Comparing Regression and Classification solutions to Feed Forward Neural Networks

Frida Marie E. Westby, Herman Brunborg, Cosmina Maria Condor,
Joanna Sulkowska, Jayeong Song

November 20th, 2021

Abstract

Prediction of quantities (regression) and labels (classification) based on previous observations are central objectives for supervised machine learning methods. This can be done using both a conventional statistical approach as well as a Feed Forward Neural Network (FFNN). In this project we have developed our own code for these methods and compared their performance on both a regression problem and a binary classification problem.

For our regression problem we tried to medel the Franke function with noise. We used both an analytical as well as a numerical approach with Stochastic Gradient Descent for linear regression, and compared the results to the ones obtained using a FFNN. For our binary classification problem, we used logistic regression and a FFNN on the Wisconsin Breast Cancer data set to predict whether a breast mass was benign or malignant.

After en extensive hyper parameter search we found that the performance of our FFNN was nearly on par with a convensional statistical method for the regression, and slightly better for classification. For the linear regression problem the results were close up to the analytical solution using both methods. For the classification problem, where we lack an analytical solution, both methods obtained good F1 scores and accuracy. The similar performance may be due to our data set being relatively uncomplicated. How these methods compare on more complicated data remain to be tested.

Contents

1 Introduction

Prediction is one of the tasks where machine learning can aid us humans. In this project we touch upon supervised learning and explore how we can both predict function values and categorical values associated with input variables .

Linear regression can be used for predicting function values and logistic regression can be used for predicting the affiliation to a class. A Feed Forward Neural Network (FFNN) can however also be used for these two tasks [1].

Through this paper we will learn how to implement these methods through a hands-on approach where we apply them on dataset examples. We will predict the output of the Franke function using both linear regression and a FFNN. This will be done with linear regression using analytical methods as well as using a numerical approach, namely Gradient Descent. For comparing the performance of logistic regression to FFNN we test our methods on the Wisconsin Breast Cancer Dataset [2].

Firstly we will introduce the reader to all the relevant methods and metrics of evaluation for both the linear regression problem and our classification problem. We will also implement our own code and compare it to the built in functionalities of the Scikit-learn library [3]. Full code for our method implementations and analyses can be found on [1]GitHub.

A topic of utmpost importance will be to optimize, or tune, a range of so called hyperparameters. These are parameters that can alter the training of our models and can be tuned in order to obtain the best predictions. The choices of hyperparameters is something that will be thoroughly covered in the discussion part.

In summary we believe this paper will provide the reader with a deeper understanding of how these methods work and compare.

2 Method

2.1 Data

2.1.1 Franke Function

The Franke function 1 is an analytical function of two variables. In this paper we use it for assessing the performance of our regression models. It is defined as follows:

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

for $x, y \in [0,1]$. Throughout this project we also added random noise to the Franke function in order to simulate values obtained in real life.

The data has also been scaled. See this report [2] for a more thorough exploration of the Franke function and the scaling of the data.

2.1.2 Wisconsin Breast Cancer Data

In order to test our implemented methods for classification we used the Wisconsin Breast Cancer Data.

This is a tagged binary classification data set consisting of 569 breast tumors that are either benign (non-cancerous) or malignant (cancerous), i.e. breast cancer vs breast mass that is not cancerous. In this set there are 212 (37 percent) malignant and 357 (63 percent) benignant tumors. For each tumor there are 30 features extracted based on a digitized images of a fine needle aspirate. A comprehensive list of features is included in the appendix along with a correlation matrix. We evaluated the performance of our code for logistic regression and FFNN on this data set and evaluated the performance. [2]

[1] https://github.com/hermanbrunborg/FYS-STK4155-project-2.txt
[2] https://github.com/hermanbrunborg/linear-regression-fys-stk4155/blob/main/report/report.pdfReport on linear regression

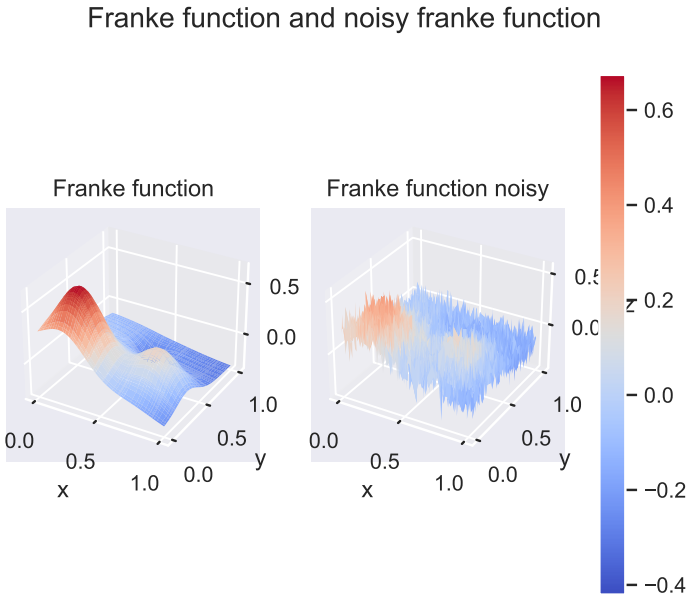

Figure 1: The Franke function and the Franke function with added noise, $\mu = 0$, $\sigma^2 = 0.2$

2.2 Model Assessment

In the following we describe the measures of evaluating the performance of the regression and classification.

2.2.1 Regression

Two metrics of evaluating the performance of a model on a linear regression task are the Mean Square Error (MSE) (which can be decomposed as the sum of bias squared, variance and error, see the report for linear regression [3]) and the R2-score. [4]:

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 \tag{1}$$

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2} \tag{2}$$

where n is the number of datapoints, $\tilde{\boldsymbol{y}}_i$ is the value we predicted of the $i - th$ sample, y_i is the corresponding true value, and the mean value $\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$.

2.2.2 Classification

In order to evaluate the performance of the models on classification we used metrics described in this section.

Accuracy score represents the fraction of correctly classified cases. This measure has limitations especially in unbalanced binary data sets (i.e. data sets where the class labels greatly differ in their numbers) as models classifying all inputs to one specific class may give a high accuracy score if the data set consists of a high number of cases corresponding to one class. As our data set was somewhat unbalanced (37 percent benign and 63 percent malignant) we explored other metrics of evaluation, described in the following.

Positive Predictive Value (PPV), also called precision, is the fraction of true positive classifications out of all predicted as positive cases (true positive + false positive). In other words it is the proportion of all positive predictions that truly are positive and in our classification problem represents the probability of having a

[3] https://github.com/hermanbrunborg/linear-regression-fys-stk4155/blob/main/report/report.pdfLinear regression

malignant tumor given that the model predicts it to be malignant.If the number of false positives is low than the PPV is high.

Negative Predictive Value (NPV) is the fraction of truly negative classifications out of all the ones predicted as negative (true negative + false negative). That is the probability of not having a malignant tumor given that the model predicts the tumor to be non-malignant. If the number of false negatives is low than the NPV is high.

Sensitivity, also called recall, refers to the fraction of true positive classifications out of all truly positive cases (true positive + false negative). In our case the probability of the model predicting the case as malignant given that the tumor is truly malignant. If the number of false negatives is low than the sensitivity is high.

Specificity is the fraction of truly negative classifications out of all the truly negative cases (true negative + false positive). For our classification problem the probability of the model predicting the case as benign given that the tumor is benign. If the number of false positives is small than the specificity is high.

