Logistic regression and neural networks

Eirik Nordgård
Geophysical Institute, University of Oslo

2019
October

Abstract

Here you can write your abstract

All material for this project may be found on `https://github.com/eirikngard/FYS-STK4155/tree/master/Project_2`

1 Introduction

In this project credit card data with 23 features is used to determine whether the clients will default or not default their debt. The data output thus is binary, 1 for default and 0 for not default, and prediction these includes solving a classification problem. In this project this is done using logistic regression and neural networks. Overall, the main goal is to assess which model best predicts the outcome based on the 23 features in the data, and how adjusting parameters of these models affect the result. The performance is measured by the accuracy score, which simply is a measure of how often the given model predict correct outcome. Accuracy score ranges from 0 to 1, 1 being a perfect classifier.

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

2 Theory

2.1 Logistic regression

Logistic regression is used to solve classification problems. These are problems concerning outcomes, y_i, in form of discrete variables. Commonly the classification problems in question has two possible outcomes, true or false. Using the credit card data as an example, the two outcomes would be if a client would pay his/hers debt or not. This type of outcome is often calleed binary outcome, and can easily be programmed as 1 or 0.

The dependent variables (outcomes) y_i are discrete, ranging from $k = 0, ..., K-1$ where K is classes. The main goal is to predict the output classes from a designmatrix $\hat{X} \in \mathbb{R}^{nXp}$ made of n samples carrying p features/predictors. The simplest output is perhaps a binary output, only having values 0 or 1 meaning yes/no, true/false, pay/dont pay etc. From there it is desired to identify the classes to which new unseen samples belong.

A descrete output can be obtained in several ways, but the simplest may be to have a sign function, which maps the output of a linear regressor to values $0, 1, f(a) = sign(a) = 1$ if a

$\geq$ 0 and 0 if otherwise. Although this model is very simple, in many cases it might be convenient to know the probability of an output belonging a given caegory rather than a single value. This is done using the *logistic function*, and is often called a "soft classifier". The classifier outputs the probability of x_i belonging to a category $y_i = \{0, 1\}$. In this case the classifier is given by a *Sigmoid* function, often reffered to as a likelihood function. The Sigmoid function $p(y)$ is given by[2]

$$p(y) = \frac{1}{1+e^{-y}} = \frac{e^y}{1+e^y} \tag{1}$$

and looks like this

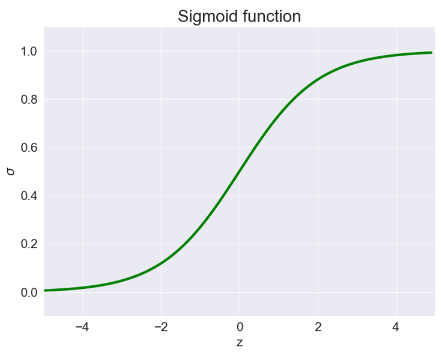

Figure 1: Plain Sigmoid as example of likelihood function

Like other likelihoods, eq. (1) here we have that the likelihood $p(y_i = 0) = 1 - p(y_i = 1)$, always resulting in a summed likelihood of 1. Assuming two classes y_i being either 0 or 1, we have

$$y_i = \beta_0 + \beta_1 x_i + ... + \beta_n x_i \tag{2}$$

Used in in eq.(1), this gives

$$p(y_i = 1 \mid x_i, \hat{\beta}) = \frac{e\beta_0 + \beta_1 x_i}{1 + e\beta_0 + \beta_1 x_i} \tag{3}$$

and

$$p(y_i = 0 \mid x_i, \hat{\beta}) = 1 - p(y_i = 1 \mid x_i\hat{\beta}) \tag{4}$$

Eq. (3) and eq. (4) can then be used to find the maximum likelihood of an event. The product of all individual probabilities of a specific outcome y_i is used to obtain a log-likelihood function, which in turn gives the cost function

$$\begin{aligned} C(\hat{\beta}) = \sum_{i=1}^{n} \big(y_i \, log \, p(y_i = 1|x_i, \hat{\beta}) \\ + (1 - y_i) \, log \left[1 - p(y_i = 1|x_i, \hat{\beta}\right]\big) \end{aligned} \tag{5}$$

Eq. (5) is called the *cross entropy*. Being convex, a local minimizer of this function will also be a global minimizer. Hence, minimizing eq. (5) with respect to each β gives

$$\frac{\partial \hat{\beta}}{\partial \beta_0} = -\sum_{i=1}^{n} \big(y_i - \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}\big) \tag{6}$$

and

$$\frac{\partial \hat{\beta}}{\partial \beta_1} = -\sum_{i=1}^{n} \big(y_i x_i - x_i \frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}\big) \tag{7}$$

