Project 2 FYS-STK4155

OLIVER ISTAD FUNCH

November 20, 2021

Abstract

Two problems are assessed; regression and classification. Several models are tested within each, OLS, Ridge, SGDregression, SGDclassification and Neural Networks. Grid-search with cross validation was used throughout the analysis for parameter tuning. For regression Neural Network performed best, whereas for classification, SGDClassifier performed best.

1 Introduction

The main objective of this project has been to study different stochastic gradient (SGD) regression and classification methods. Herein understand the effects of the different hyper parameters and how they can be tuned to cater for the specific problem. As such, the code used to create the different models are developed by the author.

Two main problems are addressed. Regression using Norwegian terrain data, and classification using UCI ML Breast Cancer Wisconsin datasets. Three methods will be tested for the regression problem; Linear regression using matrix inversion, linear regression using SGD optimization and multi-layer perceptron(MLP). Two methods will be tested for the classification problem; Logistic regression using SGD optimization and MLP. All methods are studied and compared in the context of the problem.

In this document, first a background will be given of the two data sets used in regression and classification. Afterwards an overview of the code developed and it's structure is given. Furthermore, a theoretical foundation for the methods will be given as well as an explanation on how the models are evaluated. The results for the regression problem will then first be presented, after which the results of the classification problem will be presented. In the discussion, the methods used in regression and classification will be discussed both separately and joined. Finally a conclusion on the results and preferred model for the different problems will be given.

Please note that the files found on the author's github is complementary to this report. Especially those found under *notebooks/reports* which are meant to give the reader a deeper insight into the analysis conducted. These notebooks aims at answering the specific tasks given in this project.

2 Background

2.1 Datasets

2.1.1 Terrain Data

For the regression problem a public terrain dataset is used. The dataset has been made available through the course's github[link] as a TIF file. The 'image' contains X amount of datapoints, where x and y are the coordinates and the value, z is the height at the corresponding coordinates. For linear regression and linear regression with SGD, a design matrix of polynomials of x and y up to degree 5 is used. Here there are X amount of features and z is then the target data. When using MLP the design matrix only consist of x and y, i.e, two features, but the target data remains the same.

Note that to reveal some of the interesting effects of different parameters, the dataset is downsampled by a factor of 8. I.e, every 8th x and every 8th y are used as coordinates. Furthermore, the dataset is split into %80 train and 20% test data. In some of the cases, the train data is further split into 90% train and 10% validation data as to monitor the performance during training.

Figure 1 shows the plots of the downsamples data and train and test data.

2.1.2 Wisconsin Breast Cancer Data

For classification the publicly available data set Wisconsin Breast Cancer Data is used. The data set is made available through the Scikit learn package and is directly loaded into the script by the use of a line of code. It consist of two components, the features(i.e the design matrix) and the target data. The features are a collection of 30 different measurements and other characteristics of cell nuclei obtained from patients, and so there are 30 features. The target data are the classes of each of the samples. I.e Benign(1) or Malignant(0). As such, it is a binary classification problem where one sample contains 30 features and one class(benign or malignant).

Class	Num samples	Fraction
Benign	357	0.627
Malignant	212	0.373

Table 1: Caption

Table 1 shows the distribution of the classes in the data set.

2.2 Code

All results achieved in this article are produced by the code developed by the author and can be found on the author's github repository[https://github.com/oliveristadfunch/FYS-STK4155-project2].

There is a readme.md file explaining both the structure of the repository as well as the purpose of each component. The project is structured like a self-written python package, where the folder *src/* contains all the modules and sub-modules developed for this project. It is in this folder all the code for creating the models are found.

Another folder to notice is *notebooks/reports.* Here different jupyter notebook files can be found. These notebooks aims at answering the tasks a) through e) more explicitly. As such, for a more detailed walk-through of the results obtained, refer the aforementioned notebooks.

Note that the code is extensively commented, such that no explanation of algorithms will be given here. All can be found in the code.

2.3 Parameter tuning

The analysis performed in this project, follows a certain pattern. Initially, parameters are studied and tweaked to understand their effects. After, a grid-search is conducted with cross-validation. Note that this grid-search uses the train data, and splits it further into train and test to find optimal parameters. After several such grid-searched, and optimal parameters are found. The model is then tested on the original test set. A final bench-mark is done using cross-validation on all the data with the found parameters.

2.4 Linear Regression

Note that in the literature on these topics it is more common to use y as a symbol for target data. However in this project, x and y are used as coordinates, and z is used as target data. As such, to avoid confusion, z is used as symbol for target data throughout this article.

Linear regression is used to fit the terrain data extracted. Here both Ordinary Least Squares(OLS) as well as Ridge is used. The aim is to minimize the cost function

$$\boldsymbol{C}_{OLS}(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{2n}\{(\boldsymbol{z}-\boldsymbol{X\beta})^T(\boldsymbol{z}-\boldsymbol{X\beta})\} \quad (1)$$

for OLS and

$$\boldsymbol{C}_{Ridge}(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{2n}\left(\{(\boldsymbol{z}-\boldsymbol{X\beta})^T(\boldsymbol{z}-\boldsymbol{X\beta})\} + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}\right) \quad (2)$$

for Ridge. Here $\boldsymbol{X}$ is the design matrix, $\boldsymbol{z}$ is the target data, $\boldsymbol{\beta}$ are the parameters of the model, λ is the regularization parameter and n is the number of samples. Note that the whole cost function is divided by $2n$, as opposed to the cost function derived from the maximum likelihood and MSE where division of two is omitted. This version is preferred as 2 will be canceled out when calculating the derivative yielding a cleaner gradient for SGD methods later. An additional note is that the regularization term is also divided by number of samples. This is because when used later in SGD the λ is more comparable across different batch sizes. It is used here as well for consistency. In this case, when calculating the derivatives with regards to the parameters

$$\frac{\partial C_{OLS}(\boldsymbol{X}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = \boldsymbol{X}^T \left(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta}\right) \tag{3}$$

$$\frac{\partial C_{Ridge}(\boldsymbol{X}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = \boldsymbol{X}^T \left(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta}\right) + \lambda\boldsymbol{\beta} \tag{4}$$

the $2n$ is canceled out anyways. Rearranging these derivatives gives the analytical expressions for the optimal β values

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{z} \tag{5}$$

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{z} \tag{6}$$

2.5 Stochastic Gradient Descent

When performing linear regression as described above, one has to perform matrix inversions to obtain optimal parameters. With large amount of data, this might be a computationally heavy load. Gradient Descent(GD), Stochastic Gradient Descent or Mini-batch Stochastic Gradient Descent can be used as a solver instead of matrix inversion. They differ primarily in how and what data they use during optimization. In this project Mini-batch Stochastic Gradient Descent, hereafter referred to as Stochastic Gradient Descent(SGD), is used. The method is often employed when the data set is large. In this article, SGD regression and normal regression are compared even though the data set is not particularly large.

GD is a numerical optimization method that finds parameters for a function corresponding to a local minima of the function in an iterative manner. For instance, GD can be used to find $\boldsymbol{\beta}$ values that corresponds to a local minima of either of the cost functions (1) or (2). The method updates the parameters by subtracting the gradient of the function from the previously defined parameters, as such

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta\nabla_\beta \boldsymbol{C}(\boldsymbol{X}, \boldsymbol{\beta}_k),\ k = 0, 1, \cdots \tag{7}$$

where η is the learning rate and $\nabla_\beta \boldsymbol{C}(\boldsymbol{X}, \boldsymbol{\beta}_k)$ is the gradient of the cost function. The learning rate decides the amount of the gradient that is to be subtracted from the parameters. It is a value between 0 and 1. As such, a low learning rate causes the optimization to converge to local minima slower. Note that a too high learning rate may cause the optimization to diverge, i.e overshoot the minima. As such, SGD guarantees convergence to local minima given small enough learning rate and enough iterations. The iterations are referred to as *epochs*, and it denotes the amount of times the optimizer runs through the whole data set. Note that GD uses the entire dataset $\boldsymbol{X}$ and $\boldsymbol{z}$ at each iteration. As such, it calculates the gradient across all the data at each iteration which can be computationally heavy.

SGD addresses this by calculating the gradient of a batch of data at a time. The stochasticity is introduced by picking a random batch at each iteration. As such, it may use some batches several times and leave some out. Note that an epochs is still the amount of times the optimizer has used n samples of the data. As such, with n_b as the amount of samples in each batch, one epoch consist of

$$N_b = \left\lceil \frac{n}{n_b} \right\rceil \tag{8}$$

batches. If not divisible, N_b is rounded up, and the last batch will be smaller and have the size ($n \bmod n_b$).

For OLS and Ridge the updates becomes, respectively

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta\frac{1}{n_b}\boldsymbol{X}_b^T(\boldsymbol{X}_b\boldsymbol{\beta}_k - \mathbf{z_b}) \tag{9}$$

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \eta\frac{1}{n_b}\left(\boldsymbol{X}_b^T(\boldsymbol{X}_b\boldsymbol{\beta}_k - \mathbf{z_b}) + \lambda\boldsymbol{\beta}_k\right) \tag{10}$$

where n_b is the number of samples in current batch, $\boldsymbol{X}_b$ is the chosen batch and $\boldsymbol{z}_b$ is the corresponding target data. Here $\frac{1}{n_b}$ is kept in the gradient as the batch size may differ. This also makes the employed learning rate comparable across different batch sizes. One could always change the learning rate η according to the batch size. For instance, results with $\frac{\eta}{n_b 1}$ would be comparable to $\frac{\eta}{n_b 2}$. However, dividing by n_b allows for easier analysis of learning rate across batch sizes. Note that the regularization term is divided by n_b, also for the same argument as with learning rate.

The second derivatives of the cost functions

$$\frac{\partial^2 C_{OLS}(\boldsymbol{X}\boldsymbol{\beta})}{\partial\boldsymbol{\beta}^T\partial\boldsymbol{\beta}} = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X}, \tag{11}$$

$$\frac{\partial^2 C_{Ridge}(\boldsymbol{X\beta})}{\partial \boldsymbol{\beta}^T \partial \boldsymbol{\beta}} = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X} + \frac{\lambda}{n}. \tag{12}$$

are always non-negative. As such, both cost functions are convex, which in turn means that every local minima is a global minima. In other words, using SGD with low enough learning rate, so that

$$\boldsymbol{C}(\boldsymbol{\beta}_{k+1}) \leq \boldsymbol{C}(\boldsymbol{\beta}_k) \tag{13}$$

is always true, the optimization is guaranteed to converge to a global minima(i.e analytical solution) given enough epochs.

Note that for added stochasticity, and to avoid to fit the model to any spurious correlation as a result of the order of the data, the data set is shuffled between before every epoch.

2.5.1 Momentum

A common challenge with SGD optimization is the sometimes slow convergence. Momentum is a technique used to speed up SGD thereby requiring fewer epochs for convergence. Momentum is used in all SGD based models in this project.

The method involves capturing the so-called *speed*, $\mathbf{v}_k$ at each update k.

$$\mathbf{v}_k = \gamma \mathbf{v}_{k-1} + \eta \nabla_\beta \boldsymbol{C}(\boldsymbol{X}, \boldsymbol{\beta}_k). \tag{14}$$

Here the $\mathbf{v}_{k-1}$ is the speed at the previous iteration, and must be set to zero for the first iteration. γ is the momentum parameter, i.e the amount of the former speed that is added to the new speed as well. γ has a value between 0 and 1.

The parameter update is then

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k - \mathbf{v}_k \tag{15}$$

From this it can be understood that in essence a running average is calculated, though where the former average is scaled down according to γ. This in turn causes the parameter updates to be less noisy as some of the update from the previous update is also a part of the current update. As such, consistent changes in gradient has more impact on the course of the parameters than noisy gradients. Consistent changes to the gradient should then indicate a direction worth following as opposed to sudden crevices or hills.

2.5.2 Learning Schedule

The result of an SGD optimization is often highly sensitive to the chosen learning rate. First and foremost, the learning rate needs to be low enough for convergence. Too low may cause it to require too many epochs for convergence. Too high(though still low enough for convergence) may cause it to overshoot the solution somewhat, especially when is close to the solution. This may also result in a larger required amount of epochs as the optimizer bounces around the optimal solution.

In the beginning of an SGD optimization, it is often safe to take rather big steps, however closer to the end, one should go more careful about it. As such, scaling the learning rate on a schedule based on number of epochs and batches elapsed may be beneficial. There are several ways of scheduling the learning rate, in this project a standard schedule according to

$$\eta_{kb} = \frac{t_0}{t - t_1}, \tag{16}$$

is used. Here η_{kb} is the learning rate for the current batch in the current epoch, t_0 and t_1 are constants, and t is calculated at each batch as

$$t = kN_b + b. \tag{17}$$

Here k is the current epoch, N_b is the number of batches and b is the current batch. t always increases as the optimizer elapses the epochs, such that η_{bk} will become smaller and smaller.

This method will be tested, however a constant learning rate is mostly used in this project.

2.5.3 The importance of scaling the data

Standardizing the data, and especially the features is important when employing SGD. The reason is that standardizing the features causes the landscape to be homogeneous in all directions, i.e no sudden crevice or hill. This results in a less noisy gradient much like momentum, however by changing the landscape instead. Fewer epochs are then needed for convergence. The sklearn's StandardScaler will be used in this project.
image

Figure [ref] shows an illustration of this effect. Standardization causes the error ellipsis(the contour lines of the solution space) to be more circular. Lines that lie close together in a contour map indicate steeper hills.

The target data z will in the terrain case also be scaled using standardization. The argument here is that when used later in neural networks, large initial gradients may cause instabilities when training the network.

2.6 Linear Regression with SGD

Performing linear regression with SGD is a matter of defining the cost function either as (1) or (2) as exemplified in 2.5. In this project both cost functions are tested and compared to OLS and Ridge.

Note that the SGD optimizer might not need to fully converge to reach a good solution. It may as well be that a non-converged solution performs better at the test data than a converged one, as it may be better at generalizing. Monitoring the performance on a separate validation set during training might reveal this.

Furthermore, it is not necessarily the same λ for SGD and Ridge that yields the optimal results individually. The results are often highly sensitive to the combination of different parameters. For instance a certain η might result in another optimal λ and vice-versa. Therefore, a grid search should be conducted to benchmark different combinations.

When performing a grid search of λ and η, a range of λ values are tested with a range of η values. In other words, $n_{\lambda values} \times n_{\eta values}$ models are trained using every combination of the λ and η values. Each model is trained and scored using cross validation for proper bench-marking. As such, the optimal combination of two parameters can be determined.

2.7 Logistic Regression with SGD

Logistic regression is a model used for classification. It is similar to linear regression in that the goal is to find $\boldsymbol{\beta}$ that minimizes a cost function. With a logistic regression model, the goal is to predict probabilities of a sample being in different classes. For instance, with Breast Cancer Data, the features would be the input, while the output would be probabilities of benign and malignant. Note that for a final classification one can classify every probability above 0.5 as 1 and below as 0. If z_i is the class [0,1] for the given sample, then

$$z_i = \begin{cases} 1, & \text{if } p(z_i = 1|x_i, \boldsymbol{\beta}) > 0.5 \\ 0, & \text{if } p(z_i = 1|x_i, \boldsymbol{\beta}) < 0.5 \end{cases} \tag{18}$$

where x_i are the sample's features. The cost function is different from linear regression. Since every output should be between 0 and 1, simply outputting

$$f(\boldsymbol{x}_i, \boldsymbol{\beta}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} \cdots \beta_n x_{in}, \tag{19}$$

would not give a probability. Here n is the number of features. By further processing this result through the Sigmoid function

$$p(z_i = 1|x_i, \boldsymbol{\beta}) = \frac{\exp\left(f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)}{1 + \exp\left(f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)} \tag{20}$$

$$p(z_i = 0|x_i, \boldsymbol{\beta}) = 1 - p(z_i = 1|x_i, \boldsymbol{\beta}) \tag{21}$$

one forces the values between 0 and 1. The optimal $\boldsymbol{\beta}$ should then contain values that maximizes the correctness of above guess. Or in other words, that maximizes the likelihood function

$$P(\mathcal{D}|\boldsymbol{\beta}) = \prod_{i=1}^{n} \left[p(z_i = 1|x_i, \boldsymbol{\beta})\right]^{z_i} \left[1 - p(z_i = 1|x_i, \boldsymbol{\beta}))\right]^{1-z_i} \tag{22}$$

which results in the cost function

$$\mathcal{C}(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \left(z_i(f(\boldsymbol{x}_i, \boldsymbol{\beta})) - \log\left(1 + \exp\left(f(\boldsymbol{x}_i, \boldsymbol{\beta})\right)\right)\right) \tag{23}$$

and the derivative

$$\frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{X}^T \left(\boldsymbol{p} - \boldsymbol{z}\right). \tag{24}$$

Using (24) as the cost function with SGD optimization one can find $\boldsymbol{\beta}$ that minimizes the cost function (23). Note that the gradient is equal to (3) except with a method for outputting predictions, $\boldsymbol{p}$. As such, same algorithms can be used when calculating the gradient.

The metric used for bench-marking classification models is commonly accuracy, and will be explained later.

2.8 Multi-layer perceptron

A multi-layer perceptron(MLP), also called a Neural Network is a computing system able to both model regression and classification problems depending on it's configuration. With a linear non-restricted output the MLP is a regression model, whereas with an output between 0 and 1 is a classification model. Much like logistic regression, one can feed the last output through a Sigmoid function to obtain probabilities. Both models will be tested on both problems in this project.

An MLP consist of an input layer, hidden layers and an output layer. With one hidden layer, the MLP is said to have three layers. More hidden layers can be added. Every hidden layer contains a customizable amount of neurons. Every neuron in one layer is connected to every neuron in the layers on both sides. It is therefore fully connected. The constellation of layers and nodes is called network architecture. Several network architectures has been tested in this project.

When a network predicts on input data, it uses *feed-forward* to pass the values through the layers, and finally outputs a prediction.

The value of neuron n_j is the weighted sum of the values of all connected neurons passed through an activation function,

$$a_j = f\left(\sum_{i=1}^{n} w_{ij} a_i + b_j\right) = f(z_j). \tag{25}$$

Here a_i is the value of neuron n_i, w_{ij} is the weight between them, b_j is the bias of neuron n_j, and f is the activation function. There are several applicable activation functions, some of which will be discussed later. It is the weights and biases that are learned when training a network. For the first layer, a_i is simply replaced by the input data x_i.

In the case of terrain data, x_1 and x_2 is the xy-coordinate of a sample. A trained network may then reconstruct the terrain surface given all the coordinates. In this case, the output layer has one node, which also holds the output of the network, namely z.

In the case of Wisconsin Breast Cancer data, the network is given $x_1, x_2 \cdots x_{30}$ as input. In this case, the output layer also has one node, however, it's value is further activated through the Sigmoid function to produce output probability.

2.8.1 Activation Functions

There are several options for activation function in the hidden layers. A requirement is that it is non-linear. The classic Sigmoid

$$f(z) = \frac{1}{1 + \exp(-z)} \tag{26}$$

has been much used, however Rectified Linear Unit(ReLU)

$$f(z) = \begin{cases} z, & \text{if } z > 0 \\ 0, & \text{if } z < 0 \end{cases} \tag{27}$$

and Leaky Rectified Linear Unit(LeakyReLU)

$$f(z) = \begin{cases} z, & \text{if } z > 0 \\ \alpha z, & \text{if } z < 0 \end{cases} \tag{28}$$

has gotten some attention lately much due to their inability to saturate for positive values, thereby contributing in the mitigation of *vanishing gradient problem.* All three will be tested and compared in both problems in this project. Note that the output activation function is dependent on the problem. For regression, it should be the identity activation function

$$f(z) = z \tag{29}$$

and Sigmoid (26) for binary classification.