F1 score is the harmonic mean of the precision (PPV) and the recall (Sensitivity) and is in that way combining these into a single metric of performance. The harmonic mean gives more weight to low values and this score will only be high if both PPV and sensitivity are high (**TODO: I found the content of this sentence in the book page 92-93 in Hands-on machine laerning, but i don't understand it completely, maybe we can discuss it, if we don't understand it maybe it should be deleted?**) [5, page92-93]

Overview of the formulas:

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(\hat{y}_i = y_i)}{n} \tag{3}$$

$$\text{PPV} = \frac{TP}{TP + FP} \tag{4}$$

$$\text{NPV} = \frac{TN}{TN + FN} \tag{5}$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{6}$$

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{7}$$

$$\text{F}_1 = \frac{TP}{TP + 1/2(FP + FN)} = 2 * \frac{PPV * Sensitivity}{PPV + Sensitivity} \tag{8}$$

Where I is the indicator function, 1 if $\hat{y}_i = y_i$ and 0 otherwise. TP is number of true positives, FP is the number of false positives, TN is the number of true negatives and FN is the number of false negatives.

2.3 Linear regression

The goal of regression is to explain output data based on input data along with noise. The underling assumption is there is a linear relationship between input and output. In addition, This linear relationship can be expressed using a matrix equation as follows:

$$\text{y} = X\beta + \epsilon \tag{9}$$

where output $\boldsymbol{y}$ stands for actual values, known quantity $\boldsymbol{X}$ stands for the input data matrix(i.e., design matrix), unknown quantity $\boldsymbol{\beta}$ stands for the polynomial coefficient vectors, and $\boldsymbol{\epsilon}$ stands for noise

$$\text{y} = [y_0, y_1, y_2, \ldots, y_{n-1}]^T, \tag{10}$$

$$\text{X} = \begin{bmatrix} 1 & x_0^1 & x_0^2 & \ldots & \ldots & x_0^{n-1} \\ 1 & x_1^1 & x_1^2 & \ldots & \ldots & x_1^{n-1} \\ 1 & x_2^1 & x_2^2 & \ldots & \ldots & x_2^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n-1}^1 & x_{n-1}^2 & \ldots & \ldots & x_{n-1}^{n-1} \end{bmatrix}, \tag{11}$$

$$\beta = [\beta_0, \beta_1, \beta_2, \ldots, \beta_{n-1}]^T, \tag{12}$$

$$\epsilon = [\epsilon_0, \epsilon_1, \epsilon_2, \ldots, \epsilon_{n-1}]^T \tag{13}$$

Our goal is to find a solution to the linear regression equation, but this is theoretically impossible because there is always uncertainty and noise $\boldsymbol{\epsilon}$ in real data, meaning that we will be overfitting even if we find the best fit for our data set.(i.e., find solution for this linear regression equation)

$$\tilde{y} = X\beta \tag{14}$$

where $\tilde{\boldsymbol{y}}$ stands for approximated values, known quantity $\boldsymbol{X}$ stands for the input data matrix(i.e., design matrix), and unknown quantity $\boldsymbol{\beta}$ stands for the polynomial coefficients.

Rather, we define the cost function $C(\boldsymbol{\beta})$ to see how good our model fit is. The defined cost function in matrix-vector notation is as follows:

$$C(\beta) = \frac{1}{n}\left\{(y - X\beta)^T (y - X\beta)\right\} \tag{15}$$

Regression tries to reduce the cost function by obtaining optimal parameters $\boldsymbol{\beta}$. Each regression method is distinguished by its own cost function. Among several linear regression methods, the ordinary least squares is the most often used linear regression. The cost function of OLS is defined as follows:

$$C(\beta) = \sum_{i=0}^{p-1} (y_i - \beta_i)^2 \tag{16}$$

That is, the sum of the squared difference between our predicted value and the actual value of the data. In addition, Ridge regression is slightly modified version of OLS regression. The cost function of Ridge regression is just the OLS with added regulation term, which is defined as follows:

$$C(X, \beta) = \frac{1}{n}||y - X\beta||_2^2 + \lambda||\beta||_1 \tag{17}$$

This term consists of the squared size of beta-vector multiplied by λ. Here λ regulates the weight of the $\boldsymbol{\beta}$. The ridge regression tries to penalize the slopes of model and this penalty is proportional to the hyper parameter λ. This can be interpreted as we want to reduce the power of predictors by increasing the shrinkage penalty on the beta coefficients of ridge regression. By increasing this penalty, ridge will have lower beta values than that of OLS. [4]

Based on the developed OLS and ridge regression from project 1, we try to see how these linear regression methods perform as function of learning rates, epochs and mini-batches, and epochs along with scaling of the learning rate.

2.4 Gradient Descent Methods

Gradient Descent (GD) is a numerical method for finding the minimum of a function. As the gradient of a function points in the direction of steepest ascent, the negative gradient points in the direction of steepest descent. For a small enough step size in the direction of steepest descent we will get closer to the minimum of the function. After one step in the direction of steepest descent is taken the gradient is calculated again, thereafter another step in the direction of the negative gradient is taken. This is repeated until the gradient is zero (or in reality smaller than a given tolerance). In this iterative way a minimum is obtained. The method stops when the gradient is close to zero even if we only reach a local minimum. We have no guarantees that the minimum we have reached is indeed the global minimum.

Mathematical representation:

$$\mathrm{x}_{k+1} = x_k - \eta_k \nabla F(x_k) \tag{18}$$

Where η is the step size, ∇ F is the gradient, x_k is the point we calculate the gradient for and $x_{(k+1)}$ is the next point on the function we arrive at after stepping in the opposite direction of the gradient.

The step size in the direction of steepest descent is called the learning rate (η) and can be altered. If the learning rate is too small than a great number of iterations will be needed before we reach a point close to

zero. On the other hand, a high learning rate puts us at risk of stepping over a minimum. Thus, the learning rate becomes a hyper parameter that can be tuned. The learning rate can also be scaled to first be large and to diminish in size as we approach a minimum.

2.4.1 Stochastic gradient descent

Stochastic Gradient Descent (SGD) is a variant of the gradient descent method where we only calculate the gradient on a subset of the data, called a mini-batch. This is computationally less expensive than calculating the gradient on all the data points. The subset of datapoints we calculate the gradient for is called a minibatch. One iteration over the chosen number of minibatches is called an epochs. Thus both the mini batch size and the epochs (the number of times one loop over the minibatches) become hyperparameters.

In our code we explored how our metric of evaluation changed for different numbers of mini-batches and epochs.

In our code we calculated metrics of performance using both the analytical solutions to optimize the coefficients for OLS and Ridge and using the numerical SGD method to explore how these compare.

2.4.2 Use of gradient descent methods in our project

Gradient descent methods can be used both when there is an analytical solution and when it is not. The main reason of using SGD for OLS and Ridge (even as an analytical solutions exist) is that it is computationally cheaper. For logistic regression (next section) we have no analytical solution, in this case we depend on a numerical approach.

2.5 Logistic Regression

Logistic regression models the probability of a discrete dichotomous outcome. It is a method where we assume that p(x) is related to x by a Sigmoid function [6, page621]. The logistic function is a common choice of Sigmoid function and is denoted as follows:

$$\text{p(t)} = \frac{1}{1 + exp(-t)} \tag{19}$$

The range of output of this function is between 0 and 1 and can be interpreted as the probability of an input belonging to one of two binary classes. Commonly a threshold of 0.5 is chosen for separation, i.e. if a certain input gives a probability above the this is classified into one class rather than the other. The positive class is often labeled as 1 (or True) and negative class as 0 (or False). If p(x) is close to 1, it indicates that the likelihood of belonging to the positive class is high and the input is assigned the output 1 (or True). Likewise if p(x) is close to zero it indicates a high likelihood of belonging to the negative class and the input is therefore assigned the output 0 (or False). In this way we get a binary classifier. [5, page142]

The objective in training a Logistic regression model is to set the parameters of the function in a way that gives a good classification. Logistic regression models are usually fit by maximal likelihood [7, page120]. We did this using a gradient descent method on the Cross-Entropy function (also known as the Log-loss function).

Since we want a value between 0 and 1, we apply the Sigmoid function to the hypothesis, meaning the the hypothesis is as follows:

$$h = \frac{1}{1 + e^{-\theta^T \cdot x}} \tag{20}$$

The cost function is:

$$J = -\frac{1}{m} \sum_{i=1}^{m} y_i \log(h_i) + (1 - y_i) \log(1 - h_i) + \lambda \sum_{j=1}^{n} \theta_j^2 \tag{21}$$

The gradient of the cost function is:

$$\frac{\partial J}{\partial \theta} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \cdot X_i + \frac{\lambda}{m} \theta \tag{22}$$

We performed a search for optimal hyperparameters optimizing the metrics of evaluation described above. Due to restrictions in computational power we restricted the search for optimal hyperparameters to the following ranges learning rate: $10^4 - 10^2$, regularization parameter (lambda): $10^4 - 10^2$.

2.6 Feed Forward Neural Network

Inspired by the brain, we have a machine learning technique called *Artificial Neural Networks* (ANN). [8] One type of ANN is the *Multilayer perceptron* (MLP) which we have used to implement our *Feed Forward Neural Network* (FFNN).

The MLP is the a very common method in machine learning. It consist of a input layer, output layer and one or more hidden layers.

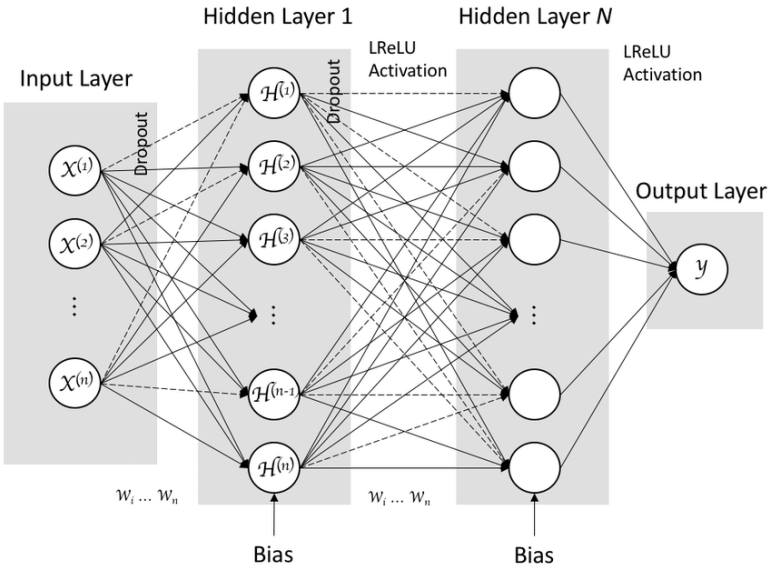

Figure 2: Multilayer perceptron [9]

For training the MLP we need to *feed forward* and *back propagate*. In the feed forward part, we are sending data through the network, by adding updating weights and adding bias for the current layer. After that, we are then going back trough the network, updating weights with the difference between the input nodes and the target target nodes. This is most commonly known as back propagation. The biases

The MLP algorithm starts with initialisation, here we have initialised all weights using a normal distribution of small random numbers.

The next step is to train the model. Our model has been implemented to handle multiple hidden layers and flexible amount of nodes. For each epoch and iteration, the training starts with the feed-forward phase, where we start on our input layer. So for each node in the input vector we send to the input layer, we are computing the activation function:

$$a_j^l = f(z_j^l) \tag{23}$$

where z_j^l is the calculated weighted sum of the input features

$$z_j^l = \sum_{i=1}^{F} w_j^i x_i + b_j^l \tag{24}$$

here j is a neuron in the hidden layer l, F represent number of features, w is the weights and b is the bias.

Then using the Sigmoid function as our activation function, we get

$$a_\zeta(x) = \sigma(x) = \frac{1}{1+e^{(-x)}}, \;\; a'_\zeta(x) = a_\zeta(x)(1 - a_\zeta(x)) \tag{25}$$

where β is the bias values.

Or if our activation function is the Leaky ReLU function, we get

$$a_\zeta(x) = \begin{cases} cx, & \text{for } x < 0. \\ x, & \text{else.} \end{cases}, \quad a'_\zeta(x) = \begin{cases} 1, & \text{for } x > 0. \\ c, & \text{else.} \end{cases} \tag{26}$$

.

If our activation function is the ReLU function, we get

$$a_\zeta(x) = \begin{cases} x, & \text{for } x > 0. \\ 0, & \text{else.} \end{cases}, \quad a'_\zeta(x) = \begin{cases} 1, & \text{for } x > 0. \\ 0, & \text{else.} \end{cases} \tag{27}$$

we see that the ReLU function can be viewed as a leaky ReLU function with $c = 0$.

The last step is to take the output and apply the last activation function. If we want to do classification, we classify all values over a threshold to True and all smaller to False. The sigmoid function is a good fit for the last layer, since it maps values $\sigma :\rightarrow [0, 1]$. In our case, we will classify to True if the value is over 0.5.

After going trough the feed forward phase as described above, we enter the backward propagation (or backward phase). Here we need to minimize the cost function, since this is what trains our network; minimizing the cost function by updating weights and biases. The simplest way to do this is by using gradient decent [4][link to SGD part]. The *learning rate* η decides how big steps we are taking for trying to reach til global minimum. For a parameter θ we get

$$\theta_{i+1} = \theta_i - \eta \nabla C(\theta_i). \tag{28}$$

In our train function, we divide the training data in to so-called *minibatches*. So instead of calculating the SGD for the whole training data set, we are slicing it up to smaller batches and calculate the approximation of the SGD on those. This is known as *Batch Gradient Descent*. For a mini-batch (B_k) with size M we have the following cost function:

$$\nabla C(\theta) = \frac{1}{M} = \sum_{i \in B_k} \nabla \mathcal{L}_i(\theta).$$

Regularization is an extra term which is normally added to the cost function. The regularization will make sure that the weights do not grow out of control by constraining their size. This will also reduce the chances for overfitting. By using *L2-norm* regularization, we get the following cost function for our mini-batches:

$$\nabla C(\theta) = \frac{1}{M} = \sum_{i \in B_k} \nabla \mathcal{L}_i(\theta) + \lambda ||\hat{w}||_2^2,$$

where λ represents the regularization parameter. For classification we used cross entropy, while we used quadratic cost for the regression case.

We also need to initialize our weights and biases. For the weights, we initialized them from a normal distribution with $\mu = 0, \sigma^2 = 0.5$, and the biases were initialized to 0. The reason for initializing the weights to zero, was to avoid having the non-zero weights breaking the symmetry.

2.7 Code overview

The implementation of our own code for linear regression, SGD, FFNN and logistic regression was the backbone in all of our code for analyses. All the results we obtained by using our own implementations of the methods were compared to the results obtained using the scikit learn library. In the following we give an summary of how we have performed our analyses.

2.7.1 Approach to the linear regression problem

We calculated and compared the MSE and R2 score using both analytical and numerical (SGD) solutions for optimizing the coefficients of a fitted 5th order polynomial using both OLS and Ridge Regression. This was also done for the predictions obtained using a FFNN. For Ridge regression we plotted the MSE obtained by the explicit solution for the function coefficients as a function of different lambdas.

