Assignment 2

Computational Intelligence, SS2023

Team Members		
Last name	First name	Matriculation Number
Carlos Franco	Verde Arteaga	
Nedžma	Mušović	
Laura Maria	Höber	

Contents

2 Logistic Regression

2.1 Gradient Descent

The first task uses gradient descent to find the optimal weights $\omega_0, ..., \omega_L$ for the basis functions $\Phi_0, ..., \Phi_L$, which are defined as:

$$\begin{aligned} &\Phi_0(\mathbf{x}) = 1 \\ &\Phi_1(\mathbf{x}) = x^{(1)}, \Phi_2(\mathbf{x}) = x^{(2)} \\ &\Phi_3(\mathbf{x}) = (x^{(1)})^2, \Phi_4(\mathbf{x}) = x^{(1)}x^{(2)}, \Phi_5(\mathbf{x}) = (x^{(2)})^2 \\ &\quad \vdots \end{aligned} \tag{1}$$

The error function, that needs to be minimized, and its gradient are given as:

$$\tilde{E}(\omega) = -\frac{1}{N}\sum_{n=1}^{N}(t_n ln(y_n) + (1-t_n)ln(1-y_n)) \tag{2}$$

$$\nabla\tilde{E}(\omega) = -\frac{1}{N}\sum_{n=1}^{N}(t_n - y_n)\cdot\Phi(x_n) \tag{3}$$

2.1.1 Fit logistic regression model to training data

Figure 1 shows the result when a step size η=0.5 is applied. The maximum number of iterations is 2000, the weight vector is initialized with ω_0=$\underline{0}$ and the tolerance for convergence is chosen as $\epsilon = 10^{-3}$.

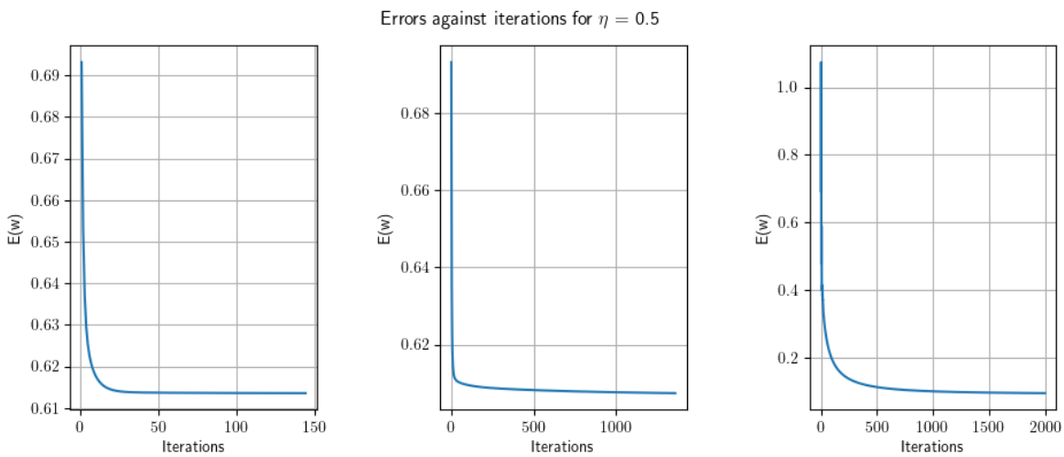


Figure 1: E(w) against iterations, $\eta = 0.5$

Table 1: Iterations for convergence

training data			
D	1	2	3
iterations	144	1357	2000

The three obtained model functions f have an analytical form of:

$$\begin{aligned} D = 1 : f(x_n) &= -0.3520 + 0.4772x^{(1)} + 0.2740x^{(2)} \\ D = 2 : f(x_n) &= -0.3254 + 0.6274x^{(1)} + 0.1392x^{(2)} - 0.3307(x^{(1)})^2 + 0.2083x^{(1)}x^{(2)} + 0.1068(x^{(2)})^2 \\ D = 3 : f(x_n) &= 1.8692 - 1.2046x^{(1)} - 1.5333x^{(2)} - 2.4994(x^{(1)})^2 - 2.1621x^{(1)}x^{(2)} - 1.8363(x^{(2)})^2 \\ &\quad + 0.9886(x^{(1)})^3 + 1.1480(x^{(1)})^2x^{(2)} + 1.3227x^{(1)}(x^{(2)})^2 + 1.2018(x^{(2)})^3 \end{aligned} \tag{4}$$

.

2.1.2 Analyze the convergence behavior of gradient descent

For the next exercise, we changed the step size that we are using for plotting the error function. In this case n=0.05, 0.5, 1.5, for D=1,2,3.