2.8.2 Backpropagation

A network is trained in much of the same way as Linear and Logistic regression using SGD. A cost function is defined, and parameters are found that minimizes the cost function. As such, gradients must be determined for every single weight and bias in the network.

The backpropagation algorithm uses the error in the output layer and distributes it to the preceding layers. The error in a specific layer is quite intuitively it's contribution to the error in the layer after it. The error in this layer, is in turn it's contribution to the error in the next, and so forth. The error in the last hidden layer is it's contribution to the output error. As such, by calculating the output error, on can determine every layer's contribution to the error through backpropagation.

With a general definition of a cost function as

$$\mathcal{C}(\theta) = \frac{1}{N}\left(\sum_{i=1}^{N}\mathcal{L}_i(\theta) + \lambda||\hat{w}||_2^2\right) \tag{30}$$

where θ are the parameters, herein both weights and biases. Note that the regularizing term is divided by N here as well, for the same reasons as previously explained.

Using matrix operations, several input samples can be process simultaneously. The output error is simply calculated as

$$\delta_L = \hat{z} - z = (n_{inputs}, n_{categories}) \tag{31}$$

where $\hat{z}$ is the output of the network and z is the target data. Note that $n_{categories}$ is 1 in both problems addressed in this project.

The gradients of the weights and biases of the output are then

$$\nabla W_L = \frac{1}{n_{inputs}}\hat{a}^T\delta_L = (n_{hidden}, n_{categories}) \tag{32}$$

$$\nabla \hat{b}_L = \frac{1}{n_{inputs}}\sum_{i=1}^{n_{inputs}}\delta_L = (n_{categories}) \tag{33}$$

Note that the division of n_{inputs} are kept in the gradient for the same reasons as described in 2.5.

The error in the last hidden layer is then

$$\delta_h = \delta_L W_L^T \circ f'(z_h) = (n_{inputs}, n_{hidden}) \tag{34}$$

which then yields the gradients

$$\nabla W_h = X^T\delta_h = (n_{features}, n_{hidden}) \tag{35}$$

$$\nabla b_h = \sum_{i=1}^{n_{inputs}}\delta_h = (n_{hidden}) \tag{36}$$

This easily extended to the rest of the hidden layers by replacing error and weight those from the last hidden layer.

Lastly, using mini-batches and momentum, the weights and biases are update such

$$w_{jk}^l \leftarrow w_{jk}^l - \eta\frac{1}{n_b}\delta_j^l a_k^{l-1} + \gamma\Delta w_{ij}^{t-1} \tag{37}$$

$$b_j^l \leftarrow b_j^l - \eta\delta_j^l \tag{38}$$

where $\gamma\Delta w_{ij}^{t-1}$ is the previous updates to the weights. This is equivalent to including the previous speed in the calculation of the next, and using the speed to update the parameters (14).

2.8.3 Initialization

For the network to learn, the weights and biases needs to be initializes to something else than 0. A previously used method has been to randomly distribute according to a normal distribution with mean 0 and variance 1. However, this often suffers from vanishing gradients. It has been recently discovered that initializing weights with random values uniformly distributed between $\pm\frac{6}{\sqrt{n_{features}+n_{neurons}}}$ [link]. This helps ensure similar variance of output as input and thereby helps mitigate the vanishing/exploding gradient problem by ensuring that the signal flows fully both ways. The method is often referred to as glorot from the author.

The bias is simply initialized to a constant value of 0.001 as to not be zero.

2.9 Accuracy

Accuracy is the most common bench-mark metric for a classification model. It will be used for all classification models used in this project. It describes the fraction of correct predictions. It is given by

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(\hat{z}_i = z_i)}{n} \tag{39}$$

where I is the indicator function, which is equivalent to a boolean test. It gives 1 if the statement inside is correct, otherwise 0.

3 Results

This section contains the results of the analysis conducted using the above described methods. First, regression results will be assessed, after which the classification results will be presented. In each problem results for all models used will be presented and compared. Note that all scores and MSE listed in the results have been produced using 5-fold cross validation.

For some of the intermediate results, the reader is referred to the authors github for more details.

3.1 Regression

3.1.1 Linear Regression

New results for OLS and Ridge has been generated as a more sparse data set is used in this project.

From a bias variance analysis it was concluded that a polynomial degree of 5 suited both OLS and Ridge best. Figure 2 and 3 shows the result of the bias-variance analysis as well as cross validation.

From an analysis of the parameter λ show in Figure 4 it was found that = 0.01 yielded the lowest loss for Ridge.

Finally using above reached parameters, both OLS and Ridge was cross-validated, with a score as follows:

	MSE		R2	
	Train	Test	Train	test
OLS	0.097	0.138	0.903	0.8497
Ridge	0.114	0.1319	0.8858	0.8539

Table 2: Linear regression results

OLS does achieve a better train score, however Ridge scores better on the test score. This may indicate that Ridge is better at generalizing, and that perhaps OLS is slightly overfit. Figure 5 shows the produced surfaces from the linear regression models.

3.1.2 Linear Regression using SGD

The linear regression problems are now solved using SGD optimizer. For the initial test following model parameters where used.

Parameter	value
batch_size	64
n_epochs	100
lmb	0.01
lr0	0.01
momentum	0
learning schedule	'constant'

Table 3: Model parameters for initial SGD regression model

The learning curve of the result 6 shows that the model is not about to overfit, as the validation loss and score is still decreasing. It is clear that with increased epochs, the loss decreases. However, if the model is too complex, the validation loss might increase with too many epochs.

The batch size may also have an impact on the result. Figure 7 shows how MSE increases with increasing batch size. An increased batch size likely requires more epochs to converge as it uses a larger portion of the data to calculate an "average" gradient. The gradient is therefore more stable towards the global solution, however requires more epochs. Note that a very small batch size leads to slow running times. Because of vectorization techniques performing matrix operations is more efficient than calculating gradients for each and every sample.

By increasing the learning rate the optimizer takes a "bigger" step at every iteration. Changing this parameter will thereby have an effect on the result. Figure 8 illustrates the effect of learning rate. The learning curve indicates that the optimizer overshoots regularly but is able to reel itself back in. This can also be seen in the MSE vs. Learning rate plot. Too low learning rate also produces a poor fit, as the optimizer has not been able to train a substantial amount.

When training an SGD model, it is often possible to take larger steps in the beginning without diverging. Later on, a smaller learning rate is more sensible, as overshooting the target might lead to unstable convergence. Applying a learning schedule aims at adapting the learning rate during training. The schedule (16) is used, and the resulting learning curve can be seen in Figure 9. The scores stagnates heavily indicating that the learning rate perhaps has decayed too much. The validation loss has stagnated as well, but the train loss is still fluctuating somewhat. Note that train loss is accumulated during training, and then averaged after an epoch. The validation is calculated after each epoch.

Furthermore, a grid-search was performed to identify the best combination of λ and η. A heatmap showing the cross validated scores with the different parameters are shown in Figure 10. From the grid-search, it was found that = 0.001 and $\eta = 0.001$ yielded the best results.

OLS, OLS SGD, Ridge, Ridge SGD and Sklearn SGD where cross-validated with model parameters found in Table 16. The results are

	MSE		R2	
	Train	Test	Train	test
OLS	0.097	0.136	0.903	0.850
OLS SGD	0.141	0.170	0.859	0.8249
Ridge	0.112	0.148	0.888	0.852
Ridge SGD	0.160	0.179	0.840	0.809
Sklearn SGD	0.134	0.153	0.866	0.840

Table 4: Results of different models both using matrix inversion and SGD

None of the SGD methods is able to reach the regular linear regression methods in terms of performance. Ridge and OLS still outperforms the others, and Ridge still has the highest test score. The Sklearn's SGD does obtain a better score than own developed models. Figure 11 shows surfaces produced by the different models. Note that both train and test data is concatenated and plotted here.

3.1.3 MLP Regression

An MLP is now fitted to the terrain data. In this case the activation function of the output layer needs to be the "identity" activation function (29). The parameters for the initial model tested are as follows

Parameter	value
batch_size	64
n_epochs	1000
hidden_layer_sizes	(50,)
w_init	'normal'
hidden_activation	'sigmoid'
lmb	0.01
lr0	0.01
momentum	0.9
learning schedule	'constant'

Table 5: Model parameters for initial SGD regression model

Here hidden_layer_sizes denotes the model architecture. A tuple is used, such that the amount of values in the tuple is the number of hidden layers, while the values them selves are the number of neurons. (50,100) denotes two hidden layers, one with 50 neurons and the next with 100. w_init specifies the initialization scheme, while hidden_activation is the activation function used in the hidden layers.

The resulting learning curve can be seen in Figure 12. The loss starts out relatively high, but is able to converge.

Changing the initialization to glorot, it is clear that the model starts off much better. As seen in Figure 13, the initial loss is much lower than when using normal initializer.

A grid-search of λ and η was conducted to find the optimal combination of the parameters. Figure 14 shows the result with $= 0.01$ and $\eta = 0.05$ as the optimal parameters. As can be seen, increasing learning rate much more will result in divergence. A too low one causes a poor score. To some extent it seems like with lr=0.1 that the score increases with increasing lmb. However, after lmb=0.01 the score starts decreasing. This may indicate too much regularization. In general it does not seem like the model needs a lot of regulairzation, or that it has much effect at all.

Further more, a grid-search the model architecture and batch size is also conducted. Here different architectures are defined and tested with all defined batch sizes. The results can be seen in Figure 15 with hidden_layer_sizes = (100,50) and batch size = 128 as optimal parameters. However, as (50,50) with batch size=128 is very close, and is a simpler architecture, it is chosen instead going forward.

The cross-validated results of the model reached so far is

	MSE		R2	
	Train	Test	Train	test
NN	0.076	0.107	0.924	0.888
OLS	0.097	0.136	0.903	0.850

Table 6: Results of different models both using matrix inversion and SGD

and actually outperforms the OLS quite a lot.