It is worth mentioning that far from all cost functions are convex. Many cost functions used in machine learning are in fact non-convex and of high dimensionality. Compressing eq. (6) and eq. (7) we get the following expressions for the first derivative. This is the gradient used in the gradient descent method explained later.

$$\frac{\partial C(\hat{\beta})}{\partial \hat{\beta}} = -\hat{X}^T(\hat{y} - \hat{p}) \tag{8}$$

where $\hat{y}$ is a vector with n elements y_i, the nxp matrix $\hat{X}$ containing the x_i values and the vector $\hat{p}$ is the fitted probabilities $p(y_i \mid x_i, \hat{\beta})$. [1] [2].

[1]here is a paragraph on Hessian. check if i use this later

2.2 Gradient Descent Method

The idea of a gradient descent (GD) method is that a function $F(x)$, with $x = x_1, ..., x_n$, decreases fastest if one goes from x in the direction of the negative gradient $-\nabla F(x)$. If

$$x_{v+1} = x_v - \gamma_v \nabla F(x_v) \tag{9}$$

where γ_v is called learning rate or step length, in this case $\gamma_v > 0$. The gradient used in eq. (9) is obtained by eq. (8). For small enough γ_v we will then have $F(x_{v+1} \geq F(x_v)$, meaning we are always moving towards smaller values and eventually a minimum of F [2].
Choosing appropriate learning rate is important. If chosen too small the method will converge very slowly, and chosen too large may result in unpredictable behaviour. The first step doing the GD is to make a guess x_0 for a minimum of F. Note that the minima is a global minima only if the function is a convex function.

Logistic regresseion er som et neuralt nettverk uten layers. [2] [2].

2.2.1 Standard/Steepest Gradient Descent

I have used this as the first gradient descent. Iterate 1000, calculated the gradient, minimizing the cost function. This iteration outputs betas, used to make prediction of y. Thereafter y is used in the sigmoid function to classify the data og default or not. Making a threshold of 0,5, where sigmoid output of less than 0,5 is 0, and above is 1. Then I count how many of the predicted values match with the values of the data y (last column in data set) to get the accuracy score for this gradient descent.

2.2.2 Stochastic Gradient Descent

Write about epocs and minibatches. Why we use them, how we use them and maby how to optimize the number of these?

Anyhow, I have used stochastic gradient descent with linear regression as well, and found it to be around 1

[2]Make sure p (sigmoid) is defined somewhere above

2.3 Artificial Neural Networks

Artificial Neural Networks (ANN) is a computational system that by training on examples can learn to perform tasks. Increasing the number of examples the ANN can learn from will increase the accuracy of the tasks performed. It is called artificial neural network because it is supposed to mimic a biological system such as a brain, where neurons interact by sending signals in the form of mathematical functions. The ANN is constructed in layers containing an arbitrary number of neutrons, which in turn is connected by a weight variable. Similarly to an actual brain, the neurons in a ANN can only communicate with each other if the incoming signal exceeds a certain threshold value. If the signal is "strong" enough, an output is sent forward, but if this activation threshold is not reached the neuron remain inactive, providing zero output[2].

The neurons described above may be on different forms. One type of artificial neuron is called the *perceptron*, and it has the general form

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b_i) = \sigma(z) \tag{10}$$

This works by taking n binary inputs $x_1, x_2, ..., x_n$ and producing one single binary output. The importance of each binary input is represented by n weights, $w_1, w_2, ..., w_n$. The binary output is determined by if the weighted sum of inputs is less or greater than a threshold (bias) value. Thus,

$$output = 0 \; if \sum_i w_i x_i \; \leq \; threshold$$
$$output = 1 \; if \sum_i w_i x_i \; \geq \; threshold$$

This is the equivalent to whether the neuron is activated or not. Described in another way, the perceptron makes a decision (0 or 1) by weighing up arguments pro or con the decision. The network may also have several layers of neurons, each taking the output of the "previous" layers as input. These layers are called *hidden layers*, and allows for very complicated decision making networks. It is apparent that logistic regression in a way is a single layered neural network. **For understanding, the next paragraph states an simple example.**

