FYS-STK 3155 Project 2 Erlend Abrahamsen

November 7, 2019

Abstract

1 Introduction

For the source code (folder Source_code) and the results (folder Results) go to my Github: `https://github.com/ErlendAbrahamsen/FYS-STK3155/tree/master/Project2`

Outline [Introduction, Methods, Results and discussion, Summary]
I will reference to source code/project part where relevant!

Our aim is to analyze logistic regression and an feed forward neural network for binary classification on the credit data presented in section 1.1. I.e. predict if costumer is creditable or not. We will also attempt continuous regression on the Franke function `https://www.sfu.ca/~ssurjano/franke2d.html` with an feed forward nural network.

1.1 Credit data format

This data is taken from an important bank in Taiwan from 2005. The original sample size is 30.000 id's/persons.

y: Default payment: (Yes=1, No=0)

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.

Cited from section 3.1 in
`https://bradzzz.gitbooks.io/ga-seattle-dsi/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf`

We look for erros in the data set by checking the following:

y is in range (Only 0's or 1's)
X2, X3, X4, X6-X11 is in range
X1, X5, X12-X17, X18-X23 ≥ 0 (Assuming bank is not in debt to costumer)

The data set is reduced to 3792 data points / costumers before we begin analysis. *(See creditDataReduction() in Part_a.py)*

Basic properties:
y shape: (3792, 1)
X shape: (3792, 24) (Including 1 as constant predictor)

Creditable: 0.633 %
Non-creditable: 0.367 %

2 Methods

2.1 Binary Logistic regression

Let $\mathbf{x} = (1, x_1, ..., x_p)$, $\beta = (\beta_0, ..., \beta_p)^T$.
Our goal is to calculate prediction probabilities on our binary response variable $y_i \in \{0, 1\}$ depending on the linear combination $\mathbf{x}\beta$.

For setting up an probability mass function, the logistic function $p(t) = \frac{exp(t)}{1+exp(t)}$ is used.

We finally get the two probabilities $P(y_i = 1|x_i, \beta) = p(\mathbf{x}\beta)$ and $P(y_i = 0|x_i, \beta) = 1 - p(\mathbf{x}\beta)$.

For optimizing β weights we use the maximum likelihood estimation (MLE), which is maximizing probability prediction of observed data.

I.e. maximazing $P(D|\beta) = \prod_{i=1}^{n}[p(\mathbf{x}\beta)]^{y_i}[1 - p(\mathbf{x}\beta)]^{1-y_i}$, where D is any possible event.

We then get the cost function by computing partial derivatives of P, since we want to maximize P:

$C(\mathbf{x}, \beta) = -\frac{\partial P}{\partial \beta}$

$= \sum_{i=1}^{n}\Big(y_i \log[p(\mathbf{x}\beta)] + (1 - y_i) \log[1 - p(\mathbf{x}\beta)]\Big)$

Now we want to find $\frac{\partial C(\beta)}{\partial \beta}$ and $\frac{\partial^2 C(\beta)}{\partial \beta \partial \beta^T}$ for minimizing cost:

Define the designmatrix (x_{ij} is the i'th sample value of j'th predictor)

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

and the vector $\mathbf{p} = (p(X_1\beta), ..., p(X_n\beta))^T$ (X_i is the i'th row of X)

The derivatives for n samples can be expressed as:

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

$\frac{\partial^2 C(\beta)}{\partial\beta\partial\beta^T} = X^T W X$, where $W = diag(\mathbf{p}(\mathbf{1}-\mathbf{p}))$. (element wise product)

We cannot minimize the cost analytically, but we can do it numerically with e.g. Newton's method.

2.1.1 Newton's method

The derivation of the 1d case $f(x)$ is derived by an 2. order taylor expansion of $f(x)$.

We get $x_{n+1} = x_n - f'(x_n)/f''(x_n)$, this is extended to higher dimensions as

$\mathbf{x_{n+1}} = \mathbf{x_n} - [\mathbf{H}f(\mathbf{x_n})]^{-1}\nabla f(\mathbf{x_n})$, with the hessian and the gradient.

Setting up our case:

$\beta^{i+1} = \beta^i - \left[\frac{\partial^2 C(\beta)}{\partial\beta\partial\beta^T}\right]^{-1}\left[\frac{\partial C(\beta)}{\partial\beta}\right]$ (Hessian, gradient computed in section 2.2)

$\beta^{i+1} = \beta^i - [X^T W X]^{-1} X^T(\mathbf{p}-\mathbf{y})$ **

Now we choose some initial guess for $\beta^0 = (\beta_0^0, ..., \beta_p^0)$ and iterate **, hoping for convergence.
For implementation see newtonsMethod in part_b.py.

2.2 Feed forward neural network using back propagation with stochastic gradient descent

2.2.1 Basic setup

Our aim is to predict probabilities on our binary response variable $y_i \in \{0, 1\}$ depending on activations of weighted linear combinations with biases/constants $W\mathbf{x} + \mathbf{b}$.
This is almost the same as in logistic regression except that we can have

multiple/iterative functions called activation functions.

Setting up our feed forward neural network we have multiple layers such as an input layer, hidden layers and an output layer.
Note that feed forward means no loops in the network.
Figure of neural network with 2 hidden layers:

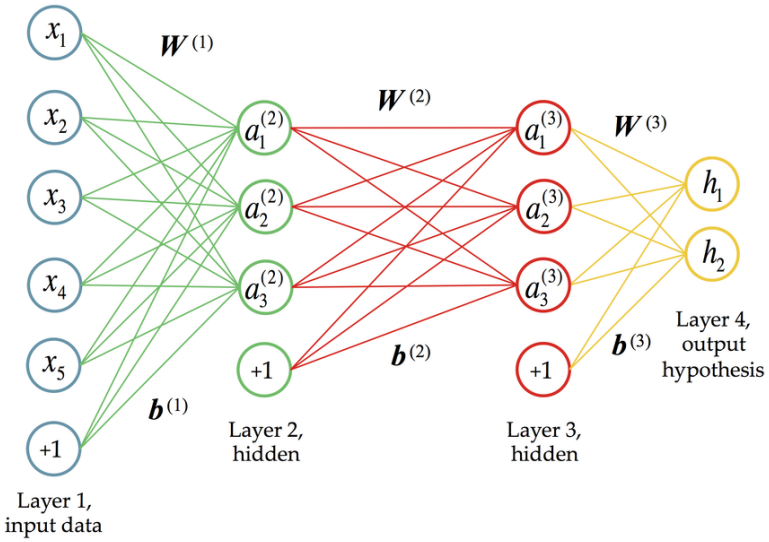


Figure source: https://www.researchgate.net/publication/299474560

We will attempt to setup an feed forward neural network (FFNN) capable of multiple hidden layers. Before we start, note that the python implementation uses slightly different notation than explained here. (Wrote the setup before implementing it)
Setting up matrices and vectors:

Given the activation function σ we have the l'th layer activation as $\mathbf{a}^l = \sigma(W^l\mathbf{a}^{l-1} + \mathbf{b}^l)$, $l = 1, ..., L$ with $\mathbf{a}^1 = \mathbf{x} \in R^{23}$.

For optimizing W and $\mathbf{b}$ we need to assign a cost function $C(\mathbf{y}, \mathbf{a}^L)$ and minimize it using the gradient.
Calculating the gradients is done with a process called back propagation

2.2.2 Back propagation

For notation purposes let $\mathbf{z}^l = W^l\mathbf{a}^l + \mathbf{b}^l$ and $\delta^l = \frac{\partial C}{\partial \mathbf{z}^l}$.

By the chain rule we have from the last layer

$\delta^L = \frac{\partial C}{\partial \mathbf{a}^L} \cdot \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} = \frac{\partial C}{\partial \mathbf{a}^L} \cdot \sigma'(\mathbf{z}^L)$

By chain rule and vector summations it can also be shown that

$\delta^l = [(W^{l+1})^T\delta^{l+1}]\sigma'(\mathbf{z}^l).$

Finally we get the scheme:

(feed_forward method in NeuralNetwork class)

Let $\mathbf{a}^1 = \mathbf{x}$. Compute $\mathbf{a}^l = \sigma(\mathbf{z}^l)$ for $l = 2, ..., L$.

(back_propagation method in NeuralNetwork class)

Let $\delta^L = \frac{\partial C}{\partial \mathbf{a}^L} \cdot \sigma'(\mathbf{z}^L)$ and compute

$\delta^l = [(W^{l+1})^T\delta^{l+1}]\sigma'(\mathbf{z}^l)$ for $l = L-1, ..., 2$.

$\frac{\partial C}{\partial b^l_i} = \delta^l_i$

$\frac{\partial C}{\partial W^l_{ij}} = a^{l-1}_j\delta^l_i$

Since we have the gradients/partials we can use gradient descent to find

minima. Also, see Cost- and activation derivatives for derivation of used formulas in the NeuralNetwork class.

2.2.3 Stochastic gradient descent

This is an approximation of gradient descent using less data.
We still have the standard gradient descent equations of weighs and biases with an given learning rate η:

$$W^l = W^l - \eta \nabla C(W^l, X)$$

$$\mathbf{b}^l = \mathbf{b}^l - \eta \nabla C(\mathbf{b}^l, X)$$

The difference is that we choose so called randomly selected mini-batches of X, instead of using the whole data set, hoping for faster convergence.
Process goes as follows:

Let n, S be number of datapoints and batch size.
Iterate $n//s$ times (whole division) with an randomly selected mini-batch of X with size S.

2.2.4 Cost- and activation derivatives

We implement cross-entropy (log cost) and square error (mse cost) as possible cost functions. cross-entropy cost at final layer L:

$$C(W) = -\sum_{i=1}^{n} \Big(y_i \log a_i^L + (1 - y_i) \log(1 - a_i^L) \Big)$$

We get the partials $\frac{\partial C}{\partial a_i^L} = \frac{a_i^L - y_i}{a_i^L (1 - a_i^L)}$.

For the square error cost $C = \frac{1}{2} \sum_{i=1}^{n} (y_i - a_i^L)$ *

we get the partials $\frac{\partial C}{\partial a_i^L} = a_i^L - y_i$. **
(* and ** are used in backpropgation() method in NeuralNetwork class

Part_c.py)

For choices of activation functions we use sigmoid, tanh and relu. These derivatives can easily be calculated.
(activ() method in NeuralNetowork class Part_c.py contains these derivatives)

$\sigma'(z^L) = \frac{exp(-z^L)}{(1+exp(-z^L)^2} = a^L(1-a^L)$ (element wise product)

$tanh'(z^L) = 1 - tanh^2(z^L) = 1 - (a^L)^2$ (element wise product)

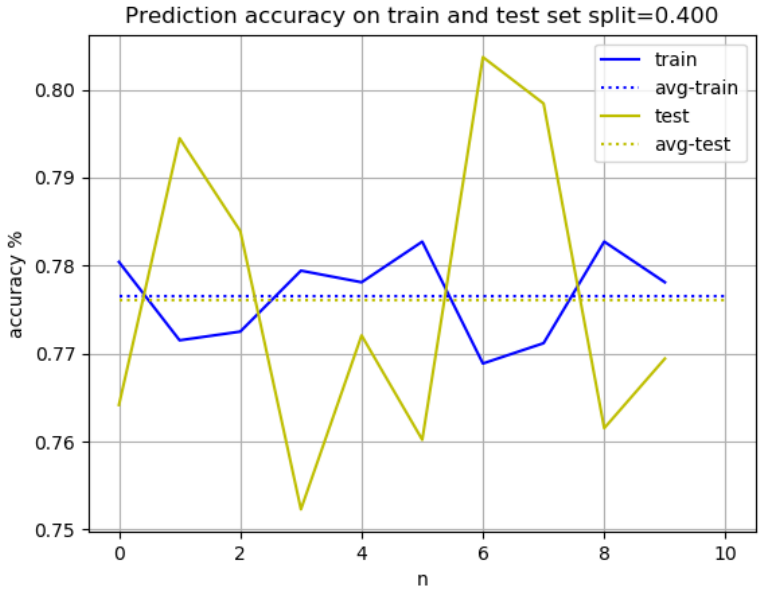
$relu'(z_i^L) = 1$ for $z_i^L > 0$ and 0 for $z_i^L \leq 0$.