A last grid-search is conducted to assess which activation function together with initialization scheme yields the best result. Figure 17 shows the result with 'glorot' and ReLU as the optimal combination. Note that it is only Glorot that is able to allow ReLU and LeakyReLU. With the final model parameters being

Parameter	value
batch_size	128
n_epochs	1000
hidden_layer_sizes	(50,50)
w_init	'glorot'
hidden_activation	'relu'
lmb	0.01
lr0	0.05
momentum	0.9
learning schedule	'constant'

Table 7: Model parameters for initial SGD regression model

the results where as follows:

	MSE		R2	
	Train	Test	Train	test
NN	0.071	0.110	0.929	0.888
OLS	0.097	0.136	0.903	0.850

Table 8: Results of different models both using matrix inversion and SGD

showing that the neural network does outperform OLS in reproducing the surface. The full reproduced surface can be seen in Figure

3.1.4 Comparison

All results obtained in the regression problem can be seen in Table 14, ranked by test score.

	MSE		R2	
	Train	Test	Train	Test
NN	0.071	0.110	0.929	0.888
Ridge	0.112	0.148	0.888	0.852
OLS	0.097	0.136	0.903	0.850
Sklearn SGD	0.134	0.153	0.866	0.840
OLS SGD	0.141	0.170	0.859	0.8249
Ridge SGD	0.160	0.179	0.840	0.809

Table 9: Results of different models both using matrix inversion and SGD

The neural network obtains the best scores, both for train and test.

3.2 Classification

3.2.1 MLP Classifier

By changing the output activation to Sigmoid(26 the MLP can be turned into a classifier. The model is now trained using the Wisconsin Breast Cancer data. Parameters for the initial model are those in Table 5.

An immediate grid-search on λ and η is conducted. With 'normal' as initialization scheme, the results are $\lambda = 0.05$ and $\eta = 0.05$ as optimal parameters. However with an additional grid-search on λ and η, now with 'glorot' and ReLU, it was found that $\lambda = 0.0005$ and $\eta = 0.005$ gave better results. These will be used further.

Two more grid-searches was conducted, first architecture vs. batch size, then activation function vs initialization scheme. All heatmaps resulting from grid-searches can be seen in Figure 18. Note that most configuration seem to give high scores.

With the final optimal parameters

Parameter	value
batch_size	16
n_epochs	100
hidden_layer_sizes	(100,100)
w_init	'glorot'
hidden_activation	'relu'
lmb	0.0005
lr0	0.005
momentum	0.9
learning schedule	'constant'

Table 10: Final optimal parameters for MLP classifier

the cross-validated result

	Accuracy	
	Train	Test
NN	0.998	0.975
NN sklearn	0.985	0.972

Table 11: Results of different models both using matrix inversion and SGD

Own developed neural network achieves slightly better score than sklearn NN. With such good results for near all configurations it is expected to achieve good results with logistic regression as well.

3.2.2 Logistic Regression

The Wisconsin Breast Cancer data is now modeled using Logistic Regression. With initial parameters

Parameter	value
batch_size	32
n_epochs	100
lmb	0.001
lr0	0.01
momentum	0.9
learning schedule	'constant'

Table 12: Initial parameters for Logistic Regression with SGD

resulting learning curve and confusion matrix can be seen in Figure 19. With train and test accuracy of 0.98 and 0.99 respectively, it is clear that logistic regression is a good candidate. Figure 20 shows that too many epochs can result in an increased validation score.

Conducting a grid-search(Figure **??** on λ and η, the optimal parameters where found to be

Parameter	value
batch_size	32
n_epochs	100
lmb	0.001
lr0	0.005
momentum	0.9
learning schedule	'constant'

Table 13: Initial parameters for Logistic Regression with SGD

Comparing to sklearn's SGDClassifier and LogisticRegression, the results are

	Accuracy	
	Train	Test
LogReg	0.99	0.970
SGDClassifier	0.987	0.982
LogisticRegression	0.989	0.977

Table 14: Results of different models both using matrix inversion and SGD

The SGDClassifier seem to be achieving the best test scores. However, all are relatively equal. It seems clear that the classification problem is a rather simple problem.

3.2.3 Comparison

All classifiers are ranked in Table 15.

	Accuracy	
	Train	Test
SGDClassifier	0.987	0.982
LogisticRegression	0.989	0.977
NN	0.998	0.975
NN sklearn	0.985	0.972
LogReg	0.99	0.970

Table 15: Results of different models both using matrix inversion and SGD

As can be seen the SGDClassifier achieves the best result, however all achieves high scores.

4 Discussion

4.1 Regression

Before it was decided to downsample the terrain data, all models developed reached very good scores. Emulating a more sparse situation did show some of the interesting effects of different parameters. A somewhat surprising result was that the Neural Network was better than OLS and Ridge at reproducing the surface. Following the Universal Approximation Theorem, neural networks are able to approximate any continuous function with an arbitrary score, a neural network should indeed be able to reproduce the polynomial function employed by OLS and Ridge. In fact, there might be an even better function out there, which the Neural Network can approximate. As such, NN's should be more powerful than Linear Regression, however it is not always easy to find the right model parameters. In this case with relatively simple tuning, an NN was found that outperformed LinearRegression.

SGD regression is mostly used when there are a lot of samples such that matrix inversion is too computationally heavy. In this case, the data was sparse, and so it is not much of a surprise that it performs worse.

4.2 Classification

High scores were obtained immediately for all models testen on the classification problem. It may indicate that the problem is easy, and that it is a clear cut between all the features. As the Neural Network and Logistic Regression scored above 96% on nearly all configurations, it may be that one could simply see the class from the data, and a simple rule based threshold model would do the trick.

For the classification problem it is unnecessary to use neural networks when logistic regression performs better. Logistic Regression is also not a 'blackbox' as opposed to neural network. In other words, one can explain the model and the output by looking at the parameters. Perhaps that is more useful since it could provide doctors with more knowledge of the correlations.

In classification problems, it is often useful to look at the confusion matrix to see how the model performs. It is not always that the sole objective is to minimize accuracy. Sometimes it is more important to minimize false negatives or false positives for that matter. In this case, one could think that minimizing false positives is important, so that no malignant tumor is classified as benign. On the other hand, this might result in too many malignant cases making it unrealistic to validate all. A balance should be found depending on the problem.

4.3 Comparison of models

OLS and Ridge are much faster given the amount of data used in this project. However they are somewhat rigid as they will only yield the one optimal solution given the amount of features.

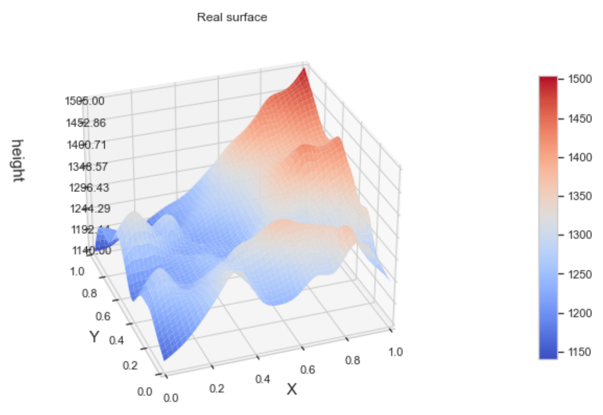
SGD based methods are flexible and can be tuned. These can also be used both as regressors and classifiers depending on the cost function. With the data used in this project, they are perhaps not the best choice for regressors, however with much more data they might be. As classifiers on the other hand, they seem to achieve good results. They also have fewer parameters to tune than neural networks, which makes them easier to work with.

Neural Networks are universal approximators, that potentially can estimate any continuous function. They can be useful in both regression and classification problem. A benefit with them is that they are extremely flexible with lots of parameters to tune. This can also be a bad side, that they can be hard to handle. One must always be on the lookout for vanishing/exploding gradients when using different configurations. Another con of NN is that it is a 'blackbox' meaning it is hard explain why a Neural Network does what it does, except for saying due to its trained weights.

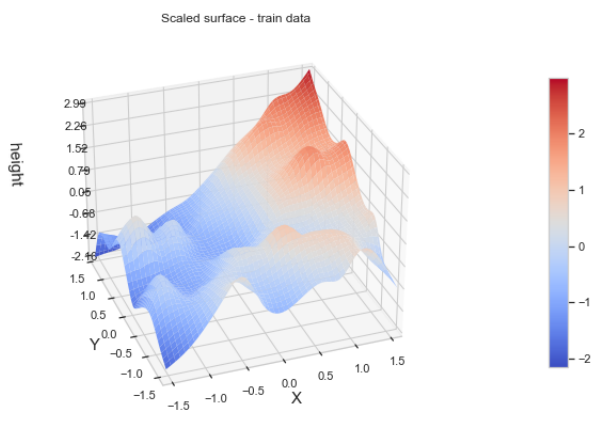
5 Conclusion

Several models have been tested on a regression and a classification problem. A nerual network was found to be the best at regression, while SGDClassifier performed best on the classification problem.