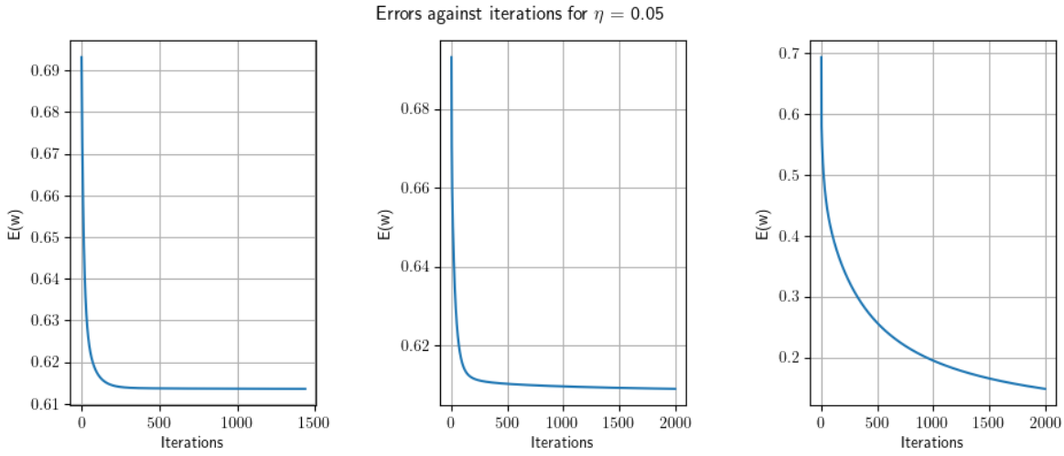


Figure 2: $\tilde{E}(\omega)$ against iterations, $\eta = 0.05$ for D=1,2,3

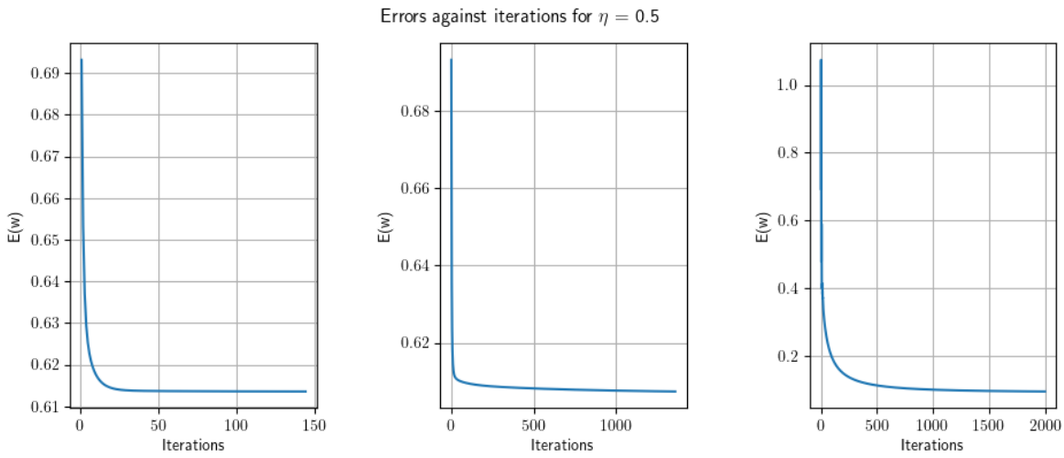


Figure 3: $\tilde{E}(\omega)$ against iterations, $\eta = 0.5$ for D=1,2,3

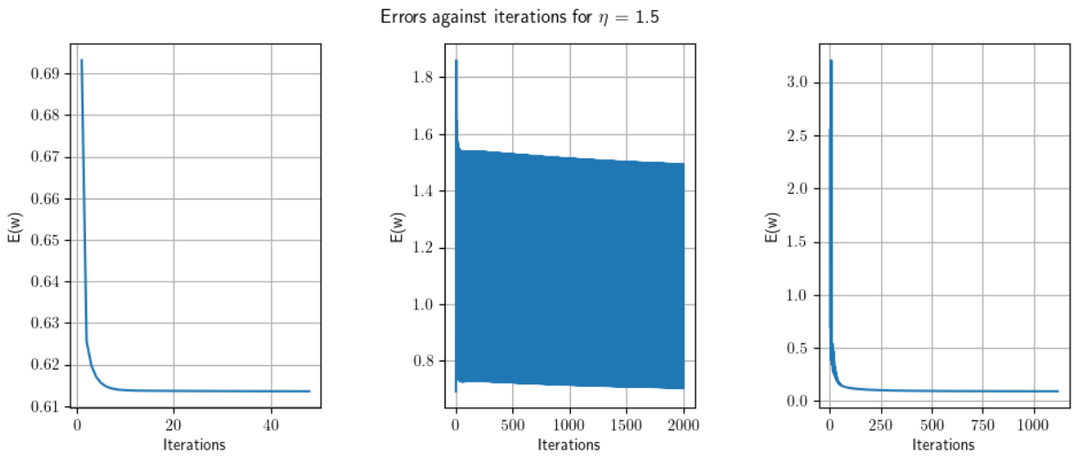


Figure 4: $\tilde{E}(\omega)$ against iterations, $\eta = 1.5$ for D=1,2,3

Conclusion:

Depending on the step size and the dimension D the error decreases or even increases at different rates. In this case when we are using the smallest step size $\eta = 0.005$ the algorithm takes longer to converge, especially for a higher D, so more iterations are needed. On the other hand, for a bigger step size the algorithm will converge faster, but this also entails the risk of oscillations, where the error jumps between higher and lower values. This might be caused if the function to minimize has narrow areas around a local minima, that the algorithm tries to reach.

The best results seem to be reached by the models with D=3, independent from η, because their error gets closer to 0.

2.1.3 Decision Boundaries

Figure 5 and 6 show the areas that the different models classify as red for the class 0 and blue for the class 1.

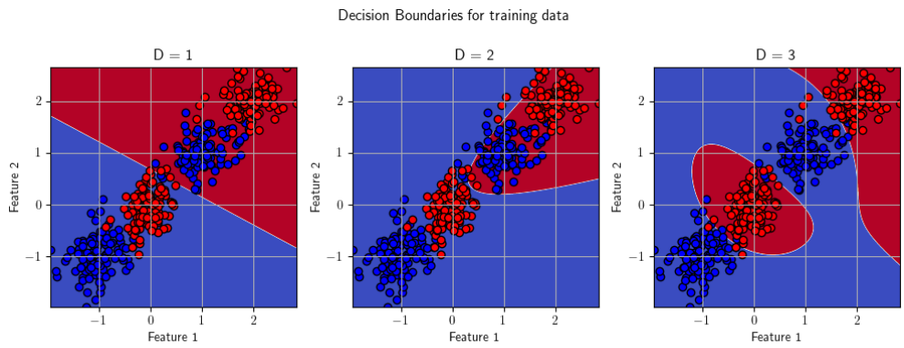


Figure 5: Decision Boundaries for D=1, D=2 and D=3 for training data

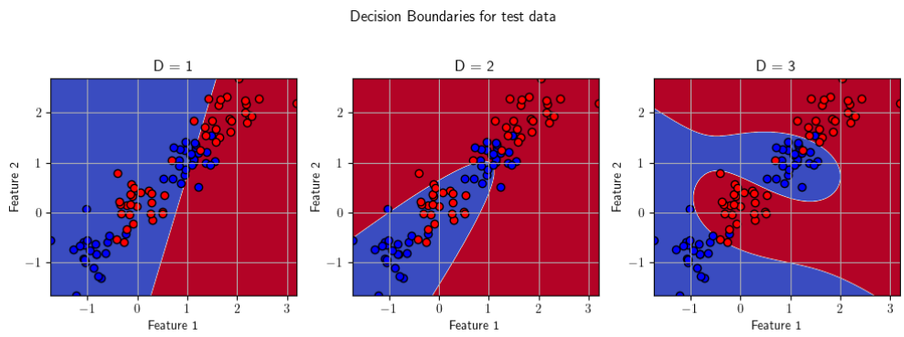


Figure 6: Decision Boundaries for D=1, D=2 and D=3 for test data

Tables 2 and 3 show what percentage of training/test sample is classified correctly by the models with different D.

Table 2: Percentage of correctly classified points for training data

training data		
D=1	D= 2	D= 3
49.25	50.0	96.0

Table 3: Percentage of correctly classified points for test data

test data		
D=1	D= 2	D= 3
52.0	54.0	96.0

A minimum degree of 3 seems to be necessary to properly separate the classes, which means that the data is not linearly separable. The accuracies further show that models with D=1 or D=2 misclassify more than half of the data, which is worse than if the labels were simply assigned at random.