Imagine you want to visit the cabin. Two inputs may be if the weather is nice or not, and if you get to see yo grandmother or not. The output is simply to go or not to go to the cabin. Then the question is what weighs most, the weather or you being able to visit your grandmother. Say the threshold is 3. The weight for nice weather is 2 and the weight for visiting the grandmother is 2. Then you actually require both nice weather and that you get to see you grandmother to go to the cabin. If the weather is nice, but you cant see your grandmother, we have $1*2+0*2 = 2 < 3$ and the threshold is not reached. Similarly if the weather is bad, but you can see your grandmother we have $0 * 2 + 1 * 2 = 2 < 3$. But as both requirements are fulfilled we have $1 * 2 + 1 * 2 = 4 > 3$ and the threshold is reached, giving the binary output of 1, which in this case means you are going to the cabin. IF the threshold were lowered to 1, you would go to the cabin even though only one of the inputs were fulfilled. Thus, changing the weights and the threshold yields different decision making.

If the perceptron where to make a wrong classification, a problem arises. The desired path of action would be to make small changes is the biases or weights of the network and check if it then made the correct classification. If not, you would repeat this process making small changes until the classification succeeds. However, making small changes in biases or weights is actually resulting in very large changes in the output. This problem is solved using another type of neuron, namely the *sigmoid.*

This neuron functions in the same way as the perceptron, but with different input and output. Now the inputs can be any value between 0 and 1. And instead of having output of 0 or 1, the sigmoid has output $\sigma(w * x + b)$, where b is the bias and σ is the sigmoid function defined by eq. (1). Hence the out of the sigmoid will be [4]

$$output = \frac{1}{1 + e^{-\sum w_i x_i - b}} \tag{11}$$

2.3.1 Backpropagation (ANN)

Backpropagation is an algorithm computing the gradient of the cost function. This quantity tells us what the partial derivatives with respect to both the bias and the weights are, and may be very useful if you want to know how fast the cost is changing when the bias or weights are changing. This actually allows us to get information on how the behaviour of the entire network changes when the bias or weights are changed. Like before, we have

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{12}$$

where a is the activation(referred to as output later), b is the bias, w is the weights, l is the layer number and σ is a function to every element in the vector$z^l = w^l a^{l-1} + b^l$. This way the activation in one layer is related to the activation in the previous layer. Thus we have $a^l = \sigma(z^l)$.(Sigma is the same as f above i think.SJEKK[3]). z^l is called the weighted input to the neurons in layer l.

Then, for backpropagation to work, it is necessary to assume the the cost function can be written as an average $C = \frac{1}{n}\sum_x C_x$ over cost functions C_x for individual training points, x. Also, the cost function must be able to be written as a function of the outputs from the neural network (SIDE 42 NIELSEN).

While computing the derivatives of the cost $\frac{\partial C}{\partial b^l}$ and $\frac{\partial C}{\partial w^l}$, an error will occur. This error is just referred to as *error*, and is written δ^l_j in the j-th neuron in the l-th layer. For later notation, the k-th neuron is belongs to the (l-1)-th layer. The quantity $\frac{\partial C}{\partial z^l_j}$ provides a good estimate of this error (EXPLENATION for why IN page 44 Nielsen), and from here

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \tag{13}$$

of neuron j in layer l. Backpropagation then relates δ to the derivatives of the cost. Rewriting eq. (13) gives

$$\delta^l_j = \frac{\partial C}{\partial a^l_j}\sigma(z^l_j) \tag{14}$$

which vectorized is

$$\delta^l = \nabla_a C \odot \sigma(z^l) \tag{15}$$

In eq. (15) $\nabla_a C$ expresses the rate of change of C with respect to the activation output. Further rewriting gives

$$\delta^l = (a^l - y) \odot \sigma(z^l) \tag{16}$$

[3]sjekk om f og sigma er det samme

which in terms of the error in the next layer, δ^{l+1}, is

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma(z^l) \tag{17}$$

The name *backpropagation* arises from eq. (17). Here the error is moved backward through the activation function in layer l, giving the error in the weighted input to layer l.

Eq. (17) finally allows us to obtain equations for the partial derivatives with respect to the bias and the weights, respectively

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{18}$$

simplified to

$$\frac{\partial C}{\partial b} = \delta \tag{19}$$

and

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{20}$$

simplified to

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out} \tag{21}$$

Eq.(20) has a very useful consequence. When the activation a is small, the derivative will also be small. This means that the gradient changes fairly little during a gradient descent, meaning the weight *learns slowly.*

The resulting backpropagation algorithm then looks something like this:

1. **Input** x: Find corresponding activation a^l for the input layer.
2. **Feedforward:** For $l = 2, 3, ..., L$ compute z^l (EQ HASENT NUMBER YET) and a^l (eq.(12)).
3. **Output error δ^L:** Compute vector $\delta^L = \nabla_a C \odot \sigma(z^L)$

4. **Backpropagate the error:** Compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma(z^l)$ for each l.