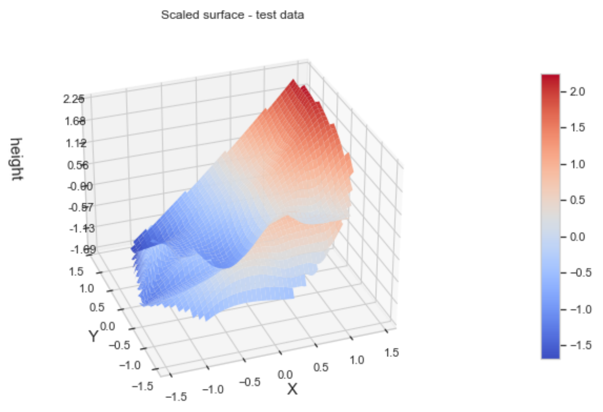
A Plots



(a) Surface



(b) Train surface



(c) Test surface

Figure 1: Terrain data

A.1 Regression

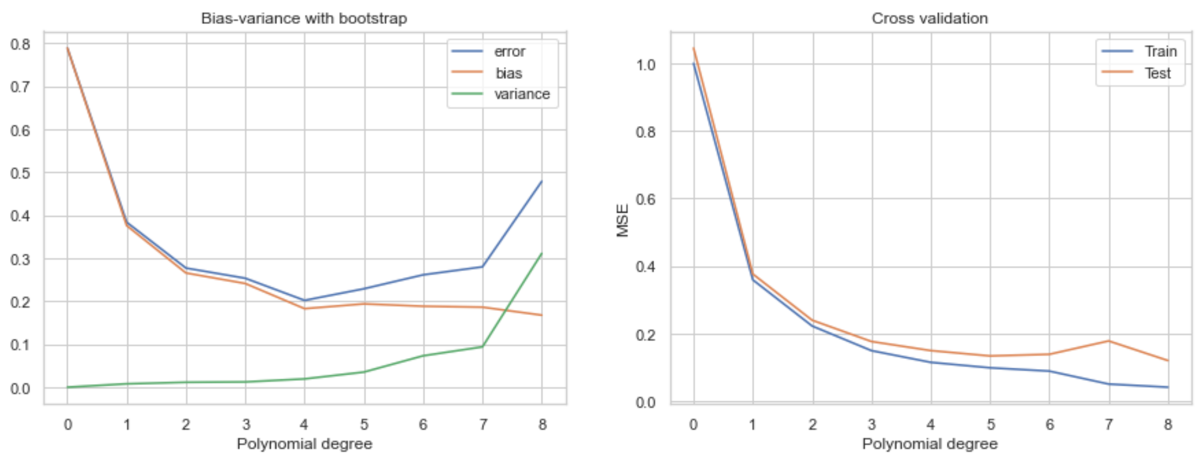


Figure 2: Bias-variance analysis shows a region of overfit after degree 7

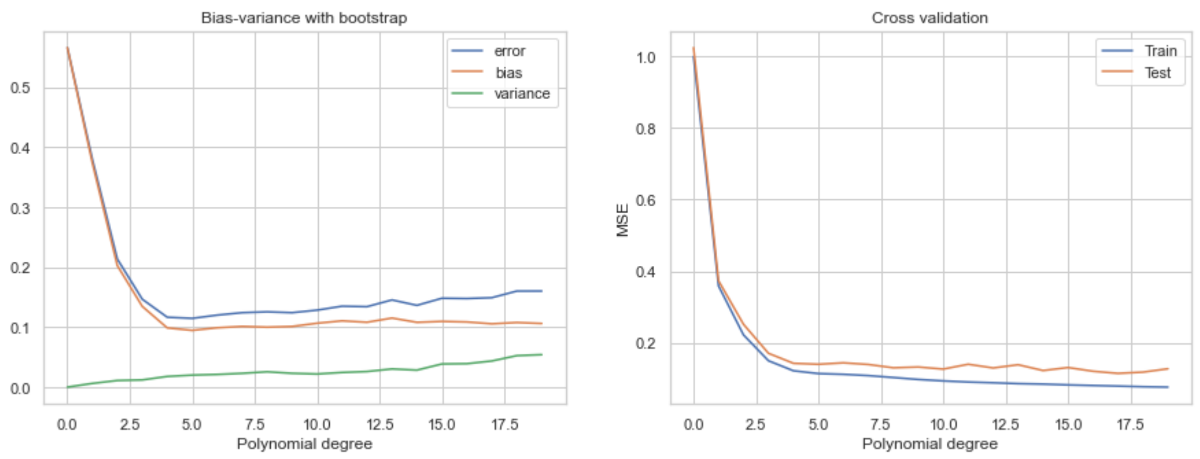


Figure 3: Bias-variance analysis shows that Ridge is more resistant to overfit

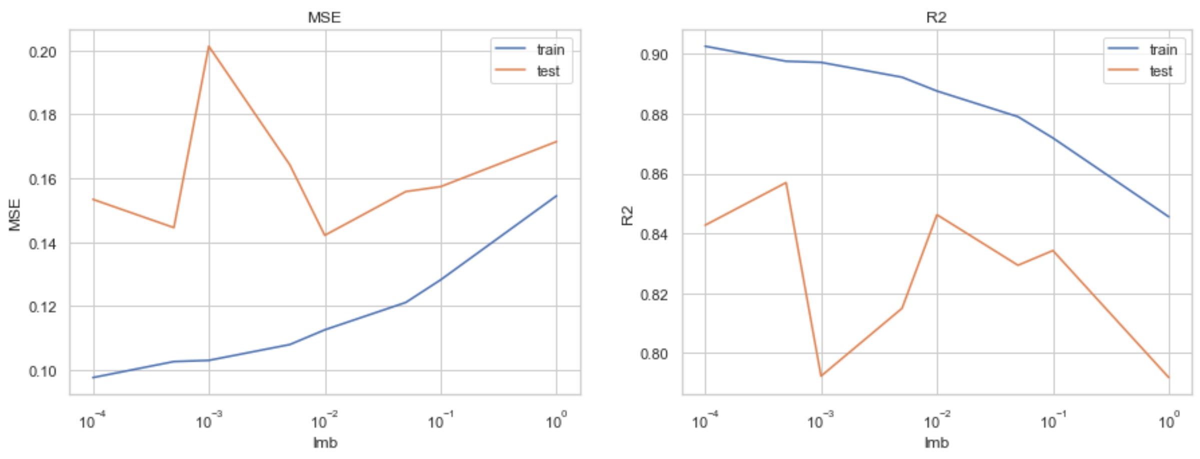
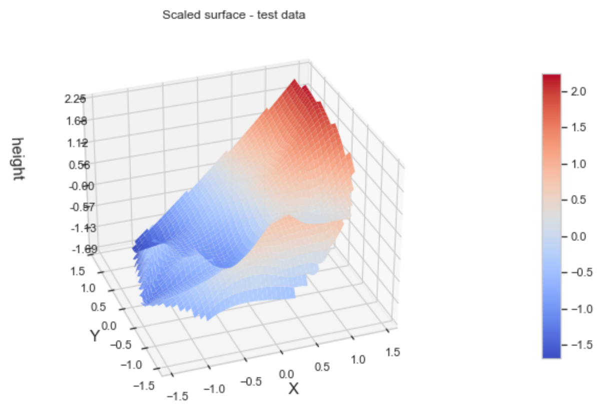
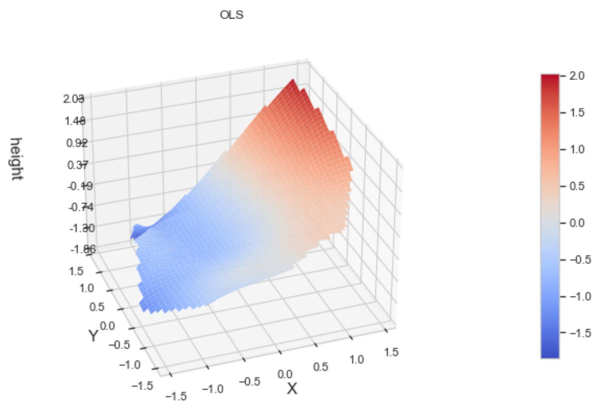


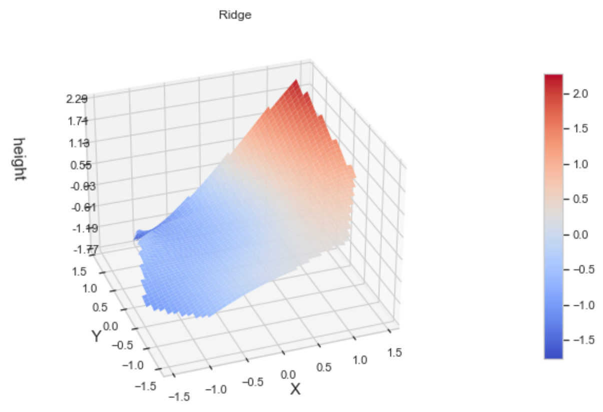
Figure 4: MSE and R2 as functions of λ. Plot shows that $\lambda = 0.01$ is optimal.



(a) Real test surface



(b) Test surface produced by OLS



(c) Test surface produced by Ridge

Figure 5: Terrain test data plots

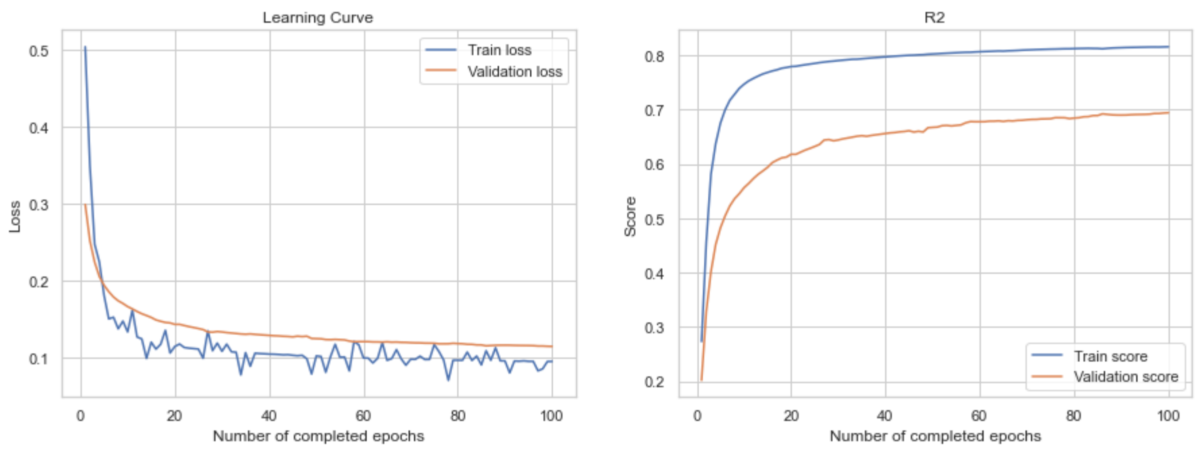
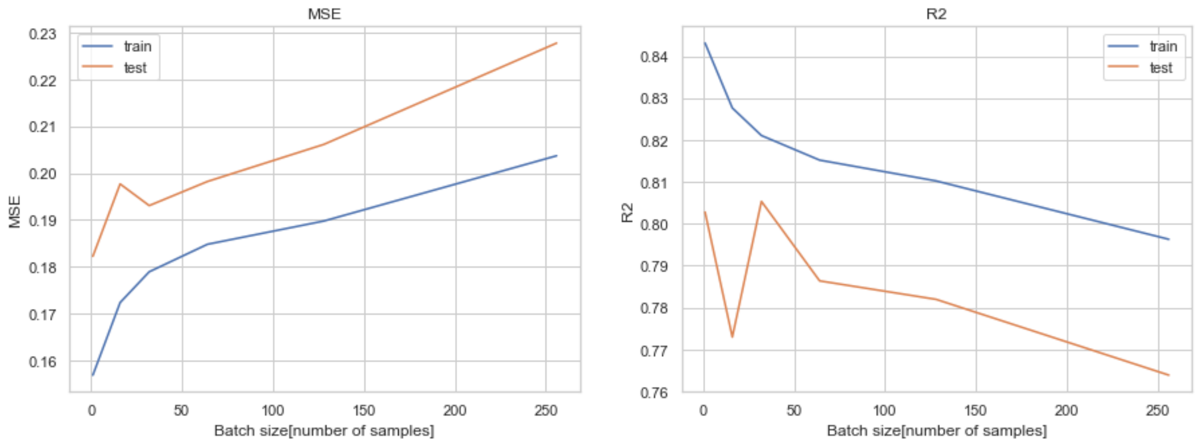
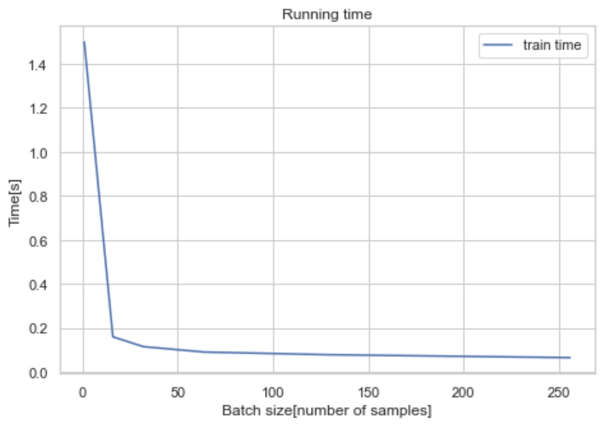


Figure 6: Learning curve showing loss and score during training of SGD regressor.

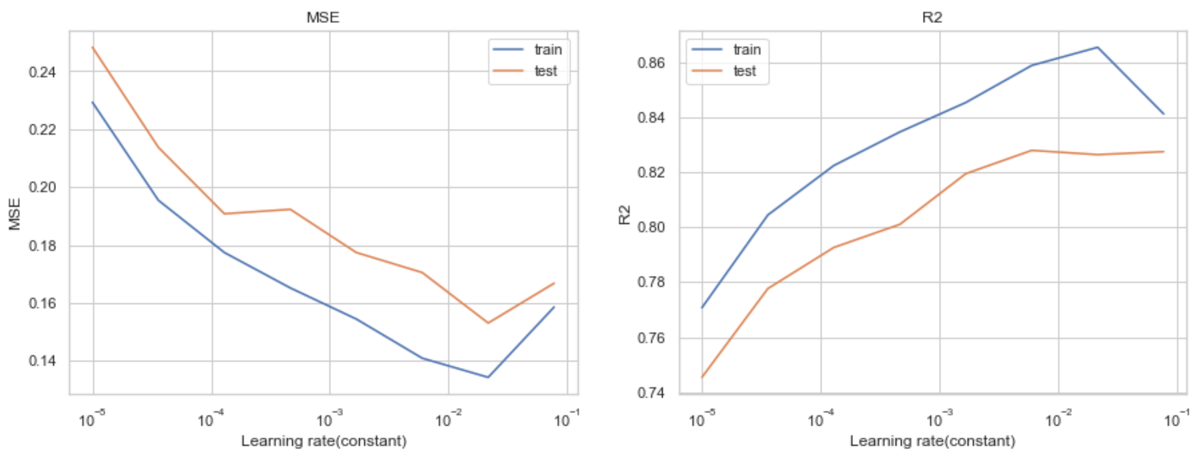


(a) The MSE and R2 as functions of batch size

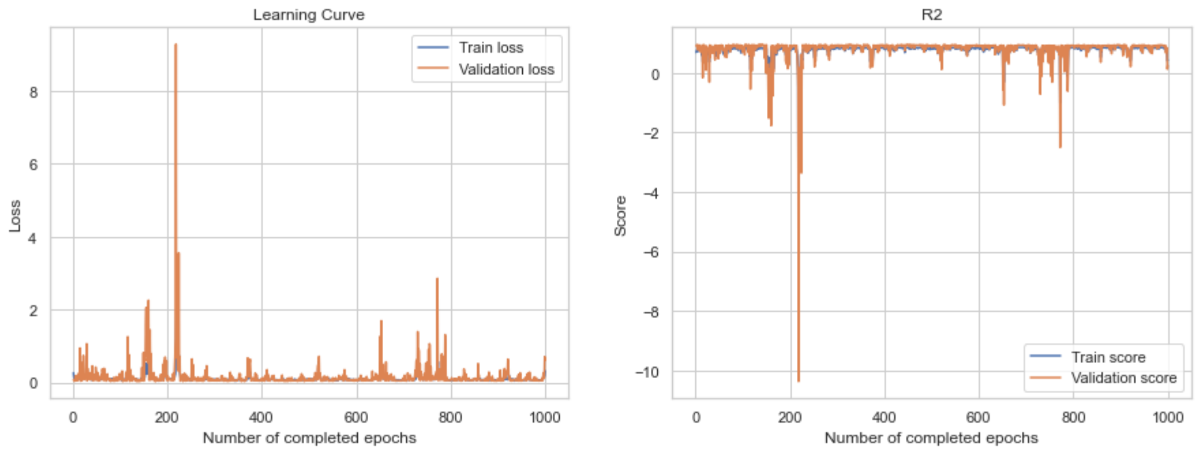


(b) Running time as a function of batch size

Figure 7: Effect of batch size



(a) The MSE and R2 as functions of learning rate. Score at lr0=0.1 shows that a high learning rate is able to reel itself back in.



(b) Learning curve with lr0=0.1. The sudden spikes indicate that the optimizer overshoots the solution.

Figure 8: Effect of batch size

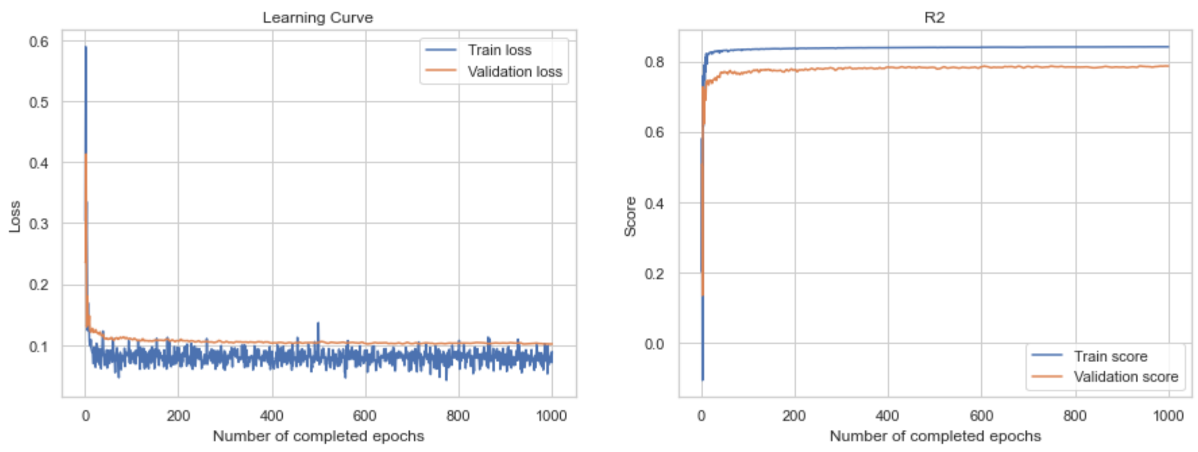


Figure 9: Learning curve using a decaying learning rate.

