Project 2 - FYS-STK4155

Sigurd Hylin

Abstract

In this project we use logistic regression and an artificial neural network to perform classification on credit card debt default data. In the logistic regression setting, I explore forward selection and backward elimination using AIC as selection criterion, for model selection. For the neural network I use grid search to identify the optimal hyperparameter values. To evaluate and compare the model performances, the data is split into training and test sets. The logistic regression and neural network models have been implemented in two Julia modules. The code for these modules and the Jupyterlab notebooks I use to run the models can be found at https://github.com/1991sig/Project-2

Contents

1 Introduction

The credit card default data set, https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients, contains 30 000 observations of several dependent variables, and has a binary dependent variable where 1 indicates that the person defaulted on his/hers credit card debt in the following month. Some variables are continuous, like the debt limit and the payment and billing history, others are categorical. The data stems from 2005 and tracks individuals' payment and billing history for 6 months from April 2005 to September 2005, and the interest is in using these data including the individuals' debt limits, marital status, gender, education, and payment delay history over the same 6 month period to determine whether or not the person will default in the next month (October 2005).

2 Code

All code as well as notebooks to run the models can be found at https://github.com/1991sig/Project-2.

Having launched in 2012, Julia is a relatively new programming language, and was created with an aim to combine the best of other programming languages, like the speed of C, the usability of Python, and the powerful linear algebra capabilities of MatLab, among others [1]. I decided to use Julia in this project, since it has a lot of interesting features which I wanted to learn more about, and since it is easy to write modular code which can easily be extended later on. In addition, it is very geared towards linear algebra, which is definitely useful.

[1] Gina Helfrich Ph.D, "Announcing Julia 1.0", https://numfocus.org/blog/announcing-julia-1-0

Before this project, I had already started developing a module for Generalized Linear Models, which I have therefore continued to develop for this project. Currently, I have only finished implementing the functionality to perfom logistic regression.

The other module I have implemented is a simple neural network module, which currently only has the ability to use a feed forward architecture.

2.1 GLM Module

To develop this module, I have drawn inspiration from the "official" GLM.jl from the StatsKit package in the way that I have organized things. Therefore, some of the structure and naming of the code may share some similarities. In terms of the functionality I wanted to incorporate, I aimed to have at least some of the capabilities that R offers.

2.1.1 Theory

GLM's consist of a random component, a linear predictor, and a link function to allow modeling dependencies where the random component is not necessarily from the normal distribution.

The random components, $\mu_i = E(Y_i|X_i)$, in a GLM model must stem from the exponential family of distributions, and is the conditional mean of the dependent variable given the observation of the independent variable(s).

The linear predictor is η such that $\eta = X\beta$, where X is the matrix containing the observations of the independent variable(s).

The link function in a GLM bears it's name due to the fact that it connects/links the conditional mean and the linear predictor η together trough a function. I.e. $\eta_i = g(\mu_i)$ and $\mu_i = g^{-1}(\eta_i)$ for a specific function g, which is then the link function.

The exponential family of distributions has the neat property that the distributions which are a part of it can all be formulated in a general way.

[2]

2.1.2 Code

One of the distributions in the exponential family of distributions is the Binomial distribution. `Binomial.jl` contains the general abstract definition of a binomial random component/distribution and functions for calculating variance, likelihood, loglikelihood, and deviance.

`Links.jl` contains the abstract definitions of link functions for the binomial distribution, and specifies the inverse link functions also.

`LinearPredictor.jl` specifies the data matrix, and generates the starting values of β before the model is fitted.

`Model.jl` contains the struct for a GLM model, i.e. where one ties together the random component, the linear predictor, and the link function.

A full GLM model struct contains the model, the "fit" and a flag indicating whether the model has been fitted or not. The fit struct is implemented in `Fit.jl`, and contains - y, the response vector - β, $\hat{\beta}$ the estimated coefficients vector - $\mathbf{D}$, the residual deviance vector - SE, the standard errors of β-hats - η, vector with $\hat{\eta} = X\hat{\beta}$ - μ, μ-hat vector - DoF, the Degrees of Freedom - AIC, the Akaike Information Criterion value for the fitted model

[2] P. McCullagh & J.A. Nelder, "Generalized Linear Models, 2nd edition", 27-28

2.1.2.1 Fisher's Scoring Algorithm

From introductional statistics, we are familiar with the principles of maximum likelihood estimation, and how maximizing the log-likelihood is equivalent to maximizing the likelihood itself.