For the numerical approach we had to tune the above mentioned hyperparameters. As a first step we searched for number of epochs and mini-batches that gave the best MSE for OLS. Thereafter we fixed these

[4] SGD analysis

values to the optimal ones while exploring different initial learning rates and regularization terms for Ridge regression. We point out that we searched for *initial* learning rates as we also added code for scaling the learning rate, a method that reduced the learning rate with each step taken in the direction of steepest descent. The ranges of search for our mini-batch size was $1 - 120$, number of epochs $1 - 120$, learning rates $0.01 - 3.00$ and regularization terms lambda 10^{-1}.

2.7.2 Approach to the classification problem

For the classification problem we made code for tuning hyper parameters for both logistic regression and the FFNN to the metrics of evaluation described earlier in this section. In the logistic regression we explored different learning rates and penalty terms and for the FFNN we in addition to these also explored different number of hidden layers and different amount of neurons in each layer. In short, our code produced one dictionary per metric of performance with the value of the metrics as value and the hyper parameters used to obtain it as keys. We then extracted the hyper parameters producing the best values for our metrics and used those parameters to make a confusion matrix to evaluate the performance.

Due to restricted computational power we restricted our search for optimal hyper parameters to the following ranges: learning rates 0.005 – 0.1, regularization parameters (λ) 0.001 – 0.1, number of hidden layers 1-3, number of hidden nodes: 30 and 40.

We again remind that all the code used in this project is available at the [5]GitHub link pasted into the introduction of this paper.

3 Results

3.1 Regression

For the OLS linear regression method of predicting function values with a 5th order polynomial we got a MSE = 0.0091 and R2 = 0.71 using the explicit solution. In comparison, using the numerical SGD method for the same task, we got the best MSE = 0.0158 and R2=0.50. This value was obtained for 60 epochs and 32 mini-batches.

Using the SGD approach for the ridge regression we got the best MSE when $\lambda = 1e - 05$ and $\eta = 0.20$. Using these hyperparameters, we got $MSE = 0.0126$ and $r2 = 0.60$.

When we were using and explicit solution for ridge, we got the optimal parameters when $\lambda = 1e - 07$. We then got $MSE = 0.009$ and $R2 = 0.44$.

[5]https://github.com/hermanbrunborg/FYS-STK4155-project-2.txt

Heat map for SGD OLS with different mini batches and epochs

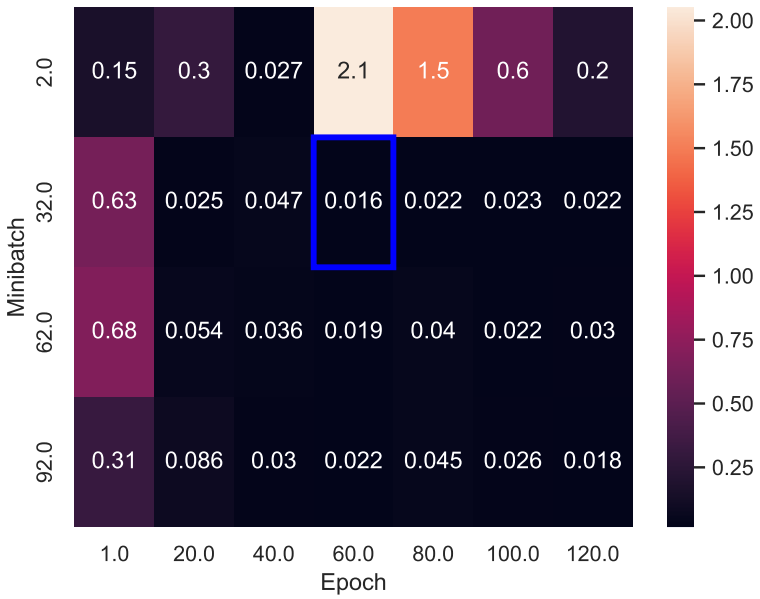

Figure 3: Heatmap showing the best MSE value for OLS with SGD implementation, when we look at different values for the regularization term λ and the learning rate η.

Below we see a heatmap for the MSE for Ridge with SGD solution as function of different lambdas and learning rates (here the number of mini-batches and epochs was set to the one obtained from the search using OLS, i.e. number of mini-batches and epochs corresponding to the lowest MSE on the heatmap above):

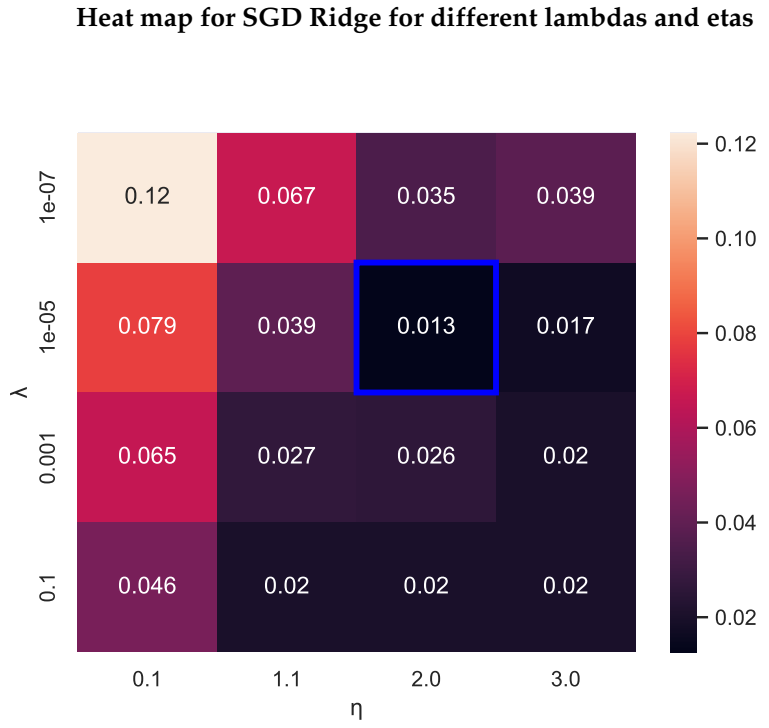

Figure 4: Heatmap showing the best MSE value for OLS with SGD implementation, when we look at different values for the regularization term λ and the learning rate η.

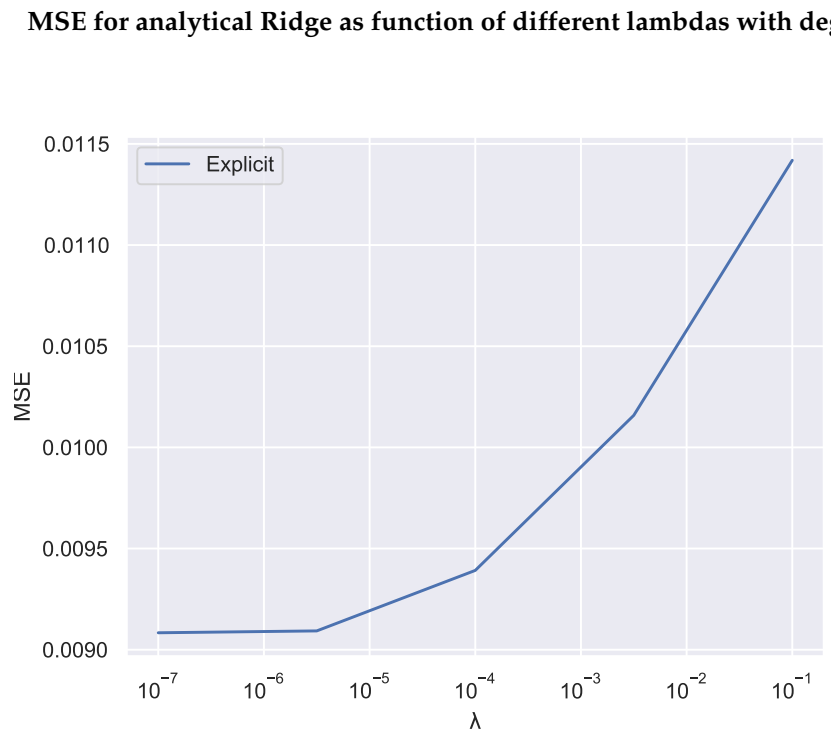

Figure 5: MSE score for Ridge regression with a given regularization term λ.

