Project 2: Solving Classification and Regression problems with a Neural Network

FYS-STK4155

Odin Midbrød Sanner, Eirik Storrud Røsvik
& Martin Heltberg

University Of Oslo

Table of Contents

Abstract

The aim of this assignment is to develop a logistic regression code and an artificial neural network to predict credit risk of costumers in Taiwan and to compare the logistic regression and MLP models. These methods have been applied to a dataset containing credit card data from a Taiwanese bank. The findings of our report are compared to the findings of a similar scientific article from 2009 where they use different data mining techniques on the same dataset. Our findings suggest that the MLP performs better and achieved higher accuracy than the logistic regression code when working with classification. However for the regression problem the MLP does not perform as well as and here the Ordinary Least Square is preferred.

Introduction

In Taiwan in, starting in 2005, there was a big credit card and debt crisis in the country, this came as a result of the banks issuing credit cards to applicants that should not have qualified for credit cards and loans. As too many customers, 700 thousands, did not have the financial capability to repay on their loans they started to default (Chang et al, 2017). The defaulters later had a hard time, becoming what locally became referred to as a 'credit card slave', as they repaid their debts. Following this, government regulations forced the lenders to introduce risk assessment. This risk prediction predicts the risk of a costumer defaulting, based on their past financial information (Wang, 2019).

To develop risk prediction a lot of different statistical methods have been used in the past, like logistic regression, Bayes classifier etc., but as machine learning and artificial neural networks (ANN) has been further developed, this has proven a good method to use to forecast credit card risk.

When making a model to predict credit card risk of costumers, it is more informative to first estimate probability of default from costumer's financial data, then classifying them into binary results. This is done by developing an artificial neural network that predicts the probability of default for costumers based on their financial history.

Data

This assignment uses payment data from a large Taiwanese banking institution, and all the targets are credit card holders from this bank. The data dates from October 2005 with information regarding 30 000 credit card holders and thus the dataset contains 30 000 observations.

There are 23 variables used as explanatory variables:

X1: Amount of given credit in Taiwanese dollars. This includes both the individual customer as well as their families.

X2: Gender, categorical, where a value of 1 is a male and 2 is female.

X3: Education, categorical, 1 for graduate school, 2 for university, 3 for high school and 4 for other.

X4: Marital status, categorical, 1 for married, 2 for single and 3 for other.

X5: Age in years.

X6-X11: History of past payments, categorical. The payments of previous monthly payment record, from April to September 2005. Where X6 is the payment status in September 2005, X7 is the payment status in August 2005, all the way until X11 for the payment status in April 2005. The measurements scale here is: -1 for pay duly, 1 is a payment delay for one month, 2 for two months, up until 9 for payment delay for 9 months and above.

X12-17: Amount of bill statement in Taiwanese dollars. X12 is the amount of bill statements in September 2005, X13 for August, up until X17 for April.

X18-X23: Amount of previous payment in Taiwanese dollars. X18 for the amount paid off in September 2005, X19 for August 2005, up until X23 for the amount paid off in April 2005.

There is also a Y variable in the dataset, a binary variable for defaulting payments (1 for yes and 0 for no) and there are 6638 of them, or about 22.12 %, of the entire dataset.

Theory

Logistic regression

Logistic regression is a method used for doing a statistical analysis of a dataset; in the dataset it will be one or more variables that determine the outcome. Suppose we are working with a dataset containing x-values, where we have p number of predictors for each data point so that $\mathrm{x(i)} = \{\boldsymbol{x_1^i}, \ldots\ldots, \boldsymbol{x_p^i}\}$. The response of the regression will give an outcome of two values, either 0 or 1. This method is used to predict a binary outcome; a binary classification is when there is an outcome with only two possible values: 1/0, false/true, or whether or not a customer will default. (Towards Data Science, 2019)

First thing that must be done when doing logistic regression is to use the logistic responds function to find the probability that an observation belongs to class 1. The logistic responds function is:

$$P(y=1) = \frac{1}{1+(\beta_0+\beta_1 x_1,\ldots\ldots,\beta_n x_n)} \qquad (1)$$

Here the beta values are the parameters of the model; these coefficients are selected to maximize the likelihood of predicting a high probability for observations belonging to class 1 and predicting a low probability for observations belonging to class 0. (Towards Data Science, 2019).

This logistic model can be used to predict an output y_i using the estimated probability:

$$y_i = \begin{cases} 1 \; if \; p(y=1|x^i) \; \geq 0.5 \\ 0 \; if \; p(y=1|x^i) \; \leq 0.5 \end{cases} \qquad (2)$$

Neural networks

Neural networks, also sometimes called artificial neural networks, are computational models that have the ability to be trained from examples and by that learn how 'behave'.

The neural networks model gets trained using the same approach used in nature in order to recognize certain information patterns. The models are shown examples, and with these it automatically refers rules to recognize whatever information it is shown in order to recognize similar

data in the future. Having seen several data examples containing the result, a neural network will be able to extrapolate the result in new real data. The more examples it is shown the more accurate it is. The biological network that is in the human/animal brain inspires this model (Nielsen, 2019).

These kinds of artificial neural networks can be created in many ways, but in this project, we will use the MLP (multilayer perceptron's) neural network that is the most common one. The architecture of this MLP neural network is built in three steps; there is the input layer in which neurons are inserted, one or more middle layers that are called the hidden layers and then there is the output layer, which contains the output neuron for linear regression or a binary output for classification. So, when an input value is introduced to the model it is propagated through the layers in one direction, being processed through each neuron and then comes out as an output value (vector). In this project we will only deal with feed forward ANNs (Artificial neural networks), which means that there are no loops and information only flows in one direction. We can look at the ANNs as complicated functions. (Nielsen, 2019)

Neurons and layers

The model function propagating information through the network is called a neuron. The simple way a neuron works is it takes several binary inputs and produces a single binary output:

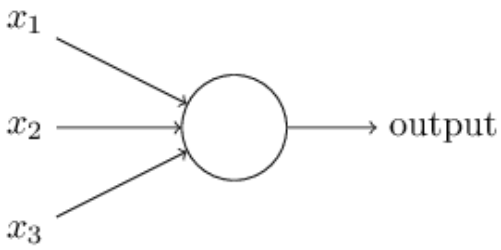

Figure 1: A simple representation of how the hidden layer get fed three inputs: x1, x2 and x3 and produces one single binary output (Nielsen, 2019)

The different inputs may be differently important, so in order to get the most accurate output the vector w_i is introduced. The w-value is number expressing the importance of each input value, so now the weighted sum determines whether the neurons output is 1 or 0. (Nielsen, 2019)

The mathematical way to look at it is:

$$\text{Output} = \begin{cases} 1 \; if \; \sum_j w_j x_j \leq treshold \\ 0 \; if \; \sum_j w_j x_j \geq treshold \end{cases} \qquad (3)$$

This can be compared to how humans make decisions by weighing up the evidence. The models give an output by taking the input-value multiplied with the weight-value. (Nielsen, 2019)

In order to simplify the description of the neutrons, we can make two notational changes. First, we can rewrite the $\sum_j w_j x_j$ to **w*x** = $\sum_j w_j x_j$ where **w** and **x** are vectors. The other change is to move the

threshold to the other side of the inequality and replace it with bias. Then we can write the neutron rule as:

$$\text{Output} = \begin{cases} 0 \; if \; w * x + b \; \leq 0 \\ 1 \; if \; w * x + b \; > 0 \end{cases} \quad (4)$$

The bias is a one value for each neuron, the bias acts as a modifier making the neurons less or more likely to fire independently of the input. (Nielsen, 2019)

The sum of **x** and **w** is transferred to an activation function where bias is introduced and alters the neurons in a specific way, resulting in an output.

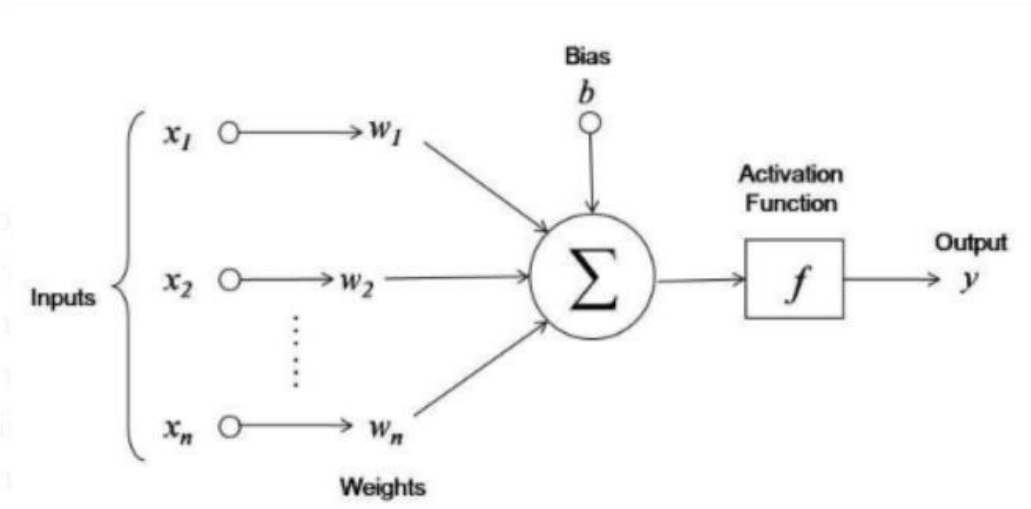

Figure 2: A model neuron, a constituent part of the artificial neural network model. The input from the previous layer p multiplied by corresponding weights w and summed. Then the bias b is added, and the activation function f is applied to the resulting w^T p + b. The output $\tilde{p}$ goes on to become input for neurons in the next layer (Shamoon, 2019).

The activation function may vary depending on the purpose, it can be either linear or non-linear, the function will disappear for small inputs and saturate for large inputs. How a single neuron goes through the ANN can be written as:

$$\text{Input} \rightarrow f\,(W^T p + b) = \tilde{p} \rightarrow \text{output}$$

The full ANN as shown Fig 2 is built up of layers of neurons. When the data is introduced to the ANN it first goes through the first layer, then the hidden layer and lastly ending up in the output layer. Before the activation can be calculated in layer **k**, the layer **k-1** need to be fully computed and transferred to the next layer **k**, for this to happen, the propagation across the network need to happen simultaneously. Each layer **k** is made up of a collection of neurons that is connected to neurons in layers **k-1** and **k+1**. If we try to make a representation of how each neuron in a layer work, we could label one neuron in layer **k** with the index i ($n\mathbf{k}_i$). The bias for the neuron is $\mathbf{b}^*\mathbf{k}_i$ and the weight connecting $n\mathbf{k\text{-}1}_i$ to $n\mathbf{k}_j$ is $\mathbf{w}_j$ i. The **wk** matrix is a representation of the combined weight

vectors for layer **k**, and **b*k** is the collection of all bias in layer **k**. The propagation from layer **k-1** to layer **k** can be written as:

$$\mathbf{y*k} = \mathrm{f}\,(\mathbf{w*k} * \mathbf{y*k\text{-}1} + \mathbf{b*k}) = f\left(\begin{bmatrix} w_{11}^k & \cdots & w_{1N}^k \\ \vdots & \ddots & \vdots \\ w_{N1}^k & \cdots & w_{NN}^k \end{bmatrix} * \begin{bmatrix} y_1^{k-1} \\ \vdots \\ y_N^{k-1} \end{bmatrix} + \begin{bmatrix} b_1^k \\ \vdots \\ b_N^k \end{bmatrix}\right) \quad (5)$$

Or in Einstein notation:

$$y_i^k = f\left(w_{ij}^k * y_j^{k-1} + b_i^k\right) \quad (6)$$

In fig. 3 it is shown the how the three neurons in a layer is fully connected in ANN.

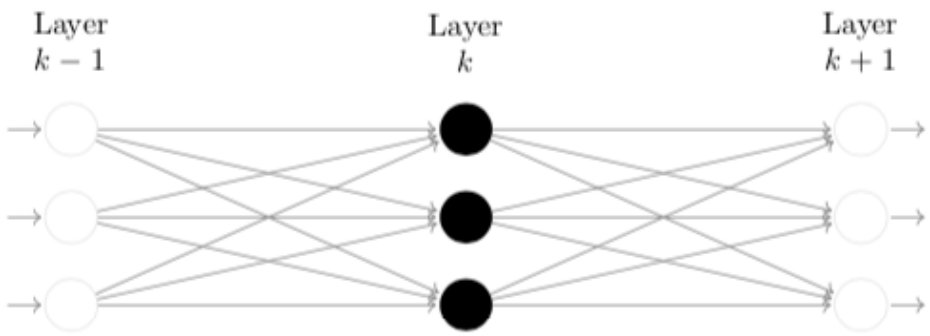

Figure 3: schematic representation of a single ANN layer. Each neuron of the layer-indexed k is connected from behind to all neurons in layer k − 1. The connection weights can be organized into a matrix, W^{k-1}, and the action of layer k can be succinctly stated as f ($W^k p^{k-1} + b^k$) where element-wise operation is assumed for the activation function (Hallstrom, 2019) .

The full network

A full network is formed from a collection of L layers, this network is basically a complex function. If the values for input and output are specified, then you can write out the action of ANN in closed form like:

$$\hat{y} = \sum_{j=1}^{M} w_{1j}^L f_L\Big(\sum_{k=1}^{M} w_{jk}^{L-1} f_{L-1}\Big(\sum_{i=1}^{M} w_{ki}^{L-2} f_{L-2}\left(\dots f_1(w_{m1}^1 * x_1 + b_1^m)\dots\right) + b_i^{L-2}\Big) + b_k^{L-1}\Big) + b_1^L$$

Here the ANN is divided up with each layer made up of M neurons, the scalar x_1 indicates the input value and $\hat{y}$ is the output value. The activation functions (f) are indicated with f(L), f(L-1), f(L-2),…, f(L0). This equation is not easily read and understood but this ANN is a universal approximation when it has at least one hidden layer with finite number of neurons. Universal approximation theorem states that “a feed-forward network with a single hidden layer containing a finite number of neuronscan approximate continuous functions on compact subsets of Rn, under mild assumptions on the activation function.”

Activation function

Without the activation functions within the layers the ANN would simply be a linear regression model, when one or more sets of activation functions are introduced to the model it introduces non-linearity by calculating weighted sum and adds bias to the neuron. The non-linearity in the model is required to make the model capable to learn and perform well with more complex cases.

There are several activation functions that are commonly used today; the simplest activation function is the identity transformation function: $f_I(x) = x$ which is used in regression network as output layers. A Heaviside step function, $f_H(x) = H(x)$, with

$$f_H(x) = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases} \quad (8)$$

This Heaviside step function is rarely used in the hidden layer because it's impossible to train with backpropagation due to the vanishing gradient, but it is sometimes used in the classification layer. The two most common functions used in the hidden layer are the sigmoid function and hyperbolic tangent:

$$f_S(x) = \frac{1}{1+e^{-x}} \quad \& \quad f_t(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (9)$$

The rectifiers activation functions are activation functions that are even simpler than the sigmoid function and are used to train deep ANN, deep ANN means that there are many more and large hidden layers in the NN, and it can solve more complex problems. The basic variant ReLU (Rectified linear unit) is defined as:

$$\boldsymbol{f_{ReLU}(x) = \max(0, x)}$$

A lot more variant of ReLu exist but the most common once are:

The leaky ReLU $\rightarrow f_{Leaky\ ReLU}(x;\ \alpha) = \begin{cases} x \; if \; x \geq 0 \\ \alpha \; x \; if \; x < 0 \end{cases}$

The noisy ReLU $\rightarrow f_{NReLU}(x) = \max(0, x + N(0, \sigma))$

The exponential linear unit $\rightarrow f_{ELU}(x;\ \alpha) = \begin{cases} x & if \; x \geq 0 \\ \alpha \; (e^x - 1) & if \; x < 0 \end{cases}$

(Theory of everything, 2017).

Training neural network

There is not much use in ANN unless you can approximate any given function f(x) which is done by obtaining suitable parameters for approximation. In order to approximate a function, you need to train the ANN, this is done in three steps:

1. Compute the output of the ANN $\rightarrow NN(x) = \hat{y}$ and then evaluate the cost function, $C(\hat{y}, y)$

2. Compute the cost function gradient, C($\hat{y}$), with respect to the parameters w_{ij}^k & b_j^k.
3. Alter the parameters according to the gradients, in order to achieve a better estimate $\hat{y}$ for the true output value y.
4. Repeat 1-4.

This is also known as supervised learning, supervised learning is when you have an input value x and an output value y and by using algorithms to learn the mapping function from input to output, y=f(x). The goal of supervised learning is that the ANN can predict the correct output (y) based on prior training data (ML4A, 2019). Step 3 in the process is to alter the parameters, which might be relatively easy to understand but difficult in practice. In order to not get random values for bias and weight, the expected change in total output needs to be measured. The most common way to do this is to use the backpropagation algorithm, backpropagation is used to decide how much the parameters need to altered in order to get a more accurate final output, this is done by comparing predicted output with desired output for the NN and then going back in the layers to alter the parameters for weight and bias. This is repeated until the NN is "happy" with its own performance. (ML4A, 2019)

Backpropagation

Before application, the backpropagation algorithm needs to be performed a forward pass on. Some input vector will be given $x \in R^{n_f}$, where $n_f$will denote the number of features in the input data (Nielsen, 2019). Briefly, only the case of a single input only will be considered. When the forward pass is executed it will calculate the activations (a^l) of the layer

$$a_j^l = f_l(z_j^l) \qquad (10)$$

Here $z_j^l$is a sum of the weighted sum of inputs from the preceding layer and the bias in layer l.

$$z_j^1 = \sum_{i=1}^{n_r} w_{ij}^1 x_i + b_j^1 \qquad (11)$$

One can assume that the layers have N_l number of neurons. If that statement is true, the calculated z_j^l of the ensuing layers can be expressed as (Nielsen, 2019),

$$z_j^l = \sum_{i=1}^{N_{l-1}} w_{ij} x_i + b_j^l \qquad (12)$$

Here $w^l \in R^{N_{l-1} \times N_l}$ is true.

When the forward pass is completed, one can calculate the cost function and the derivative of the cost function using the weights in the output layer W^L.

$$\frac{\partial C(W^L)}{\partial w_j^L} = \frac{\partial C(W^L)}{\partial a_j^L}\left[\frac{\partial a_j^L}{\partial w_{jk}^L}\right]$$

$$\frac{\partial C(W^L)}{\partial w_j^L} = \frac{\partial C(W^L)}{\partial a_j^L}\left[\frac{\partial a_j^L}{\partial z_{jk}^L}\frac{\partial z_j^L}{\partial w_{jk}^L}\right]$$

$$\frac{\partial C(W^L)}{\partial w_j^L} = \frac{\partial C(W^L)}{\partial a_j^L} f'\left(z_j^L\right) a_k^{L-1} \qquad (13)$$

Here we used that

$$\frac{\partial C(W^L)}{\partial a_j^L} = \frac{\partial}{\partial a_j^L}\left[\frac{1}{2}\sum_{i=1}^{N_O}\left(a_j^L - t_j\right)^2\right]$$
$$\frac{\partial C(W^L)}{\partial a_j^L} = a_j^L - t_j \quad (14)$$

And

$$\frac{\partial z_J^L}{\partial w_{jk}^L} = \frac{\partial}{\partial w_{jk}^L}\left[\sum_{i=1}^{N_O}\left(w_{jP}^L a_j^{L-1} - b_j^L\right) \right]$$
$$\frac{\partial z_J^L}{\partial w_{jk}^L} = a_j^{L-1} \quad (15)$$

One can define equation 13 except for a_j^{L-1} as δ_j^L, that means that $\frac{\partial C(W^L)}{\partial w_j^L} = \delta_j^L \, a_j^{L-1}$. Applying the chain rule (Dawkins, 2018) to δ_j^L gives out the derivative of the cost function with respect to the output layer biases, just like we want it to (Lin et al., 2015).

$$\delta_j^L = \frac{\partial C(W^L)}{\partial a_j^L}\frac{\partial f^L}{\partial z_j^L}$$
$$\delta_j^L = \frac{\partial C(W^L)}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}$$
$$\delta_j^L = \frac{\partial C(W^L)}{\partial z_j^L}$$
$$\delta_j^L = \frac{\partial C(W^L)}{\partial b_j^L}\frac{\partial b_j^L}{\partial z_j^L} \quad (16)$$
$$\delta_j^L = \frac{\partial C(W^L)}{\partial b_j^L} \quad (17)$$

Where the following has been used

$$\frac{\partial b_j^L}{\partial z_j^L} = \left[\frac{\partial z_j^L}{\partial b_j^L}\right]^{-1}$$
$$\frac{\partial b_j^L}{\partial z_j^L} = \left[\frac{\partial}{\partial b_j^L}\sum_{i=1}^{N_{L-1}} w_{ij}^L a_i^{L-1} + b_j^L\right]^{-1}$$
$$\frac{\partial b_j^L}{\partial z_j^L} = 1 \quad (18)$$

Therefore, the derivatives of the cost function are calculated, with respect to both the weights and the biases in the output layer (Sazli, 2006). $\frac{\partial C(W^L)}{\partial z_j^L}$ Hold true for any layer, except the output in equation 16. Relating this to the derivatives, on layer l+1, one can find

$$\delta_j^l = \frac{\partial C(W^L)}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{z_j^l}$$
$$\delta_j^l = \sum_k \delta_j^{l+1} \frac{\partial z_k^{l+1}}{z_j^l}$$
$$\delta_j^l = \sum_k \delta_j^{l+1} w_{kj}^{l+1} \frac{\partial f^l}{z_j^l} \quad (19)$$

And

$$\frac{\partial z_k^{l+1}}{z_j^l} = \frac{\partial}{\partial z_j^l}\left[\sum_{i=1}^{N_l} w_{ik}^{l+1} a_k^l + b_k^{l+1}\right]$$
$$\frac{\partial z_k^{l+1}}{z_j^l} = \frac{\partial}{\partial z_j^l}\left[\sum_{i=1}^{N_l} w_{ik}^{l+1} f^l\left(z_k^l\right) + b_k^{l+1}\right]$$
$$\frac{\partial z_k^{l+1}}{z_j^l} = w_{ik}^{l+1} f^l\left(z_k^l\right) \quad (20)$$

Now we iterate equation 19, and compute the gradients, δ_j^l and $\delta_j^L\ a_j^{L-1}$ for each layer. When the gradients are known, one must update the weights and biases to improve the network and its performance. One forward pass and one backpropagation is known as an epoch (Sharma, 2017).

Gradient Descent

Gradient descent is a good optimization algorithm that is used to find the local minima of any function (Weissstein, 2019). To do this one first has to minimize the cost function, C($\hat{\beta}$), where $\hat{\beta}$ is a vector with elements for every new value. Using this we want to find a new sequence of points, $\hat{\beta}(i)$, that is closer to the actual solution. This is done by using the derivative of the cost function that was shown earlier (Marsland, 2014). One important thing to keep in mind is the direction of the derivative, as we always want the gradient descent to down as fast as possible. The equation for gradient descent is as following (Weinberger, 2018)

$$\hat{\beta}_{n+1} = \hat{\beta}_n - \eta \nabla_\beta C(\beta) \quad (21)$$

Where η is the rate of learning and the derivative of the cost function is set to

$$\nabla_\beta C(\beta) = -\hat{X}^T(\hat{y} - \hat{p}) \quad (22)$$

Where $\hat{X}$ is the feature matrix, $\hat{y}$ is a vector of targets and $\hat{p}$ is the sigmoid function, a vector of fitted probabilities. The gradient descent method has some small problems, like if $\hat{\beta}$ might move directly towards a local minima (Luhaniwal, 2019). Some of the shortcomings could be addressed by using

the stochastic gradient descent. Equation 21 will be changed by adding a randomly shuffled data, and therefore adding stochasticity (Patrikar, 2019).

Calculating the descent on the entire dataset might be inefficient and computably expensive, as well as a high probability of misinterpreting the local minima as the global minima. To get past these problems, the stochastic gradient descent is computed using mini-batches. Mini-batches are essentially a subset of data to make computations faster. This will also compute the gradient against more training data, which makes the mean value more averaged out over more training data. For each epoch, the mini-batches are tested before $\hat{\beta}_{n+1}$ is calculated using equation XXX. After the calculation, the data is randomly reordered before moving on to calculate $\hat{\beta}_{n+1}$ for the next epoch. This continues until the last epoch is done.

Model evaluation

The model is evaluated, and the performance is measured by the accuracy score

$$A = \frac{\sum_{i=1}^{n} I(y_{i=}t_i)}{n} \qquad (23)$$

Where A is the accuracy, y_i is the output of either the logistic regression or the classification neural network, t_i is the target and I is an indicator function with value one if $y_{i=}t_i$, and 0 for any other values. The linear regression neural network is computed using the R^2 score

$$R^2(y_i, t_i) = 1 - \frac{\sum_{i=1}^{n}(t_i - y_i)^2}{\sum_{i=1}^{n}(t_i - \bar{y}^i)^2} \qquad (24)$$

Where $\bar{y}$ is the mean of the predicted values. The area ratio was also used in this assignment. In the area ratio there are three curves, one model curve, one theoretically best curve and one baseline curve. The larger the area between the baseline curve and the model curve, the better the model is (Yeh & Lien, 2009). Area ratio is defined as

$$Area\ ratio = \frac{area\ between\ model\ curve\ and\ baseline}{area\ between\ the\ theoretically\ best\ curve\ and\ baseline\ curve}$$

Results

The results for the credit card data has been produced with two hidden layers with 25 neurons each, using an output layer consisting of 2 neurons for classification, later converted to a binary output. The data for the Franke function was produced with three hidden layers with 25, 25 and 10 neurons, with a single output neuron output. The activation function used on the hidden layers is the hyperbolic tangent (tanh, equation 9) and a linear function between the last hidden layer and the output.

Classification

The model hyper-parameters were set using the heat maps seen in figures 4, 5, & 6. The heat maps show that the parameters have some ranges where the model preforms better. The parameter for the logistic regression was set using a lower learning rate, as this is a less complex model. However, the number of iterations for the logistic regression model is several orders of magnitude larger.

Table of the best values for shrinkage and learning rate for logistic regression and neural network models.

Model	Shrinkage, λ	Learning rate, η
Logistic regression credit data	-	0.001
Neural network credit data	0.001	0.1
Neural network franke	0.0001	0.01
Neural network franke noise	0.0001	0.01

Table 1: Table of the best values for shrinkage and learning rate for logistic regression and neural network models.

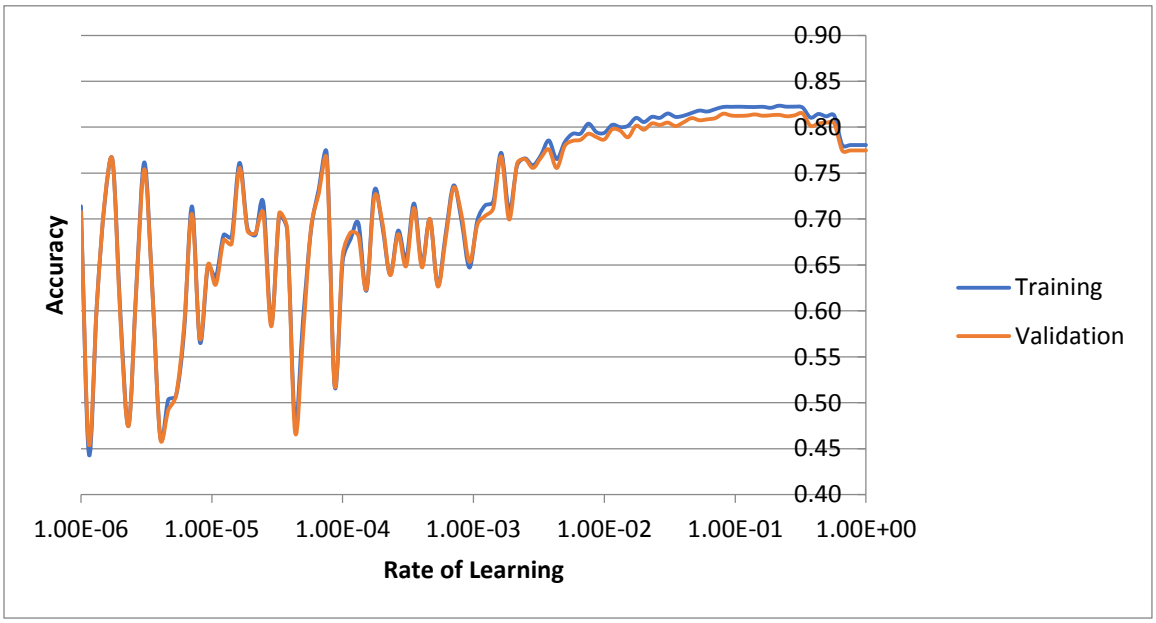

Figure 3: Accuracy score of the validation set of the credit card data using logistic regression for values of the rate of learning, shrinkage = 0.001.

Figure 3 shows the accuracy score for the logistic regression on both the credit card training and validation data set for 100 different logarithmically scaled values for the rate of learning. As seen in

the figure the accuracy increases steadily with an increasing learning rate, with a small drop off when the rate of learning gets closer to 1. The value of the learning rate value which maximizes the accuracy, as seen in table 1, is 0.81.

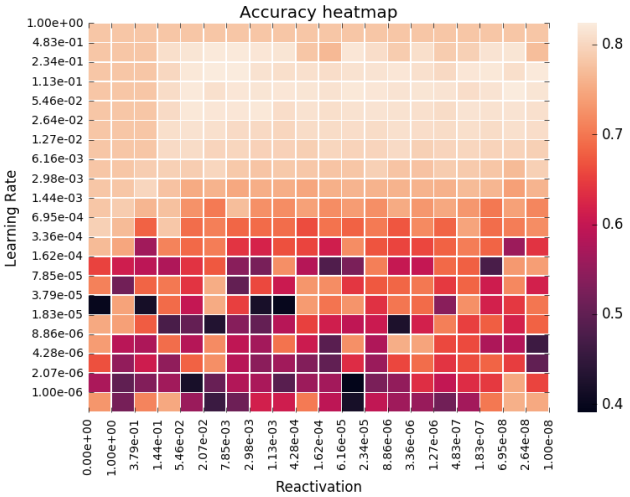

Figure 4: Accuracy score of the validation set of the credit card data using MLP for various shrinkage parameters and learning rates.

Figure 4 shows the accuracy for the MLP classifier used on the credit card data. The data was once again set for 100 different combinations of the value for learning rate and the shrinkage parameter. As seen in the figure the best accuracy happens near the top, in the middle of the heatmap. Generally, a low learning rate gives out lower accuracy scores. The better accuracy also happens when the shrinking parameter is lower than one, at least by an order of one magnitude.

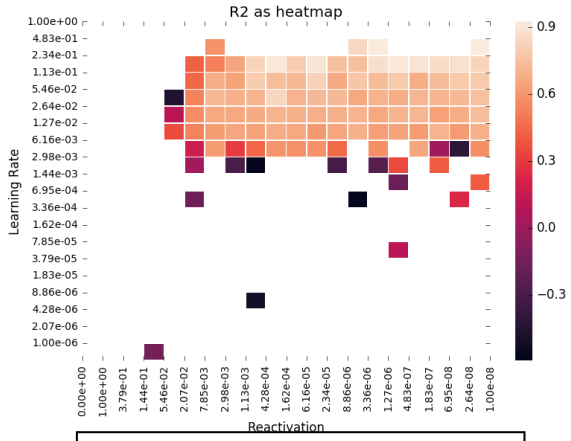

Figure 5: R^2 score of validation set on Franke function data without noise using MLP-regressor for various shrinkage and learning rates.

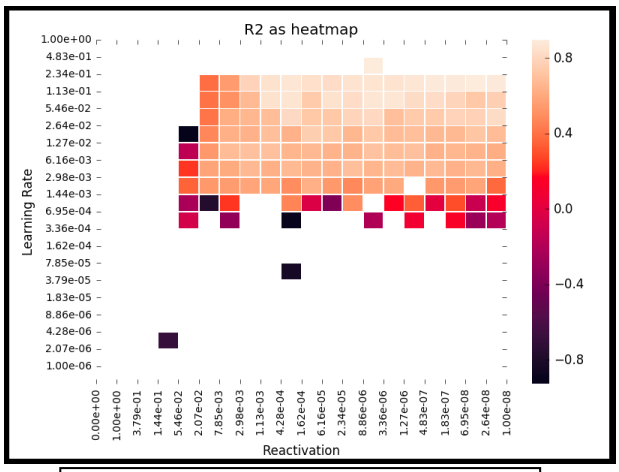

Figure 6: R^2 score of validation set on Franke function data with noise using MLP regressor for various shrinkage and learning rates.

Figures 5 and 6 shows the validation scores (R^2) for the MLP regressor on the Franke data sets. In figure 5, the Franke function is without noise, while figure 6 includes noise. The noise is set to α=0.3. As seen, they are fairly similar, both with higher R^2 values with lower values of both shrinkage and learning rate.

The models were running for 100 different combinations of learning rate and shrinkage. For several of the learning rate and reactivation combinations, the model diverged and resulted in the R^2 values being computed to NaN values, R^2 values not in the span [-1, 1] were also set to NaN. This appears to happen for all configurations where the reactivation parameter is set higher than the learning rate, as well as for all the lesser aggressive learning rates ($<5*10^{-5}$).

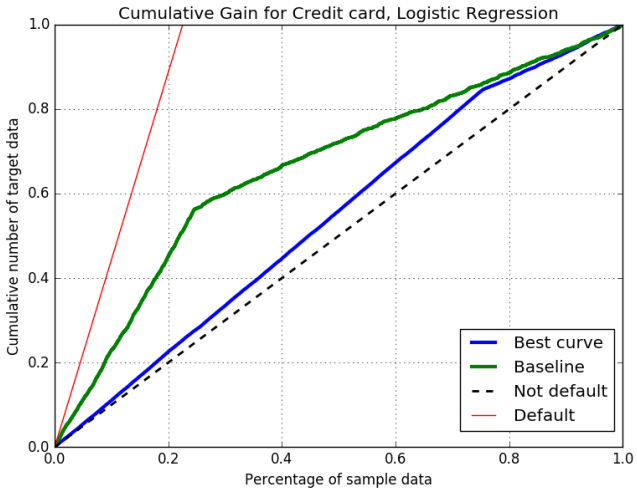

Figure 7: Cumulative gains plot for the logistic regression model on credit card data.

Figure 7 shows the cumulative gains plot from the logistic regression. From the figure, we can see that the logistic regression model performs better when predicting default outcomes, rather than non-default.

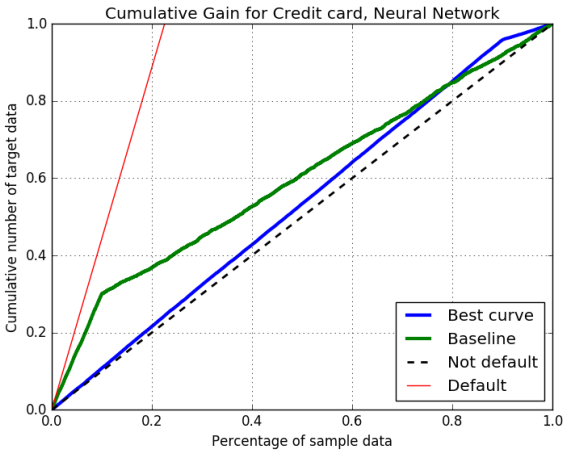

Figure 8: Cumulative gains plot for the MLP model fit on the credit card data

Figure 8 shows the cumulative gains plot from the MLP classifier. The model is, once again, better at predicting default outcomes, but the non-default rises above the default near the end of the sample data.

Method	Error Rate	Area Ratio Default	Area Ratio non-default
Logistic Regression	0.22	0.42	0.18
Neural Network	0.19	0.25	0.08
Logistic Regression Original	0.18	0.44	N/A
Neural Network Original	0.17	0.54	N/A

Table 2: Error rates and area ratio using our own logical regression and the neural network used on the credit card data, compared with the logistic regression and neural network from (Yeh & Lien, 2009)

In table 2, one can see the error rate and area ratio (both default and non-default) for the logistic regression and neural network from this assignment and its comparison to Yeh & Lien.

	Not default, correct	Default correct,	Not default, incorrect	Default, incorrect
Neural Network	22637	1973	4663	717
Logistic Regression	19936	3720	2916	3428

Table 3: Confusion matrix for MLP and Logistic regression

Table 3 shows the number of correct/incorrect predictions made by the two models. The MLP preforms better when it comes to not incorrectly predicting not defaulters to default. Logistic regression on the other hand preforms better when predicting the total number of defaulters as well as correctly predicting defaulters.

Linear Regression

Linear regression was performed on the Franke function computed over a 1000 by 1000 grid, resulting in one million samples. The regression was applied to the dataset both with and without noise. The noise was generated following a normal distribution, N(1, 0.05)

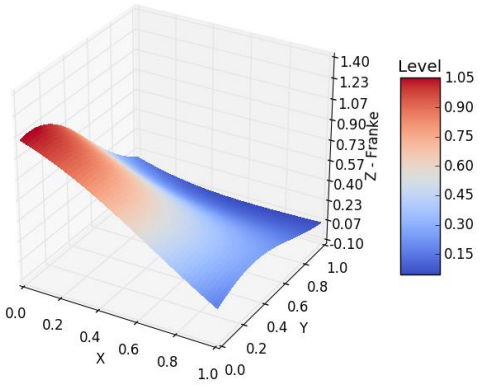

Figure 9: The MLP model of Franke function without noise

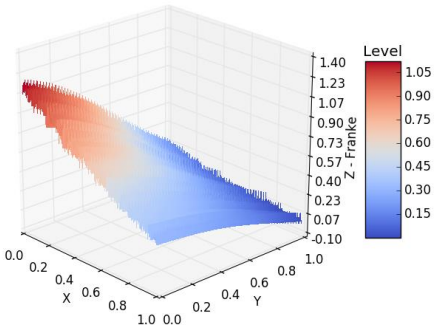

Figure 10: MLP model prediction with noise

Figures 9 and 10 shows the MLP models prediction for the Franke data, the noise appears to make the modeled surface less smooth.

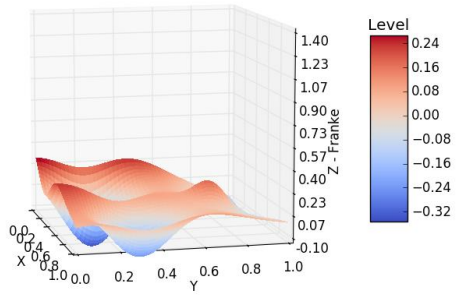

Figure 11: Residuals Franke function without noise

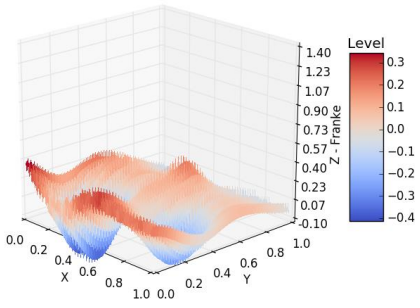

Figure 12: Residuals Franke function with noise

Figures 11 and 12 shows the residuals of the Franke function, with and without noise, created by the neural network.

R^2	Franke function	Noisy Franke function
NN Complete	0.818	0.792
NN Validation	0.817	0.791
OLS Cross validated	0.976	0.911

Table 4: R^2 scores for the neural network (NN) and the ordinary least squares (OLS) model from project 1, used on the Franke function

Table 4 is showing the R^2 scores for the Franke function data with noise and without noise, compared to the OLS solution from project 1.

Discussion

Classification

The accuracy score of the credit card data using logistic regression, as seen in figure 3, behaves strange until the learning rate hits about $1*10^{-3}$. Then the accuracy score gradually climbs before it takes a short dive when the learning rate gets closer to 1. When the learning rate is lower than $1*10^{-3}$ the accuracy becomes rather volatile with large swings in accuracy. The best accuracy is therefore closer to $1*10^{-1}$.

As seen in figure 4, the best accuracy for the MLP on credit card data happens, once again, when the rate of learning is close to $1*10^{-1}$, and generally gets gradually worse with lower learning rate. What happens with the prediction when the shrinkage parameter changes is not as clear. When the shrinkage is closer to one, the data with the lower rate of learning falls in accuracy, while the data with lower learning rate had less changes. The shrinkage parameter seems to be of less importance, as long as it's at least one order of magnitude smaller than the learning rate. Looking at the confusion matrices for the MLP predictions and the Logistic regression predictions in table 3, it can be seen that the models preform quite differently. The logistic regression is better when it comes to predicting correctly the total number of defaulters, but preforms poorer when it comes to determining who these defaulters are. The model has an accuracy score for predicting the defaulters correctly of around 54 percent. The MLP is better at correctly guessing true defaulters amongst predicted defaulters with a score of 73 percent. This leads us to believe that the MLP model could be further tuned in order to yield even better results in the future.

The classification models created in this paper had accuracy scores of 78.8 percent for the logistic regression and 82.0 percent for the MLP. These scores were calculated giving score for all correct predictions. For some problems, it might make more sense to only calculate the score based on only correctly predicted defaults.

The cumulative gains functions in figures 7 and 8 shows that the models are generally better at predicting default outcomes, rather than non-default. One exception from this however is that the non-default rises above the default outcome, near the end of the neural network's way through the sample data.

The area ratios shown in table 2 along with figures 7 and 8 are similar to the results from the original paper written by Yeh and Lien. The neural network had a default area ratio that was much lower than what Yeh & Lien calculated. This could be due to the fact that out neural network is not as

strong as the one Yeh & Lien used. The logistic regression however, had a calculated area ratio default that was very close to what Yen & Lien got. The error rates in table 2 shows that the error rates achieved in this assignment are almost as low as the error rate calculated by Yeh & Lien.

Regression

The R^2 score was calculated for several hyper-parameters trying to model the Franke function, as seen in figures 5 and 6 for un-noisy and noisy respectively. As the plots are fairly similar, it is safe to say that the model preforms better when the learning rate is in the span ($1*10^{-3}$, $1*10^{-1}$), with the shrinkage parameter being at least one order of magnitude smaller. The optimal models for the regression are the models where the R^2 score is maximized. The shrinkage and learning rate values that correspond to the best R^2 score can be found in table 1. As can be seen in figures 9 and 10, the modeled surface is a very general approximation, and the R^2 scores seen in table 4 reflect this. The residuals can be seen in figure 11 and 12, and the noise can be seen in figure 12 to make much rougher surface. The noise seems to be giving some values closer and some further from the actual values, in the end giving a smaller effect on the R^2 score than the visual.

The regression preformed quite a bit worse than the OLS method developed in project 1, witch preformed with a R^2 score of over 0.9.

Conclusion

For the classification problem, the MLP model scored better. However, if the lender wanted to predict the number of defaults, the logistic regression model would be preferred. For the regression problem, the MLP scored worse than the OLS model on R^2, but the MLP model comes with the possibilities to be tuned further, either by changing activation functions or the number of layers or neurons, none of which was tested for in this paper. When it comes to regression, the OLS model is a far simpler to implement and preforms better.
For improving the accuracy of the MLP, different activation functions and classification functions could be implemented and explored in order to generate an even better fit. Several hyper-parameters were not changed through the calculations, such as batch size, number of epochs, number of hidden layers and number en neurons were all kept static. Tuning these could also lead to better predictions. Lastly, a gradient booster could also be implemented to improve the result.

References

Chih, Hsiung Chang et al. (2017)"A Study on the Coping Strategy of Financial Supervisory Organization under Information Asymmetry: Case Study of Taiwan's Credit Card Market "From: http://www.hrpub.org/download/20171030/UJM3-12110203.pdf

Dawkins, Paul (2018) "Chain Rule" from: http://tutorial.math.lamar.edu/Classes/CalcI/ChainRule.aspx Downloaded: 05.11.19

Deeplearning (2019) "Parameter optimization in neural networks" from: https://www.deeplearning.ai/ai-notes/optimization/ Downloaded: 05.11.19

Geeksforgeeks (2019) "Activation function in neural network" from: https://www.geeksforgeeks.org/activation-functions-neural-networks/ Downloaded: 04.11.19

Hallstrom, Erik "Backpropagating from the beginning" From: https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d Downloaded: 12.11.2019

Hjort-Jensen, Morten (2019) "Data Analysis and Machine Learning: Logistic Regression" from: https://compphysics.github.io/MachineLearning/doc/pub/LogReg/html/LogReg.html

Hjort-Jensen, Morten (2019) "Data analysis and machine learning lectures: optimization and gradient methods" from: https://compphysics.github.io/MachineLearning/doc/pub/Splines/html/._Splines-bs008.html

Yeh, I-Cheng & Lien, Che-Hui (2007) "The comparison of data mining techniques for the predictive accuracy of probability of default of credit card clients" from: https://bradzzz.gitbooks.io/ga-seattle-dsi/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf Downloaded: 17.10.19

Lin, Bin et al. (2015) "Application of back-propagation artificial neural network and curve estimation in pharmacokinetics of losartan in rabbit" From: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4729999/ Downloaded: 05.11.19

Luhaniwal, Viashraj (2019) "Why Gradient descent isn't enough: A comprehensive introduction to optimization algorithms in neural networks" from: https://towardsdatascience.com/why-gradient-descent-isnt-enough-a-comprehensive-introduction-to-optimization-algorithms-in-59670fd5c096 Downloaded: 07.11.19

Marsland, Stephen (2014) "Machine Learning: An Algorithmic Perspective. Chapman and Hall/CRC"

ML4A (2019) "How neural networks are trained" from: https://ml4a.github.io/ml4a/how_neural_networks_are_trained/ Downloaded: 28.10.19

Nielsen, Michael (2019) "Neural Networks and Deep Learning" from: http://neuralnetworksanddeeplearning.com/index.html Downloaded: 04.11.19

Patrikar, Sashant (2019) “Batch, Mini Batch & Stochastic Gradient Descent” from: https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a Downloaded: 06.11.19

Sharma, Sagar (2017) “Epoch vs Batch Size vs Iterations» from: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9 Downloaded: 04.11.19

Siddiqui, Shamoon. “How would we find a better activation function than ReLU?“From: https://medium.com/shallow-thoughts-about-deep-learning/how-would-we-find-a-better-activation-function-than-relu-4409df217a5c Downloaded: 12.11.2019

Sazli, Murat (2006) “A brief review of feed-forward neural networks” from : https://www.researchgate.net/publication/228394623_A_brief_review_of_feed-forward_neural_networks Downloaded: 03.11.19

Theory of everything (2017) “Understanding activation functions in neural networks” from: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 Downloaded: 28.10.19

Towards Data Science (2019) “Logistic Regression for Xummies : a Detailed Explanation” from: https://towardsdatascience.com/logistic-regression-for-dummies-a-detailed-explanation-9597f76edf46 Downloaded: 01.11.19

Weinberger, Kilian (2018) “Gradient Descent (and Beyond)” from: http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote07.html Downloaded: 05.11.19

Weisstein, Eric W. "Method of Steepest Descent." From: http://mathworld.wolfram.com/MethodofSteepestDescent.html Downloaded: 06.11.19

Wang, Eric. ”Taiwan credit card crisis.” from: https://sevenpillarsinstitute.org/case-studies/taiwans-credit-card-crisis/ Downloaded: 12.11.19