Classification and Regression, from linear and logistic regression to neural networks

Johan Nereng

Department of Physics, University of Oslo, Norway

Nov 13, 2019

Abstract

——-

—-

Authors' comment: —–

1 Introduction

methods sections Results discussion of results conclusions

Scaling Logistic Regression Neural Networks Code Implementation Evaluating The Implementation

Regression vs classification crossvalidation mean-squared error and/or the R2 or the accuracy score (classification problems)

Assumes prior knowledge of fundamentals of machine learning, such as what a design matrix, predictors and response is.

A network such as this is a universal approximator, which means that it can approximate any suitably smooth with arbitrary accuracy [3][p.564].

2 Theory

2.1 Qualitative response modeling

In this project, both qualitative and quantitative response variables are modeled. In the case of quantitative [2][p.130]. [LOGREG INSTEAD OF LINREG - EXPLAIN WHY 0-1 etc]

2.2 Logistic Regression for two classes

Logistic regression models the probability that a qualitative response falls within a category. In the case of two mutually exclusive categories or classes, such as "True" ($Y = 1$) and "False" ($Y = 0$), this translates to modeling the relationship between the probability of $Y = 1$ given X as $P(Y = 1|X; \beta) = p(X; \beta)$, using the *logistic function* (1) [2][p.132], which due to mutual exclusivity, models $Y = 0$ as a function of X, or $P(Y = 0|X) = p(X'; \beta)$, as $1 - p(X; \beta)$.

$$p(Y = 1|X, \boldsymbol{\beta}) = p(\boldsymbol{X}; \beta) = \frac{e^{\boldsymbol{\beta X}}}{1 + e^{\boldsymbol{\beta X}}} \tag{1}$$

where $\boldsymbol{\beta} = [\beta_0, \beta_1, ...\beta_p]$, $\boldsymbol{X} = [1, X_1, X_2, ...X_p]$, and p is the number of predictors that the the binary response Y is modeled upon. When fitting the model (estimating the β coefficients) to a data set $\mathcal{D} = \{y_i, \boldsymbol{x}_i\}_{i=1}^n$, of n observations, the desired fit is the one where the predicted probabilities of $p(\boldsymbol{x}_i)$ most precisely corresponds to y_i. In other words, one seeks the model which has the highest probability of predicting the data set, which is achieved by utilizing the *maximum likelihood* method (2)[2][p.133].

$$P(\boldsymbol{\beta}) = \prod_{i:y_i=1} p(\boldsymbol{x}_i; \boldsymbol{\beta}) \prod_{i':y_{i'}=0} (1 - p(\boldsymbol{x}_{i'}; \boldsymbol{\beta})) \tag{2}$$

Taking the log of the maximum likelihood, one obtains the log-likelihood. The (reordered) negative log-likelihood constitutes the cost function, $\mathcal{C}(\hat{\beta})$ (3)

[1][p.120] (in Hastie et al, the log-likelihood is maximized) of the logistic regression, or the *cross entropy.* When minimized, this convex function yields the desired β estimates.

$$\mathcal{C}(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \big(y_i(\boldsymbol{\beta x}_i) - log(1 + e^{\boldsymbol{\beta x}_i})\big) \tag{3}$$

2.3 Gradient Decent

The minimization of a cost function may in some cases be carried out analytically, for example when using least squares error. For some cost function however, the analytic expression may be inconvenient to use. Factors such as the size of a matrix which requires inverting may be to computationally intensive, and in yet other cases, close form solutions are entirely unavailable. For problems such as these, numerical methods for obtaining the desired model fit is instead applied. One such method is the gradient descent (GD) (4) [3][p.247], also known as steepest descent. When utilizing GD, one makes an initial guess for $\boldsymbol{\beta}$, evaluates the gradient, $\boldsymbol{g}$ of the cost function in order to obtain an estimate closer to the desired minimizing value. This process is then iteratively repeated till the sufficient convergence of the coefficients is reached. In order to not overshoot the $k+1$'th estimate, a *learning rate* of η is applied to the gradient - which may also serve to lengthen the step in an iteration. The value of the learning rate can either be fixed, usually by trial and error, such that it works well for the problem at hand, or it may be adapted to each step.

