FYS-STK4155
Applied data analysis and machine learning

Project 2

Classification and Regression, from linear and logistic regression to neural networks

Anna Eliassen
Aron Jansson Nordberg
Kristina Othelia Lunde Olsen

https://github.com/kristinaothelia/FYS-STK4155/tree/master/Project2

November 13, 2019

Abstract

Machine learning is an important tool in today's handling of data. We have studied classification of credit card data by developing our own logistic regression code and a feed-forward neural network code. The two models were compared to each other, where we concluded that our logistic regression method gave the best accuracy and f_1 score, while the area under curve (AUC) score were similar for both methods. For neural network we used a learning rate $\eta = 1 * 10^{-4}$ and the hyperparameter $\lambda = 0.01$. The reason neural network underperformed, could be caused by too little fine-tuning of various parameters used in the calculations. We then modified the neural network code to perform a regression analysis on Franke's function, using linear regression. This result was then compared to the linear regression methods OLS, Ridge and Lasso, studied in the first FYS-STK4155 project. To measure the performance of the models, we compared the mean squared error and R^2 score, showing that OLS and Ridge regression fitted the data best, with neural network close behind. Again we believe that the neural network method underperformed due to bad adjustments of various parameter inputs. This would have been better tested if the time allowed it.

Contents

1 Introduction

In this project we will look into both logistic regression and neural networks, studying both classification and regression problems. In a classification problem, the goal is to make a model that predicts an outcome with discrete and labelled categories, while a regression problem has an outcome that is continuous values. Artificial neural network (ANN or just NN) is a more complicated system, designed to mimic a biological system where neurons interacts between layers. In the numerical approach, this is achieved by neurons sending out mathematical functions. The first ANN were developed by McCulloch and Pitts in 1943. They used the method to study signal processing in the brain [16]. A NN can be used to solve both classification and regression problems, while logistic regression is mainly applied to classification.

As a starting point for building our models for classification, we looked into the Taiwan credit card data [11]. This data set has been well studied with machine learning algorithms over the past few years, and we will in this report try to recreate some of the results from the scientific article *The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients* , written by I-Cheng Yeh and Che-hui Lien [19].

After developing our logistic regression and neural network models, we will evaluate the credit card data and compare the two methods. After this we will study Franke's function and perform a regression analysis using neural network. Since this is not a classification problem, but a regression problem, we will compare the results with the linear regression methods we evaluated in the first project of the FYS-STK4155 course [9]. To make this comparison, we will perform a regression analysis using the mean squared error and R^2 scores for evaluation, as well as plotting a 3D image of the function.

In the following two sections, we will present the two data sets used in this project and some theory about the models we have developed - Logistic Regression and Neural Networks. This will be accompanied by theory about the different metrics used to evaluate the models. We will then give a short explanation about how we executed the project, in the method section. The result and discussion section will be presented after this, before we sum up our main achievements and thoughts in the conclusion section. At the end of the report we have added acknowledgements, references and an appendix explaining the credit card data.

2 Data

2.1 Classification: Credit card data

For our studies about classification we used the Taiwan credit card data, provided by UCI [11]. This data set consists of research about credit card users default payments in Taiwan, and is divided into various categories. More information about how the data set is assembled can be found in Appendix A and the excel file with the raw data can be found in the git repository. The credit card data is flawed data, most likely caused by human errors when creating the data set. When examining the raw data, we could see deviations from the expected values explained in the data dictionary in Appendix A. We therefor had to remove these faulty data entries before using the data set further.

For the variable *EDUCATION*, we had to remove entries below 1 and above 4, and for *MARRIAGE* we have to remove entries below 1 and above 3. For the repayment status *PAY_i* we had to make sure that we only kept the entries $i = -1$ and $i = 1$ to 9. In figure 1, we can see how the *EDUCATION* entries is divided into the four categories defined by the data dictionary. Histograms for all categories can be found under **Hist** on Git.

In total, removing flawed entries, we reduced the credit card data from 30k data points to 4030 data points. This reduction were done in the Python file *credit_card.py*. We also saw some weird features in *BILL_AMT* and *PAY_AMT*, some credit card users have all entries for the two categories as zero, which we found a little odd. But we decided not to remove these entries. In the end, the data set had 1436 defaulters, and 2594 credit card users who did not default.

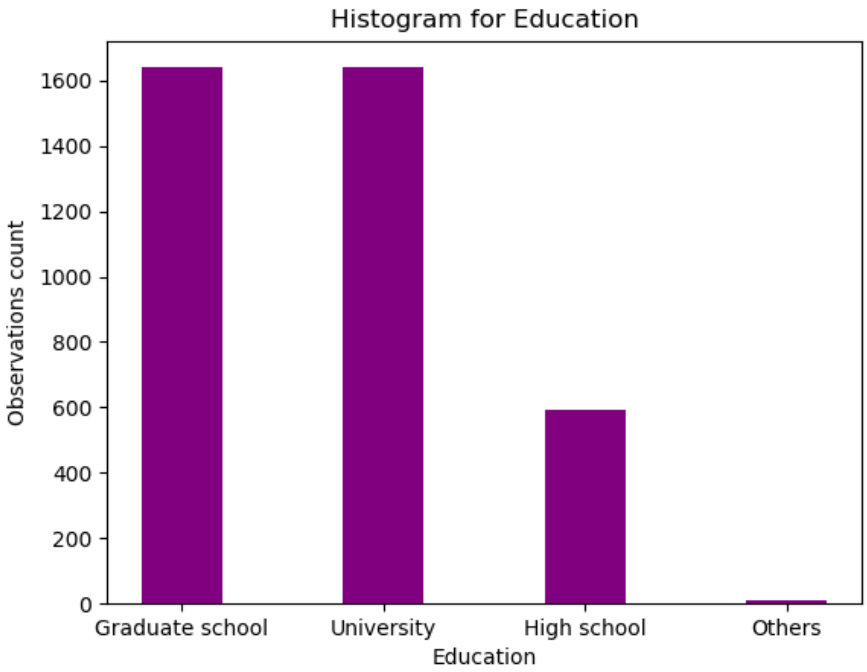


Figure 1: A histogram of the category *EDUCATION*, after the faulty data had been removed.

Next we made a correlation matrix, figure 2, showing correlation coefficients between the variables in the credit card data. The value of each cell is color coded, with a colorbar showing the value.

We can see from the correlation matrix that especially the *PAY* and *BILL_ATM* columns correlate decently with themselves. This is easily observed by the lighter squares in the figure. We can also see that the correlation between *MARRIAGE* and *AGE* is especially low. This is also an intuitive assumption, since a persons marital status, education, sex and age shouldn't have a big impact on the persons capability of defaulting or not. On the other hand, if we had information about the test subjects salary, this could cause more correlations. So in general we can say that this correlation matrix gives little useful information. To make the correlation matrix, we used Panda's *DataFrame.corr()*, with default values [10].

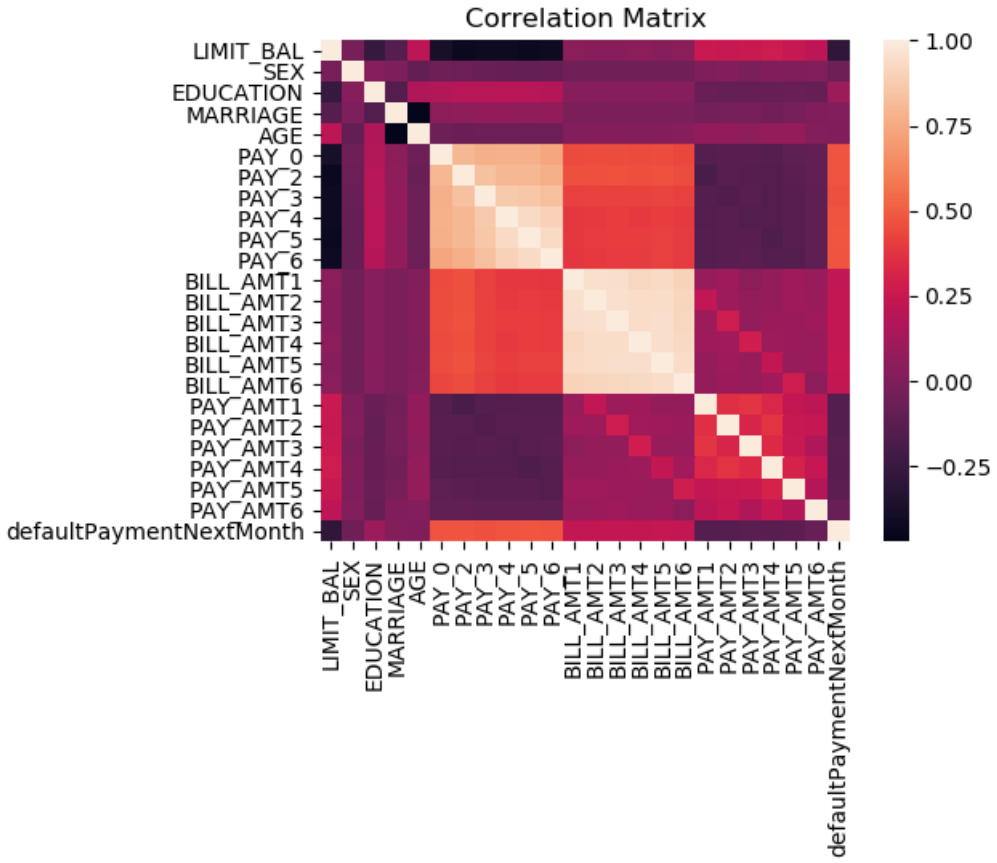


Figure 2: Correlation matrix, made before the credit card data were scaled.