2.1.4 Lowest Error

In the next exercise, we will plot the training/test errors against D =1,2,...,10. Figures 7 8 show the final error for each model after it finished training and the optimal weights were obtained.

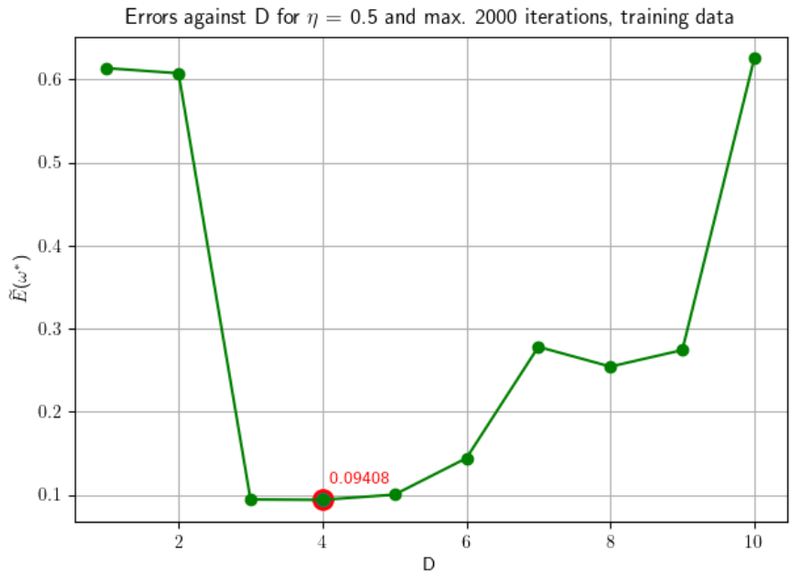


Figure 7: Error against D for the training data

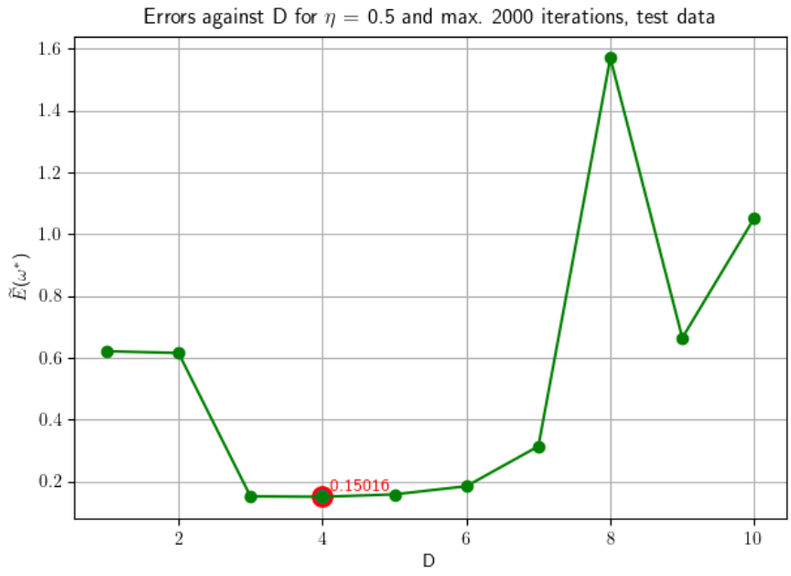


Figure 8: Error against D for the test data

Table 4: Minimum error for D=4, eta=0.5, iterations=2000 for the training and test data

	training	test
$\min(\tilde{E}(\omega^*))$	0.09408	0.15016

Conclusion:
Analysing the graphs that we obtained, D = 4 resulted in the lowest error for both data sets. A D of 3, 5 and 6 also leads to relatively good results, but below that undefitting occurs and above that overfitting occurs.

2.1.5 Analytical Expression

For solving this exercise, we first observe the number of parameters for the values of D, that we used: f(D=1)=3, f(D=2)=6, f(D=3)=10, f(D=4)=15, ...,f(D=8)=45,f(D=9)=55, f(D=10)=66.

Applying :

f(D=2) - f(D=1)= 6 -3 = 3
f(D=3) - f(D=2) = 10 - 6 = 4

...

f(D=9) - f(D=8) = 55 - 45 = 10
f(D=10) - f(D=9) = 66 - 55 = 11

We can notice from the last information the difference from each previous value increases by 1 every time. So the number of parameters for D is the sum of all previous values and D+1, which corresponds to the triangular number and can be written as:

$$f(D) = \sum_{d=1}^{D+1} d = \frac{[D+1] \cdot ([D+1]+1)}{2} = \frac{D^2 + 3D + 2}{2} \tag{5}$$

The same result is obtained, if the function is assumed to follow a second-degree polynomial relationship.

Considering: f(D) = aD^2 + bD + c

D=1 => f(D) = a+b+c = 3

D=2 => f(D) = 4a + 2b +c = 6

D=3 => f(D) = 9a+3b+c = 10

Solving the equation system that we obtained before. The result for a=0.5, b=1.5 and c=1. As result, we can also find our analytical expression, f(D) = 0.5D^2+ 1.5D + 1

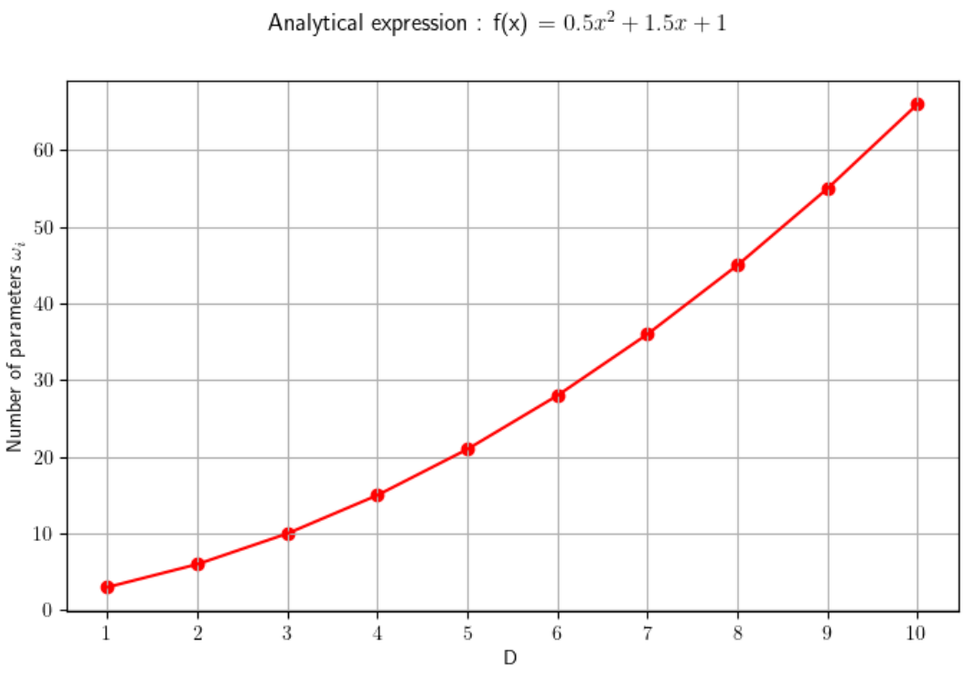


Figure 9: Analytical Expression

2.2 Newton-Raphson algorithm

Now, instead of calculating the best weights by gradient descent, the Newton-Raphson algorithm is used. The weights are obtained iteratively with the formula:

$$\omega_{(k+1)} = \omega_{(k)} - \eta \cdot (\mathbf{H}(\omega_{(k)}))^{-1} \nabla \tilde{E}(\omega_{(k)}) \tag{6}$$

$\nabla \tilde{E}$ stands for the gradient of the cross entropy error from equation 2 and $\mathbf{H}(\omega_{(k)})$ describes the Hessian matrix of $E\tilde{(\omega)}$, which contains all of its second derivatives as follows:

$$\mathbf{H}(\omega_{(k)}) = \begin{bmatrix} \frac{\partial^2 \tilde{E}(\omega)}{\partial \omega_0 \partial \omega_0} & \frac{\partial^2 \tilde{E}(\omega)}{\partial \omega_0 \partial \omega_1} & \cdots & \frac{\partial^2 \tilde{E}(\omega)}{\partial \omega_0 \partial \omega_L} \\ \vdots & \ddots & & \vdots \\ \frac{\partial^2 \tilde{E}(\omega)}{\partial \omega_L \partial \omega_0} & \cdots & & \frac{\partial^2 \tilde{E}(\omega)}{\partial \omega_L \partial \omega_L} \end{bmatrix} \tag{7}$$

2.2.1 Proof, that the Hessian matrix can be written as:

$$\mathbf{H}(\omega) = \frac{1}{N} \sum_{n=1}^{N} y_n (1 - y_n) \Phi(x_n) \Phi(x_n)^T \tag{8}$$

First, we derive the cross entropy error by every weight ω_l with l = 1, ..., L:

$$\begin{aligned} \frac{\partial y_n}{\partial \omega_l} &= \frac{\partial}{\partial \omega_l} \sigma\left(\omega^T \Phi(x_n)\right) = \sigma\left(\omega^T \Phi(x_n)\right)\left(1 - \sigma\left(\omega^T \Phi(x_n)\right) \Phi_l\right) = y_n(1 - y_n)\Phi_l \\ \frac{\partial^2 \tilde{E}(\omega)}{\partial \omega_l} &= -\frac{\partial}{\partial \omega_l} \frac{1}{N} \sum_{n=1}^{N} (t_n ln(y_n) + (1 - t_n) ln(1 - y_n)) \\ &= -\frac{1}{N} \sum_{n=1}^{N} (t_n \frac{1}{y_n} y_n(1 - y_n)\Phi_l(x_n) - (1 - t_n) \frac{1}{1 - y_n} y_n(1 - y_n)\Phi_l(x_n)) \\ &= -\frac{1}{N} \sum_{n=1}^{N} (t_n(1 - y_n)\Phi_l(x_n) - (1 - t_n) y_n \Phi_l(x_n)) \\ &= -\frac{1}{N} \sum_{n=1}^{N} (\Phi_l(x_n)(t_n - t_n y_n - y_n + t_n y_n)) \\ &= -\frac{1}{N} \sum_{n=1}^{N} (t_n - y_n)\Phi_l(x_n) \end{aligned} \tag{9}$$

For the Hessian matrix this expression is now derived a second time by ω_m with m = 1, ..., M:

$$\begin{aligned} \frac{\partial^2 \tilde{E}(\omega)}{\partial \omega_l \partial \omega_m} &= -\frac{\partial}{\partial \omega_m} \frac{1}{N} \sum_{n=1}^{N} (t_n - y_n)\Phi_l(x_n) \\ &= \frac{1}{N} \sum_{n=1}^{N} y_n(1 - y_n)\Phi_m(x_n)\Phi_l(x_n) \end{aligned} \tag{10}$$

These are the unique components of the Hessian matrix with l as the row index and m as the column index. Because H is a square matrix m = 1, .., L and the final matrix corresponds to the given formula 8.

2.2.2 BONUS: Proof, that H(ω) is positive definite

Positive definite means, that the expression $\mathbf{v}^T\mathbf{H}(\omega)\mathbf{v}$ is positive. Because all of the terms inside the sum are positive this criteria is fulfilled for the Hessian matrix.

$$\begin{aligned}\mathbf{v}\mathbf{H}(\omega)\mathbf{v}^T &= \frac{1}{N}\sum_{n=1}^{N} y_n(1-y_n)v^T\Phi(x_n)\Phi(x_n)^T v \\ &= \frac{1}{N}\sum_{n=1}^{N} y_n(1-y_n)\left(v^T\Phi(x_n)\right)^T\Phi(x_n)^T v \\ &= \frac{1}{N}\sum_{n=1}^{N} y_n(1-y_n)\left(v^T\Phi(x_n)\right)^2\end{aligned} \tag{11}$$

$$\begin{aligned}(\Phi(x_n))^2 &> 0 \quad v \neq \underline{0} \\ 1 > y_n = \sigma\left(\omega^T\Phi(x_n)\right) &> 0 \\ (1-y_n) &> 0\end{aligned} \tag{12}$$

2.2.3 BONUS: Proof, that $\tilde{E}(\omega)$ is convex

Per definition a function is convex, if their second derivative is positive. In the case of partial derivatives, the matrix of second derivatives(Hessian matrix) needs to be positively semi-definite. This means, that the Eigenvalues of the Hessian matrix need to be greater or equal to 0.
We have already proven, that the Hessian matrix is positive definite, which means all Eigenvalues of it are greater than 0. Therefore $\tilde{E}(\omega)$ is convex.

