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

autoths: left right mult. confusing in looking at notes, example code and nielsens book 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 [4][p.564].

2 Methods

2.1 Qualitative response modeling

In this project, both qualitative and quantitative response variables are modeled. In the case of quantitative [3][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) [3][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)[3][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.

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

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) [4][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.

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

For logistic regression, $\boldsymbol{g}$, may be expressed as: [4][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 $p(y_i|x_i;\boldsymbol{\beta})$.

A common improvement on GD is the stochastic gradient descent (SGD), which involves splitting the data into min batches, then randomly selecting one batch at a time from which to compute the gradient. More detailed information is available in literature such as Kevin P. Murphy's book on machine learning [4][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 [4][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.

The remaining subsections on neural networks are based on the informative and well written book by Michael Nielsen book [5] (using a slightly different matrix indexation than what this project does), and lectures notes in FYS-STK4155 [2].

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 such that $a^L \approx y$ with sufficient accuracy is unlikely. The quality of these predictions is quantified by the cost function $C(w, b)$, which as indicated is a function of the weights and biases of network. The effect on prediction quality when adjusting w and b is thus measurable by the cost function, which, as in the case of the logistic regression, is to be minimized. In the case of FFNNs, this is done through backpropagation, which is the process of calculating the error of each layer, (6) and (7), and the gradient of the cost, ∇C, function w.r.t the biases (8) and

weights (9).

$$\delta_L = \nabla_a C \odot f'(z^L) \tag{6}$$
$$\delta_l = \delta^{l+1}(w^{l+1})^T \odot f'(z^l) \tag{7}$$
$$\frac{\partial C}{\partial b^l} = \delta^l \tag{8}$$
$$\frac{\partial C}{\partial w^l} = (a^{l-1})^T \delta^l \tag{9}$$

Having obtained the gradient ∇C, gradient decent is applied in order to calculate new weights and biases. Once the weights and biases are updated, a new feed forward process is initiated, followed by backpropagation and corresponding adjustments of parameters until the cost function is sufficiently minimized or the number of iterations (feed forward + backpropagation) is reached.

2.4.3 Neural network for classification

In the logistic regression case, this project did not use one hot encoding - using cross entropy ensures equal importance of both target options when training. For the NN implementation however, one-hot encoding is used, along with cross entropy (3) as the cost function. As the one-hot encoding means that the network is expected to output two values, representing the likelihood of each of the two categories for each data point, the activation function of the output layer is the softmax function (10) (output of neuron $j = 1, 2$ in L), which normalizes the sum of the outputs to one.

$$a_j^L = \frac{e^{z_j^L}}{\sum e^{z^L}} \tag{10}$$

Using the cross entropy as the cost function, sigmoid for hidden layer(s), and softmax as the activation for layer L, means that the backpropagation equations (6)-(9) for the two last layers of network are:

$$\delta_L = a_L - y$$
$$\delta_{L-1} = \delta_L (w^L)^T \odot a_{L-1} \odot (1 - a_{L-1})$$
$$\frac{\partial C}{\partial b^L} = \delta^L$$
$$\frac{\partial C}{\partial w^L} = (a^{L-1})^T \delta^L$$
$$\frac{\partial C}{\partial b^{L-1}} = \delta^{L-1}$$
$$\frac{\partial C}{\partial w^{L-1}} = (a^{L-2})^T \delta^{L-1}$$

It's worth noting that in the output error, when using softmax with cross-entropy, the $f'(z)$ term cancels with parts of the derivative of C w.r.t a.

2.4.4 Neural network for regression

In this project, the neural network for regression is set up using the quadratic cost, with $tanh(z)$ as activation function in hidden layer(s), and with $f(z^L) = z^L$, the identity activation, as activation for the output layer. Due to the change in activation functions, the backpropagation changes slightly, however, as the cost function is also changed, the output error remains the same.

$$\delta_{L-1} = \delta_L (w^L)^T \odot (1 - tanh(z_{l-1}^2))$$

2.5 Implementation, testing, and evaluation

Logistic regression for classification is implemented in it's own program using python 3.7.5. The program also uses functionalities from scikit-learn [6] for splitting data into test and training sets, metrics such as producing a confusion matrix, and scaling functions. A neural network for classification is also implemented, building on the template provided in the lecture slides on neural networks from the course FYS-STK4155 [2]. The neural network uses one hidden layer, and activation functions and cost functions as described in the subsection on neural networks for classification.

Initial testing of the logistic regression model is carried out on data from the breast cancer data set available through scikit-learn, which contains 569 data points with 30 predictors and targets (212 WDBC-Malignant, 357 WDBC-Benign), using both GD and SGD. The program features an option to plot (set plot=True in function call for GD or SGD) the cost function vs. number of iterations and epochs (time) for GD and SGD respectively - which shows that the cost function decreases with iterations/epochs, thus indicating a working implementation. Having established implementation integrity, the project now moves on to evaluation.

Evaluation of accuracy The accuracy score of a classifier (11), where I is the indicator function, reflects the number of correctly predicted targets t_i - where a perfect classifier will have a score of 1.

$$\text{Accuracy} = \frac{\sum_{i=1}^{i=n} I(t_i = y_i)}{n} \tag{11}$$

area ratio score

2.6 Preprocessing

remove outliers, one hot encoding categories, stratified split, Scaling, smote (upsampling of y), train, predict

2.7 Scaling

In order to write a program using logistic regression with gradient decent, matrix representation greatly greatly improves readability and effectiveness.

2.8 Code Implementation

2.9 Evaluating The Implementation

2.9.1 Preprocessing

Scaling, before and after Stratification, before and after

3 Results

1 shows the gains chart produced by the logistic regression program on the credit card data with no other pre-processing of the data than one-hot encoding of categorical predictors with values other than 0/1. As the plot suggests, the model performs the same as a random classifier.

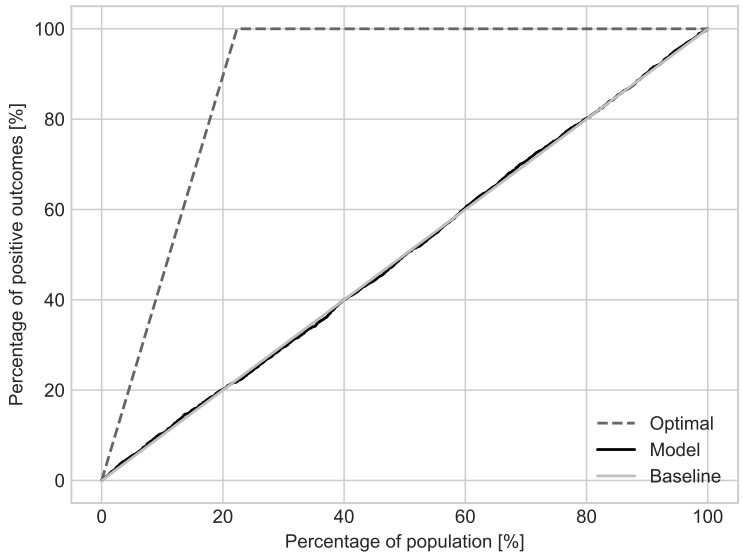

Figure 1: Gains plot for logistic regression using GD with 5000 iterations and $\eta = 0.1$ on credit card data, no pre-processing of data other than one-hot encoding. Accuracy score=0.78, Area ratio score=0.00

4 Discussion of results

4.1 Implementation, testing, and evaluation

The accuracy score of the logistic regression model on the raw credit card data (with one-hot encoding) is 0.78, a deceivingly high score, considering that the area ratio score is 0.00. This indicates that that the model is classifying all the test data as 0 (1 being defaulting on payment) - which the confusion matrix (see appendix 1.) confirms. The discrepancy in evaluation scores shows hat accuracy is not necessarily a meaningful measurement on model performance, especially when the whole goal of the operation is to, in the case of the credit card data, evaluate who will default. It is not unreasonable to imagine that one would prefer a model which produces a few false default flags over on which produces too few.

5 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] Morten Hjorth-Jensen. Lectures notes in fys-stk4155. data analysis and machine learning: Neural networks, from the simple perceptron to deep learning, Oct 4 2019.

[3] 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.

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

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

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Appendices

Appendix 1.

Confusion matrix for linear regression classifier on un-processed credit card data.

$$\boldsymbol{X} = \begin{bmatrix} tn = 7591 & fp = 0 \\ fn = 2178 & tp = 0 \end{bmatrix} \tag{12}$$