2.2 Regression: Franke's function

The second set of data used in this project is generated using Franke's function, with random numbers $x, y \in [0, 1]$. This single-valued function of two variables is a weighted sum of four exponentials, which reads as follows:

$$f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)^2}{10}\right) \quad (1)$$
$$+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right)$$

We also added a normal distributed noise ($N(\mu, \sigma)$ where $\mu = 0$ and $\sigma = 1$), to match the data set used in project 1. Our data was then defined as $z(x,y) = f(x,y) + N(0,1)$.

3 Theory

3.1 Logistic Regression

This section is based on the article *Logistisk regresjon – anvendt og anvendelig*, by Magne Thoresen [15]. Within statistics, regression analysis in an important and common tool. A regression analysis looks at the correlation between a dependant variable and one or more independent variables. In contrast to a pure correlation analysis, which only shows if it is correlation between the variables, a regression analysis can show the degree to which one variable varies with another variable. We often divide regression into linear and non-linear.

First let us look briefly into linear regression to refresh our memory. This method is used when the response variable is measured on a continuous scale, and the model can be written as follows

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon, \tag{2}$$

where y is our response variable. $x_1..x_n$ is the set of n explanatory variables and ϵ represents the error of the model. The regression coefficients, $\beta_1...\beta_n$ is the ones we want to estimate. They give us the relationship between the explanatory variables and the response variable.

In many different research areas, we are often interested in binary response variables. In for instance medicine, we can have a typical response variable indicating illness or not illness. Or as in this project where we look into credit card data, our response variable tells us if someone defaults or not defaults. The most common regression model for these kind of responses, variables dependent on categories, is the logistic regression model.

In logistic regression, we want to model the likelihood for our response, for instance the probability that someone defaults in the credit card case or not. The logistic function is given by

$$p(t) = \frac{1}{1+e^{-t}} = \frac{e^t}{1+e^t}, \tag{3}$$

which is also known as the Sigmoid function. The variable t is given by the linear relation above ($t = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$). This function tells us about the probability that a data point x_i belongs to a category $y_i = \{0, 1\}$. Or as mentioned above, it provides the likelihood for an event. To easier see the similarity and difference between the logistic and the linear model, we can write the logistic function as

$$ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n. \tag{4}$$

The quantity $p/(1-p)$ is the odds, and the function $ln(odds)$ is often called the logit function. Now it is easier to see the linear dependence (the right hand side of the equation), however, we notice that this is not strictly related to the response variable as in the linear regression model.

In figure 3, we can see that linear regression gives us predicted probability values outside the 0 to 1 range, while logistic regression gives us a Sigmoid curve, with predicted probability values between 0 and 1. With logistic regression, we want to make probability predictions with a binary response outcome (0 and 1). In figure 4, we have a simple illustration of how logistic regression works. Firstly, we need a more complex cost function than in linear regression, the Sigmoid function. When the data input is sent through the Sigmoid function, we get values between 0 and 1. With a threshold we then decide if the predicted value is to be set to either 0 or 1, giving us binary variables.

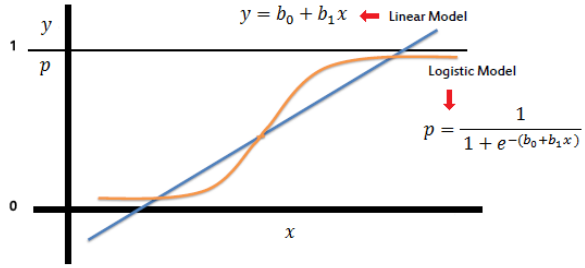


Figure 3: Illustration of linear versus logistic regression (Sigmoid curve) [13].

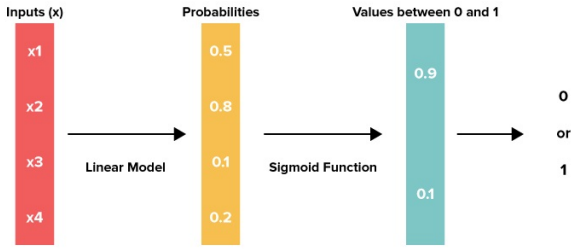


Figure 4: Illustration of logistic regression, or the Sigmoid function [12].

3.2 Neural Networks

This section is largely based on the lecture note *Data Analysis and Machine Learning: Neural networks, from the simple perception to deep learning*, written by Morten Hjorth-Jensen [2].

In supervised learning, the goal is to train a model to predict a known outcome. Neural Networks (NN) are supervised learning techniques that emulate the design of biological neural networks. The simplest method is the Feed-Forward Neural Network, and it is the one used in this project. It is a structure that consists of L layers of nodes, each connected to every node in the previous layer and the next, but not to others in the same layer. The connections between each node is characterized by a number called a weight. The nodes function like a neuron in a brain, receiving a signal, processing it, and then sending it on to the next neuron. The connections between the nodes function like synapses in brains, carrying the signals between the nodes and having varying signal strength. A higher value in the weight indicates a stronger neural connection.

The Universal Approximation Theorem loosely states that a neural network with one or more hidden layer, can give outputs that approximate other functions to an arbitrary degree. This is why it can be used to make models for machine learning, both classification and regression problems.

There are three types of layers in this network. An input layer, L hidden layers and an output layer. Broadly speaking, what happens is that the input layer receives input from the data set, then each node in the first hidden layer receive signals containing the data points from each node in the input layer. Each node computes the weighted sum of all the numbers from the input layer and their respective weights, then uses a so called activation function, and use that output to send to every node in the next hidden layer. Each node also has a bias that is added to the weighted sum of each node.

Let us first consider what happens in a single node j in the layer l. It will receive signals from layer $l-1$ on the form $y_1^{(l-1)}, y_2^{(l-1)}, ..., y_i^{(l-1)}, ..., y_{N_{l-1}}^{(l-1)}$. The processing will compute the following weighted sum

$$z_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^l y_j^{l-1} + b_j^l \tag{5}$$

w_{ij}^l is the weight between node i in previous layer $l-1$ and node j in curreny layer l. b_j^l is the bias of the curren node j. WHAT IS BIAS??? N_l is the number of nodes in layer l.

The next step to to use the activation function, f, on this weighted sum, before sending to to every node in the next layer $l+1$. So the output from this node is.

$$y_j^l = f(z_j^l) = f\left(\sum_{i=1}^{N_{l-1}} w_{ij}^l y_j^{l-1} + b_j^l\right) \tag{6}$$

This is recursive in the sense that at each layer l the output y_j^l is dependent on the output

from the previous layer $l-1$. And so we have

$$y_j^{l-1} = f(z_j^{l-1}) = f\left(\sum_{i=1}^{N_{l-2}} w_{ij}^{l-1} y_j^{l-2} + b_j^{l-1}\right)$$
$$y_j^{l} = f\left(\sum_{i=1}^{N_{l-1}} w_{ij}^{l} f\left(\sum_{i=1}^{N_{l-2}} w_{ij}^{l-1} y_j^{l-2} + b_j^{l-1}\right) + b_j^{l}\right)$$

This goes back to the input layer, $l = 0$, so the total expression (bad formulation...) for node i the final layer $l = L$ becomes

$$y_j^L = f\left(\sum_{i=1}^{N_{L-1}} w_{ij}^L f\left(\sum_{i=1}^{N_{L-2}} w_{ij}^{L-1} f\left(...f\left(\sum_{i=1}^{N_0} w_{ij}^0 y_j^0 + b_k^0\right)...\right)\right)\right) \tag{7}$$

Here we have assumed the activation function is the same for every layer.

7 can be expressed compactly by making use of vector and matrix notation from linear algebra. Let $\mathbf{y}^l$ and $\mathbf{b}^l$ be the output and biases for layer l. We introduce the weight matrix W^l that contain all the weights between every node from layer $l-1$ to nodes in layer l. Thus 6 becomes

$$\mathbf{y}^l = f\left(W^l \mathbf{y}^{l-1} + \mathbf{b}^l\right) \tag{8}$$

We see from this that W^l must be a $N_l \times N_{l-1}$ matrix. And accordingly, we can express the final output of the process as

$$\mathbf{y}^L = f\left(W^L\left(f\left(W^{L-1}\left(...f\left(W^1\mathbf{y}^0 + \mathbf{b}^0\right)...\right) + \mathbf{b}^{L-2}\right)\right) + \mathbf{b}^{L-1}\right) \tag{9}$$

3.2.1 Activation functions

The function f that is used in each node to process the signals before sending them forward is called an activation function. In order for the universal approximation theorem to apply, we have the following requirements on the activation function

1. Non-constant.
2. Bounded. Meaning it has a non-infinity limit. (explain better)
3. Monotonically increasing.
4. Continuous.