5. **Output:** Gradient of the cost function: $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

[4]

2.3.2 Measuring performance

Maybe include some words on how i measure the performance of the models. Have used accuracy score. Possibly use area under curve and F1 score.

2.4 Data description

The data set used in this analysis is credit card data from credit card holders in Taiwan. In total the dataset contains 23 variables, which is used to employ a response variable, *default payment* with value *1 = default = not pay, 0 = not default = pay.* The dataset contains the following information:

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.

- X2: Gender (1 = male; 2 = female).

- X3: Education (1 = graduate school; 2 = university;3 = high school; 4 = others).

- X4: Marital status (1 = married; 2 = single; 3 = others).

- X5: Age (year).

- X6–X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005;...;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: 1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; ...; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

- X12–X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005;...;X17 = amount of bill statement in April, 2005.

- X18–X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005;...;X23 = amount paid in April, 2005.[5]

All of these variables determines the probability of whether a client defaults or not on his/her debt the following month.

3 Method

3.1 Data Preparation

Not processed in any way, the dataset is roughly 30000 × 23 data points. Within these values are several illegal ones, not mentioned in the data description. Thus, the following data are removed from the set:

$$EDUACTION = 0, 5, 6$$

$$MARRIGE = 0$$

$$PAY\_X = -2$$

$$BILL\,AMT\_X < 0$$

$$PA\,AMT\_X < 0$$

Encode categorical integer features as a one-hot numeric array

Additionally, according to Vladimir G. Drugov [6] $86,5\%$ of PAY_X has the illegal value 0. Removing these would mean loosing too much of the total data, so these values are not removed. We are then left with 22455 $\times$ 23 data points. A One-hot-encoder from scikit-learn is used to encode categorical integer values into one-hot numeric arrays. This feature give all the categorical columns value of 0 or 1. Our data matrix is now of dimensionality 22455 $\times$ 81. Finally, the remaining data is standardised using scikit-learns feature StandardScaler.
Containing a lot more not default (0) than default (1), the data is very skewed. This may result in models gaining better performance in predicting zeros than ones. To correct this skewed distribution the data is down-sampled so that the model can be trained on equally many zeros and ones. This leaves 12977 $\times$ 81 data points.

4 Conclusion

Something here

5 Conclusion

And we'd like the network to learn weights and biases so that the output from the network correctly classifies the digit.

For example, suppose the network was mistakenly classifying an image as an "8" when it should be a "9". We could figure out how to make a small change in the weights and biases so the network gets a little closer to classifying the image as a "9". And then we'd repeat this, changing the weights and biases over and over to produce better and better output. The network would be learning.

6 REMEMBER

- Contro all references, that they are written correctly
- Just include formulas you actually use in the theory part.
- Maybe include illustrative figures on how neural network work.
- Write about Softmax somewhere. You are using this in your code if i use this.
- Drop the example of how a NN works if you have aslot of text in theory

Example of multilined eq:

$$\begin{aligned}\frac{dC(\beta)}{d\beta_j} = \frac{d}{d\beta}[\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \beta_0 x_{i,0} \\ - \beta_1 x_{i,1} - ... - \beta_{n-1} x_{i,n-1})^2] = 0\end{aligned} \tag{22}$$

7 Appendix

8 References

References

[1] Hastie, Trevor. Tibshirani, Robert. Friedman, Jerome. *The Elements of Statistical Learning. Data Mining, Interference, and Prediction.* Second Edition. Springer, 2009. Chapter 3, Chapter 7

[2] M. Hjorth-Jensen Lecture Notes in FYS-STK4155. *Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis.* URL: https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/Regression.html Unpublished, 2019.

[3] Wikipedia: Bias-Variance tradeoff URL: `https://en.wikipedia.org/wiki/Bias\OT1\textendashvariance_tradeoff` Read: 23.09.2019

[4] Nilsen BLABLABLA URL

[5] The comparisons of data ining techniques for the predictive accuracy of probability of default of credit card clients. I-Cheng Y. and Che-hui L. URL: Det er et problem med å få lagt til linken

[6] URL: