Using logistic regression and neural networks for forecasting crude oil prices and detecting depression among the rural population in Siyaya county, Kenya (Project 2)

Ronald Mathew Baysa Payabyab
FYS-STK4155 - Applied Data Analysis and Machine Learning
University of Oslo

November 11, 2020

Abstract

The main purpose of this project is to create a neural network for analyzing real-life regression and classification problems. This will include studying the neural networks performance by tuning different hyperparameters, namely regularization, epochs and learning rate. More classical regression approaches such as OLS and Ridge regression are also briefly revisited to compare the results from the neural network regression. In the classification set, logistic regression will be benchmarked with the perfomance of the neural network. Data consisting of historical oil prices was gathered from the international monetary fund for the regression fit, and data with information about the living condition and depression diagnostic of a rural population in Kenya was gathered from Busara Institute for the classification case. The neural network obtained an optimal R2-score of 0.98, which is far ahead from OLS and Ridge with an R2-scores of 0.745 and 0.75 respectively. Likewise, the forecastability of the neural network performs better than OLS and Ridge with a MAPE-value of 2.34 % compared to 27.26 % and 34.0 %. As for the classification set, the neural network achieved an accuracy score of 0.84, which is 0.06 more than the accuracy score obtained by logistic regression. It is also discovered the error metrics are more dependent on a regularization parameter for the regression case compared to the classification case, and that the error metric improves for larger number of epochs for both cases.

1 Introduction

Neural networks and deep learning are becoming a big topic in computer science and in the industry sector whether its aim is to classify data points into discrete categories based on different features or to predict a continuous quantity for forecasting data. For instances, long-term forecasting of market prices of food products, medicine and energy commodities may support the policymakers in the decision-making process for future investments. The use of neural networks for forecasting has already been proved to be quite effective in predicting the development of wheat prices in China (Zou et al., 2007). Like many other machine learning techniques, the neural network algorithms are also not exclusively used to solve regression problems. Big companies like IBM and Google have already been harnessing neural network algorithms to guide personalized medicine or attempting to classify depressive states in bipolar patients based on social media activity (Durstewitz et al., 2019). This has raised hopes that neural networks can assist in tackling open issues in psychiatry such as reliable diagnostics decisions.

To show the versatility of neural networks, we will study two real-life data sets from different fields for applying regression and classification analysis. For the regression case, we will investigate a data set of historical oil prices provided by the International Monetary Fund (IMF) for the main energy commodities of the world. The model performance of the neural networks will be benchmarked against the linear regression methods OLS and Ridge with applied stochastic gradient descent. Additionally, we will also study the mean absolute percentage error (MAPE) to analyze the forecastability of the model. In the classification problem, the same processes will be applied, but replacing linear regression with logistic regression instead. Here, we will use a data set from Busara Center for behavioral Economics, Kenya which contains information about the living conditions of people in Siyaya county to detect depression. This will be an epidemiological measure of depression rather than a clinical one. The performance, properties, and its advantages and disadvantages of each method will be studied for both regression and classification problem.

2 Background

2.1 Neural Networks

In its simplistic form, a neural network is either a two-stage regression or classification model, that is typically represented as a network diagram. The network is built upon three types of layers, namely: an input layer, a hidden layer, and an output layer. Each node in the neural net performs a calculation, which is forwarded to other nodes deeper in the neural nets. These neural networks are composed of layers of nodes, where each node is designed to behave similarly to a neuron in a brain. The neurons in these models acts like nodes where data and computations flow. They receive one or more input signals, and these signals can either stem from a raw data set or from neurons positioned at a previous layer of the neural net. Each connection of between neurons are represented by weights. Once a neuron receives its inputs from the neurons of the preceding layer of the model, it sums up each signal multiplied by its corresponding weight and passes them further to an activation function. The general aim of the neural networks is to tune the weights of every connection to properly estimate an output or reaction with a given a set of inputs. In this project, we will look closer to the Feed Forward Neural Network (FFNN) algorithm.

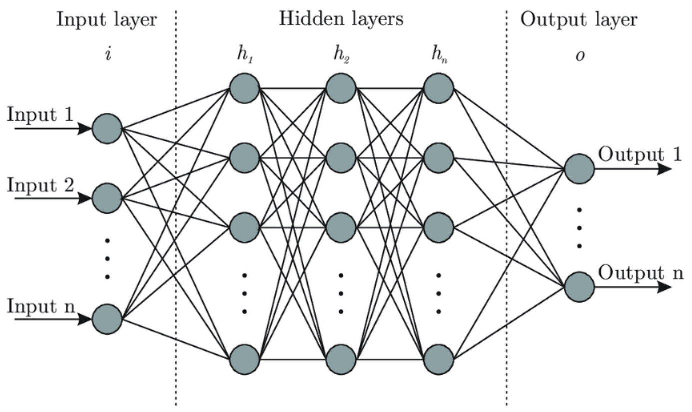


Figure 1: A simple visualization of a neural network. Each circle represents a node and belongs either in a input layer, hidden layer or output layer. The arrows show that all nodes of one layer is connected to each node in the next layers.

2.2 The structure of ANN

2.2.1 The activation functions

The biological neuron is simulated in an ANN by an activation function. In a classification task, an activation function has to have the switch-on characteristic, which mean that the output should change state. This simulates the "turning on" of a biological neuron. In a regression problem on the other hand, the final layer of the neural network will have one neuron and the value it returns is a continuous numerical value.

A common activation function for both classification and regression problem is the sigmoid function:

$$f(z) = \frac{1}{1+e^{-z}} \tag{1}$$

The sigmoid function moves from 0 to 1 when the input z is greater than a certain value. It is not a stepwise function, and the output does not change instantaneously. This activation function is often utilized due to its smooth behavior where the function goes from 0 to 1 in a well-behaved way. In a binary case, the output class depends on the value of the activation function used in the last layer. If the $f(z^l) \geq 0.5$, the output will be defined as class type 0, and class type 1 otherwise.

Another activation function is the rectified linear unit (ReLU):

$$f(z) = \begin{cases} 0, & \text{for } z < 0. \\ z, & \text{for } z \geq 0. \end{cases}$$

ReLU is linear for all positive values and 0 for all negative values. This makes it computationally cheap compared to sigmoid as the model takes less time to train or run. Additionally, it also converge faster since it does not have the vanishing gradients, and it is sparsely activated. Sparsity results in concise models often leads to better predictive powers and less overfitting/noise. Unlike sigmoid, ReLU has no upper limit of the value it can take.

Another variant is the leaky ReLU. Leaky ReLU has a small slope for negative values, instead of 0 altogether. Leaky ReLU may for instances have $y = 0.01z$ when $z < 0$. An advantage of leaky ReLU over ordinary ReLU is that we can worry less about the initialization of the neural network. In the case of ReLU, it is possible to end up with a neural network that never learns if the neurons are not activated in the beginning. The network may have many "dead" ReLU, e.g ReLU always gives values under 0, without even noticing.

In a classification problem, one might also consider softmax as an activation function for the output layer. Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly. Softmax is implemented through a neural network layer just before the output layer. The softmax layer must have the same number of nodes as the output layer. The equation of softmax is as follows:

$$p(y = j|\mathbf{x}) = \frac{e^{\mathbf{w}_j^T \mathbf{x} + b_j}}{\sum_{k \in K} e^{\mathbf{w}_j^T \mathbf{x} + b_j}} \tag{2}$$

The equation above expresses the full softmax, where the activation function calculates a probability for every possible class. This form of activation function is usually computationally cheap when the number of classes are small but becomes more expensive when the number of classes increases. For binary classification, using softmax should give the same result as sigmoid, since softmax is a generalization of sigmoid for a larger number of classes.

2.2.2 The basic notations

Before diving in to the algorithm, it is essential to explain some basic notation first. For the upcoming equations, each weight is identified with $w_{ij}^{(l)}$. The i represents the node number of the connection in layer $l + 1$, while j refers to the node number of the connection in the

l-th layer.For instances, the connection between node 1 in layer 1 and node 2 in layer 2 will gain the weight notation $w_{21}^{(1)}$. The notation of the bias weight is considered as $b_i^{(l)}$ where i is the node number just like the weight notation. Note that the bias generally has no input value since it is not a true node with an activation function. Both the values $w_{ji}^{(}1)$ and $b_i^{(l)}$ must be calculated during the training phase of the neural network. The ANN also consist of an output notation $h_j^{(l)}$, where j is the node number in layer l of the network.

2.2.3 Feed Forward propagation

In a feed forward process, a new variable $z_i^{(l)}$ is introduced, which represents the sum of inputs into node i of layer l, with the bias term included. For example, in the case of the first node in layer 2, z is equal to:

$$z_1^{(2)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)} = \sum_{j=1}^{n} w_{ij}^{(1)}x_i + b_i^{(1)} \tag{3}$$

It is possible to forward propagate the calculation through any given number of layers in the neural network. The generalized term is then:

$$z^{(l+1)} = W^{(l)}h^{(l)} + b^{(l)} \tag{4}$$

where

$$h^{(l+1)} = f(z^{(l+1)}) \tag{5}$$

The output of layer l becomes the input of layer $l+1$. $h^{(1)}$ is simply considered as input layer x, while the last layer is the output layer.

2.3 Cost function

It is always important iteratively minimize the error of the output the neural network by varying weights and gradient descent. This maes it possible to prevent overfitting of the data, and the optimization revolves around minimizing the cost function. In neural networks , the equivalent cost function of a single training pair (x^z, y^z) is expressed as:

$$\begin{aligned} J(w,b,x,y) &= \frac{1}{2}\left|\left|y^z - h^{n_1}(x^z)\right|\right|^2 \\ &= \frac{1}{2}\left|\left|y^z - y_{pred}(x^z)\right|\right|^2 \end{aligned} \tag{6}$$

The equation above shows the cost function of the z-th training sample where h^{n_1} represents the final layer of the neural network, the output layer in other words. The error is represented in the L^2 norm, which is a common way of representing the error of machine learning algorithms. Instead of taking the absolute value between the predicted value and the actual value, the square of error is calculated.

The downside of using a quadratic function in equation (6) is that there's a risk for getting a learning slowdown. This means that when the neuron's output is getting closer to 1, the curve flattens, and the derivative of the cost function gets very small, which in turn leads to $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ being very small as well (Nielsen, 2015). To address the issue of learning slowdown, the cost function can be replaced with a different cost function called the cross-entropy function:

$$J(w, b, x, y) = -\sum [y \ln(h^{n_1}(x^z)) + (1 - y) \ln(1 - h^{n_1}(x^z))] \tag{7}$$

The cross-entropy is positive and tends toward zero as the neuron gets better at computing the desired output, y, for all training inputs. If we replace $h^{n_1}(x^z)$ with $\sigma(z)$, the partial derivative of the cost function can be described as:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum \frac{\sigma'(z) x_j}{\sigma(z)(1 - \sigma(z))} (\sigma(z) - y) \tag{8}$$

The terms $\sigma'(z)$ and $\sigma(z)(1 - \sigma(z))$ cancel in the equation and gets simplified to become:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum x_j (\sigma(z) - y). \tag{9}$$

where x_j are the input variables and n is the total number of items in the training data. The equation above indicates that the learning rate of the weights are controlled by the error in output. The larger the error, the faster the neuron will learn. In particular, it avoids learning slowdown caused by the term $\sigma'(z)$ as it is cancelled out using cross-entropy. $\sigma(z)$ in this equation is the probability of an input data point being class type 1, while the term $1 - \sigma(z)$ represents the probability of class 0. In a multiclass case, the softmax function discussed in 2.2.1 are used. One-hot encoding will be utilized, and in a binary case, the classes may for instances be represented as $(1, 0) = 0$ and $(0, 1) = 1$

When applying regression analysis to evaluate forecasting, the mean absolute percentage error (MAPE) can be useful to measure. This is given by:

$$MAPE = \frac{1}{N} \sum_{t=1}^{N} |(\frac{y_t - \hat{y_t}}{y_t})| \tag{10}$$

where N is the number of data in the testing set, and y_t and $\hat{y_t}$ are real value and estimated value at a given time t respectively. It is fairly easy to calculate, and does not depend on scale. However, MAPE is also asymmetric and reports higher errors if the forecast is more than the actual and lower errors when the forecast is less than the actual. It should therefore be used along with other metrics as well.

2.3.1 Backpropagation

In equation (3) as discussed in section 2.2.3, we have defined the simple foundational equation of the neural network by using three layers as an example. One of the purposes of the backpropagation process is to find out how much a change in the weight $w_{12}^{(2)}$ has on the cost function J in order to evaluate the chain function:

$$\frac{\partial J}{\partial w_{12}^{(2)}} = \frac{\partial J}{\partial h_1^{(3)}} \frac{\partial h_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial w_{12}^{(2)}} \tag{11}$$

For the weights connecting to the output layer, the cost function can be directly calculated by comparing the output layer to the training data set. The output of the hidden nodes does on the other hand have no such direct reference. They are connected to the cost function only through mediating weights and other layers of nodes. This is where backpropagation will do its work. The term that needs to propagate back through the network is denoted as $\delta_i^{(n_1)}$. This term represents the neural network's ultimate connection to the cost function, and it contributes through the weight $w_{ij}^{(2)}$. The output layer δ is communicated through the hidden node by the weight of the connection. The hidden layer δ can in a vectorized form be described as:

$$\delta_j^l = \sum_{i=1} (\delta_1^{(l+1)} w_{ij}^{(l)}) f'(z_j)^l \tag{12}$$

It is then possible to calculate the gradient descent for the weights and biases:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b, x, y) = \delta_i^{(l)} h_j^{(l-1)} \tag{13}$$

$$\frac{\partial}{\partial b_j^{(l)}} J(W, b, x, y) = \delta_i^{(l)} \tag{14}$$

for all layers l. The derivatives in the activation function enters in equation (13), behaving like a weight to error used in the gradients. This might cause a vanishing gradient when using the sigmoid function. If the weighted inputs to the of the sigmoid function is substantially large such that $|z| >> 0$, then its derivatives will be close to zero. This will, as a consequence, wind up with very small gradients for the weights and biases, and it will take a lot of iterations to optimize the weights and biases for the stochastic gradient descent. This will become less of an issue when using ReLU as an activation function. Since the output of its derivative is either 1 or 0, the vanishing gradient will not cause too much of an issue since not all of the weighted inputs become less than zero.

2.4 Stochastic Gradient Descent

The stochastic gradient descent forms the basis of the neural networks. It is an iterative algorithm that starts from a random point on a function and travels down its slope stepwise until it hits the global minimum of that function. In the stochastic gradient descent (SGD), an instance in the training set is randomly picked for each step, and the gradients are computed based solely on that particular instance (Géron, 2017). This will make the algorithm much faster since there is very little data to manipulate for every iteration, and due to its randomness, it can also escape local minima in the function. Despite its functionality of escaping such minima, the randomness of the SGD makes the algorithm never settle at the global minimum. It it is therefore essential to reduce the learning rate η by starting with large steps at the beginning, and then get smaller and smaller after a while in order to settle for a minimum.

In a linear regression case such as OLS and Ridge regression, the purpose is to minimize the cost function:

$$\min_{w,b} \sum_{i=1}^{n} (y_i - y_{pred})^2 \tag{15}$$

The prediction values y_{pred} are based on the weights w_i and test data points X_i. We multiply them together and ad a bias b to calculate the predicted value. The predicted value will then look like:

$$y_{pred} = w^T x_i + b \tag{16}$$

The loss function is then minimized by finding the optimal weights and biases:

$$\mathcal{L}(w, b) \rightarrow \min_{w,b} \sum_{i=1}^{n} (y_i - (w^T x_i + b))^2 + \lambda \sum w^2 \tag{17}$$

Here, the equation is performing a summation of over a complete n training data points. For SGD, the summation will be performed over a random sample of batch size k where $k < n$. By introducing stochasticity to the algorithm, the data are approximated by these batches instead in order to reduce the probability of the gradients to stop in a local minimum of the loss function. The gradient descent is then performed on each batch instead of the entire data set , iterating over a number of these batches. A full iteration over these batches are denoted as an epoch.

To find the optimal w, w_{j+1} will be updated constantly per error difference in the previous weight w_j. In order to do this, w_j is subtracted with the derivative of the loss function with respect to w:

$$\frac{\partial \mathcal{L}}{\partial w} = \sum_{i=1}^{n} (-2x_i)(y_i - (w^T x_i + b) + \lambda w) \tag{18}$$

Note that for Ridge regression, it is also needed to subtract the equation with $2\lambda w$. The updated weight value will then be expressed as:

$$w_{j+1} = w_j - \eta(\frac{\partial \mathcal{L}}{\partial w})_{w_j} \tag{19}$$

where η is the learning rate which is multiplied to the derivatives. Similarly, the bias b_{j+1} will be updated as well. The bias b_j will be subtracted with the derivative of the loss function with respect to b, giving:

$$b_{j+1} = b_j - \eta(\frac{\partial \mathcal{L}}{\partial b})_{b_j} \tag{20}$$

2.5 Logistic regression

The aim of logistic regression is to model the posterior probabilities of K number of classes through a linear function in x, while at the same time ensures that the sum to one and remain in the interval $[0, 1]$. Logistic regression aims to predict discrete point data rather than continuous ones. Thus, logistic regression classifies data in different groups or classes based on a set of inputs or predictors just like in neural network classification. The theory behind logistic regression is therefore quite similar to neural networks excluding hidden layers and backwards propagation.

The sigmoid function will however behave differently for a logistic regression problem. it will act like a probability of a given output $p(t)$ instead of representing if a neuron activates or not $f(z)$. A set of predictors $x_i = [1, x_1, \dots, x_p]$ corresponding to the binary output given $y_i = [0, 1]$, and its respective weights $(W) = [w_0, \dots, w_p]$ will determine the value of the sigmoid function, which will be expressed as:

$$p(y_i = 1|x_i, \mathbf{W}) = \frac{1}{1 + e^{x_i^T \mathbf{W}}} \tag{21}$$

$$p(y_i = 0|x_i, \mathbf{W}) = 1 - p(y_i = 1|x_i, \mathbf{W}) \tag{22}$$

It is also common in logistic regression to add regularization to prevent overfitting by adding an additional regularization term in the cost function.

In L2-regularization the cost function can be described as:

$$J(w) = \sum_{i=1}^{m} Cost(h(x^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2 \tag{23}$$

λ is the denotion of the regularization parameter, which controls the trade-off between fitting the training data properly and keeping the parameters small to avoid overfitting. The cost function will in this case be the cross-entropy function defined in equation (7) using sigmoid as an activation function. Furthermore, the gradient will then become:

$$\frac{\partial}{\partial w_i} J(w) = \frac{1}{m} [\sum_{j=1}^{m} (h(x^{(j)}) - y^{(j)}) x_i^{(j)} + \lambda w_i] \tag{24}$$

The regularization term will heavily penalize larger weight values such that the growth of w is controlled.

2.6 Error metrics

It is necessary to apply error metrics to judge the credibility of a model. For regression problems, the R2-score is commonly used:

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{25}$$

where y represents the true value of the data set while $\hat{y}$ is the predicted value on y. $\bar{y}$ on the other hand represents the mean value of the true data sets. In percentage, the R2 score gives a value between 0 and 1, with 1 being a perfect fit while 0 representing a completely faulty model.

In a classification problem on the other hand, two types of error metrics can be applied. The first one being the accuracy score:

$$Accuracy = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n} \tag{26}$$

The accuracy is measured by the number of correctly guessed targets t_i divided by the total number of targets n. I is the indicator function which will give the value 1.0 if $t_i = y_i$ and 0 otherwise in the binary case. This metrics is simple and easy to understand, but can be very biased if the data set are very biased.

With biased data sets, the Area Under Curve (AUC) might be preferred in a binary classification problem. AUC provides an aggregate measure of performance across all possible classification threshold, and represents the probability that a random positive example is positioned to the right of a random negative example. It ranges in value from 0 to 1, where a model with 100 % wrong prediction gets a score of 0.0, while a perfect model gets a score of 1.0. AUC will be tested to observe how different the results could be when applying different error metrics

3 Methodology

3.1 Data sets

In this project, a feed forward neural network algorithm will be utilized for both classification and regression problem. For the regression problem, we will study the historical prices of oil, coal and natural gas provided by International Monetary Fund (IMF). The original data set contains 13 columns containing the monthly oil, gas and coal price in USD per barrel/metric for 6 different international markets and their respective log-return in a time series from January 1980 to June 2017(Herrera et al., 2019a). For this current regression problem, we are only interested in predicting the oil price of West Texas Intermediate 40 API with the following set of columns:

Table 1: The data used in the susceptibility assessment, the data sources, and the associated factor classes for the landslide susceptibility mapping in the study area.

Col no.	Name	Description
1	Oil Brent	Crude oil. Dated Brent, light blend 38 API, US $/barrel
2	log-return Oil Brent	log-return of Oil Brent's time series
3	Oil WTI	Crude Oil. West Texas Intermediate 40 API, Midland Texas, US$/barrel
5	Coal AU	Australian thermal coal. 12000-BTU/ pund, FOB Newcastle/Port Kembla, US$/metric ton
6	log-return Coal AU	log-return of Coal AU's time series
9	Gas Russia	Russian Natural Gas border price in Germany, US$/Million Metric BTU
11	Gas US	Natural gas spot price at the Henry Hub terminal in Louisiana, US$/Million Metric BTU
12	log-return Gas US	log-return of Gas US' time series

Column number 3 will be the target value while the rest will function as input data for the Neural Network. Due to lack of data for the US gas prices from the time series from 1970 to 1990, we will only consider the monthly prices and their respective log-return value from January 1991 and onwards. The data set has been studied before using multiple machine learning methods random forests and ANN. Their respective model accuracy has also been discussed in Herrera et al. (2019b).

For the classification problem, a data set from Kaggle used for recognizing depression diagnostics. This can be found in: https://zindi.africa/competitions/busara-mental-health-prediction-challenge/data. We will use a rather simplified version by only considering 8 features. These data come from a 2015 study conducted by the Busara Center in rural Siaya County, near Lake Victoria in western Kenya. The original data set contains more than 70 features while the sample set only contain 23 features with 1143 surveys of people. For simplicity, it was decided to only include the following 8 features:

1. Sex
2. Age
3. Married
4. Number of children
5. Education level
6. Meals skipped
7. Incoming salary
8. Durable investments

The aim of the classification problem is to detect people with depression diagnostics based upon their living conditions. The target value consist of a binary column where 0 indicates no depression while 1 confirms depression. Other binary values are within the other features as well. In the "sex" category, females are labeled as 1, while in the "married" category a person who is married is labeled as 1. Incoming salary is also binary, where 1 means that person interviewed received any form of salary, meaning that 0 indicates that the person is unemployed.

3.2 Polynomial regression of OLS and Ridge vs. SGD

To compare the results of the regression model in neural networks, we will implement regression analysis in OLS and Ridge to give estimates of what values to expect from the neural networks. OLS is considered to be a simple regression model. Ridge on the other hand is quite similar, but utilizes a hyperparameter λ to avoid overfitting. Some random noise were also added to the target value to avoid overfitting of the dataset. The following python-files involved for printing the optimal R2-score and generating a heat map of the R2-scores relation to model complexity and hyperparameter λ can be found on the three following object-oriented programs: **Regress.py**; for finding β-values for given regression method, **Resampling.py**; for resampling the data set using KFold cross-validation, and **results.py**; for calculating the error metrics.

20 % of the data set were used as test data, and both OLS and Ridge were ran for increasing degrees of freedom from 1 to 10. In Ridge, the hyperparameter where studied within the range $[10^{-10}, 10^{2}]$.

3.3 Logistic Regression

The logistic regression algorithm was built upon a class function stored in **Logreg.py** and the class execution will happen at **logreg_execution.py**. This was utilized to compare with the performance of neural networks classification problem. The class function fits the design matrix $\mathbf{X}$ and the target $\mathbf{y}$ in order to make a prediction depending on the number of epochs and the learning rate. The number of epochs is equivalent to the number of iterations used in the stochastic gradient descent. Just like the Neural networks algorithm, mini-batches were also applied along with L2-regularization as discussed in section 2.5. A default value for the inverse of regularization strength was set to $\frac{1}{\lambda} = 0.1$. To study the behavior of the logistic regression, the code was run for different set of epochs and learning rates, with the learning rates 10^n where n is defined between -5 and 0 while the epochs contained the values: 100, 200, 400, 800, 1600, 3200, 6400, and 12800.

3.4 Neural Network architecture

The aim of the neural networks is to use it for both regression and classification problems. The entire structure is defined as an object-oriented function with 4 class functions in total, two for each problem. The general neural network architectures contains the feed forward and back propagation algorithms, including SGD, training the network, and initializing the weights and biases. A separate neural network for regression and classification can be found in **neural_net_regression.py** and **Classify_run.py** respectively. Additionally, the cost functions and activation functions used for each problem are stored in two separate class functions, namely **Regression.py** and **Classify.py**. Sample runs of the regression problem can be found in **Regression_run.py** while the runs from the classification lies in the same place as its neural network structure. Whether initializing a regression or classification problem through their respective class functions, the type of activation functions used in hidden layers and the output layers are fixed. This also applies to the cost function as well. The feedforward and backpropagation algorithms in equations (3) and (12) are also converted in a matrix format, giving:

$$z^l = (w^l)^T h^{l-1} + b^l \tag{27}$$

and

$$\delta^l = \delta^{l+1} (w^{l+1})^T \sigma'(z^l) \tag{28}$$

3.4.1 Inputs and outputs

The number of neurons depends on the input data set, while the output layer on the other hand depends on the type of problem. The data set for depression detection was considered to be a binary classification problem. However, if another data set with multiclasses was applied in the code, the MNIST data set for instances, it is necessary for the output layer to classify multiple samples to digits between 0 and 9. A sensible neural network architecture would thus have an output layer consisting of 10 nodes, which each of these nodes representing a digit from 0 to 9. To make the neural network as flexible as possible, a function was created to convert a single vector that lines up with the n-node output layers. For example if the MNIST-data set was applied and the target data set was the digit 3, the number will be converted into the vector $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$.

The regression case with the oil price data set, did only contain a single output neuron, giving an output vector for each input point. The choice of activation function used in the output layer along with the cost function did also depend whether the data set is considered a regression or classification problem. For classification problems, the activation function was set to softmax with the cost function defined by cross-entropy. Additionally, the activation function used in the output layer depends on the type of problem. If the problem is a classification case, softmax and and cross-entropy will be used with an output error of:

$$\delta^L = h^l - y \tag{29}$$

For the regression case, the mean squared error will be used, giving an output error of:

$$\delta^l = \frac{1}{n_{ouput}}(h^l - y)\sigma'(z^L) \tag{30}$$

3.4.2 Training the neural network

First and foremost, all of the input data sets are scaled in order to make it fit between either 0 to 1 or centered around 0. This is to help with the convergence of the neural networks, which is in particular important when combining different data types. The scaling was applied by using the standard scaler of scikit-learn. Splitting the data sets into training and testing cases where also applied in this module with 20% of the data set were used as test-cases.

The code produced in the neural network is built upon defining the structure of the neural network. In the classification problem, it is known that the data set contains 8 features and 8 nodes will be needed to cover these 8 features in the data set. Since the data set is binary, two output layer nodes were applied. In the regression problem, 7 nodes are needed, but only a single output layer is necessary. As for the hidden layer, a reasonable number of nodes should be somewhere in-between the numbers given in the input and output layer. In Python the structures of the hidden layers of the neural network for regression and classification will be defined in a list such as $hidden\_neuron\_list = [5, 5, 5]$ and $nn\_structure = [2]$. The first case gives 3 hidden layers with 5 neurons each, while the second examples defines one hidden layer with 2 neurons. It is also possible to define a flexible number of neurons for each hidden layer.

After defining and implementing different activation functions, the weights were randomly initialized for each $W^{(l)}$-layer. This was performed by using a loop over the number of iterations/epochs where the weight-matrix ΔW and its respective bias Δb were initialized to zero.

A feed forward propagation was performed through all the n_l number of layers where the activation function output $h^{(l)}$ are stored. Then, the $\delta^{(n_1)}$ value for the output layer are calculate while the backpropagation algorithm are calculated for the layers. Finally, the stochastic gradient descent step are implemented in the code by the principles explained in the equations 18 and 19. Mini-batches were also added before setting the data into training. The chosen cost function is averaged over a small number of samples. In this particular case, 100 mini-batches, bs, were created in default. The cost function will then have the structure of:

$$J(W,b,x^{(z:z+bs)},y^{(z:z+bs)}) = \frac{1}{bs}\sum_{z=0}^{bs} J(W,b,x^{(z)},y^{(z)}) \tag{31}$$

The stochastic gradient descent routine are repeated until the average of the chosen cost function has reached a minimum. At that stage, the network is trained and should ideally be ready for use.

3.5 Testing of hyperparameters

Just like any other statistical models, it is preferable to tune different hyperparameters to obtain the optimal values of the hyperparameters. This will involved tuning every free parameters and testing out different values such as the learning rate, epochs and the regularization parameter. Every combination of hyperparameters should be tested as they often are strongly related to each other.

Unlike logistic regression, neural networks has a large number of hyperparameters. In its simplest form, the neural networks can have up to 5 to 10 hyperparameters. This could for instances be the type of activation function used, the cost function and the number of neurons in a hidden layer. This is a recurring problem for both the regression and classification problem as there will be some issues with runtime for optimizing every hyperparameter. If 10 hyperparameters were tested for 3 values each, then we would obtain approximately 59 000 combinations to test. (Each configuration must be averaged).

Another issue comes to visualizing what combinations of hyperparameters yields the optimal results since they often are strongly related to each other. For instances, tuning the learning rate often requires to tune the number of epochs.

It was therefore decided to limit the testing to approximately 4 hyperparameters, which are: the epochs, the learning rate, the regularization value λ, and the number of neurons per layer. The type of activation function will be tested as well. A full list of values to test for the chosen hyperparameters can be seen below for both regression and classification problem. Every combination was run 30 times with a randomly selected test data which represents 20 % of the data set. The test score will be calculated as an average of the 30 runs. Since the size of the data set is quite small, it would be suitable to not include too many epochs and neurons per layer. The number of hidden layers in the analysis will contain 3 hidden layers by default.

regularization value :	0.0, 0.1, 0.01, 0.001
Neurons per layer:	2, 5, 6
Epochs:	100, 500, 1000
Learning rate:	0.001 0.01, 0.1

Table 2: Table showing set of values tested for hyperparameter optimalization in neural networks

4 Results and discussions

4.1 Data

4.1.1 Oil prices

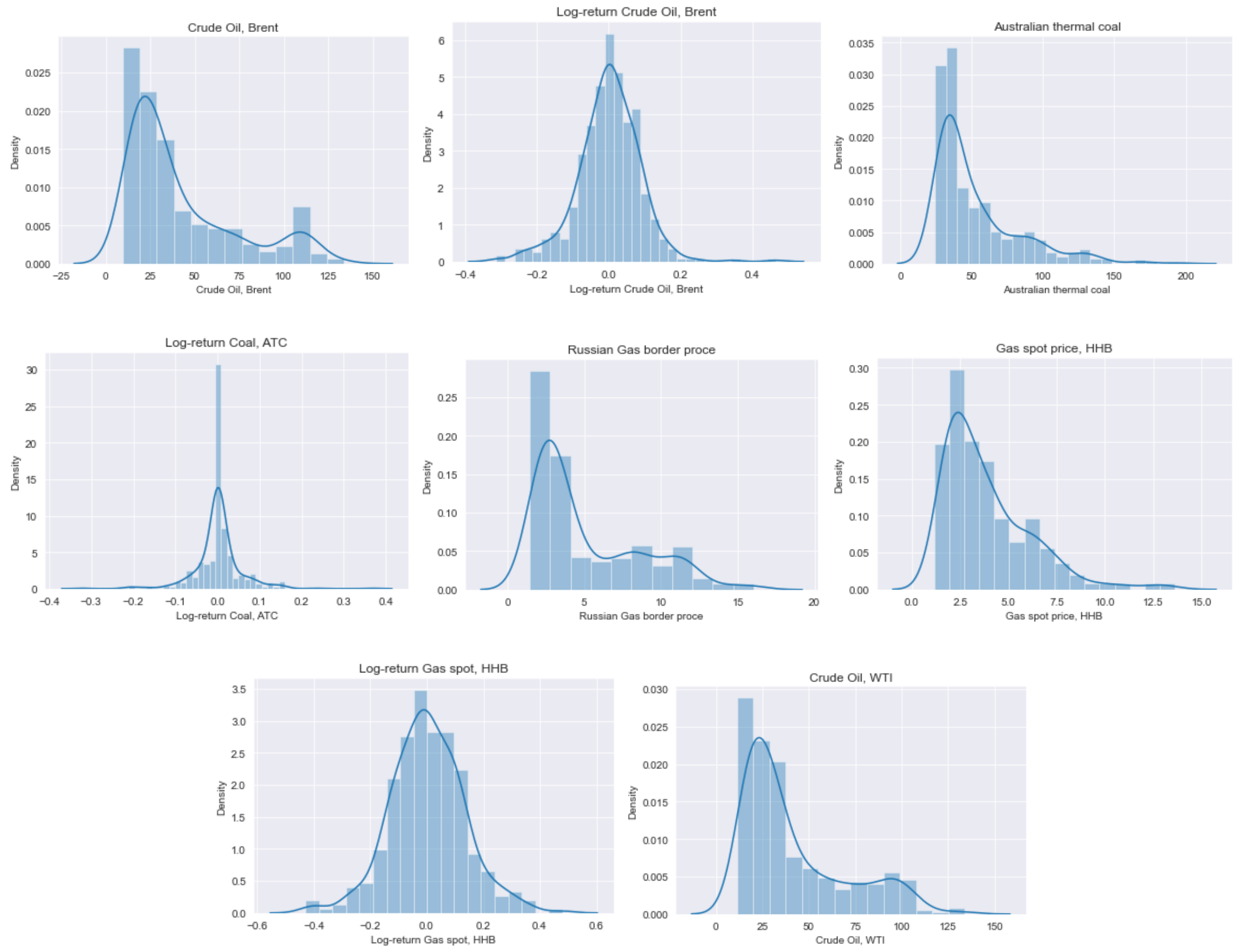
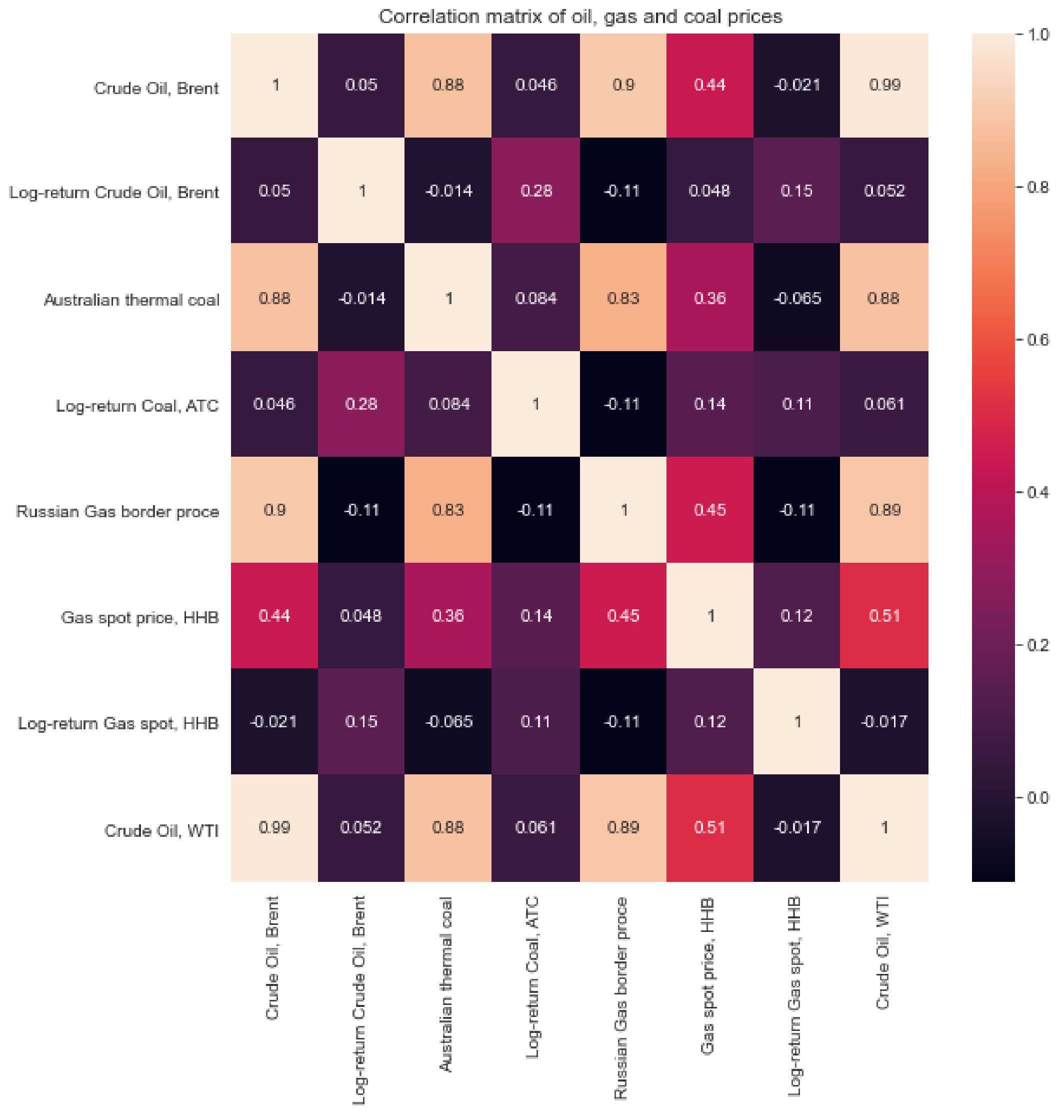


Figure 2: The distribution of the feature data set

The data set from the International Monetary Fund remained mostly unchanged with no data cleaning necessary. However, information about Russian gas prices was not available until January 1985 while American gas prices were not available until January 1991. It was therefore decided to perform a forecast analysis of the data set from January 1991 to 2017 rather than the entire data set. The analysis consists of 318 data points containing the monthly prices and its log-return changes of petroleum.

The figure below shows the correlation matrix of the data set. Most of the variables are positively correlated with the target value "Oil, WTI". The Crude Oil from West Texas intermediate appears to be especially strongly correlated to Oil Brent, with almost a perfect correlation of 0.99. The target value appears to be also very strongly correlated to Australian coal and Russian gas prices.



	Crude Oil, Brent	Log-return Crude Oil, Brent	Australian thermal coal	Log-return Coal, ATC	Russian Gas border proce	Gas spot price, HHB	Log-return Gas spot, HHB	Crude Oil, WTI
Crude Oil, Brent	1	0.05	0.88	0.046	0.9	0.44	-0.021	0.99
Log-return Crude Oil, Brent	0.05	1	-0.014	0.28	-0.11	0.048	0.15	0.052
Australian thermal coal	0.88	-0.014	1	0.084	0.83	0.36	-0.065	0.88
Log-return Coal, ATC	0.046	0.28	0.084	1	-0.11	0.14	0.11	0.061
Russian Gas border proce	0.9	-0.11	0.83	-0.11	1	0.45	-0.11	0.89
Gas spot price, HHB	0.44	0.048	0.36	0.14	0.45	1	0.12	0.51
Log-return Gas spot, HHB	-0.021	0.15	-0.065	0.11	-0.11	0.12	1	-0.017
Crude Oil, WTI	0.99	0.052	0.88	0.061	0.89	0.51	-0.017	1

Figure 3: Correlation matrix of oil prices

4.1.2 Depression detection

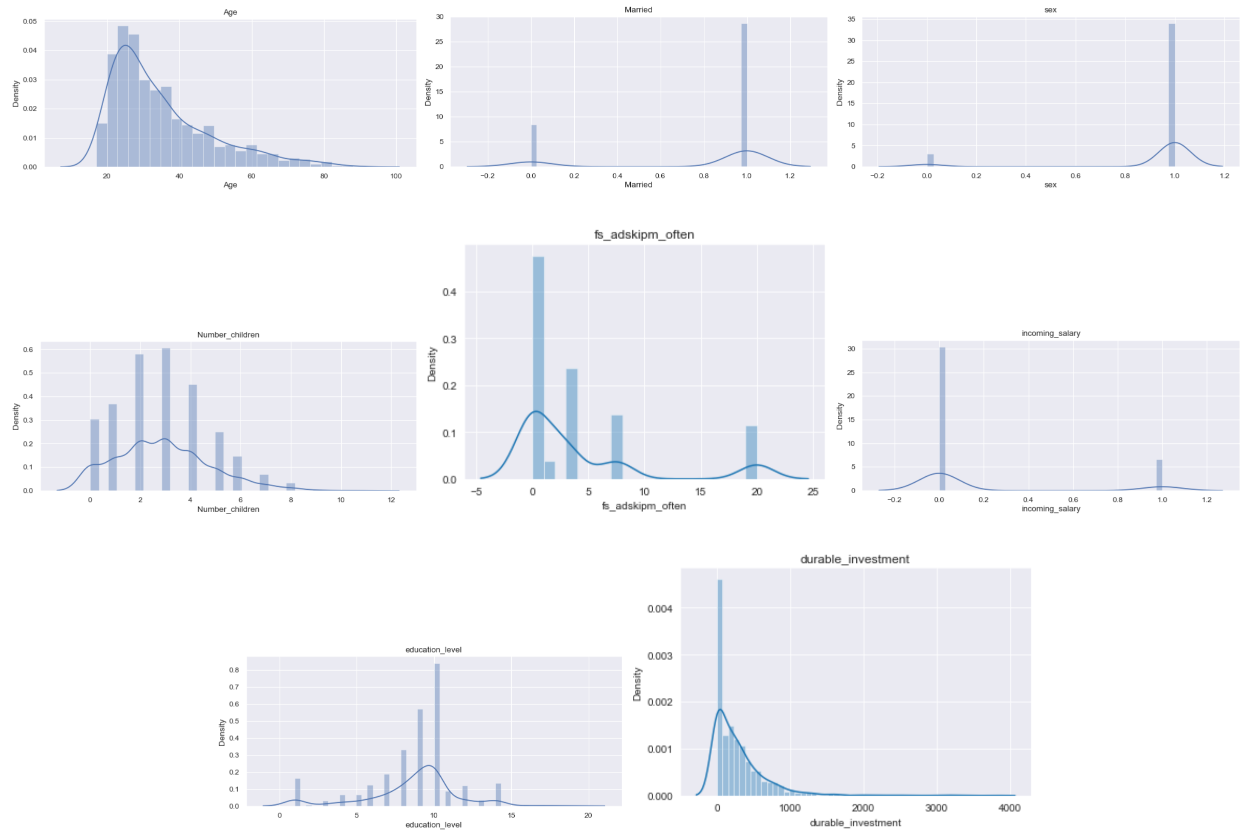


Figure 4: The distribution of age, marital status, sex, number of children, number of meals skipped monthly, income salary, years of education and durable investment investment.

In this data set, the vast majority of respondents were females with a median age of 25 and at least 10 years of education behind. Most of the respondents are married and with 2-3 children in the household on average. Interestingly, the majority also do not have any stable income salary. Based on the context of this data set, most respondents are housewives where the man often are breadwinner of the household. Most of the respondents appears to have no durable investments, but few also have about 1000 USD in spare. Most respondents do not skip a single meal in a month, but some do skip a meal at least 20 times in a month.

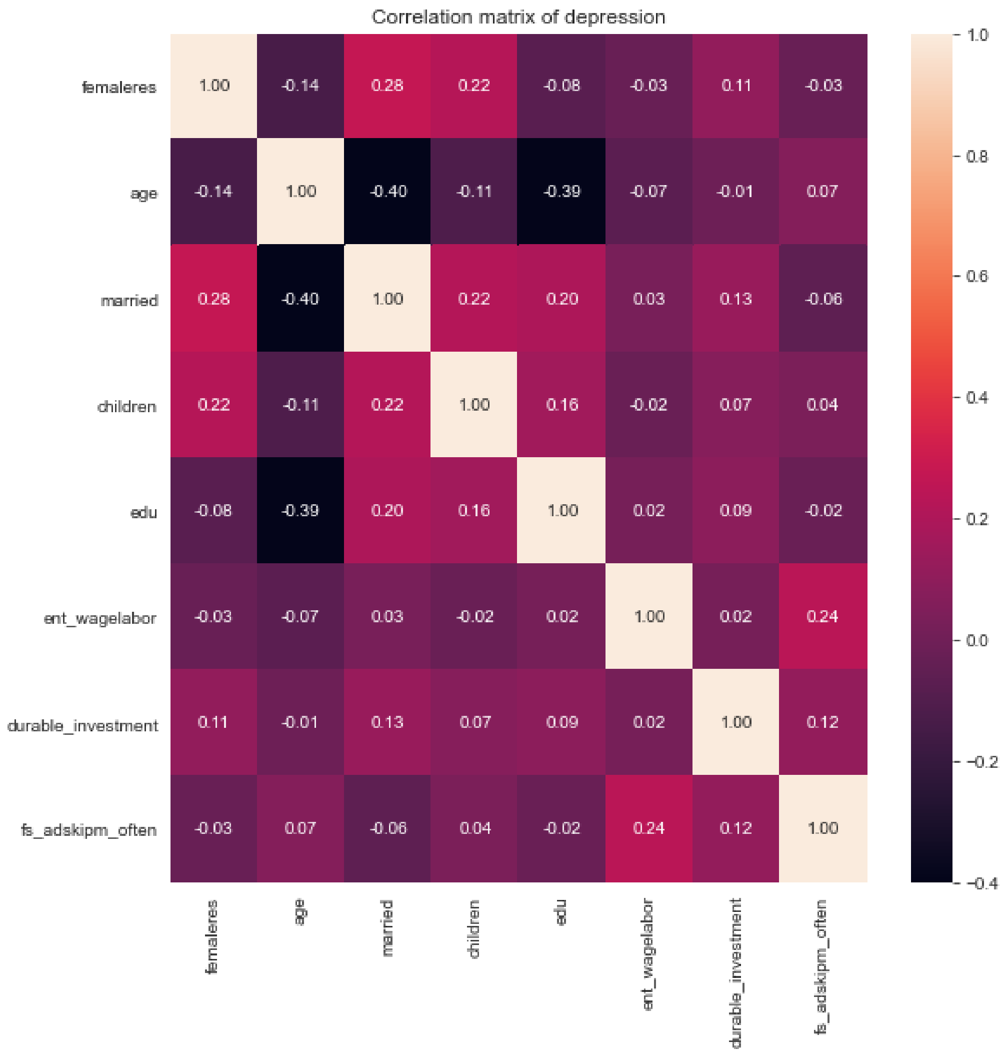


Figure 5: Correlation matrix of respondents living conditions

The features in the data set are less correlated to each other than the regression data set. There appears to a slight negative correlation between marital status and age. Furthermore, there is also a slight positive correlation between durable investment, working status and meals skipped per month.

4.2 Regression problem

4.2.1 OLS, Ridge and SGD

Table 3: List of R2-scores in OLS by degrees of freedom

Degree	R2-score
1	0.743
2	0.735
3	0.692
4	0.737
5	0.745

The table above shows the R2 scores when performing OLS regression on the oil data set with different polynomial degrees, while the table below illustrates a heat map R2-score based upon polynomial order and the λ parameters. It is observed that the R2-scores of OLS are already fairly high even on the first order polynomial, and seems to reach a minimum R2-score at the 3rd-order polynomial, and then reach its maximum score at the 5th degree with 0.745. Due to the data set's limited size, it is to be assumed that OLS might overfit the data if applied on higher polynomial degrees.

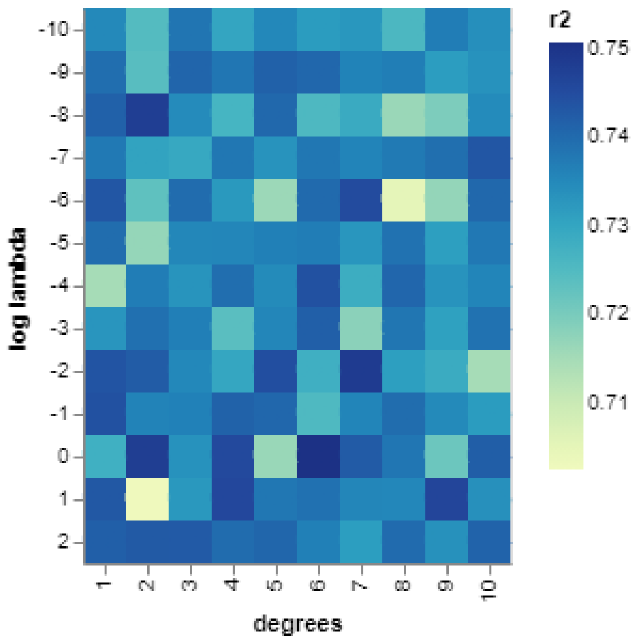
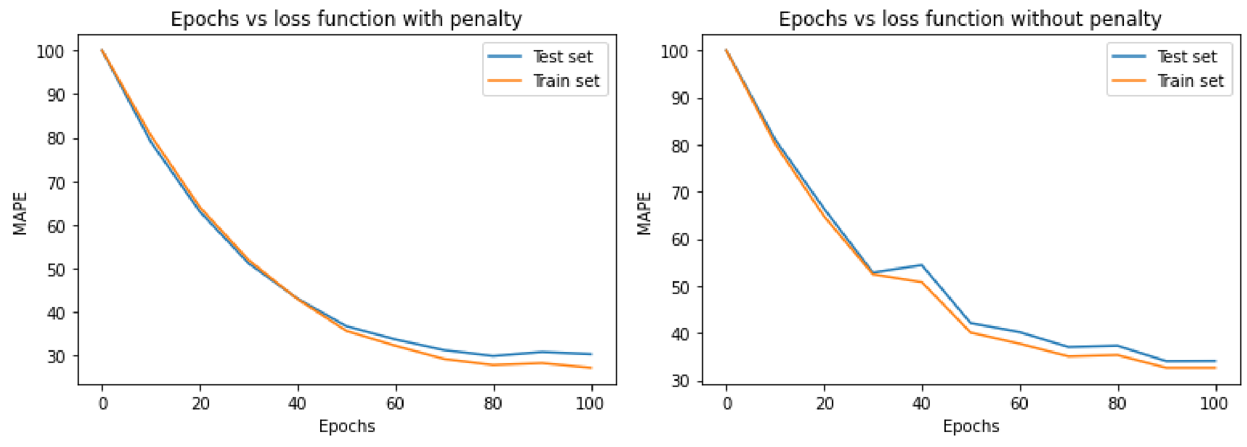


Figure 6: Heat map of Ridge regression showing R2-score based on different hyperparameters and degrees of freedom

Ridge on the other hand seems to barely outperform OLS, reaching a maximum R2-score of 0.75 at the 7th degree polynomial with $\lambda = 10^{-2}$. Since Ridge at a certain extent outporforms OLS, it could be expected that there is some relation between the parameters. On the other hand, the data set are quite small in size and could thus result in overfitting. On the other hand, figure 3 still shows reasonable values up to the 10th degree polynomial which evidently shows that Ridge are more resistant to overfitting.

The analysis was run again replacing OLS and Ridge by applying stochastic gradient descent with and without a regularization of the weights as discussed in section 2.4. In this analysis, the learning rate was set to 0.01 with 10 batches and a regularization $\lambda = 0.1$ when applying Ridge. The R2-score when applying the regularization seems to decrease a bit as the R2 score attained a value of 0.729 and 0.668 for the test data set and train data set respectively for the plot on figure 4. When running the program 50 times and calculating the average R2-score based on these 50 runs, the R2-score gains an average value of 0.739.



(a) With regularization

(b) No regularization

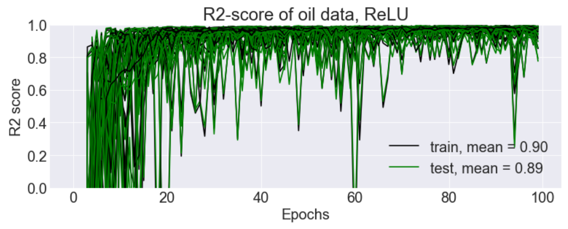
Figure 7: Mean absolute percentage error by increasing number of epochs.

The figure above shows an example run of SGD plotting the loss function against the number of epochs with a given regularization (4a) and without (4b). The loss function used is the mean absolute percentage error as explained in section 2.3. Both the test set and training set gradually decrease for the increasing number of epochs reaching a minimum MAPE of 29.97 % and 27.26 % for testing and training set respectively. When removing the regularization value, i.e setting $\lambda = 0$, the R2-score appears to improve slightly with maximum R2-scores of 0.752 and 0.738 for test and training sets respectively. However, this seems to come at the expense of the MAPE-value as the minimum values for MAPE has now reached 34.0 % and 32.63 % for test data set and training data set.

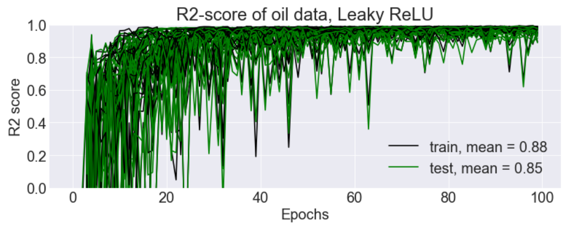
Similar MAPE-values are shown by Herrera et al. (2019b) where the MAPE-values for predicting the prices of Oil WTI were estimated to be 35.96 % when they applied a random walk model without drift. According to Lewis (1982), a reasonable forecasting can already be achieved when running SGD with 40 epochs as MAPE-score most likely will stay below 50 %. On the other hand the MAPE values can be improved by tuning the hyperparameters involved to obtain better forecastability. This can for instance be observed when changing the regularization value. When the regularization $\lambda = 0$. The regularization has no effect on the error and will be equivalent to OLS. The model minimizes the error as much as possible and eventually overfit data for higher epochs. However, if the regularization value gets to large, the error from the regularization may become as less as possible as its weightage is very high. In such scenario, the model performance will drastically worsen.

It would therefore be quite naive to set an arbitrary percentage in MAPE and interpret them as a reasonable measure of forecast, which will be noticed when applying neural networks.

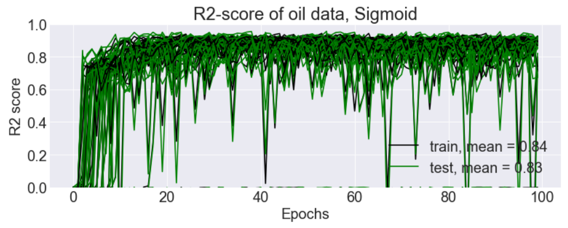
4.2.2 Neural networks of oil data



(a) ReLU



(b) Leaky ReLU



(c) Sigmoid

Figure 8: The results of calculating the R2-score of oil data forecasting up to 100 epochs using three different activation functions for the hidden layers. The trend of the test sets are unstable, and much higher number of epochs are needed.

The figures above shows the visualization of 30 random runs of the implemented neural network algorithm using the activation functions: ReLU, leaky ReLU and sigmoid respectively. The main purpose of the illustrations is to confirm that the activation functions work properly and to show how differently the R2-score behaves for each run. The input values for the three figures are mostly the same with 5 neurons in each hidden layer, λ-value of 0.1 with 100 epochs. There were however some issues with the learning rate, as both ReLU and leaky ReLU needs a much smaller learning rate in order for it to run properly. The neural network using sigmoid had no problems with a learning rate of 0.1 while ReLU and leay ReLU both were assigned a learning rate of 0.001 instead. A possible explanation for this is that the ReLU units "die" during training. A large gradient flowing towards the ReLU neuron could cause the weights to update that the neuron will never activate on any data point again. This leads to the ReLU-units irreversibly dying during the training since they can get knocked off the data manifold, which may lead to most of the neurons never activate again if the learning rate is set too high. Leaky ReLU attempts to solve this issue by adding a small slope, but the result may not always be consistent. With that in mind, all three activation functions seems to perform quite similar to each other with a mean R2-score within the range 0.80 to 0.92.

The hyperparameter optimization were tested manually using the combinations mentioned in section 3.6. The best performing networks are given in the table below, where the maximum R2-score reaches as high as 0.98 when using ReLU. There are however no clear consistencies of which combinations of hyperparameters performs the best, since every possible combination appears at least once in the list, with the exception of using ReLU and Leaky ReLU with a learning rate higher than 0.001.

Table 4: List of best performing R2-scores by manual tuning

Activation function	**regularization**	**nr of neurons**	**Epochs**	**learning rate**	**R2-score**
ReLU	0	5	500	0.001	0.95
Sigmoid	0.01	6	500	0.01	0.9815
Leaky ReLU	0.001	2	1000	0.001	0.8465
ReLU	0.001	6	1000	0.001	0.9880
Sigmoid	0.001	5	500	0.001	0.9336

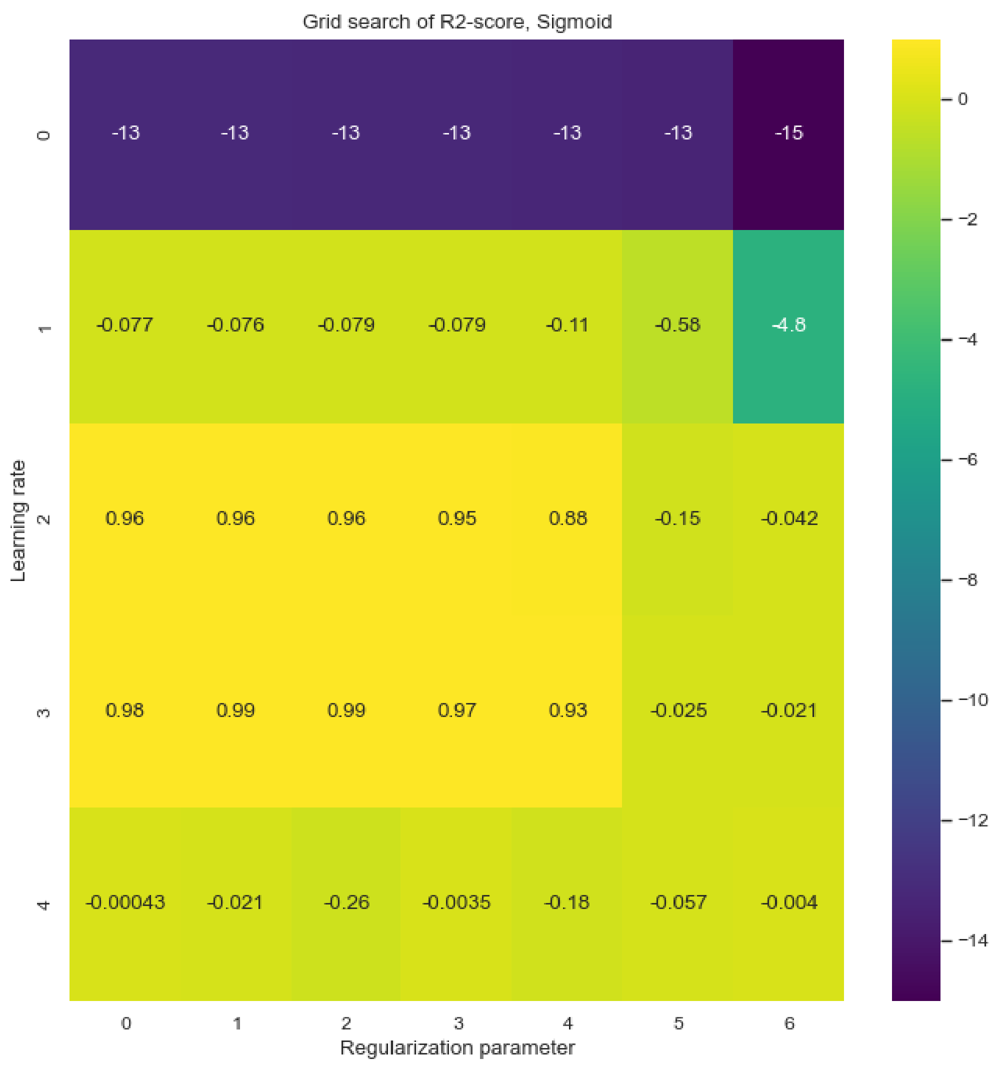


Figure 9: Grid search between learning rate and regularization parameter. Since the R2-score appears to flatten for larger epochs, the number of epochs is fixed at a value of 500.

Although there are no clear answer of which combinations of hyperparameters suits the best. The results slightly indicates that there are correlations between the regularization parameter and the learning rates. To prove this, a grid search of the optimal parameters based on learning rate and a regularization parameters with the other parameters fixed with values from the second row of table 4. The learning rate was checked for the values 0.00005, 0.0001, 0.001 , 0.01, and 0.1. The heat map in figure 6 will assign the lowest value as 0 and the largest value as 4. The regularization parameter was tested for 7 values, being: 0.00001, 0.0001, 0.001 , 0.01, 0.1, 1.0 and 10. Again, lowest value gets labeled as 0 and highest as 6. The optimal results seems to be obtained in a learning rate between 0.001 and 0.01. Any smaller or larger learning rates drastically worsens the results. Within this range of learning rate, higher R2-scores can be obtained when $\lambda \in [10^{-5}, 10^{-1}]$. It also appears that the results get better the smaller the λ-values get.

The number of hidden layers was also tested besides the major hyperparameters. There were no significant differences between one or three hidden layers, but appears to give negative R2-scores when applying more than 3 hidden layers. The number of neurons slightly improves the R2-score if increased.

Table 5: Optimal R2-scores

R2-scores		
OLS	**Ridge**	**Neural network**
0.745	0.75	0.98

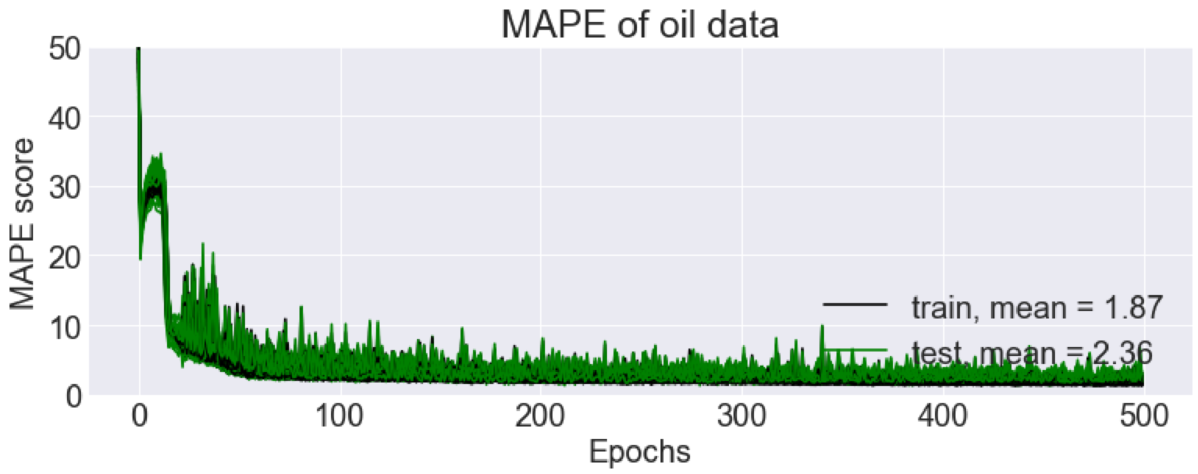


Figure 10: Mean absolute percentage error using sigmoid

In order to measure how well the neural network algorithm can forecast the oil price, it was decided to investigate the mean percentage error as well. Figure 6 shows an example run of how the mean absolute percentage error of the forecast changes as the number of epochs increases. The trend of the test data appears to be much more volatile in comparison to the training data, which appears to be quite smooth. The test data however appears to mostly have a MAPE-value under 10 % after 100 epochs. With manually tuning the hyperparameters, the neural networks seems to be superior in comparison to OLS and Ridge regression.

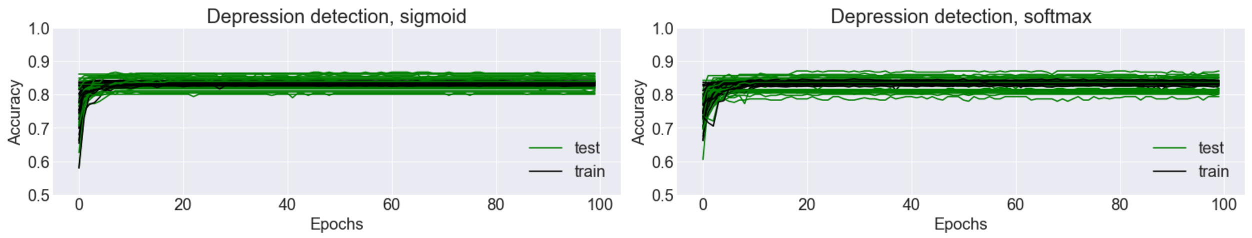
Table 6: Optimal MAPE

MAPE-value			
OLS	**Ridge**	**Neural network**	**Neural network (Herrera et al.)**
27.26	34.0	2.34	16.55

As seen from the table above, the optimal MAPE-value has drastically improved using neural networks in comparison to only use OLS and Ridge with stochastic gradient descent. The applied neural network also seems to perform much better than the results gathered by Herrera et al. (2019b), but this should be taken with a grain of salt as their neural network was run over 1000 times, and the MAPE was averaged over these runs, while the results from this analysis only included 30 runs.

4.3 Classification problem

4.3.1 Neural networks of depression data



(a) Sigmoid (b) Softmax

Figure 11: The results of calculating the accuracy-score of the classification problem up to 100 epochs using two different activation functions for the output layer. The results are relatively the same for sigmoid and softmax

An example run of the neural networks classification is shown in figure 8. For all 30 runs a convergence in the accuracy score happens at a faster rate, and appears less volatile than the regression problem regardless of activation function used for the output.

Table 7: List of best performing models for classification by manual tuning

regularization	nr of neurons	Epochs	learning rate	Accuracy score	AUC score
0.0	6	100	0.001	0.8295	0.5256
0.01	2	500	0.01	0.8372	0.5103
0.001	6	1000	0.001	0.8166	0.5034
0.001	5	100	0.001	0.8323	0.5109
0.0	6	500	0.001	0.8168	0.5135

As for the regression problem, the hyperparameters were first manually tuned for the classification problem. The accuracy score of the best runs lie between the range 0.8 to 0.83 and the

AUC-score tends to stay unchanged between 0.5 to 0.52. There are however no substantial differences as any combination of hyperparameters may give an optimal accuracy score.

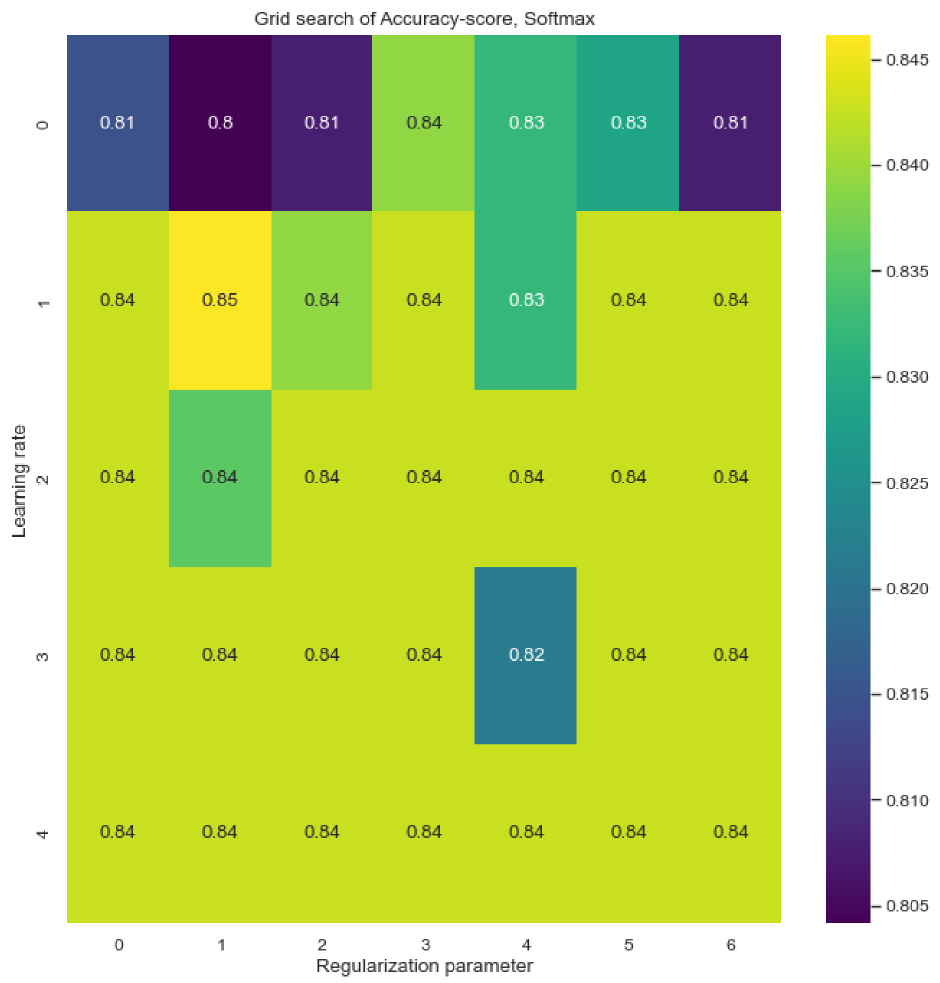
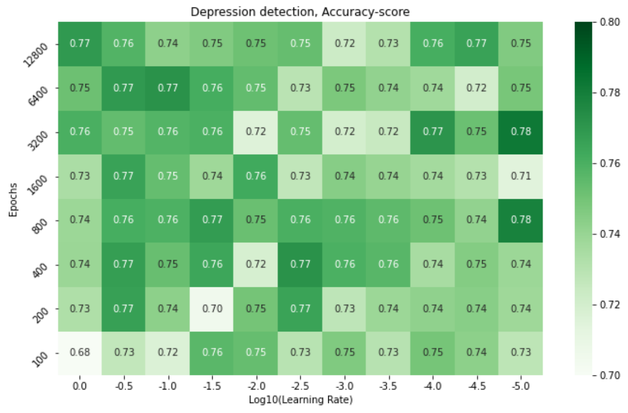


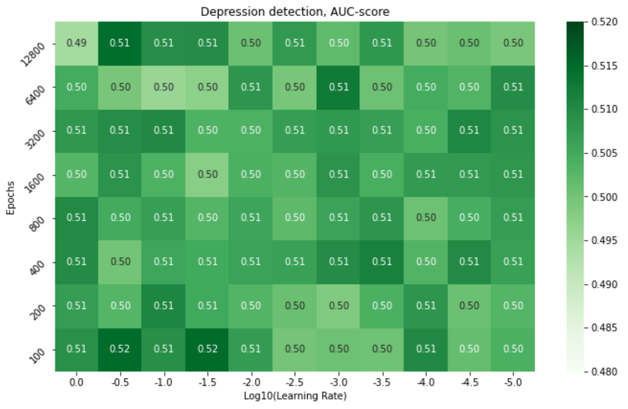
Figure 12: Grid search between learning rate and regularization parameter. Since the R2-score appears to flatten for larger epochs, the number of epochs is fixed at a value of 500.

Unlike the regression problem, the error metrics of the classification problem is improving for larger values of learning rate where the accuracy score does not stay below 0.84 when having a learning rate of 0.1. It also appears that the regularization value does not have too much impact on the accuracy score regardless of the learning rate value. The lowest learning rate value, $\eta = 10^{-5}$, still give reasonable accuracy scores between 0.8 and 0.83. When it comes to the other hyperparameters, such as the number of layers and the number of neurons, the accuracy score and the AUC-score does not appear to change too much at all, producing similar accuracy scores and AUC-scores within the values around 0.8 and 0.5 respectively. Interestingly enough, there seems to be a very small preference in learning and the regularization parameter in comparison to the regression problem. The error metrics, R2-score, of the regression problem suffers heavily when applying very small learning rates or when the regularization parameter is too large. The classification problem also appears to perform just as well for smaller number of epochs as for larger numbers of epochs.

4.3.2 Logistic regression



(a) Accuracy score



(b) Area Under Curve

Figure 13: Heat maps of accuracy score (left) and AUC (right) of logistic regression along with different combinations of epochs and learning rate. The result consisted of the average of 30 runs with 20 % test data.

It is also interesting to test the classification problem with other algorithms as well. Since the results of regularization parameters appears to have little effect, a grid search map tuning the learning rate and higher number of epochs were tested for logistic regression. The figures above shows the accuracy score and AUC-score of the logistic classification dependent on epochs and learning rate. The optimal values were given as 0.78 and 0.52 respectively. The accuracy score seems to perform better than the AUC-score with the AUC-score having almost identical values throughout the entire heatmap, making the model quite naive. The distribution accuracy distribution of the accuracy score appears to be somewhat inconsistent as lower values can be found regardless of number of epochs or the value of the learning rate. However, the accuracy score slightly improves with increasing epochs in general.

The area score was expected to perform worse the longer the data has been trained as this would often lead to overfitting the data. Considering that the dataset contained 1143 surveys of people based on 8 features, this might be the case. However, it is still quite difficult to interpret based on the current heat map. On the other hand, all the lower AUC-scores did undergo the most number of iterations.

Table 8: Optimal error metrics for classification problem

Error metrics, Classification			
Accuracy (Neural net)	**AUC (Neural Net)**	**Accuracy (LogReg)**	**AUC (LogReg)**
0.84	0.53	0.78	0.52

The final results of the error metrics are shown in table 8. The performance of the models appears strikingly similar for the AUC-score, while the neural network slightly outperforms logistic regression by a margin of 0.06. The results of the AUC-score must however be under further observation as the results throughout the entire analysis are barely passing over 0.5. An AUC-score of exactly 0.5 means that the classifier is not able to distinguish between positive and negative class point, which indicates that the classifier is either predicting random class or a constant class for all data points. The latter is not the case as the predictions did not return a constant class for all data points. None of the predicted AUC-score did have an exact value of 0.5, but quite close to it. Provided that the class representation in the data set is reasonably balanced, having a lower AUC-score compared to accuracy score should not matter too much as long as the classifier is deployed at a specific threshold. However, this was not the case for the chosen data set, as less than 50 % of the participants are confirmed to be diagnosed with major depression. It would therefore be a smarter choice to consider other metrics such as confusion matrix or precision score.

5 Conclusion

The goal of this project is to explore regression and classification problems by using neural networks and logistic regression. The main focus is to test how different hyperparameters impact the performance of those models. A neural network and logistic regression was thus built from scratch to study its inner structure with theoretical depth. OLS and Ridge with SGD was used as a comparison to the regression problem in neural net. Logistic regression did likewise for the classification problem. In the regression set, a data set containing the development of different oil prices was studied, while a data set connecting the living conditions and depression diagnostics among the population in Siyaya county, Kenya was used.

For the regression problem, neural network drastically outperforms OLS and Ridge with SGD, acheiving an R2-score of 0.98 compared to 0.745 and 0.75 for OLS and Ridge respectively. It appears that the optimal learning rate lies between 0.001 and 0.01 with a regularization parameter $\lambda \leq 1.0$, but smaller learning rates can also give reasonable R2-scores when using ReLU and leaky ReLU as activation functions as long as the regularization parameter is less than 1.0. The forecastability of the model seems to be much better for neural networks as well with a mean absolute percentage value of 2.34%. It is also noteworthy to mention that the MAPE-value from Herrera et al. (2019b) running neural networks 1000 times also gave much better results compared to OLS and Ridge with MAPE-value of 16.55%. The number of layers did not affect the results too much, but increasing the neurons per layer seems to slightly improve the results.

In the classification problem it can be concluded that the neural network outperforms the logistic regression method by a small margin with an accuracy score of 0.84 while the logistic regression gave a maximum accuracy sore of 0.78. Unlike the regression problem, the choice of regularization value appears to matter less, and a larger learning rate can be chosen without worsening the the error metric. The accuracy score did however slightly decrease when the learning rate is 10^{-5} or presumably lower. The optimal area score remained mostly unchanged regardless of tuning different hyperparameters with area scores 0.53 and 0.52 for the models of neural network and logistic regression respectively. This shows how important it is to choose an error metric accordingly to the data set. Less than 50 % of the participants have been diagnosed with depression, which in turn gives a large number of zeros compared to ones in the target value when performing a binary classification. The issue of such class imbalance is that it can result in a serious bias towards the majority class, reducing the classification performance and increasing the number of false negatives (*Why Can't I just Use the ROC Curve?*, 2018). The ROC-curve used in this analysis may be quite insensitive for imbalanced settings due to this. The accuracy score is thus a preferred error metric in this project, and applying undersampling techniques should be considered for improving the results.

6 Further improvements

In this project, there is a lot potential to explore a larger set of hyperparameters to optimize the results, especially for the classification problem. It would be beneficial to implement other activation functions such as tanh or to use other gradient descent optimization algorithms such as AdaGrad or RMSProp. Additionally, it would be interesting to use other target values in the regression set to predict and forecast the prices of other energy sources such as Australian coal or Russian gas and check if the model performances are similar as the prices of non-renewable sources seems to be highly correlated to each other.

It was also taken for granted that an unbalanced data set for the classification problem would not cause too much of an issue when measuring error metrics, but as seen from the AUC-scores, the ROC-AUC-score is not ideal as an error metric in the classification data set. A common remedy to handle imbalanced data set is to use undersampling techniques. This could for instances be an application of a "NearMiss"-algorithm where the distances between all instances of the majority class and the minority class are calculated. Then k-instances of the majority class that have the smallest distances to those in the minority class are selected. If there are n-instances in the minority class, the nearest method will result in $k \times n$-instances of the majority class.

References

Durstewitz, D., Koppe, G. and Meyer-Lindenberg, A. (2019). Deep neural networks in psychiatry, *Molecular Psychiatry* **24**: 1.

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st edn, O'Reilly Media, Inc.

Herrera, G., Constantino, M., Tabak, B., Pistori, H., Su, J.-J. and Naranpanawa, A. (2019a). Data on forecasting energy prices using machine learning, *Data in Brief* p. 104122.

Herrera, G., Constantino, M., Tabak, B., Pistori, H., Su, J.-J. and Naranpanawa, A. (2019b). Long-term forecast of energy commodities price using machine learning, *Energy* **179**.

Lewis, C. D. (1982). Industrial and business forecasting methods: a practical guide to exponential smoothing and curve fitting, *Butterworths Scientific* .

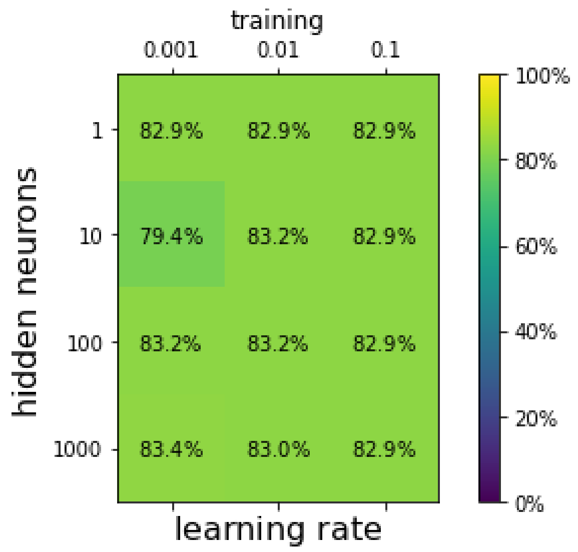
Nielsen, M. (2015). *Neural Networks and Deep Learning*, Determination Press.

Why Can't I just Use the ROC Curve? (2018).
URL: *https://towardsdatascience.com/sampling-techniques-for-extremely-imbalanced-data-281cc01da0a8*

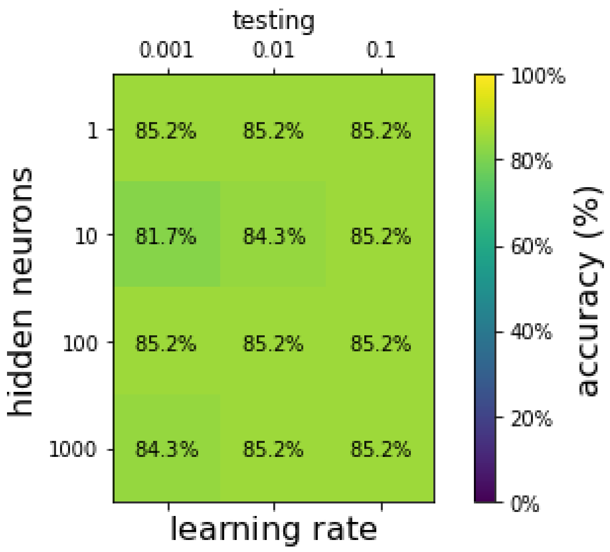
Zou, H., Xia, G., Yang, F. and Wang, H. (2007). An investigation and comparison of artificial neural network and time series models for chinese food grain price forecasting, *Neurocomputing* **70**: 2913–2923.

Appendix A Using Scikit and Keras to build a neural network

This section of appendices shows the performances of the neural networks when applying Scikit and Keras. Most of the codes are borrowed from the lectures notes. The same data sets has been used as for the regression and classification set in the main analysis. The main code for neural network has technically been compared with Keras and Scikit giving similar performances, but was dropped from the main report due to time limitations, and that there's already much to explore and discuss in the main analysis.



(a) Train



(b) Test

Figure 14: Accuracy scores of depression data sets using Keras with ReLU as activation function.

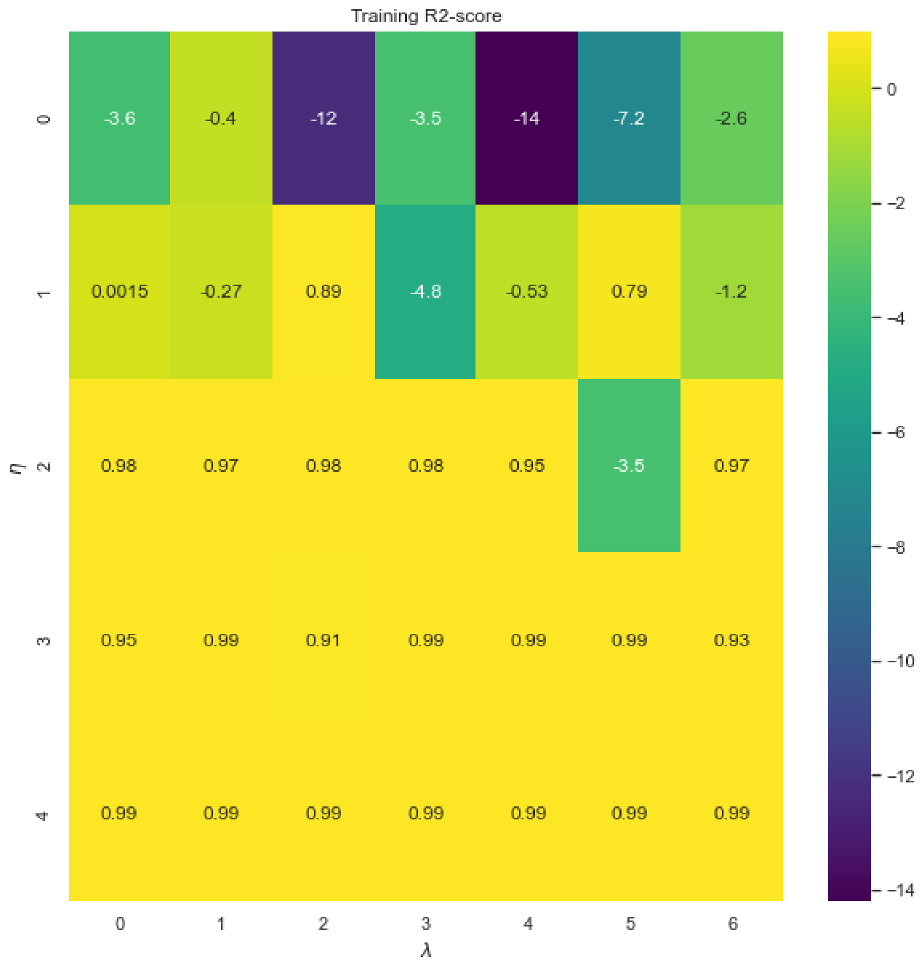


Figure 15: Train R2-score of oil data, same grid-search values are applied as for the main analysis, using ReLU as activation and Adam as optimization parameter

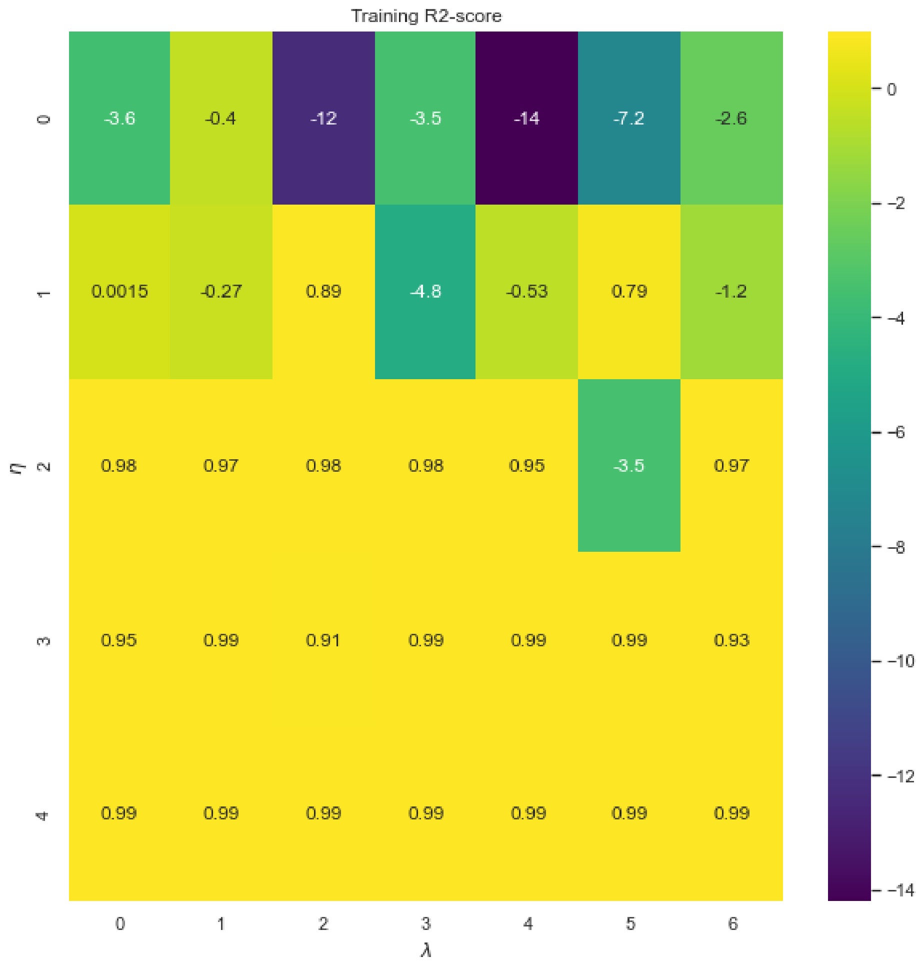


Figure 16: Test R2-score of oil data