The output from the weighted sum z is linear, so we must impose an additional function in order to not violate condition 2. In a classification problem, where the outputs to predict are discrete categories that are labelled, it is most common to use Sigmoid because it has outputs between 0 and 1, and then we use the softmax function to assign a probability to each category at the output layer. The softmax is convenient for this purpose because it sums up to 1, and thus conserves the basic understanding of probability.

$$f(x) = \frac{1}{1+e^{-x}}$$
$$f(\mathbf{x}, i) = \frac{e^{x_i}}{\sum_{i=0}^{N} e^{x_i}}$$

Here i is the specific node where this is applied, and the sum is over all the nodes in the inputs from the previous layer.

For regression problems, where the outputs to predict are continuous values, it is most common to use Rectified Linear Unit (ReLU). The reason for selecting it was because the data set for this project only has positive values, including over 1, which the Sigmoid cannot give as output.

$$f(x) = \max(0, x)$$

3.2.2 Back propagation

We can see from 9 that what must be determined in order to tune the outcome are the weights and bises. We must define a cost function $C(W)$ that measures the error between our predicted outcome $\hat{\mathbf{y}} = \mathbf{y}^L$ and the actual known data $\mathbf{y}$. The learning part is to repeat the process through the layers of the network where the weights and biases are adjusted each time to make the outcome approximate the correct values. This is done by minimizing the cost function. The way this is done is through back propagation. After each run through the network, the error is estimated, and then the weights are adjusted.

The method of choice for minimizing the cost function is called gradient descent, where we find the combination of all the weights that yield the lowest output value for the cost function. We can see that this method is similar to a regression problem, but the difference is that we instead of using an orthonormal set of functions that span a vector space, we instead determine them iteratively.

The cost function used will, like the activation function, depends on the nature of the known quantity that we want to approximate. If we are using the NN for a classification problem, then the cost function used is the so-called cross entropy

$$C(W) = -\left[\mathbf{y}^T \ln(\hat{\mathbf{y}}) + (1-\mathbf{y})^T \ln(1-\hat{\mathbf{y}})\right] \tag{10}$$

For regression problems, we will use the mean squared error (MSE).

$$C(W) = \frac{1}{N}(\mathbf{y}-\hat{\mathbf{y}})^T(\mathbf{y}-\hat{\mathbf{y}}) \tag{11}$$

Here $\mathbf{y}$ is the known target data and $\hat{\mathbf{y}} = \mathbf{y}^L$ is the output of the final layer of the NN. How 10 and 11 are used in gradient descent will be discussed in the dedicated section about the method.

3.3 Gradient descent

Gradient descent is a method used in both logistic regression and neural networks. In both cases we want to minimize a cost function that measures the error we commit when using the method. There are two slightly different variants of gradient descent: Steepest descent and stochastic gradient descent (SGD).

3.3.1 Steepest descent

Given a function $F : \mathbb{R}^n \mapsto \mathbb{R}$, which takes input $\mathbf{x}$. We can minimize the function $F(\mathbf{x})$ is to move in the parameter space $\mathbb{R}^n$ such that each step decreases the singlue value output of $F(\mathbf{x})$.

Given an initial $\mathbf{x}_0$, which is either generated randomly or made using an educated guess, we can calculate the gradient $\nabla F(\mathbf{x}_0)$. The gradient is a factor that points in the direction of the steepest increase in the value of $F(\mathbf{x}_0)$. We can therefore move in the negative direction with a given step length γ, and then repeat the process until one has an error that is sufficiently small, or when the maximum number of steps is reached.

Algorithm 1: Steepest descent

Given $\mathbf{x}_0$;
for $k \in (1, 2, ..., N)$ **do**
  $\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \nabla F(\mathbf{x_k})$;
  if $||\nabla F(\mathbf{x_k})|| \leq \epsilon$ **then**
    terminate loop;
  else
end

Where ϵ is an arbitrary lower limit on accuracy. γ_k is called the step size, and it must be sufficiently small so that $F(\mathbf{x}_{k+1}) \leq F(\mathbf{x}_k)$. [17]

3.3.2 Stochastic gradient descent

Stochastic Gradient Descent (SGD) is a modification of the steepest descent method, and uses what can be described as a stochastic approximation to it. Instead of using the entire data set each time the cost function is computed, it instead uses a randomly selected subset. This saves computing time, but in turn the method converges to the cost function's minimum slower than a steepest descent method [18].

3.3.3 Finding the gradient of the cost function

Because we want to minimize the cost function C by finding the best combination of weights and biases, we need the gradient of C with respect to these parameters.

$$\frac{\partial C}{\partial W^l} = \frac{\partial C}{\partial \mathbf{y}^l}\frac{\partial \mathbf{y}^l}{\partial \mathbf{z}^l}\frac{\partial \mathbf{z}^l}{\partial W} \tag{12}$$

$$\frac{\partial C}{\partial \mathbf{b}^l} = \frac{\partial C}{\partial \mathbf{y}^l}\frac{\partial \mathbf{y}^l}{\partial \mathbf{z}^l}\frac{\partial \mathbf{z}^l}{\partial \mathbf{b}^l} \tag{13}$$

Here we have used the chain rule to expand the expression with matrix products. We will handle regression and classification separately, since they use the different cost functions 11 and 10.

Let us first consider regression. We see from 12. That we need three partial derivatives to compute the matrix product. Starting with the first we see that

$$C = \frac{1}{2}(\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}})$$
$$\frac{\partial C}{\partial \mathbf{y}} = \frac{1}{2}\Big((1-0)^T(\mathbf{y} - \hat{\mathbf{y}}) + (\mathbf{y} - \hat{\mathbf{y}})^T(1-0)\Big) = \mathbf{y} - \hat{\mathbf{y}} \equiv \delta^L$$

For the second one, we use that $\mathbf{y}^l = f(\mathbf{z}^l)$

$$\frac{\partial \mathbf{y}^l}{\partial \mathbf{z}^l} = \frac{\partial}{\partial \mathbf{z}^l} f(\mathbf{z}^l) = f'(\mathbf{z}^l)$$

For the third factor we use that $\mathbf{z}^l = W^l \mathbf{y}^{l-1} + \mathbf{b}^l$. Then we have that

$$\frac{\partial \mathbf{z}^l}{\partial W^l} = \left(\mathbf{y}^{l-1}\right)^T \tag{14}$$

Where we transpose the expression keep the dimensions correct. Putting it all together 12 becomes

$$\frac{\partial C}{\partial W^l} = \delta^l \left(\mathbf{y}^{l-1}\right)^T$$

Where we have defined $\delta^L \equiv \hat{\mathbf{y}}^L - \mathbf{y}$ as the error made after each layer. We also have that $f'(z^L) = 1$. Let us now consider 13, and we begin with the first factor.

$$\frac{\partial C}{\partial \mathbf{y}^l} = \frac{\partial C}{\partial \mathbf{z}^{l+1}} \frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{y}^l} = \left[\left(\delta^{l+1}\right)^T W^{l+1}\right]^T = \left(W^{l+1}\right)^T \delta^{l+1}$$

We already have the second factor from previously. The third and final product is simple

$$\frac{\partial \mathbf{z}^l}{\partial \mathbf{b}^l} = 1$$

Putting it all together we now have the following expressions for 12 and 13, as well as definition of the error and a recursive formula for it:

$$\frac{\partial C}{\partial W^l} = \delta^l \left(\mathbf{y}^{l-1}\right)^T \tag{15}$$

$$\frac{\partial C}{\partial \mathbf{b}^l} = \delta^l \tag{16}$$

$$\delta^L = \hat{\mathbf{y}} - \mathbf{y} \tag{17}$$

$$\delta^l = \left(W^{l+1}\right)^T \delta^{l+1} \tag{18}$$

The derivative of the identity function $f(x) = x$ is just 1. The same procedure can be followed with the cost function being the 10 instead. We will only list the results here.

$$\frac{\partial C}{\partial W^l} = \delta^l \left(\mathbf{y}^{l-1}\right)^T \tag{19}$$

$$\frac{\partial C}{\partial \mathbf{b}^l} = \delta^l \tag{20}$$

$$\delta^L = \hat{\mathbf{y}} - \mathbf{y} \tag{21}$$

$$\delta^l = \left(W^{l+1}\right)^T \delta^{l+1} \circ \sigma(\mathbf{z}^l) \left(1 - \sigma(\mathbf{z}^l)\right) \tag{22}$$

Where $\sigma(x)$ is the Sigmoid function.

So what the back propagation ultimately does is this: It goes through the neural network until the last layer L, then it is computes 21, then it uses 22 recursively backwards through the layers. Then it computes the gradients in 19 and 20 to perform the gradient descent, then it updates W^l and $\mathbf{b}^l$ and then starts over again.

3.4 Model evaluation

3.4.1 Accuracy score, Confusion matrix and AUC

To measure the performance of a classification problem we can use the Accuracy score. The Accuracy score is defined as how many correctly guessed targets (t_i) divided by the total number of targets. A perfect score is 1, and only a perfect classifier will achieve this. The accuracy score is given by

$$\text{Accuracy} = \frac{\sum_{i-1}^{n} I(t_i = y_i)}{n} \tag{23}$$

where y_i is the model output and n is the number of samples. I is the indicator function, where

$$I = \begin{cases} 1, & t_i = y_i \\ 0, & t_i \neq y_i \end{cases}$$

Since the credit card data is biased, the accuracy score is not a good enough measurement for evaluating the classifier performance. For a better optimization, there are other metrics which can be used, such as precision, recall and the F1 score. Before looking further into these metrics, lets introduce what is called a confusion matrix, given in figure 5.

	Predicted: Not default	Predicted: Default
Actual: Not default	True Negatives (TN)	False Positives (FP)
Actual: Default	False Negatives (FN)	True Positives (TP)

Figure 5: Confusion matrix. *Default* - The credit card user do not pay. *Not default* - The credit card user pays.

The confusion matrix tells us that the classifier indicates the class correctly if we have *True*, and incorrectly if we have *False*. *Positive* and *Negative* tells us if the classifier predicted the desired class. When using the credit card data, *Positive* corresponds to *Default*, since these are the users we want to foresee/predict.

Thus, the confusion matrix can be explained as the following [1]:

- **TN**: The classifier predicts *Not default*, but the user actually pays.
- **TP**: The classifier predicts *Default*, and the user actually do not pay the credit card bills.
- **FN**: The classifier predicts *Not default*, but the credit card users actually defaults.
- **FP**: The classifier predicts *Default*, but the credit card user actually pays.

From the information provided by the confusion matrix, we can now calculate the precision, recall and f_1 score [1]. Precision is defined as the proportion of positive predictions (default), that is actually correct. In other words, when the classifier predicts default, the precision tells us how often this prediction is correct. Thus, the precision is given as

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{24}$$

The proportion of positive observed values (defaults), correctly predicted as such, is called the recall. So, in order to find the recall, we must calculate

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{25}$$

In the case of the credit card data, a false negative can be considered worse than a false positive. It is better that the model predict a person to default, and it turns out that the person actually pays. If we have a false negative, the person is predicted as non-default, but turns out to default. In this case, one could look for a better recall to optimize the classifier.

Using the values for precision and recall, we can now calculate the harmonic mean, called the f_1 score

$$f_1 = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{26}$$

This metric us useful when we want to have classification model with the optimal balance between precision and recall, then we can try to maximize the f_1 score ([4]).

To visualize precision and recall, the Receiver Operating Characteristic (ROC) is a common option. This curve illustrates the true positive rate (TPR) versus the false positive rate (FPR), as a function of the model's threshold for classifying a postive. In figure 6, we can see an illustration of a ROC curve.

The red line in figure 6 indicates a random classifier. The calculated classification model is given by the blue line. As the figure demonstrates, we can sort of "move along the curve" by adjusting the threshold (the number given above the blue dots). As we can see, decreasing the threshold, we move upwards to the right along the curve. In the upper right corner at a threshold equal to 0, we have the case where $TPR = FPR = 0$, which tells us that all the predictions are positive. In the opposite case, with a threshold of 1, we identify no predictions as positive. This means that we have no true positive and no false positives, $TPR = FPR = 0$.

Further, the ROC brings us into the Area Under Curve (AUC) metric, which provides the overall performance of the classification model, based on the area under the ROC curve. The AUC is given by [19]

$$\text{AUC} = \frac{\text{area between model curve and baseline curve}}{\text{area between theoretically best curve and baseline curve}} \tag{27}$$

The AUC metric is a number between 0 and 1. For a good classification model, we want AUC to be as close to 1 as possible. If the AUC $= 0.5$, this means we have a random classifier.

Another way to evaluate the effectiveness of a binary classifier, is the cumulative gain chart ([5]). This plot is generated using the true target values, and the predicted probabilities for each class returned by the classifier. An illustration for a cumulative gain chart can be viewed in figure 7.

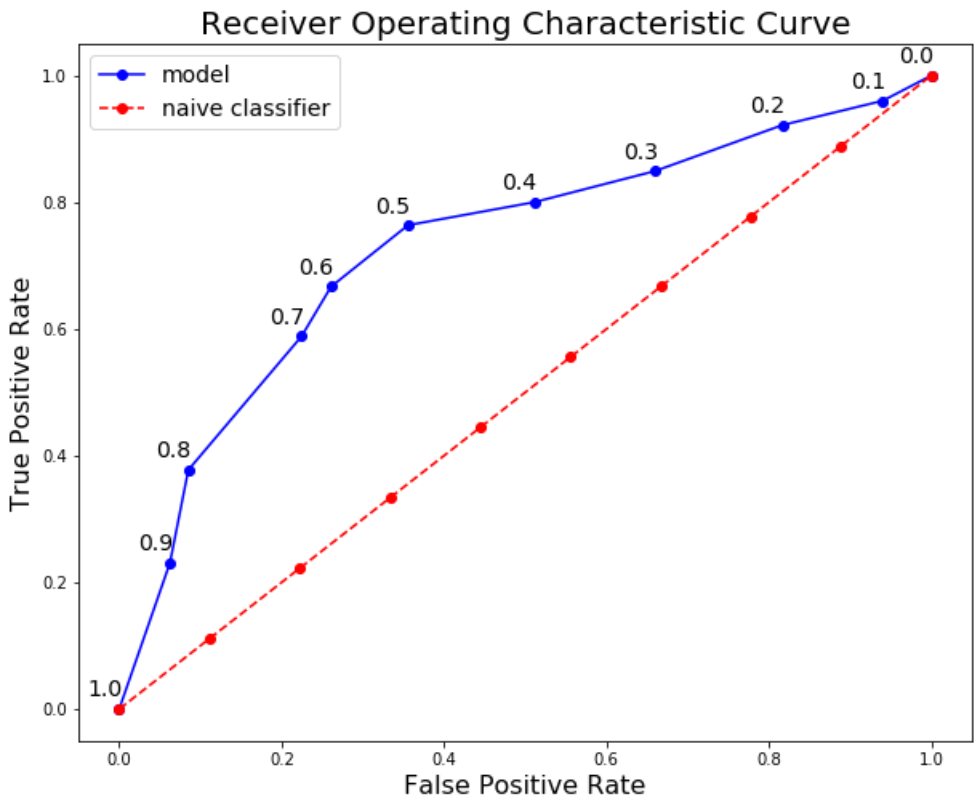


Figure 6: Receiver Operating Characteristic Curve, [4].

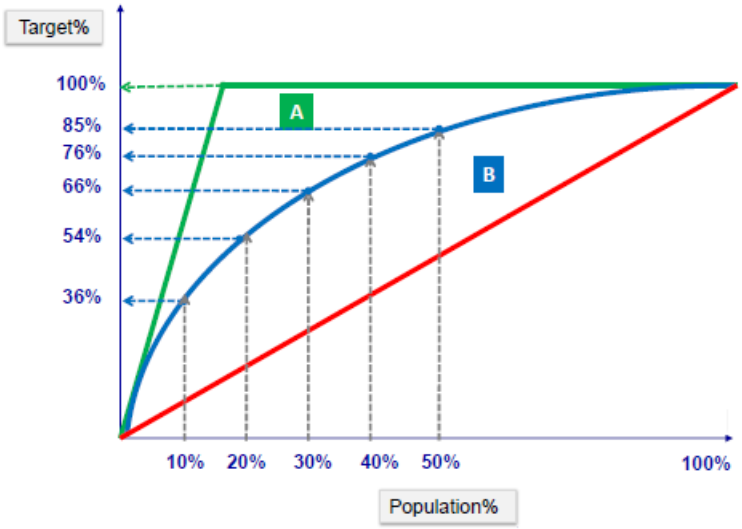


Figure 7: Cumulative Gain Chart, [14].

The gains chart associated to the model, is illustrated as the blue curve. The random classifier, or baseline, is shown in red. The optimal model is the green curve. So, what does this tell us? If we take 10 % of the targets with the highest probability for a positive response, we will get 36 % of all the positive responses ([7]).

3.4.2 Mean squared error & R^2

When performing regression analysis with neural network, we can evaluate the performance of the model by considering the mean squared error (MSE) and R^2 score, taken from section 2.2 in Project 1 [9]. The MSE function is given by:

$$\text{MSE}(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = \frac{1}{n} \sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2 \tag{28}$$

where $\boldsymbol{z}$ represents the data and $\tilde{\boldsymbol{z}}$ the model (fitted data). With other words, $\tilde{z}_i$ is the predicted value of the $i-th$ sample and $\tilde{z}_i$ is the corresponding true value. The MSE gives an average value for the difference between an known data point and the predicted data point. The closer the result is to zero, the better the model.

The statistical coefficient of determination, R^2, score is used for indicating how well the model is predicting the actual values in the data set. The closer the R^2 score is to 1, the better the model. This is because it tells us the proportion of the variance in the dependent variable which is predicted by the independent variable. With other words, $\text{R}^2 = 1$ means that the model is a 100% fit to the original data set. The R^2 score function is given by:

$$\text{R}^2(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = 1 - \frac{\sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1} (z_i - \bar{z}_i)^2} \tag{29}$$

where the mean value of $\boldsymbol{z}$ is defined as

$$\bar{z} = \frac{1}{n} \sum_{i=0}^{n-1} z_i \tag{30}$$

4 Method

4.1 Logistic Regression on the Credit Card data

To calculate the β-values used in our logistic regression model, we tried to use both the Stochastic Gradient Descent (SGD) method and steepest descent method. Both methods yielded pretty good results, however the steepest descent method seemed to perform slightly better. Since we also wanted to look at the threshold parameter, it was easier to use the steepest descent, since this does not include the parameter η in addition. Also, since our data set is not so big, we really don't need the computational time gain that Stochastic Gradient Descent provides.

We tested for different λ values in the range $0.1 - 1e^{-7}$, and we found that $\lambda = 0.001$ gave the best performance. Then we plotted precision, recall and F_1 score to observe how these metrics depended on the threshold (8). To get a slightly higher recall for our model, we adjusted our threshold from 0.5 (default value) to 0.44. After deciding the parameters, we plotted the Confusion Matrix (figure 9), ROC curve (figure 11) and The Cumulative Gain chart (figure 10). In the end, we printed our final values for accuracy, precision, recall, F_1 score and AUC. These results can be viewed in table 1.

We used scikit-learns built in package to calculate the AUC, while the other metrics is calculated using our own functions. The ROC curve is plotted entirely with scikit-learn, while the Cumulative Gain Chart is plotted using scikit-learns plotting function ([6]) together with prediction probabilities calculated with our own function and own model. We also plotted the latter entirely with scikit-learn for comparison.

4.2 Neural Network on the Credit Card data

The NN together with back propagation and gradient descent is made to find the combinations of weights between neurons and biases of each neuron that minimizes the cost function. However, there are a number of other parameters that will affect the quality of the output that the NN will produce. These are so-called hyperparameters, and in project we made efforts to determine their values.

We primarily concerned ourselves with two such parameters: The learning rate η of the SGD, and the regularization λ that regulates output to counteract divergent behaviour in the gradient .

The other hyperparameters were handled like this. Because fine-tuning all these parameters would be difficult with the computing power accessible, we made several simplifying assumptions.

1. The number of layers were kept to just one hidden layer. So the whole NN has one input layer, one hidden and one output layer.
2. The number of nodes in that hidden layer was set to 50.
3. The activation functions were the same for every layer: Sigmoid and softmax.
4. The number of epochs and the size of the batches were kept constant at 100 and 80 respectively. There was some trial and error in determining these.

5. The initial random values for the weights and biases. They were given random values drawn from a Gaussian distribution, $N(0,1)$.

The values of these parameters are listed in the program code.

The way we determined η and λ was by iterating over every combination of these as inputs to the training of the NN. To keep computing time reasonable, as tested these values for the both of them: $10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}$. For every combination of value, the NN was initialized and trained, and the R^2 score was computed and recorded. We then displayed the results in a 2D grid that revealed which combination yielded the best score. We then tested this best NN with R^2, $F1$, AUC etc.

4.3 Neural Network Regression on Franke's function

Training a NN for a regression problem was largely the same as for classification discussed above. We still focused on optimizing η and λ for the network, and the other hyperparameters were left as they were. The only exception was the initial values for the weights and biases. To avoid divergent behaviour they had to be reduced to 10% of $N(0,1)$. We also used Mean Squared Error (MSE) instead of R^2 to track the success or failure of the NN.

As a test, we then used scikit learn's *sklearn.neural_network.MLPRegressor* to get accurate results. We used the default values, beside from max_iter = 1000, ReLU, SDG and tol=$1 * 10^{-4}$. The results from this can be found in table 2, as an extra comparison factor after our own program was correctly fixed.

5 Results and discussion

5.1 Logistic Regression

As we see in table 1, we get an $AUC = 0.83$. This means that we have a 83% chance that our model will be able to distinguish between a positive class and a negative class ([3]). We also observe that our model gets a pretty high accuracy (about 80%) and a f_1 score of 0.705. These are the results with $\lambda = 0.001$ and a threshold of 0.44.

The relationship between precision, recall and f_1 score, can be viewed in figure 8. From this figure we observe that the f_1 score would be pretty much the same if choosing a threshold between 0.26 and 0.6 (roughly). In addition, we see that the the recall is higher in the lower part of this range.

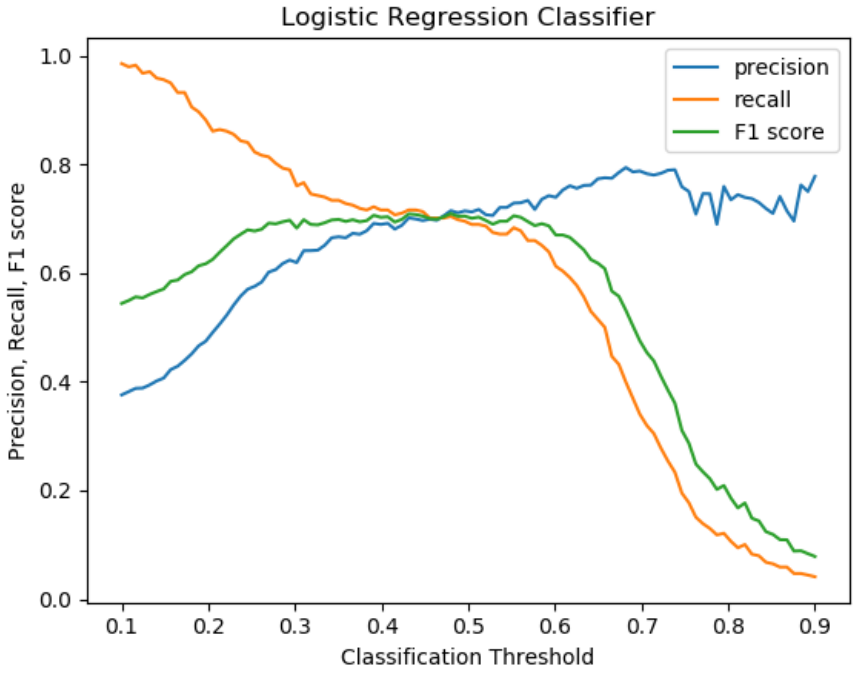


Figure 8: Precision, recall and f_1 score as a function of threshold

The Confusion Matrix obtain by our Logistic Regression classifier is represented in figure 9.

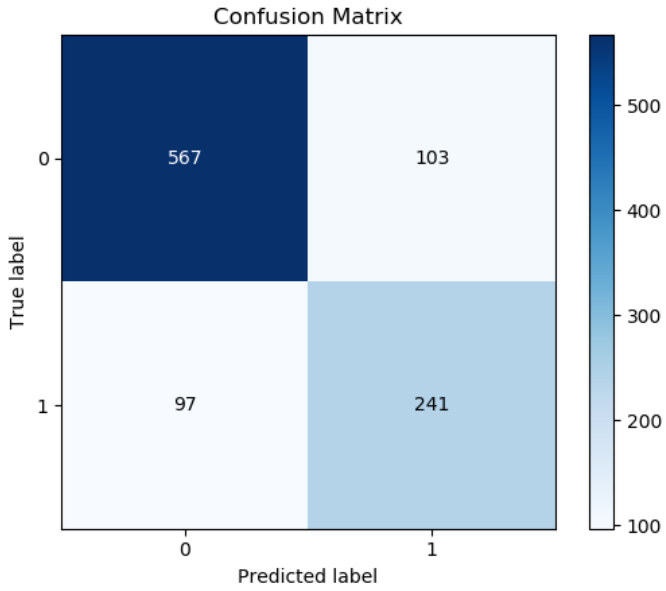


Figure 9: Confusion Matrix for Logistic Regression, where 1 means default and 0 means non-default.

We can see that the true predictions dominates, with 567 true negatives (TN) and 241 true positives (TP). We have 103 false positives (FP), meaning that the model predicts 103 of the targets as defaulters, when they are actually non-defaulters. Further, the model predicts 97 targets as non-defaulters, when they were actually defaulters. This is the number of false negatives (FN).

What we would expect to observe with the confusion matrix if we chose a higher recall, is that the number of false negatives would decrease. However, this goes at the expense of the accuracy, so it becomes an assessment one has to do. We checked that the confusion matrix behaved as expected from the theory, and in the GitHub repository one can see the confusion matrix for the extreme cases when the threshold is 0 and 1 (actually 0.9 due to issues with zero division).

The Cumulative Gains Curve achieved with Logistic Regression on the credit card data, is given in figure 10 below. As we may observe from the figure, we may have just below 40 % of data available, but still be able to say with 70% security that we detect a default. Having 60% of the data with the highest probability of a positive response, we will get above 80% of the positive responses.

The ROC curve for our binary classifier (using scikit-learn) is given by figure 11. We can clearly see that the classifier is better than having a random classifier (the linear line) where $AUC = 0.5$.

A similar plot as figure 10 using the probabilities obtained from scikit-learn can be found in the GitHub-repository for comparison.

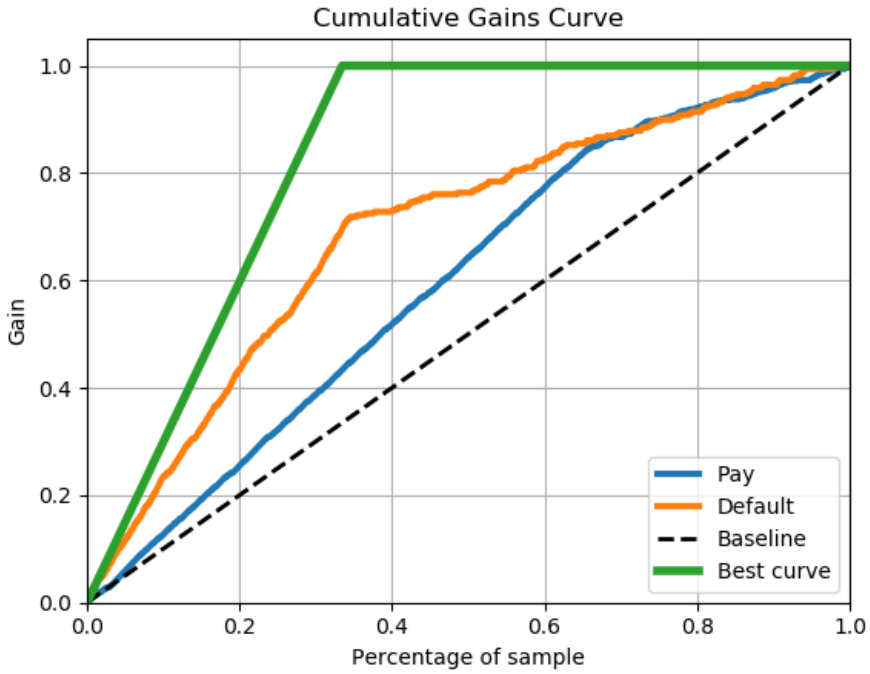


Figure 10: Cumulative gains curve for Logistic Regression

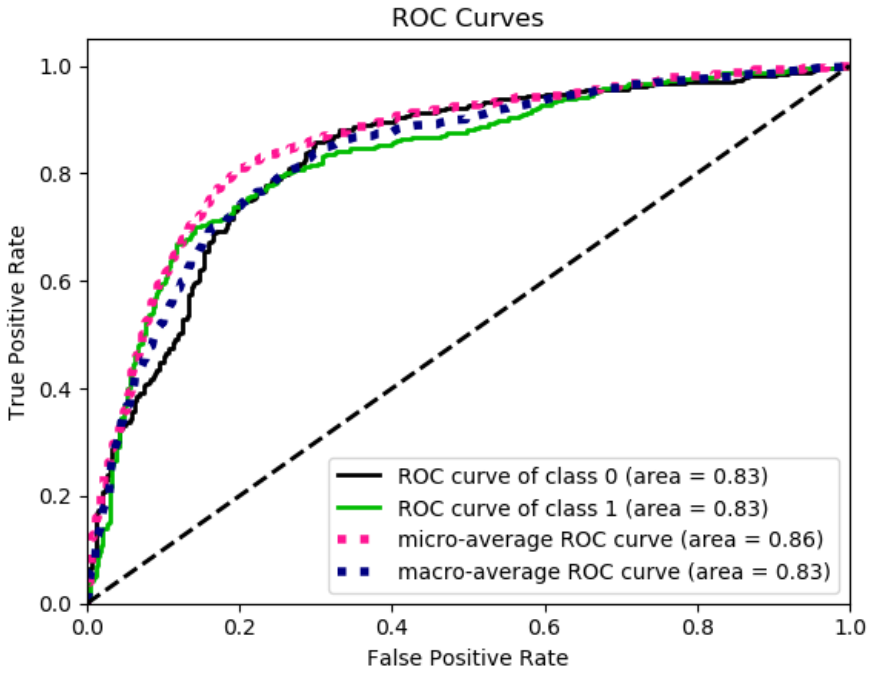


Figure 11: ROC for Logistic Regression, with scikit learn

5.2 Neural Network

The Confusion Matrix for our Neural network is given by figure 12. As for logistic regression, we can see that the true predictions dominates, with 520 true negatives (TN) and 237 true positives (TP). Further, we have 150 false positives (FP) and 101 false negatives.

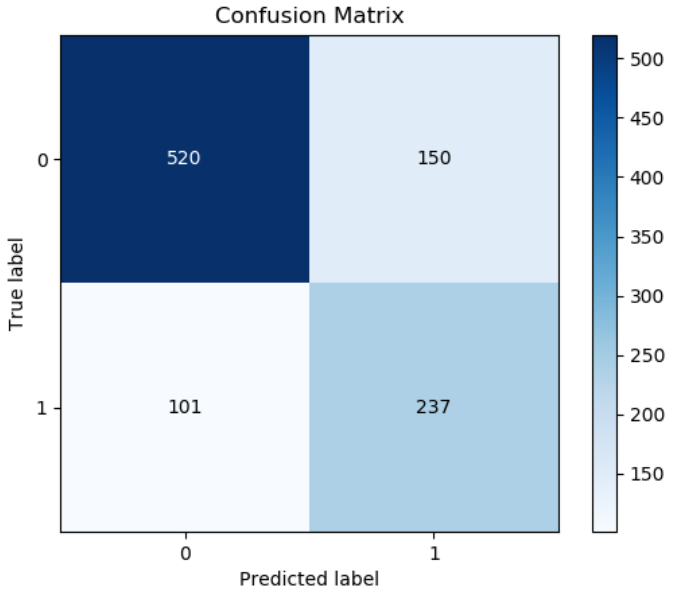


Figure 12: Confusion Matrix for Neural Network, where 1 means default and 0 means non-default.

To find the best hyperparameters, we made a heatmap of accuracy scores for NN. We found that $\eta = 10^{-4}$ and $\lambda = 10^{-7}$ yields the best accuracy, which is 78%. Although, further in our calculations we decided to use $\lambda = 10^{-2}$, which still gave an accuracy of 76%. This was mainly to avoid possible numerical error by using so small numbers.

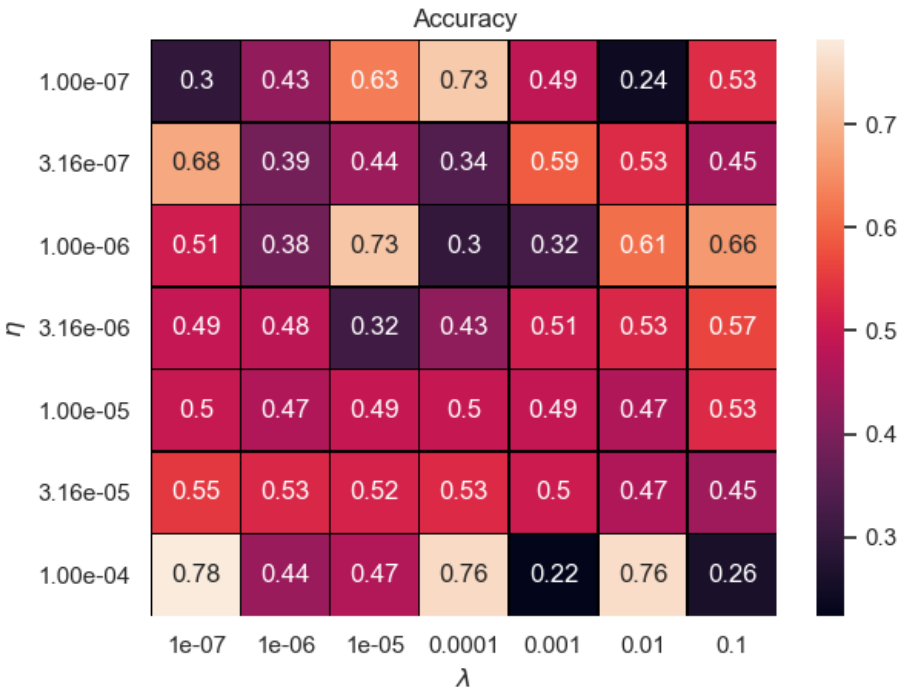


Figure 13: Accuracy heatmap for neural network

The Cumulative Gains Chart produced from our neural network code is shown in figure 14. Also here we are able to predicate the outcome with a 70% certainty with only 40% of the credit card data. To reach 80% certainty, we have to increase the amount of data to 60%.

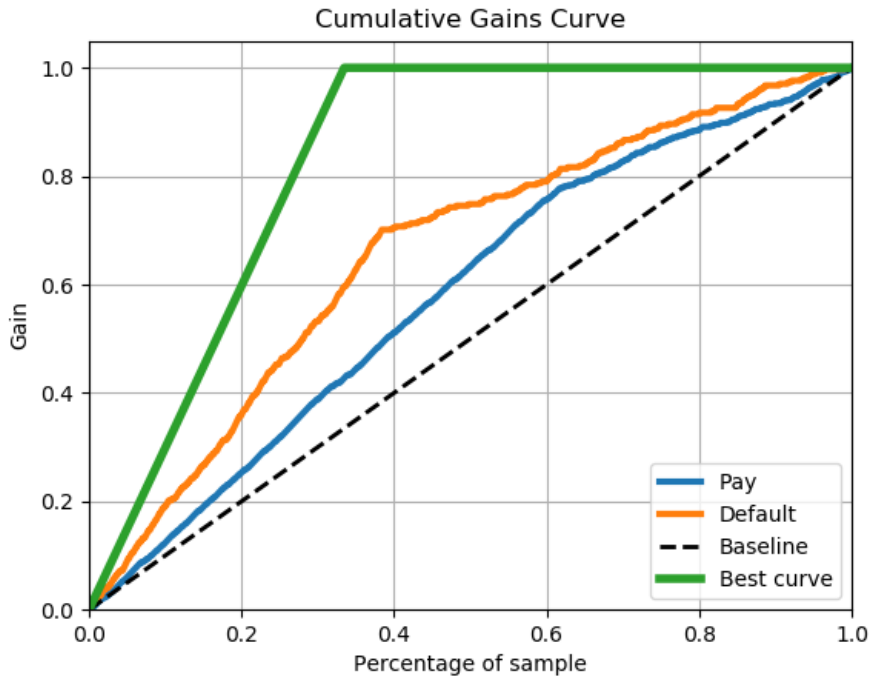


Figure 14: Cumulative gains curve for Neural Network

5.3 Neural Network Regression

In the last part of the project, we used our neural network solver to perform a linear regression on Franke's function. To evaluate this method we calculated the MSE and R^2 score. In figure 15 and 16, we can see heatmaps for MSE and R^2 scores, over various hyperparameters. From these maps we chose $\eta = 1 * 10^{-2}$ and $\lambda = 1 * 10^{-4}$, since this combination gave the highest R^2 score and the lowest MSE. See table 2 for final MSE and R^2 scores for both neural network regression methods, as well as figure 18 for 3D plots of the final models.