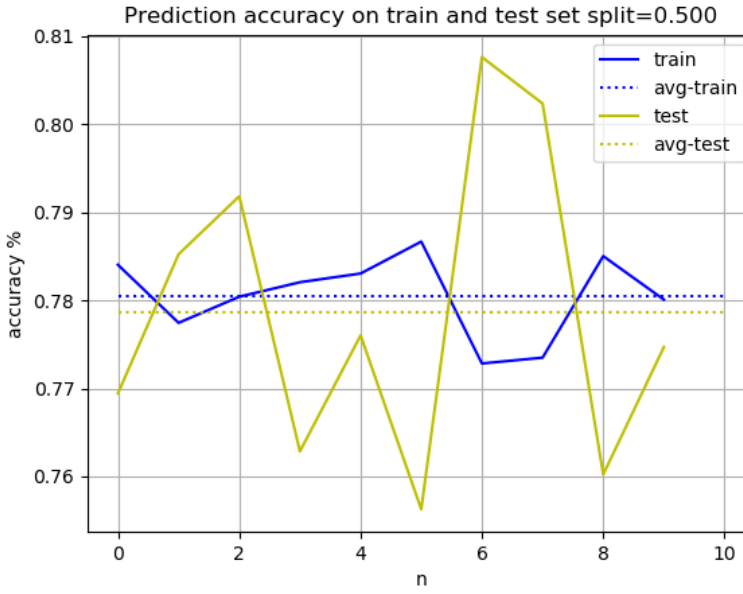
3 Results and discussion

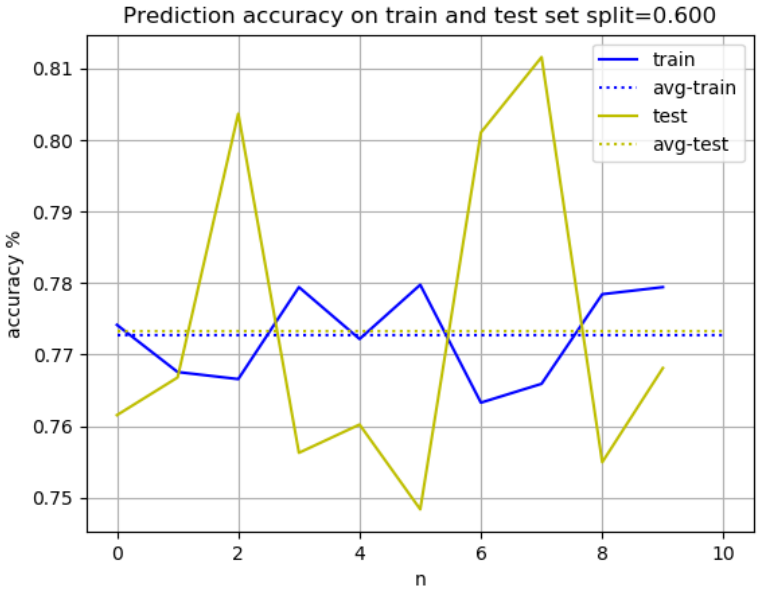
3.1 logistic regression for credit predictions (Part B)

We split the data $X, \mathbf{y}$ into training and testing data using 20% as testing data, and perform binary logistic regression.
Using the training data to find β coefficients we do predictions on both the training set and testing set.
The output is a vector containing probabilities of $y = 1$.

We compute the accuracy of our predictions (Correctly guessed / total) using some different splitting threshold.
For splitting threshold $= 0.5$, $\tilde{y} = 1$ if $prob(y = 1) \geq 0.5$.
Average accuracy scores over $n = 10$ runs:







Average Accuracy with split threshold = 0.400 Training: 0.777 Testing: 0.776

Average Accuracy with split threshold = 0.500 Training: 0.781 Testing: 0.779

Average Accuracy with split threshold = 0.600 Training: 0.773 Testing: 0.773

We see that we get the best results by predict according to greatest probability (split = 0.5).

3.2 FFNN classification for credit predictions(Part c)

3.3 Continuous regression on Franke function using FFNN (Part d)

4 Summary and conclusions

4.1 Best results (Part e)

4.2 Pros and Cons (Part e)

5 References

1: https://bradzzz.gitbooks.io/ga-seattle-dsi/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf

2: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

3: https://compphysics.github.io/MachineLearning/doc/pub/LogReg/html/LogReg.html

4: http://neuralnetworksanddeeplearning.com/chap2.html