2.2.4 Fitting Results

Same as in simple gradient descent the parameters were chosen as 2000 maximum iterations, $\eta = 0.5$ and $\epsilon = 10^{-3}$ and the model was fitted for D = [1,2, ..., 10]. Now the number of iterations until $|\nabla\tilde{E}(\omega)|$ gets smaller than ϵ is compared for all D.
Figure 10 shows how many iterations the model needed with this algorithm.

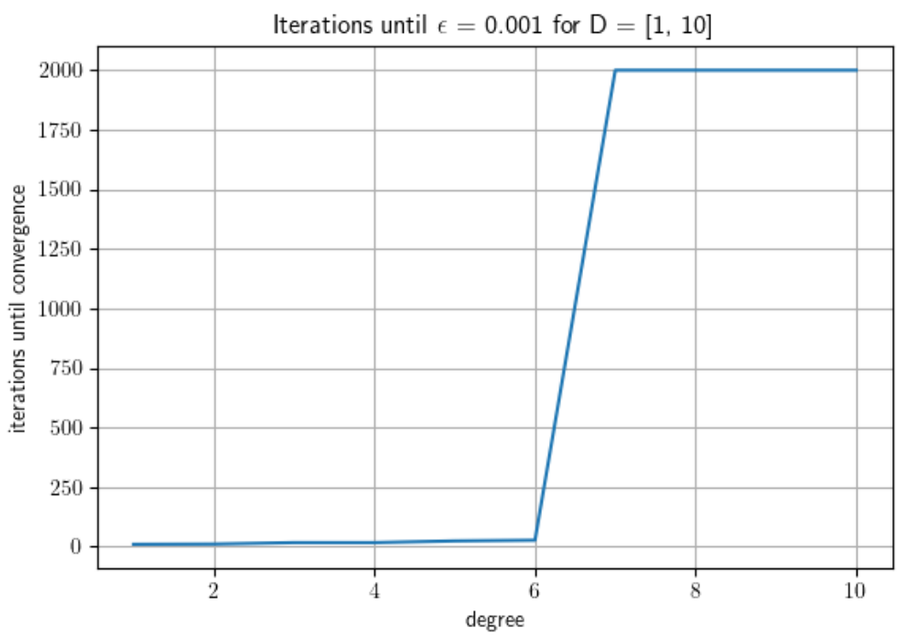


Figure 10: Iterations for every D with Newton-Raphson alrogithm

Up to a degree of 6 the algorithm converges extremely fast, after less than 27 iterations. However, for higher values of D it suddenly doesn't converge anymore, which is still better than the previous models, who failed to converge after 2000 iterations even for a D of 3.

3 Neural Networks

3.1 Regression with Neural Networks

3.1.1 Learned function

In the first subsection of the task on neural networks, an MLPRegressor neural network model was used and trained on the given train data sets, such that it outputs a target prediction on the test data set. The target predictions of neural networks containing 2, 5, and 50 neurons in the hidden layer are shown in the plots of figure 13. A random seed that lead to the most representative function flow of every model was chosen.

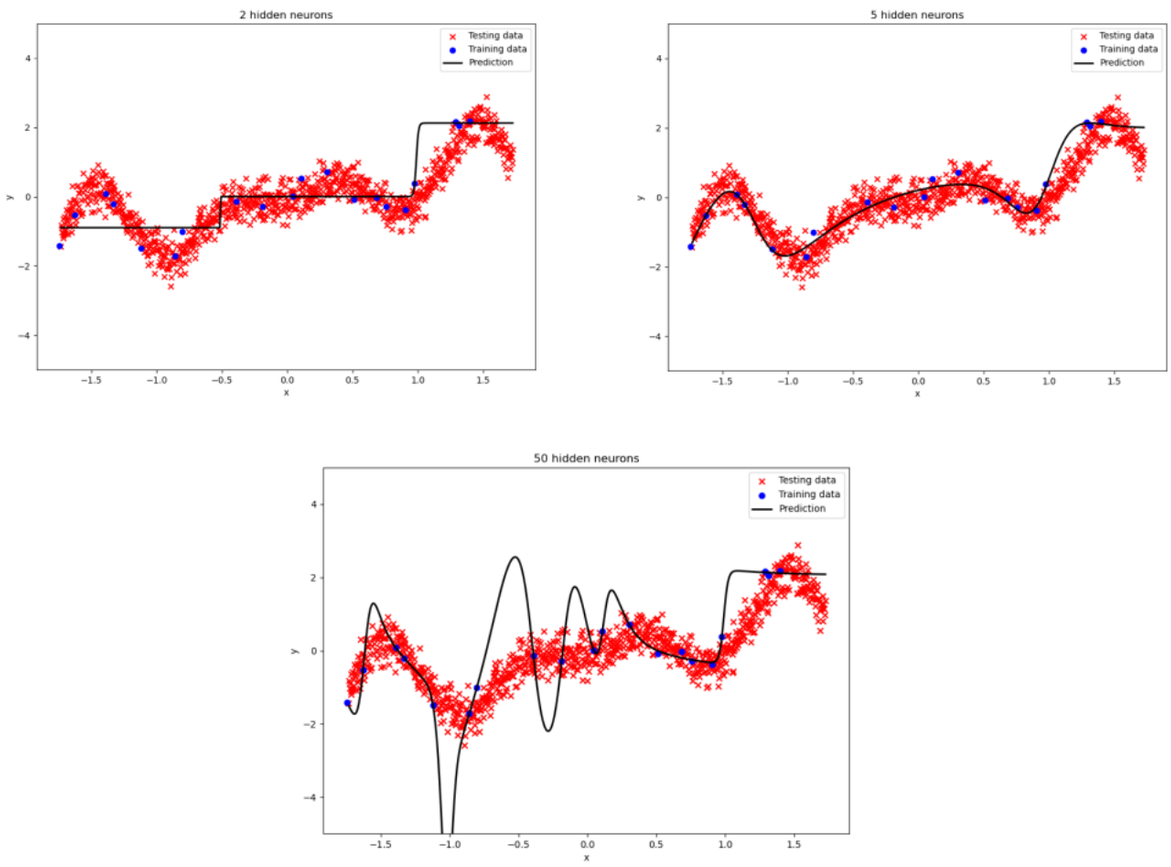


Figure 11: Learned plots given different number of neurons in the hidden layer

As expected, the number of neurons in the hidden layer, and thus the architecture of a neural network, significantly impacts the results.

In the first plot, the network with only two neurons in the hidden layer does not seem to have sufficient capacity to capture the data pattern and thus deliver a satisfactory prediction on the test data.

The third plot shows the performance of a neural network with 50 neurons in the hidden layer and is a clear example of overfitting. A neural network with a very complex architecture may have learned the training data well but performs poorly on generalizing data (e.g., it can capture even noise but not the pattern behind the data set). As soon as the test data slightly differs from the training data, the outcome of the neural network is not as expected.

The neural network with five neurons in the hidden layer produces not the ideal but, in this exercise, the best fitting prediction and seems to find a balance in the architecture concerning under- and overfitting.

3.1.2 Variability of the performance of deep neural networks

The optimal version of the previously observed neural networks gets investigated further in the second subtask by evaluating the loss function as the mean square error function with non-fixed random seeds.
Varying the random seed produced different results for MSE (its max, min, mean, and standard deviation), as shown in the following table.

training data				
	MSE	random seed	mean	σ
max	0.2001	4	0.1301	0.0647
min	0.0266	3		

Table 5: Extracted information on MSE of trained and tested data

tested data				
	MSE	random seed	mean	σ
max	0.5367	5	0.4249	0.0991
min	0.2256	7		

As shown in the tables, the extracted minimal and maximal MSE observed on two different data sets do not come from the same random seeds meaning that the random seed performance partially relies on the characteristics of a particular data set. As solely observing the outcome does not indicate the suitability of the chosen seed, the best way to find the appropriate init randomness is to run the network with multiple random values and evaluate the outcome using a so-called validation set.
The validation set is a not seen part of the training data used not in the training process but afterward for evaluating the trained process (e.g., for under/overfitting) or for selecting the most appropriate random seed - the one showing the best performance on the validation data.

Having non-fixed seeds (in this task derived from the for-loop iteration variable as an increment by 1) impacts the initial weights and thus the starting point for the algorithm. The neural networks do not deal with simple linear or convex shapes, that have one global minimum, but introduce non-linearity in their hidden layers, which leads to not one but multiple local minima in addition to a global one. Starting at an optimal position may lead the algorithm quickly converge to the global (or "good enough") minima, whereas initiating with non-optimal weights may let the algorithm stay in a local minima. That is why the random seed choice can highly impact the performance and correctness of a neural network.

Stochastic gradient descent, a commonly used optimization algorithm in the training process of neural networks, introduces randomness by its own stochasticity. In this approach, the samples for gradient calculation get selected randomly in every iteration. Instead of evaluating the whole data set, only subsets or single values flow into the gradient calculation, and these particular samples get selected randomly.
In the standard gradient descent, the whole data set gets involved in every iteration, and the randomness relies solely on the randomness of the parameters (random seeds, as previously discussed).

3.1.3 Varying the number of hidden neurons

In this task, we observed the mse averaged over multiple neural networks running with different seeds (see source code) while varying the number of neurons. The generated plot is given as follows:

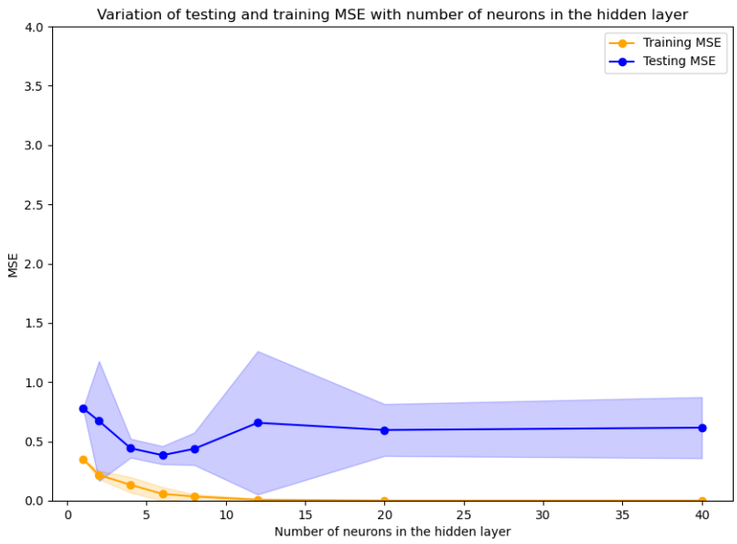


Figure 12: MSE given different number of hidden neurons

Considering the averaged mse, the most optimal behavior is show by the network with six neurons in the hidden layer. As expected, increasing the number of hidden neurons increases the network's capability, leading to a decreasing mse.

For one, two and four neurons, we can observe a relatively high mse (compared to the optimal version) implying the underfitting in a neural network with a simple architecture which can't capture enough information.
On the other side, starting with eight and twelve neurons, we can observe the trending change, and mse remains relatively constant (or slowly increases), which clearly implies that increasing the number of hidden neurons brings no benefits or the data is overlearned (capturing non-informative parts as well) - overfitting.

3.1.4 Change of MSE during the course of training

In the last subtask, we had to observe the mse escalation during the iterations given the different numbers of neurons in the hidden layer.

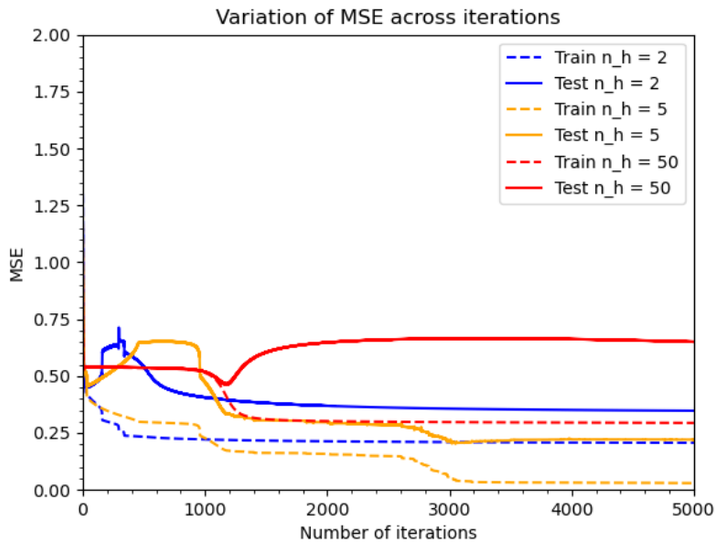


Figure 13: MSE development during the iterations

As expected, given its low capability, the neural network with two neurons converges relatively fast toward its best performance but does not train data in the best way, implied by the higher mse compared to the optimal architecture shape.

On the other side, the convergence of the optimal model takes longer as the training process is more involved in this case, but this model reaches the lowest mse.
The overfitted model reaches the highest mse as the noise and other disturbances information eventually fly in within the training process.

Increasing the number of neurons in the hidden layer automatically increases the network's capability potentially leading to capturing not informative patterns, as previously discussed.

The randomness of the SGD leads the model to find the solution that best fits the not seen data and avoid getting stuck in local minima, which may lead to overfitting.

3.2 Classification with Neural Networks: Fashion MNIST

In this task the multi layer perceptron class "MLPClassifier" of sklearn is used for multi-class classification of images of clothing items. The given Data contains 70.000 images, of which 60.000 are used as training data, and the remaining 10.000 images are used as test data.
The model consists of 3 layers:

- The first layer serves as the input layer with 794 input features, each belonging to one pixel of the 28 x 28 images.
- The second layer is a hidden layer, which contains n_{hidd} hidden units. n_{hidd} was chosen as 20 to achieve a relatively good classification performance, while still being efficient, with about 3 minutes of processing time on the used machine.
- The third layer is the output layer with 10 different nodes corresponding to the 10 different class labels: T-shirt/top, trousers/pants, pullover shirt, dress, coat, sandal, shirt, sneaker, bag, ankle boot.

All of the nodes from one layer are connected to the following layer with certain weights. During training, these are adjusted to correctly predict the output label $y((x)_n)$ for the input features/pixel values $(x)_n$.
The activation function, which calculates the final prediction based on the values of the output layer, is chosen as "tanh".

Prediction Accuracy

First, 5 different MLP classifiers with arbitrarily chosen initial weight vectors were fitted. For each of them their classification accuracy for the test images was tested and the best performing model was chosen for further analysis. All accuracies of the test and training predictions are visualized in figure 14. The top and bottom lines of the boxplot show the max. and min. accuracy of the best and worst fitted models. Here, the best accuracy for the test set is an outlier with 87.6%, which belongs to the model with seed 4.
The upper rims of the boxes mark the upper quartile of the accuracies, which indicates that 75% of the models reached an accuracy of max. 91.35% in the training set and 87.1% in the test set. Analogously the lower rim indicates that 25% of the models reached an accuracy of max. 91.25% in the training set and 86.9% in the test set.The mean accuracy is marked by the orange line.
As expected, the model performed slightly better on the training set than on the test set, but the mean accuracy only decreased by about 4%, so a number of 20 hidden units seems like a good choice to neither cause a good performance with neither under- nor overfitting.

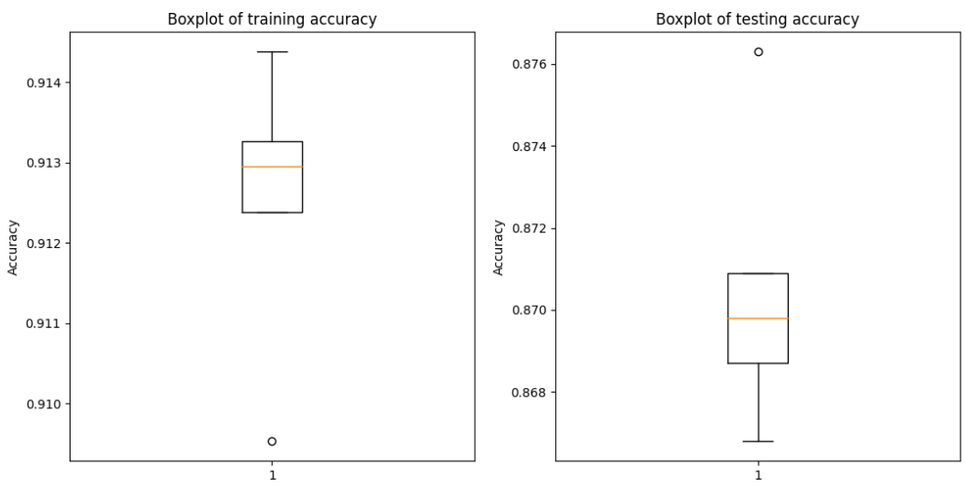


Figure 14: Accuracies of 5 MLP classifiers with different seeds

Confusion Matrix

The main diagonal of the matrix in figure 15 shows the number of correctly classified images for each class label. With a relatively high accuracy, most results lie on this diagonal. The clothing items that the model seems to have the most difficulties with, are coats and shirts. For example, 139 shirts were wrongly classified as T-shirt/top and 109 coats were wrongly classified as pullover shirts. This makes sense, because these items don't have easily distinguishable features.
On the other hand, trousers/pants, bags and ancle boots already come pretty close to a perfect score at about 960 of 1000 correctly classified.

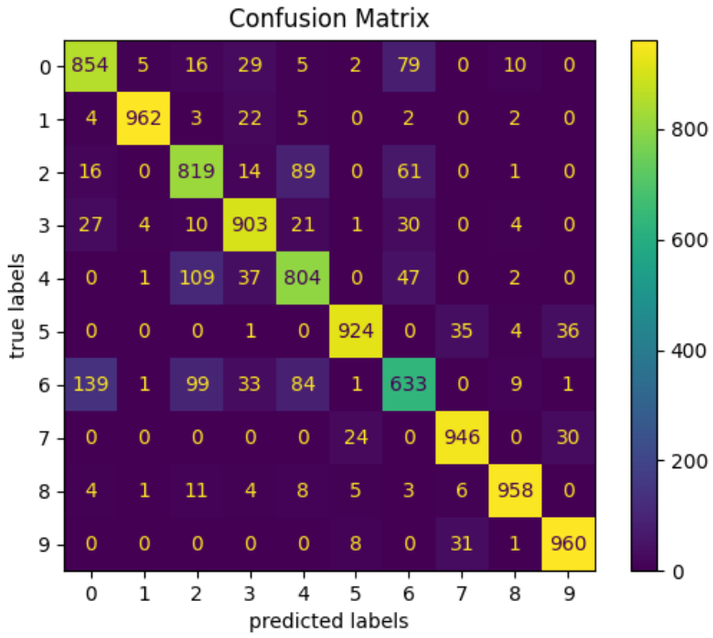


Figure 15: Confusion matrix of the best performing MLP classifier model

Model weights

Figure 16 shows the weights between the 748 input pixels and the first 10 hidden units of the hidden layer. The bright areas mark pixels with high weights, who strongly influence the output of the classification, while the dark areas mark pixels with little influence on the outcome. Not much can be interpreted directly into these abstract weight images. It seems like the border of the images is not taken much into account, because its similar for most clothes. Only small areas with higher weights can be seen around areas where sleeves or contours are generally located in the images. For example in weight matrix 16 the shape of a dress could be seen, or weight matrix 8 might be used to detect collars of coats or pullover shirts. This could indicate, that a specific hidden unit is mainly responsible for detecting a specific clothing item.