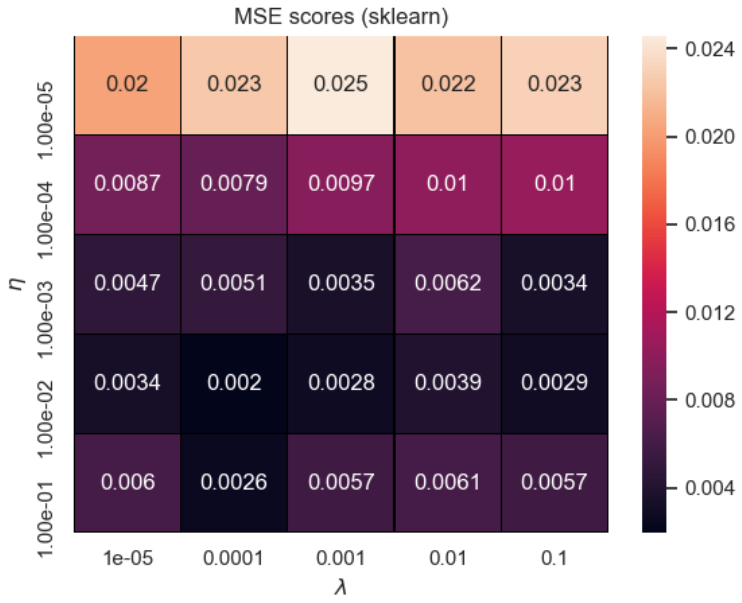


Figure 15: MSE heatmap for neural network regression, on Franke's function

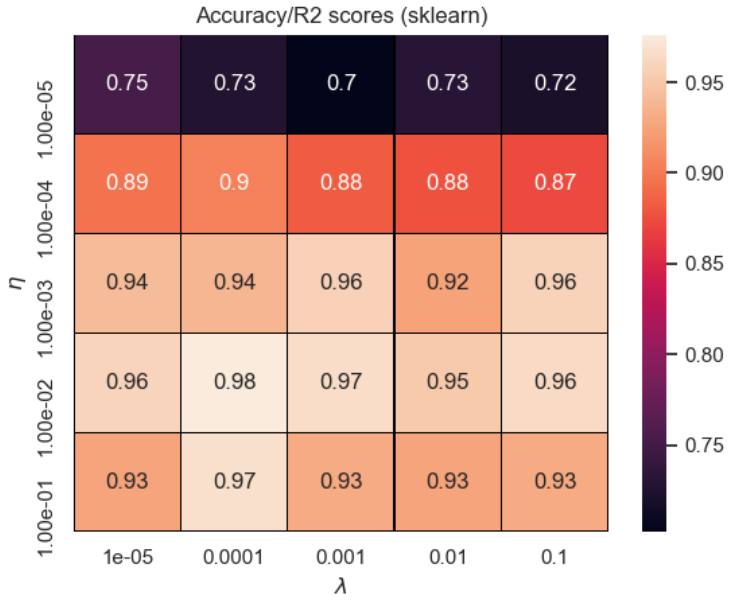


Figure 16: R2 heatmap for neural network regression, on Franke's function

5.4 Comparison of logistic regression and neural network

Table 1: Comparison of Accuracy, Precision, Recall, f_1 and AUC values for logistic regression and neural network. For neural network we use $\eta = 1 * 10^{-4}$ and $\lambda = 0.01$. Epochs: 100, batch size: 80, hidden neurons: 50. For logistic regression we used $\lambda = 0.001$.

Classification	Logistic Regression	Neural Network
Accuracy	0.803	0.751
Precision	0.706	0.612
Recall	0.704	0.701
F_1	0.705	0.654
AUC	0.831	0.832

From table 1 we have made a comparison of the accuracy of the logistic regression and the neural network classification. From section 2.5, we know that we want an accuracy score as close as possible to 1. From the table we can see that the best accuracy score were for logistic regression, with $\text{Accuracy}_{LG} = 0.803$, while $\text{Accuracy}_{NN} = 0.751$. From the table we can see that logistic regression in general does better than neural network, while the AUC score only differ with 1 one the third decimal. Since the score is above 0.5 we know that we don't have a random classifier, and values as high as 0.8 is pretty good for this data set. As stated above, the credit card data is biased , and not so easy to make predictions of.

In both the Confusion Matrices, we see that the true predictions dominates. The false negatives are lower than the false positives in both cases, but for the neural network we have quite a few more false positives. The difference between these results only affects the accuracy with a couple of percentage, and in most cases a slightly higher rate of false positives is not a big issue. But again, it depends on the purpose of the model.

In figure 17, we have gathered the Cumulative Gain plots for logistic regression (left) and the neural network (right) to easier compare them. As mentioned earlier, they both show better performance than a random classifier, and they look pretty much similar. However, we observe that the class default and the class pay overlap with a large portion of the sample in logistic regression.

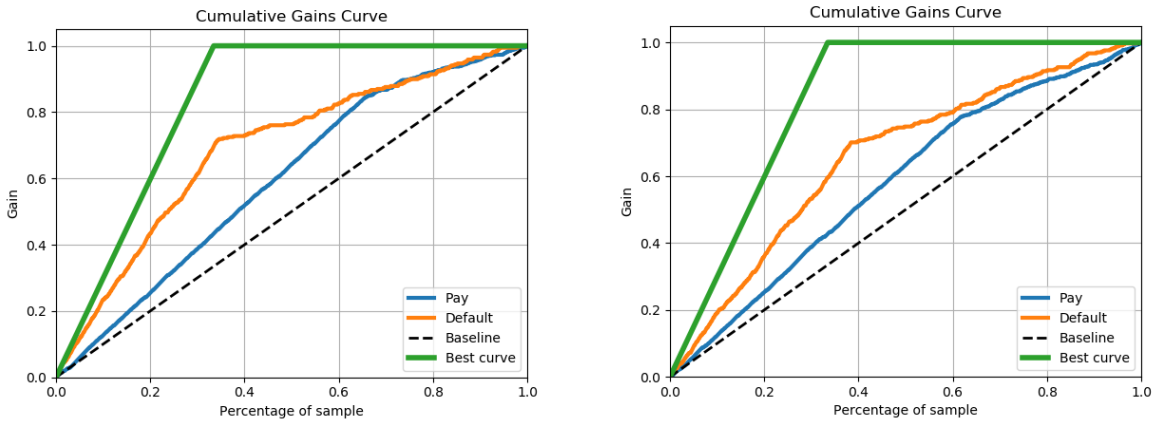


Figure 17: Cumulative gains curves for logistic regression (left) and neural network (right). Threshold 0.44.

5.5 Comparison of neural network regression with Project 1

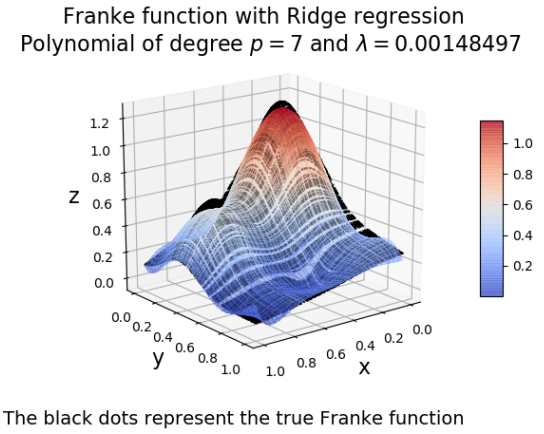
In table 2, we have listed all MSE and R^2 scores for neural network regression and linear regression from project 1. In project 1, we used OLS, Ridge and Lasso regression with k-Fold resampling to find the best models. All methods were performed on Franke's function, with the same random data points. For neural network regression, we have result from our own model, as well as scikit learn's MLPRegressor (default values and activation="relu", solver="sgd", learning_rate='constant', learning_rate_init=0.1, max_iter=1000 and tol=1e-5) [8].

Table 2: Comparison of MSE and R^2 with neural network regression and the best OLS, Ridge and Lasso regression on Franke's function, from project 1. Table includes the chosen polynomial degree and λ value for OLS, Ridge and Lasso. As well as the optimal λ and η values for neural network regression, with and without using scikit learn (SL). Calculated on the whole data set.