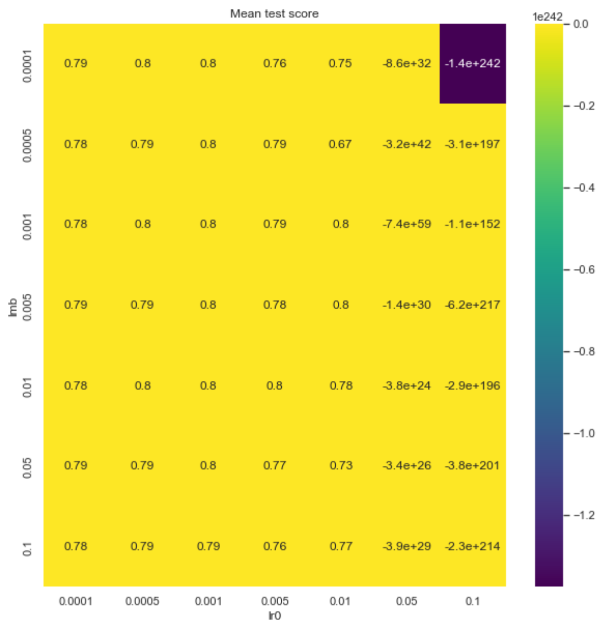
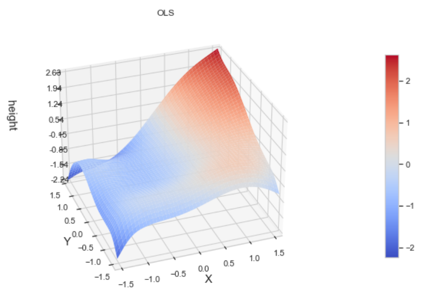
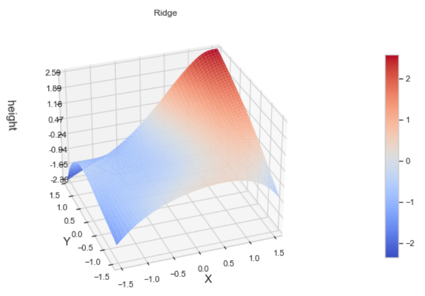


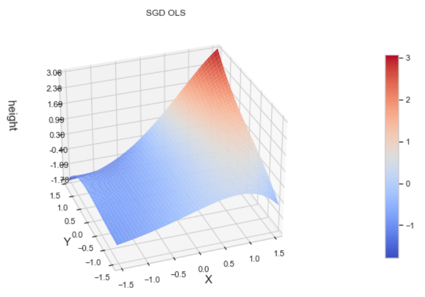
Figure 10: Heatmap showing R2 score with different combinations of lmb and lr.



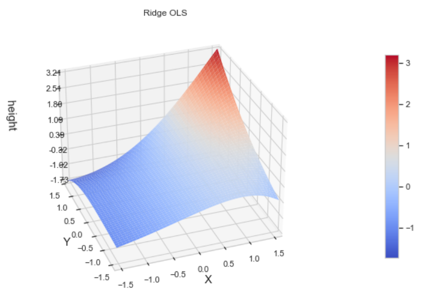
(a) OLS surface



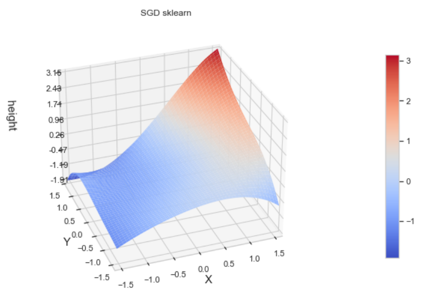
(b) OLS SGD surface



(c) Ridge surface



(d) Ridge SGD surface



(e) Sklearn SGD surface

Figure 11: Surfaces generated by different models. Test and train predictions are concatenated such that the whole surface can be recreated.

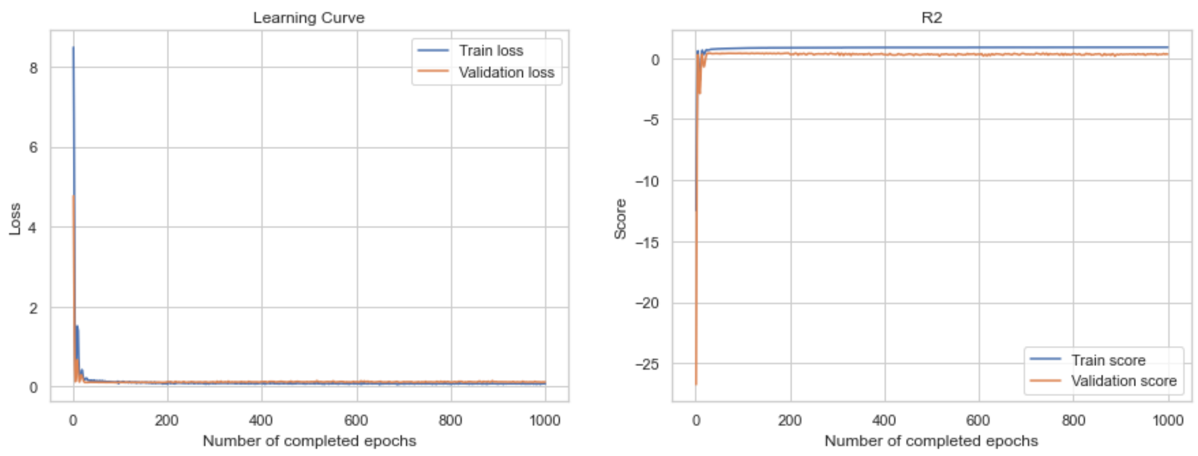


Figure 12: Learning curve of first MLP regression tested. Results show little sign of overfit and a good convergence.

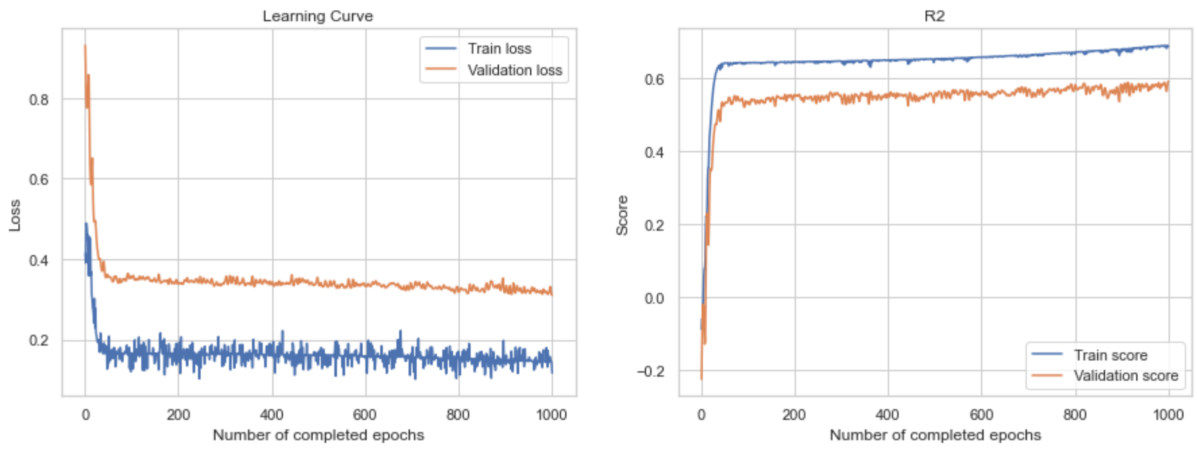


Figure 13: Learning curve of MLP using glorot as initializer.

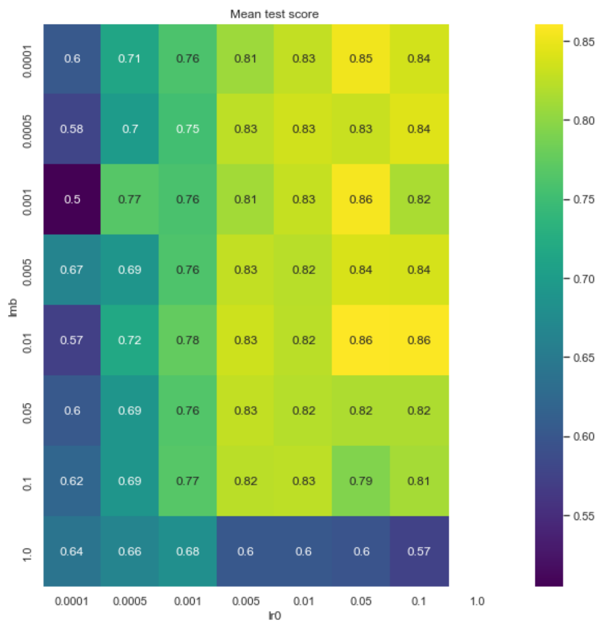


Figure 14: Heatmap showing R2 score as a function of lmb and lr.

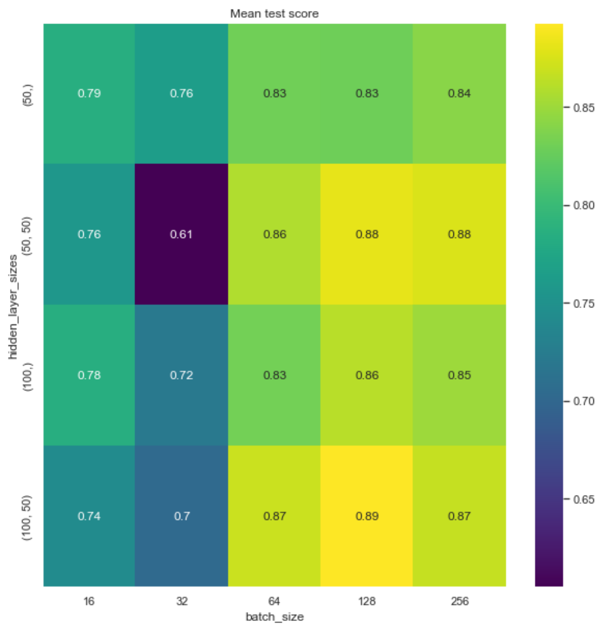


Figure 15: Heatmap showing R2 score as a function of architecture and batch size.

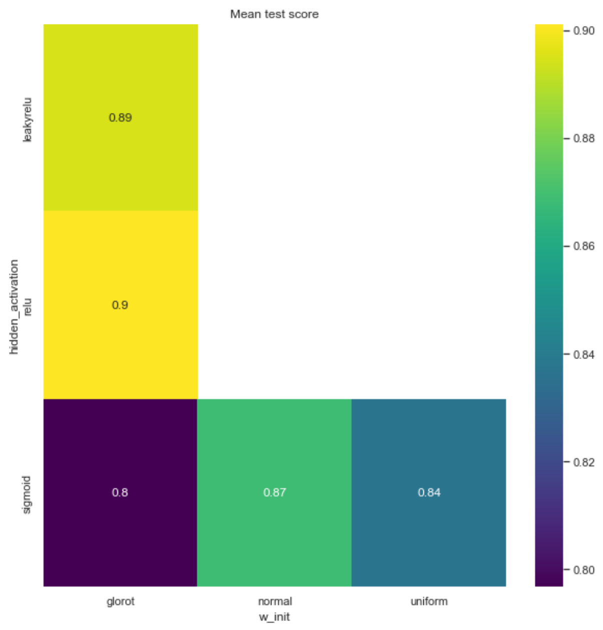


Figure 16: Heatmap showing R2 score as a function of activation function and initialization scheme.