We see that the gains of ridge are minimal for a model of degree only 5, compared to one without regularization.

3.1.1 Feed Forward Neural Network

When running the feed forward neural network for different learning rates (0.005 0.024 0.043 0.062 0.081 0.1), lambdas (0, 0.005, 0.024, 0.043, 0.062, 0.081, 0.1) and testing for 1, 2 or 3 hidden layers, with 30 or 40 as the hidden layer size, we found that we got the best performance when the learning rate was 0.005, lambda was 0, hidden layers was 2, hidden layer size was 30. We got performance $MSE = 0.19$ and $R2 = 0.38$.

We also plotted a cost function for these hyperparameters

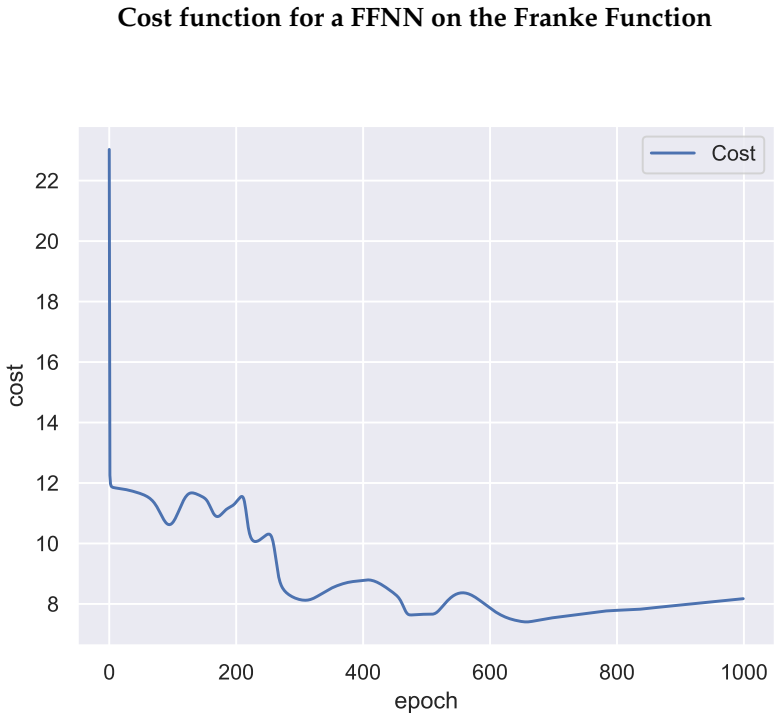

Figure 6: Cost value for OLS with SGD implementation over different epoch sizes when $\eta = 0.005$, $\lambda = 0$ with two hidden layers, each of size 30

3.1.2 Performance for different activation functions

We then preceded to test different activation functions with the best hyperparameters which were found from the hyperparameter search.

	Error function	**MSE**	**R^2**
Activation functions	Sigmoid Layer	0.019	0.31
	Linear Layer	NaN	NaN
	Leaky ReLU Layer	NaN	NaN
	ReLU Layer	NaN	NaN

Table 1: MSE and R^2 scores for different activation functions using learning rate 0.005, lambda = 0, two hidden layers, each with size 30

We then proceded to test the activation function with the same hyperparameters, except introducting $\lambda = 0.001$.

	Error function	MSE	R^2
Activation functions	Sigmoid Layer	0.027	0.17
	Linear Layer	0.014	0.50
	Leaky ReLU Layer	0.011	0.58
	ReLU Layer	NaN	NaN

Table 2: MSE and R^2 scores for different activation functions using learning rate 0.005, lambda = 0.001, two hidden layers, each with size 30

3.2 Classification

In the following we present the binary classification results obtained by both logistic regression and a Feed Forward Neural Network.

3.2.1 Logistic Regression

Optimal hyper parameters

Hyperparameters		Measures of performance for the given hyperparameters						The given hyperparameters maximized the following measures of performance
Learning rate	Penalty term	PPV (%)	NPV (%)	Sensit. (%)	Specif. (%)	Accuracy (%)	F1 score (%)	
0.1	1.0	100.0	97.3	95.3	100	98.2	98.6	Accuracy, PPV, specificity, F1 score
100	0.001	89.4	98.5	97.7	93.0	94.7	93.7	NPV, sensitivity

Figure 7: Optimal hyper parameters with the associated measures of performance

This table states for which hyper parameters we got the maximal performance estimates along with other measures of performance for those hyper parameters.

In the ranges of our search for optimal hyper parameters we found that the combination of learning rate = 0.1 and penalty term = 0.001 gave the best accuracy, F1 score, PPV and specificity. A learning rate of 100 and penalty term of 0.001 however gave us the highest NPV and sensitivity. The accuracy and F1 score obtained with our code were up to numerical precision the same as we obtained with the scikit-learn library for logistic regression.

We obtained two confusion matrices for classification using logistic regression.

	Predicted Benign	Predicted Malignant
True Benign	71	0
True Malignant	2	41

Table 3: Confusion matrix for classification using logistic, tuned for the F1 score.

The confusion matrix above represents the classification by our model where the hyper parameters (learning rate = 0.1 and the penalty term = 1.0) were tuned to optimize the F1 score. In this case as the same hyper parameters that optimized the F1 score also optimized the accuracy score, PPV and specificity, all confusion matrices that optimize these metrics of performance look the same.

The confusion matrix below is obtained by tuning the hyper parameters to optimize the NPV and sensitivity (learning rate = 100 and penalty term = 0.1).

	Predicted Benign	Predicted Malignant
True Benign	55	5
True Malignant	1	42

Table 4: Confusion matrix for classification using logistic, tuned for NPV and sensitivity.

For visualization we plotted a heat map of the F1 score for different lambdas and learning rates. The plot for F1 score and accuracy looked exactly the same.

Heatmap for the F1 score for different lambdas are learning rates:

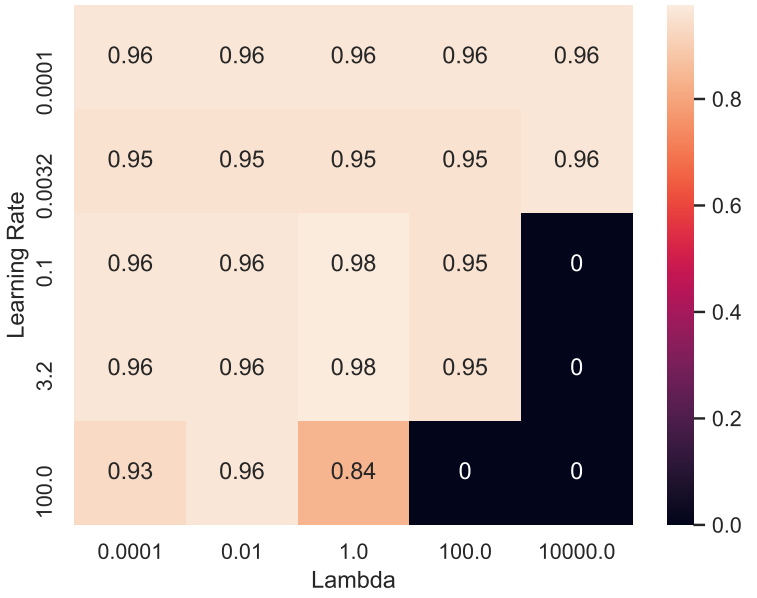

Figure 8: Heatmap showing F1 scores for different values for the learning rate η and regularization term λ.

Heatmap for the Accuracy score for different lambdas and learning rates:

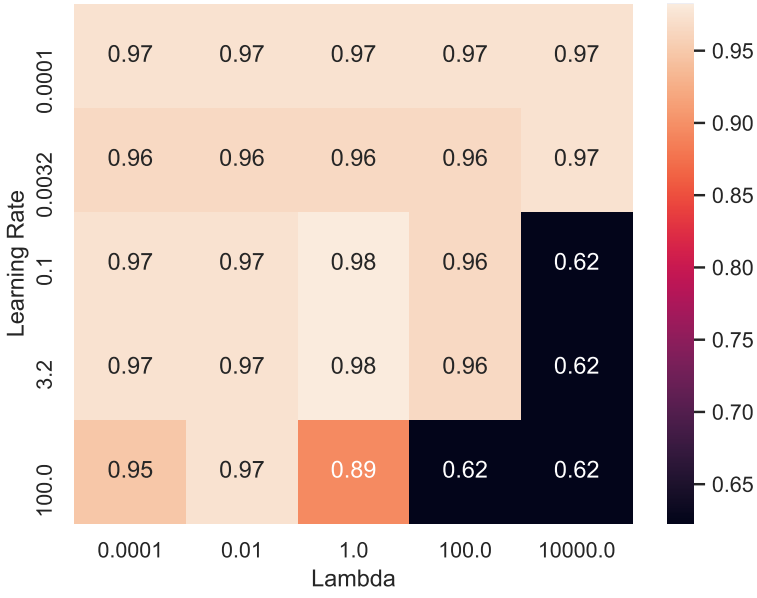

Figure 9: Heatmap showing accuracy scores for different values for the learning rate η and regularization term λ.

We also added a visualization of the NPV values obtained for different learning rates and lambdas:

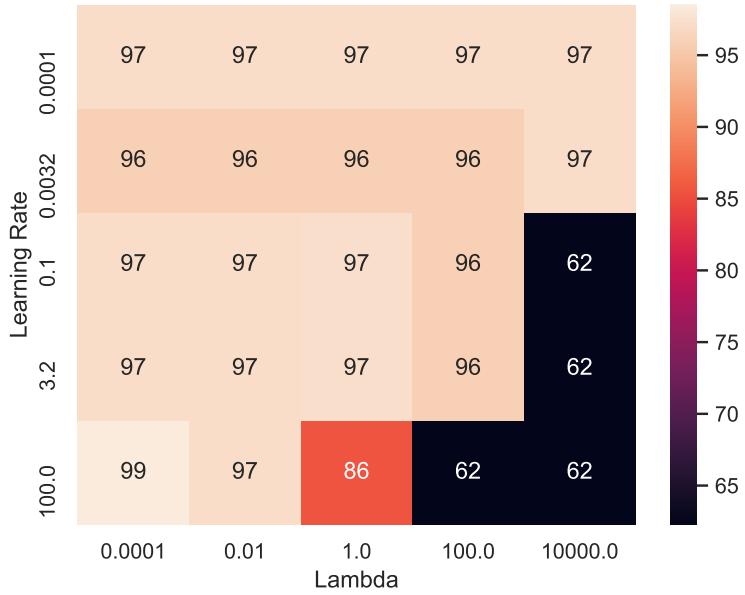

Figure 10: Heatmap showing different cost function for the breast cancer data set given the learning rate η and regularization term λ.

3.2.2 Feed Forward Neural Network

All our metrics of performance (PPV, NPV, sensitivity, specificity, accuracy, F1 -score) were optimized for the following hyper parameters: learning rate 0.005, penalty term = 0.001, number of hidden layers = 3, number of hidden nodes in each layer = 30. These were however not the only parameters which gave a good performance, with most of our tested parameters having as good or nearly as good performance.

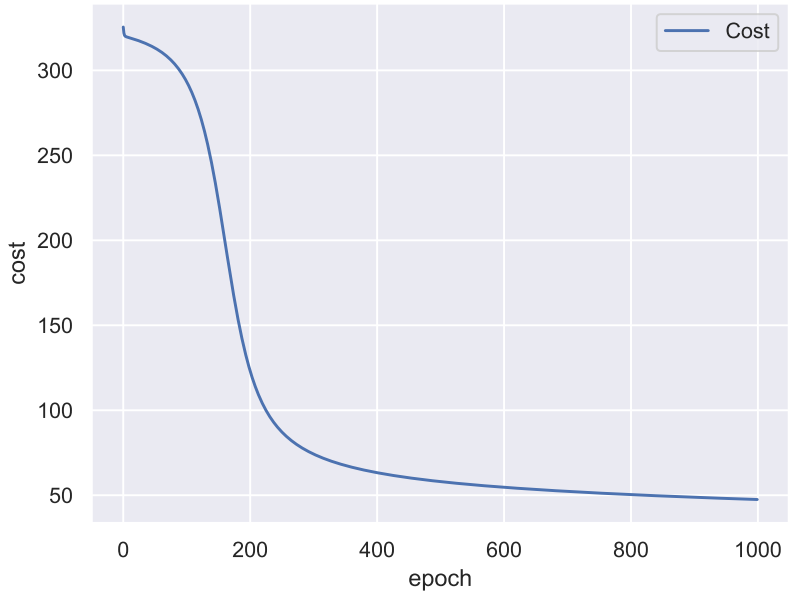

Figure 11: Cost function for different epochs for the breast cancer data set.

	Predicted Benign	Predicted Malignant
True Benign	71	0
True Malignant	2	41

Table 5: Confusion matrix for classification using FFNN, tuned for the F1 score.

Optimal hyper parameters

Hyperparameters				Measures of performance						Maximal measure of performance
Learning rate	Penalty term	Number of hidden layers	Number of hidden nodes	PPV (%)	NPV (%)	Sensit. (%)	Specif. (%)	Accuracy (%)	F1 score (%)	
0.005	0.001	3	30	100	*97.3*	*95.7*	100	99.1	98.8	**All: PPV, NPV, sens., spec., accuracy, F1.**

Figure 12: Optimal hyper parameters with the associated measures of performance

Confusion matrix for the tuned hyper parameters:

3.2.3 Performance for different activation functions

		Accuracy score	F1 score
Activation functions	Sigmoid Layer	0.97368	0.97362
	Linear Layer	0.97368	0.9736
	Leaky ReLU Layer	0.99122	0.99120
	ReLU Layer	0.96491	0.96491

Table 6: Metrics of evaluation obtained from the different activation functions on the breast cancer data set.

4 Discussion

4.1 Regression

With explicit solutions for OLS we got MSE of 0.0091 and with SGD 0.0158, with the FFNN and leaky relu as activation function in the hidden layers we got a MSE of 0.0111. The result with FFNN is in other words a tiny bit better than the one obtained with OLS using the numerical SGD method for finding coefficients.

While using the explicit solution for Ridge regression we found that the regularization, intended to reduce overfitting to the training set, actually increased the MSE obtained from the test set. With the given amount of noise added to the Franke function (described in the method section) OLS does a good job.