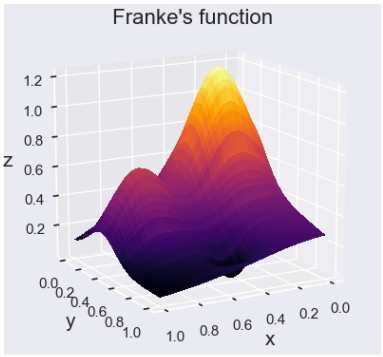
Regression analysis	**Neural Network Regression**	**NNR SL**	**OLS Regression**	**Ridge Regression**	**Lasso Regression**
MSE	0.0028	0.0038	0.0021	0.0015	0.0104
R^2	0.9657	0.9539	0.9751	0.9822	0.8735
p	5	5	5	7	4
λ	$1 * 10^{-4}$	$1 * 10^{-4}$	-	0.00148497	$1.67683 * 10^{-6}$
η	$1 * 10^{-2}$	$1 * 10^{-2}$	-		-

The best R^2 score we obtained were for Ridge regression with polynomial degree 7. This method had the best MSE value as well. OLS regression has the second best results, with our own neural network second to last. As we concluded in project 1, also this time Lasso regression gives the worst result. Although, this could be caused by to bad fine-tuning of the hyperparameters. The neural network regression using scikit learn also underperformed to our expectations, which we believe is caused by wrong input variables when using the module. Still the result were good.

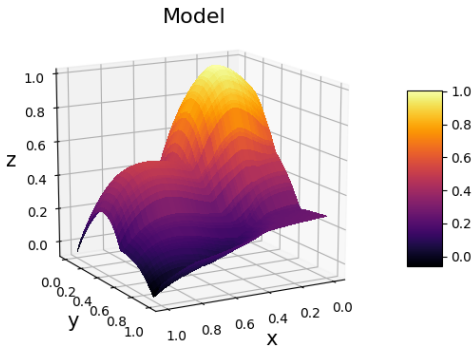
In figure 18, we have 3D figures of Franke's function using Ridge and both neural network methods. As well as the original function, as shown in the upper right corner. From the general shape of the three models, we can clearly see that Ridge regression fitted the data best. Again, we expected a model produced by scikit learn, lower right corner, to fit the data best, but this was not the case. But with a bit more time, and a bit more fine-tuning of parameters, this would probably produce the best result. The reason all methods performed very well, is likely caused by a large data set. And the 200x200 points data set easily gave us a smooth curve. The model were also plotted against the true Franke function.



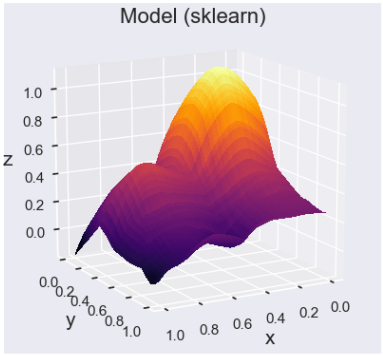
The model of FF, using Ridge. Project 1



The original Franke's function



The model of FF, using neural network



The model of FF, using Scikit Learn NN

Figure 18: Franke's function (FF) plotted with OLS, neural network and scikit learn's neural network.

6 Conclusion

We have coded our own logistic regression method, as well as neural network codes working for both classification and regression data sets. Comparing our results for logistic regression and neural network we have to conclude that logistic regression gave the best result for the credit card data, with not only an higher accuracy, but also an AUC score that only differ slightly on the third decimal. As stated in the introduction, we used the I-Cheng Yeh and Che-hui Lien scientific paper as reference point, but we were not able to compare our results with theirs. A difference however, which likely would give some difference results, is that we removed all bad data entries, while we could have re-categorised them into one category, and kept a larger data set to work with.

As we have seen in the credit card data, sometimes a data set may be biased, or we could be more interested in predicting one response variable more accurate than another. Therefor, we sometimes need to go beyond the accuracy metric to evaluate the performance of our model. It is something to keep in mind that in particular for such imbalanced problems, there is reasons to be sceptical if someone just refers to the accuracy as a way of substantiate their model's predictions [4]. The AUC score is a more accurate evaluation.

Overall, our results for the credit card data is better than the expectations we had. This is probably because we chose a fairly small amount of the total data set. We also expected neural networks to yield better predictions than logistic regression. Adding more layers to the network and doing a more thorough analysis of the different hyperparameters, could indeed have increased the performance of the neural network. For the last part of the project, we are happy with the results we gained for the linear regression. We expected neural network to give the best result, but all methods except Lasso regression gave good results.

Acknowledgement & Collaboration

This report is a collaboration between Anna Eliassen, Aron Jansson Nordberg and Kristina Othelia L. Olsen. We would like to thank Jon Andre Ottesen and Nils Johannes Mikkelsen for helpful programming input. We would also like to thank Morten Hjorth-Jensen for helpful input and an interesting project.

References

[1] Hugo Ferreira. *Confusion matrix and other metrics in machine learning*. https://medium.com/hugo-ferreiras-blog/confusion-matrix-and-other-metrics-in-machine-learning-894688cb1c0a. Published: 05-04-2018. Retrieved: 12.11.1019.

[2] Morten Hjorth-Jensen. *Neural networks, from simple perceptron to deep learning*. https://compphysics.github.io/MachineLearning/doc/pub/NeuralNet/pdf/NeuralNet-minted.pdf. Retrieved: 09-11-2019. 2019.

[3] Tomáš Jurczyk. *Gains vs ROC curves. Do you understand the difference?* https://community.tibco.com/wiki/gains-vs-roc-curves-do-you-understand-difference. Last updated: 10.09.2019. Retrieved: 12.11.2019.

[4] Will Koehrsen. *Beyond Accuracy: Precision and Recall*. https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c. Published: 03.03.2018. Retrieved: 10.11.2019.
[5] Alexey Grigorev ML Wiki. *Cumulative Gain Chart*. http://mlwiki.org/index.php/Cumulative_Gain_Chart. Published: 08.06.2014. Retrieved: 12.11.2019.
[6] Reiichiro S. Nakano. *Scikit-plot Documentation*. https://buildmedia.readthedocs.org/media/pdf/scikit-plot/stable/scikit-plot.pdf. Published: 19.08.2018. Retrieved: 12.11.2019.
[7] Sarang Narkhede. *Understanding AUC - ROC Curve*. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. Published: 26.06.2018. Retrieved: 10.11.2019.
[8] *Neural network: MLP regressor*.
[9] Kristina Othelia L. Olsen. *FYS-STK4155 - Project 1: Regression analysis and resampling methods*. https://github.com/kristinaothelia/FYS-STK4155/tree/master/Project1. Published: 9-10-2019. 2019.
[10] Pandas. *pandas.DataFrame.corr*. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html. Retrieved: 11.11.2019.
[11] UCI Machine Learning Repository. *Default of credit card clients Data Set*. https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients. Retrieved: 29-10-2019.
[12] Priyankur Sarkar. *What is Logistic Regression in Machine Learning*. https://www.knowledgehut.com/blog/data-science/logistic-regression-for-machine-learning. Published: 24.09.2019. Retrieved: 13.11.2019.
[13] Dr. Saed Sayad. *Logistic Regression*. https://www.saedsayad.com/logistic_regression.htm. Retrieved: 13.11.2019.
[14] Dr. Saed Sayad. *Model Evaluation - Classification: Confusion Matrix*. https://www.saedsayad.com/model_evaluation_c.htm. Retrieved: 11.11.2019.
[15] Magne Thoresen. *Logistisk regresjon – anvendt og anvendelig*. https://tidsskriftet.no/2017/10/medisin-og-tall/logistisk-regresjon-anvendt-og-anvendelig. Published: 16.10.2017. Retrieved: 10.11.2019.
[16] Wikipedia. *Artificial neural network*. https://en.wikipedia.org/wiki/Artificial_neural_network. Last edited: 08.11.2019. Retrieved: 09.11.2019.
[17] Wikipedia. *Gradient descent*. https://en.wikipedia.org/wiki/Gradient_descent#Description. Last updated: 10.11.2019. Retrieved: 11.11.2019.
[18] Wikipedia. *Stochastic gradient descent*. https://en.wikipedia.org/wiki/Stochastic_gradient_descent. Last updated: 03.09.2019. Retrieved: 09.11.2019.
[19] I-Cheng Yeh and Che-hui Lien. *The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients*. https://bradzzz.gitbooks.io/ga-seattle-dsi/content/dsi/dsi_05_classification_databases/2.1-lesson/assets/datasets/DefaultCreditCardClients_yeh_2009.pdf. 2009.

Appendix A - Credit card data: Dictionary

The data dictionary formulated by Vladimir G. Drugov, from https://rstudio-pubs-static.s3.amazonaws.com/281390_8a4ea1f1d23043479814ec4a38dbbfd9.html#data-dictionary.

There are 25 variables:

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit)
- SEX: Gender (1=male, 2=female)
- EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others)
- MARRIAGE: Marital status (1=married, 2=single, 3=others)
- AGE: Age in years
- PAY_0: Repayment status in September, 2005 (-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)
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)
- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment in June, 2005 (1=yes, 0=no)