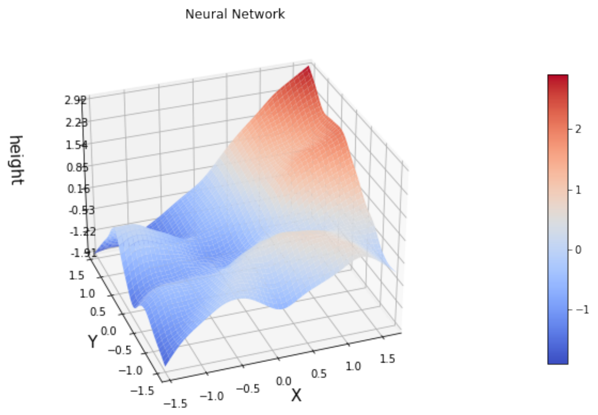
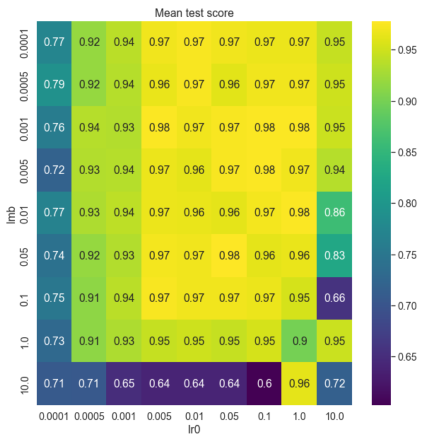
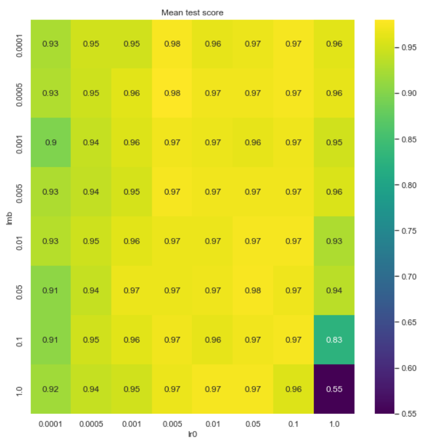


Figure 17: Complete surface produced by Neural Network. The details are rather convincing.

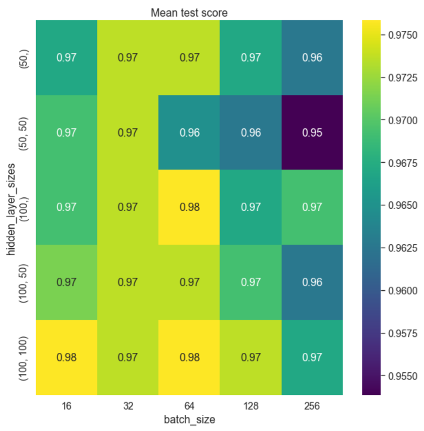
A.2 Classification



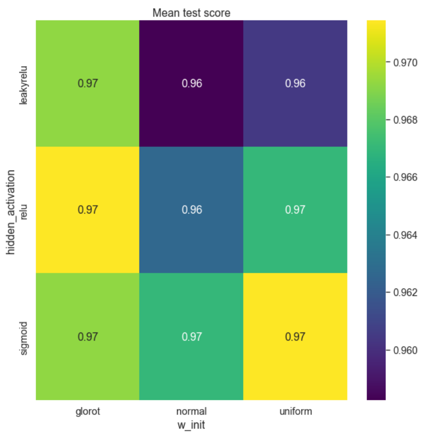
(a) Accuracy as a function of lr and lmb with 'normal' and sigmoid.



(b) Accuracy as a function of lr and lmb with 'glorot' and relu.

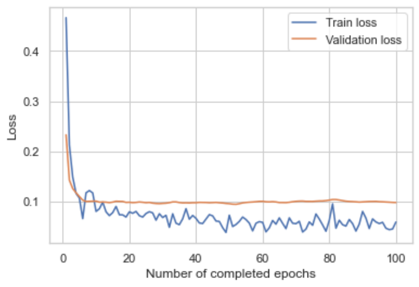


(c) Accuracy as a function of architecture and batch size with 'glorot' and relu.

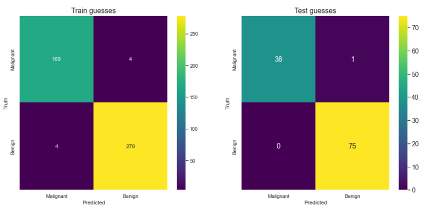


(d) Accuracy as a function of activation function and initialization scheme.

Figure 18: Grid search results on MLP classifier



(a) Learning curve with initial parameters.



(b) Confusion matrix with initial parameters.

Figure 19: Results of initial Logistic Regression model.

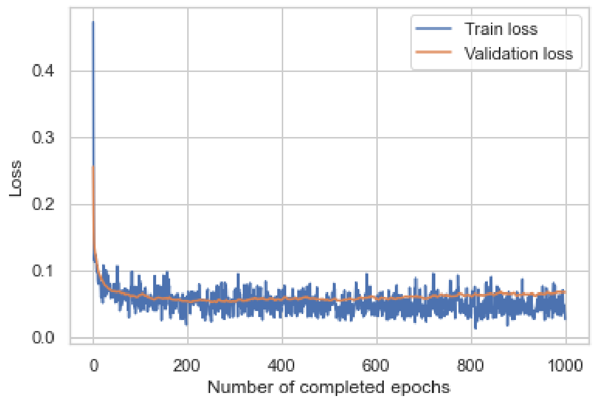


Figure 20: Learning curve using 1000 epochs, Logistic Regression

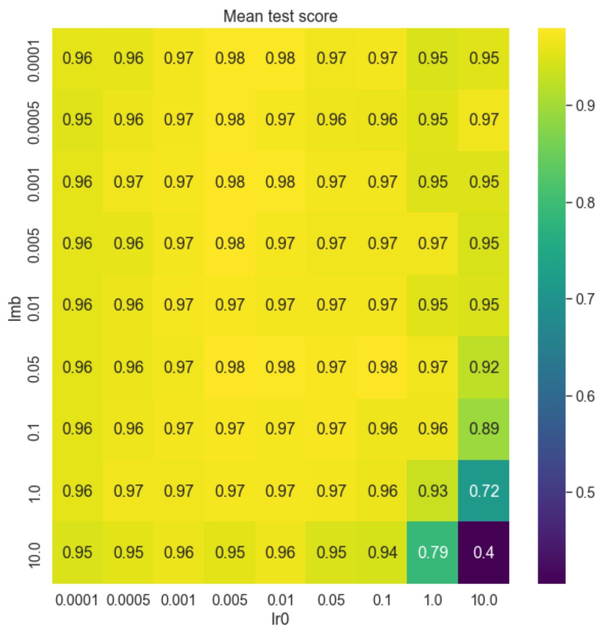


Figure 21: Accuracy as a function of learning rate and lmb for Logistic Regression

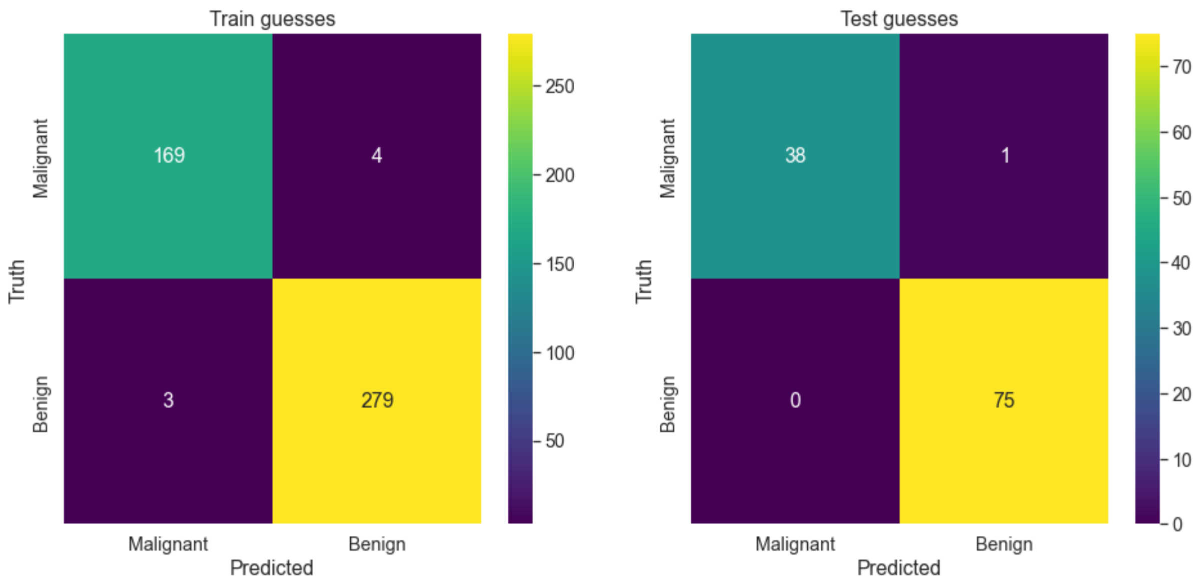


Figure 22: Confusion matrix of final achieved model.

B Tables

Parameter	OLS	OLS SGD	Ridge	Ridge SGD	Sklearn SGD
batch_size	-	32	-	32	1
n_epochs	-	1000	-	1000	1000
lmb	0	0	0.01	0.001	0
lr0	-	0.001	-	0.001	0.001
momentum	-	0.9	-	0.9	0.9
polydegree	5	5	5	5	5

Table 16: Model parameters used for benchmarking Regression vs SGD regression