The reason for the worsening performance for the ridge method, is likely because traning for only degree 5. For increased degrees, the results might have looked different.

For the main parameter search for the best hyperparameters for FFNN, we got the best performance when $\lambda = 0$. From the cost function it also seems that the network might have overfitted a bit, since the cost function is quite unstable, and since it does not seem to converge evenly. This might mean that the sigmoid function is not really suited for regression problems.

The FFNN gave us very different values for the MSE and R2 scores with different activation functions for the hidden layers. The best results were obtained using Leaky ReLU. The next best was for a linear activation function. The Sigmoid function gave us quite substantially worse scores. A possible explanation for this is that the Sigmoid function is only sensitive to input values around the midpoint. It doesn't discriminate between very high vales and very low values as the outputs 1 for above high positive numbers and 0 for very low values. ReLU gave us not a number as results for both MSE and R^2, we suspect that this was due to overflow. This may be resulting from the way we initialized our weights. In our implementation of the FFNN the weights were

initialized randomly from the standard normal distribution. Lower starting values for the weights could have maybe prevented this.

It must also be noted that the values obtained when assessing the performance of the Leaky ReLU are all from one run, with hyperparameters choosen by an educated guess. This means that the potential for this activation function is likely to be even greater than the one obtained by us.

We also saw that finding the optimal parameters for a neural network can be both challenging and time consuming. Not only does training a single network take time, but there are also a lot of parameters to tweak. One approach would be to find each parameter separately, but trying this did not work, since many of the parameters are dependent on each other. For instance the learning rate and lambda.

4.2 Classification

Various classification tasks pose different demands for performance evaluation. In this classification task the goal was to predict malignancy based on given features. Classification of a tumor as cancerous or not cancerous has serious real life consequences. If a malignant breast tumor is wrongly classified as benign the patient will not receive potentially life saving cancer therapy.

Using logistic regression with the hyper parameters that gave us the highest F1-score 2 out of 43 truly malignant tumors were wrongly classified as benign (false negative) (see the the confusion matrix in the results part).

If we want to optimize our model to reduce the number of false negatives rather than seek out the highest F1 score we can tune the hyper parameters that maximize the negative predictive value (NPV). When this is done we see that the confusion matrix (see result part) changes, now only 1 out of 43 truly malignant tumors is classified as benign.

However, as these hyper parameters reduce the false negatives, we also see that the number of false positives increases from 0 to 5. That is 5 truly benign tumors are classified as malignant. This can also have devastating consequences as it in this case leads to 5 patients being wrongly diagnosed with cancer and potentially receiving chemotherapy/radiation therapy/surgery unnecessarily. As these treatments may have potentially life threatening side effects this also an important aspect aspect to take into account when deciding which metric to optimize for.

This exemplifies that choosing the correct measure of evaluation in order to tune the hyper parameters is of high importance. And it is an important argument that data scientists/programmers/developers need to collaborate closely with professionals from the field where the classification potentially should be implemented. If this classification was to be implemented in real life it may have been a good strategy to consult specialist oncologists, an ethics committee and even lawyers before choosing how to tune the hyper parameters.

Acceptable classification is dependent on the problem at hand. For classification tasks with lower stakes there may not be the same need for such a thorough evaluation.

From the heat maps in the logistic regression we see that there is quite a high spectrum of penalty terms(lambdas) and learning rates that give somewhat similar F1 scores. To reduce the computational burden it would be better to choose the highest learning rate that gives acceptable classification.

Using a FFNN we obtained very similar results as with logistic regression. Within our field of search for both learning rate, penalty term, number of hidden layers and number of hidden neurons we found that the same hyper parameters optimized the all of our evaluation parameters. So choosing which hyper parameters to use for our confusion matrix didn't pose any challenges. We could of course have expanded our range of search something which may have differentiated the optimal hyper parameters for our metrics of performance evaluation, however for that we would have needed a more computational power.

As seen from the confusion matrices for both logistic regression and FFNN used for classification where we optimized the F1 score we see that the methods performed the same. As implementation of own code for logistic regression is simpler one can argue that there may be little point in implementing a more complicated FFNN for this task.

5 Conclusion

For the linear regression problem, there was quite a bit step from the performance of the analytical OLS and the results obtained from the SGD OLS. The Ridge performance was quite comparable to that of the OLS, sometimes scoring a bit better and sometimes a bit worse. The FFNN with a sigmoid activation function also

did not perform that well, but it was a different story for the leaky ReLU, which performed nearly as well as the analytical function.

For our binary classification problem we found that both logistic regression and the FFNN managed to classify our binary data set quite good, but the performance of the FFNN was slightly better. We have also experienced that choosing different metrics of performance as targets may lead to different tuning of hyper parameters. We saw that for the FFNN, the Leaky ReLU had the best performance among the networks once again.

The fact that the FFNN turned out to classify our binary data set pretty similarly with a lot of different parameters, and also with very simple structures, tells us that the data set was relatively uncomplicated, meaning that even a simple network was able to capture the differences. Testing the performance of these on more complicated data sets might have given us deeper insights to the classification differences of these two methods.

An obvious downside for the FFNN is the implementation, the other methods are easier to implement. If we look at the classification problem, we can see that we actually get nearly as good results with the logistic regression as we get with the FFNN. With a large amount of hidden layers and nodes can make the network quite computational heavy and slow during the feed forward phase and the back propagation, especially if we have a fully connected FFNN like the one we have implemented.

5.1 Limitations and thoughts for future work

There is potential in our work for further refinement. We could for example have scaled the learning rate in the FFNN and added more advanced SGD methods, for example Stochastic Gradient Descent with Momentum. SGD with momentum is an extension of the SGD method which takes the previous update of the iteration into account when updating the parameters. This helps accelerating convergence [10].

In our analysis for the FFNN, we also only did a big parameter search for the sigmoid activation function. The was due to time constraints. Running the search took approximatly 17 hours, so testing more parameters or different activation functions would have been too time consuming.

Further we could have gained more precise estimates for our metrics of evaluation using k-fold cross validation. We did however exclude that from our project as it turned out to be computationally expensive.

If we had more computational power we could have perform better fine tuning of the optimal hyper parameters as we could have both searched for more numbers of nodes in each hidden layer and more amounts of hidden layers.

One obvious improvement and though for further work is to parallelize the code. If this was done, we could have both ran multiple parameters at the same time, meaning that the result could have been found a lot quicker.

References

[1] Andreas C. Mueller and Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016.

[2] Olvi L. Mangasarian Dr. William H. Wolberg, W. Nick Street. Breast cancer wisconsin (diagnostic) data set. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[4] Morten Hjorth-Jensen. Data analysis and machine learning. https://compphysics.github.io/MachineLearning/doc/web/course.html.

[5] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc, USA, 2019.

[6] Kenneth N. Carlton Matthew A. Devore, Jay L. Berk. *Modern Mathematical Statistics with Applications*. Springer Nature Switzerland AG, 2021.

[7] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning : Data Mining, Inference, and Prediction,*. Springer-Verlag New York Inc., 2017.

[8] Stephen Marsland. *Machine learning : an algorithmic perspective*. 2015.

[9] George Loukas, Tuan Vuong, Ryan Heartfield, Georgia Sakellari, Yongpil Yoon, and Diane Gan. Cloud-based cyber-physical intrusion detection for vehicles using deep learning. *IEEE Access*, 6:3491–3508, 12 2017.

[10] SEBASTIAN RUDER. An overview of gradient descent optimization algorithms. https://compphysics.github.io/MachineLearning/doc/web/course.html.

A Different feature characteristics for the breast cancer data set.

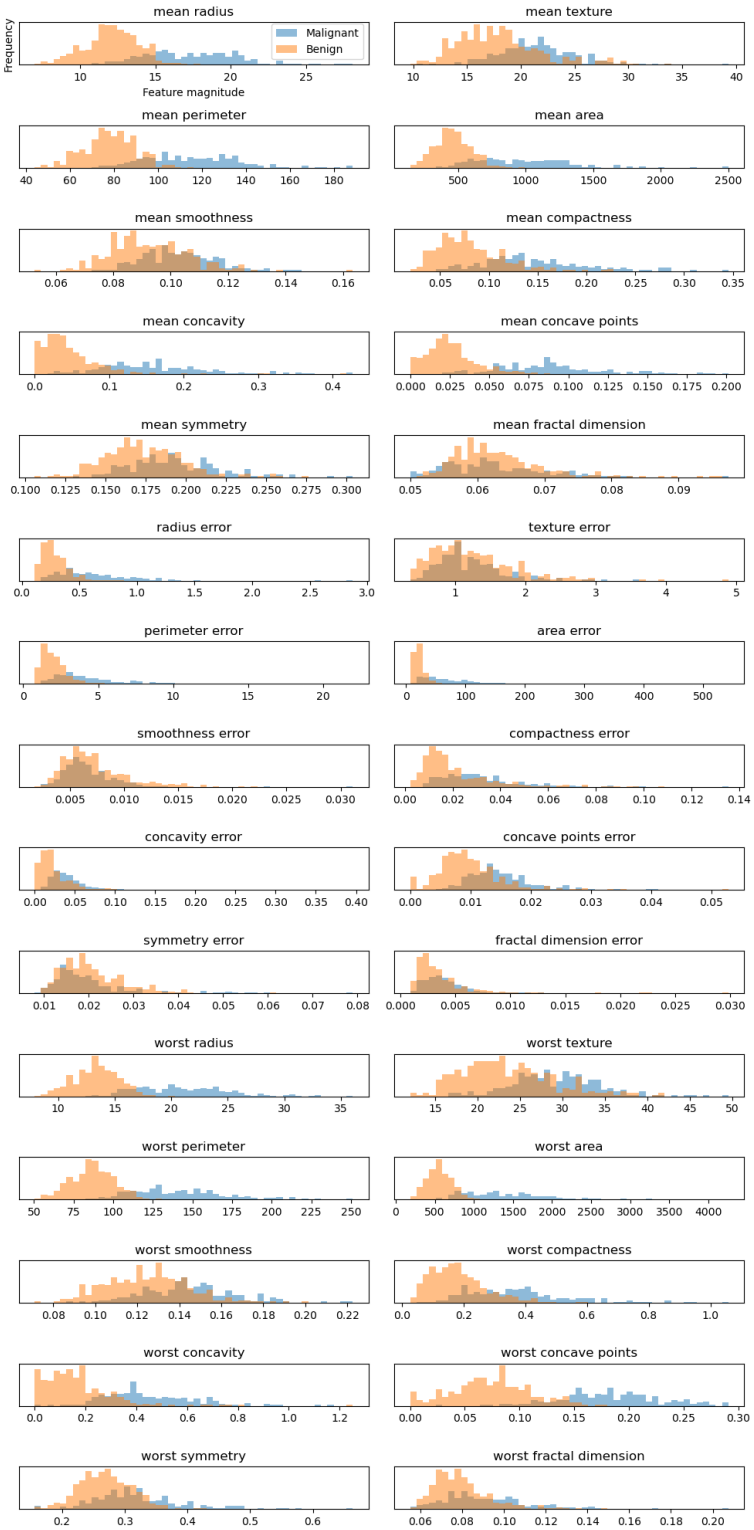

Figure 13: Feature characteristics for the breast cancer data set.

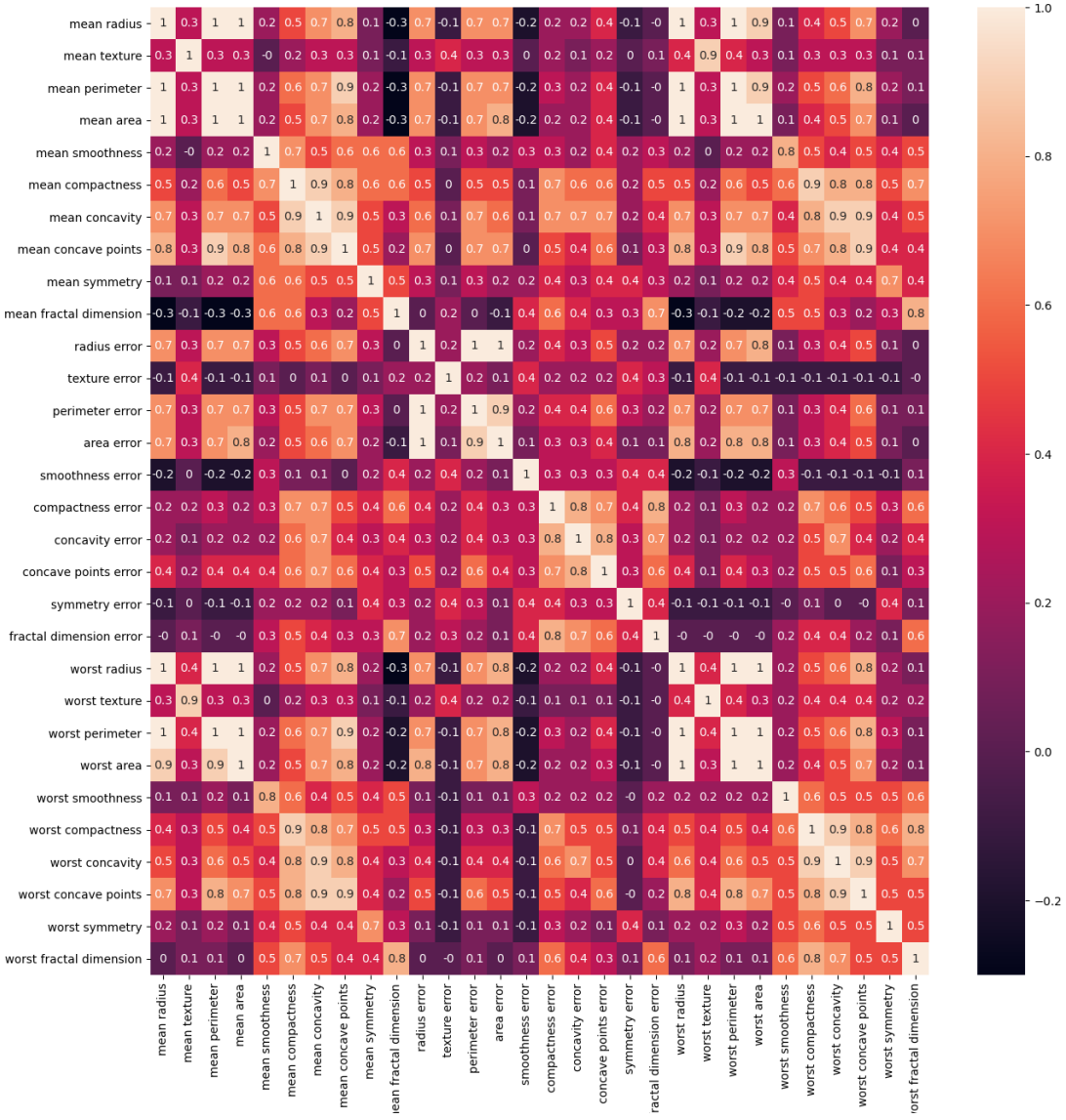

Figure 14: Feature characteristics