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta_k \boldsymbol{g}(\boldsymbol{\beta}_k) \tag{4}$$

A common improvement on GD is the stochastic gradient descent (SGD), which one may read more about in literature such as Kevin P. Murphy's book on machine learning [3][p.262].

2.4 Neural Networks

For simplicity's sake, matrices are denoted without bold font or hats from here on forwards. The feed forward neural network (FFNN), or multi-layer perceptron model (MLP), is an artificial neuron network consisting of an ordered series of regression models with connected inputs and outputs [3][p.563]. The network consists of L layers, with a number of neurons (regression models), or nodes, in each layer. The neurons are not necessarily of the same type, as one can for example use logistic regression models for all nodes but the ones in L, which could be linear regression models if the problem is a regression problem. The outputs, a^1, of the M_1 nodes in layer $l = 1$ (one), the input layer, are fed to the nodes in layer $l = 2$, and so on, until the outputs, a^{L-1}, from layer $l = L-1$ is fed to layer $l = L$, the output layer. Outputs from layer L, a^L are the network's outputs, and predict the target(s) of the model. The layers in-between layer 1 and layer L, are deterministic functions of the input, and are called hidden layers.

Each node in each layer uses weights for each input, similar to the coefficients corresponding to the predictors described in the logistic regression subsection. In addition, each node has it's own *bias*, b, which regulates the output tendency of neuron. The sum of the weighted inputs and the bias, z, is usually called the activation of the neuron, not to be confused with it's output. The output is however a function of the activation, such that $a^l = f^l(z^l)$, where $f(z)$ is called an activation function.

2.4.1 Feed forward

The propagation of neuron outputs in the network, resulting in prediction, is called feed forward. The network used in this project is set up such that it takes n_{inputs} observations, of $n_{predictors}$ predictors, thus working through the entirety of the data in one go. As with other regression methods, the design matrix X ($n_{inputs} \times n_{predictors}$) is the input, which in the end outputs a matrix y ($n_{inputs} \times n_{targets}$) of predicted values for the $n_{targets}$ targets. By arranging the weights of layer l, w^l ($M_{l-1} \times M_l$), the activations of layer l is calculated by $z_l = Xw^l$ ($n_{inputs} \times M_l$). Taking $a^l = f^l(z^l)$ ($n_{inputs} \times M_l$), means that a^l holds the output for every neuron in l.

2.4.2 Cost function and backpropagation

Once a^L is obtained through the feed forward process, the network has made a prediction on the target(s), y, for each of the n_{inputs} observations. However, unless one has divine insights, initiating the weights and biases just so that $a^L \approx y$ with sufficient accuracy is unlikely. The quality of these predictions is quantified by the cost function $C(W, b)$. By using the backpropagation, the error of each layer,

3 Methods

3.1 Scaling

3.2 Matrix representation, and application of gradient decent

In order to write a program using logistic regression with gradient decent, matrix representation greatly greatly improves readability and effectiveness. The gradient, $\boldsymbol{g}$ of cost function, $\mathcal{C}(\boldsymbol{\beta})$ (3), may be expressed as [3][p.247]:

$$\boldsymbol{g}(\boldsymbol{\beta}) = \frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\hat{X}^T(\boldsymbol{y} - \boldsymbol{p}) \tag{5}$$

Where $\hat{X}$, is the design matrix, and $\boldsymbol{p}$ a vector with elements $p(y_i|x_i; \boldsymbol{\beta})$.

3.3 Code Implementation

3.4 Evaluating The Implementation

3.4.1 Preprocessing

Scaling, before and after Stratification, before and after

4 Results

5 Discussion of results

6 Conclusions

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer New York, New York, NY, second edition edition, 2009.

[2] Gareth James. *An Introduction to statistical learning : with applications in R.* Springer texts in statistics. Springer, New York, corrected at 8th printing 2017. edition, 2017.

[3] Kevin P. Murphy. *Machine learning : a probabilistic perspective.* Adaptive computation and machine learning. MIT Press, Cambridge, 2012.

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

Appendices