \boldsymbol{b}^l} = \delta^l$;
    Compute $\frac{\partial C}{\partial \boldsymbol{w}^l} = \delta^l(\boldsymbol{a}^{l-1})^T$;
```

Algorithm 2: The backpropagation algorithm.

Adapting neural networks to regression

In order to adapt the network to regression, some changes must be made in Algorithm 1 and 2.

Implementation

2.4 Data sets

In this report, the default of credit card clients data set (Yeh and Lien, 2009) was used to study the performance and compare the logistic regression method and neural networks. A description of the attributes of the data set can be found in Yeh and Lien (2009). Upon inspection, it is notable that the data set contains a considerable amount of values different from their valid values as described by Yeh and Lien. Mostly, this apply for the categorical variables, where the given value does not correspond to any category. By removing data points with invalid values, as well as entries where the client does not have any history of past payments or bill statements, the data set is reduced from 30000 data points to 3792 points. As this is a considerable reduction of data points, a second reduced data set was constructed, where OGSÅ HVA VI GJORDE

In addition the default of credit card clients data set, data produced with Franke's function was applied to neural networks. For details on Frnake's function, see Jørgensen et al. (2019).

3 Results

Figure 2

4 Discussion

5 Conclusion

References

Eina B. Jørgensen, Anna Lina P. Sjur, and Jan-Adrian H. Kallmyr. Using linear regression for fitting terrain data. 2019.

Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473–2480, 2009.