Let the log-likelihood be $l(\beta; y) = \sum log\mathcal{P}(y_i)$

Taking the derivative, we obtain the Score Equations: $U(\beta) = \nabla l(\beta; y) = (\frac{dl(\beta; y)}{d\beta_1}, \cdots, \frac{dl(\beta; y)}{d\beta_p})$

The Hessian of the log-likelihood, $\nabla^2 l(\beta; y)$, is a matrix and the Fisher Expected Information matrix, is equal to the expectation of this negated Hessian matrix: $I(\beta) = E[-\nabla^2 l(\beta; y)]$

Fisher's Scoring Algorithm is defined as: $\hat{\beta}^{(t+1)} = \hat{\beta}^{(t)} + I(\hat{\beta}^{(t)})^{-1} U(\hat{\beta}^{(t)})$

I will not derive the whole expression here, but when we use a canonical link function, such as the logit in the case of a binomial random component, Fisher's Scoring Algorithm and Newton's Method are equivalent.

I have implemented this fitting algorithm in the file `FisherScoring.jl`.

2.2 NeuralNetworks Module

Most of this code is based on the code from the lectures.

`ActivationFunctions.jl` currently only contains the sigmoid/logistic function, but I have more planned for the future.

`Common.jl` contains the structure for a neural network model, which in my implementation consists of an "architecture" (e.g. feedforward with 1 layer), the parameters of the model, and a flag indicating whether or not the model has been fitted yet.

`NeuralNet.jl` currently only contains the setup for a one-layer feedforward model.

`Parameters.jl` contains the parameters struct with constructors. The parameters I keep in this struct are: - W^H,the hidden layer weights - b^H, the hidden layer bias term - W^O,the output layer weights - b^O, the output layer bias term - z^H,the hidden layer calculated values, i.e. $z^H = XW^H + b^H$ - a^H, the hidden layer output values after activation function is applied to zˆH, i.e. $a^H = g(z^H)$ - z^O the output layer calculated values, i.e. $z^O = a^H W^O + b^O$ - P, the probabilities after applying the logistic function/multinomial function depending on number of categories in the response.

`Train.jl` contains the code to train the neural network. FeedForward! is a mutating function that performs a forward pass and applies the results to the parameters it takes directly. BackPropagation! performs the backpropagation algorithm by first calculating the loss of the hidden layer and the output layer. The cost function used is the square loss: $C(\theta) = \frac{1}{2}\sum(p_i - y_i)^2$.

3 Methods & Implementations

3.1 Implementations

In `Credit Card Data.ipynb` all the cleaning and processing of the data is performed.

In `Logistic Regression - Credit Card Data.ipynb`, I fit the logistic regression model to the data and evaluate the results.

In `Neural Network - Credit Card Data.ipynb`, I use a neural network to predict on the data.

3.1.1 Comparisons

I have tested my implementations against R's glm methods, and these perform more or less the same.

I have not tested my implementations of the neural network using TensorFlow.jl yet.

4 Results

Table 1: Part 1

Model	Accuracy Train	Accuracy Test
Logistic Regression full model	0.81695	0.81767
Logistic Regression model 2	0.81723	0.81844
Neural Network - pre grid search	0.23486	N/A
Neural Network - grid search	0.69943	0.70433

The logistic regression model performed better in this case. However, I have not had the time to build the same familiarity with neural networks as I have with GLM's, so therefore I haven't been sure how this model could have been optimized to perform better. As is often the case, a more complex solution to a problem does not necessarily yield better results.

In this case, where we study credit card data obtained from a bank, it is not likely that a solution based on neural networks would be the optimal choice even if it had performed better, since we lose the inferential apparatus which is readily available with other methods. For a model to provide actionable insight to a bank, it would likely have to offer a certain level of interpretability. Logistic regression is often preferred in settings such as these, since one can easily interpret how the values of the input variables have an effect on the predicted probabilities, and we also have the ability to identify coefficients/variables which have no statistically significant relation to the outcome variable.

Interestingly, for both of the models, the performance is every so slightly better on the test set compared to the training set. This could indicate that there may be one or more outliers in the training set, which the fitted models are not able to correctly label. For the logistic regression model, R has some functionality to identify this type of stuff, but I have not had the time to develop the same functionality in my package.

I have not managed to complete my code for part 2 in time, so I do not have anything that I want to show for the second part of the assignment.