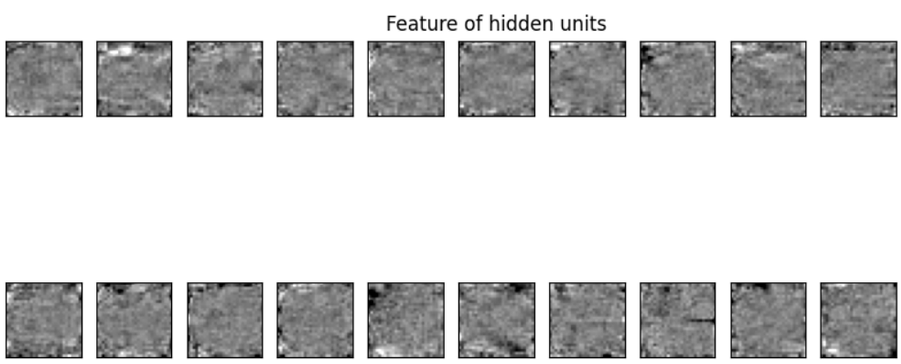


Figure 16: Weights between the first two layers of the MLP classifier

Misclassified images

The folowing figures show some randomly picked images out of the test data, that were wrongly classified. The given function "HW2_NN_classification_plot" was slightly altered to include the true and predicted label in the title. As already mentioned in the confusion matrix, T-shirts/tops, shirts, pullover shirts and coats seem to be the hardest to differentiate between. Therefore they lead to the most misclassifications.

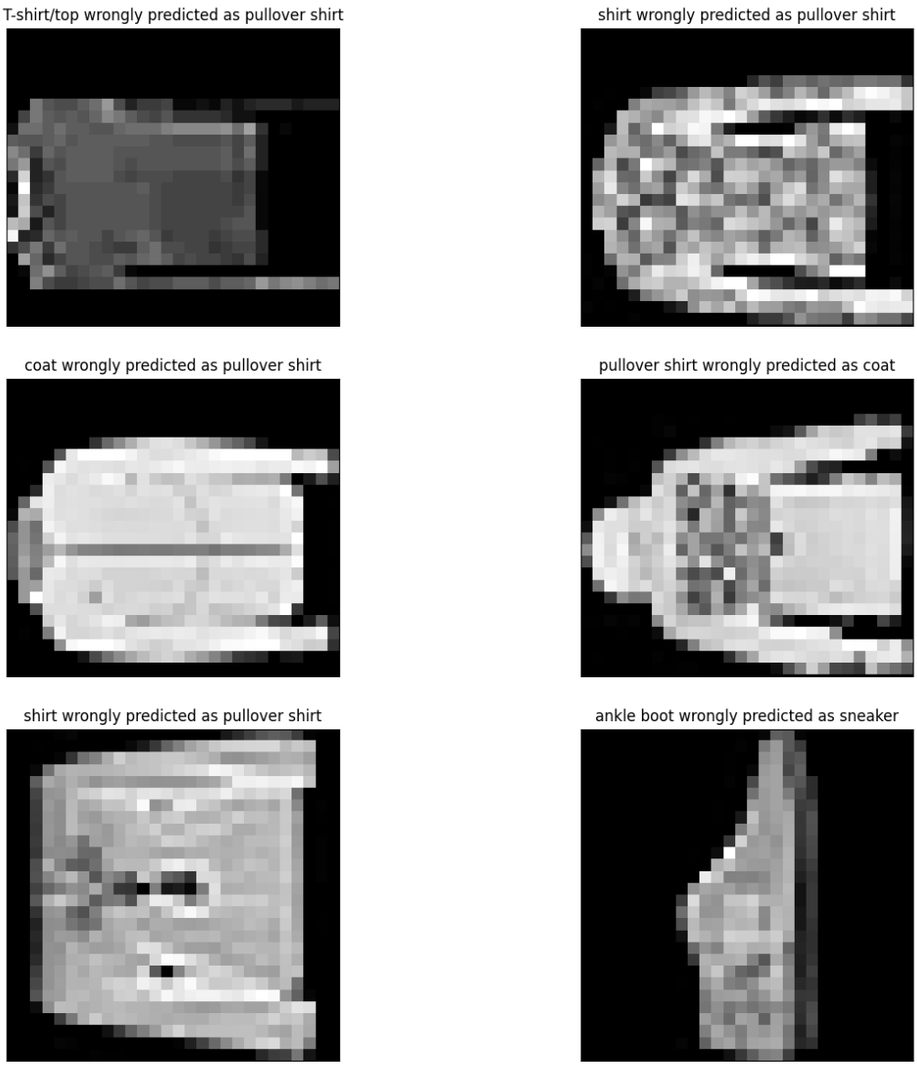


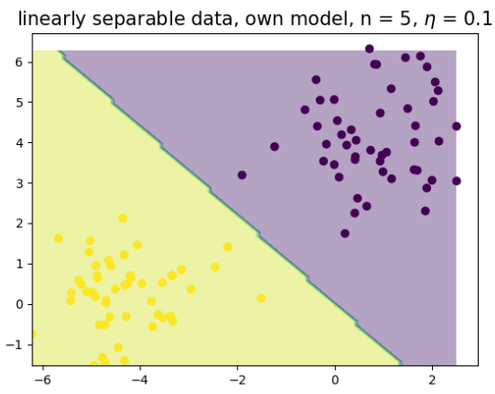
Figure 19: Misclassified images of the test set

3.3 BONUS: Implementation of a Perceptron

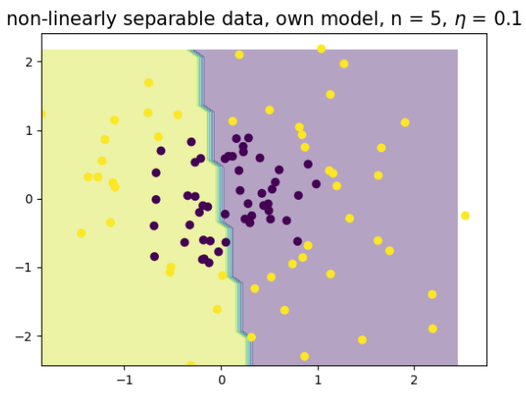
In the bonus task a classification model with a single perceptron has to be implemented and used for prediction on linearly and non-linearly separable data. The results of this self-implemented model were then compared to the perceptron model of scikit learn.

3.3.1 Results of self-implemented perceptron

As a baseline the learning rate was chosen as 0.1 and the number of iterations with 5, shown in the following figure for the linearly and non-linearly separable data. Following graphs visualize how the decision boundary changes when altering one of the parameters.

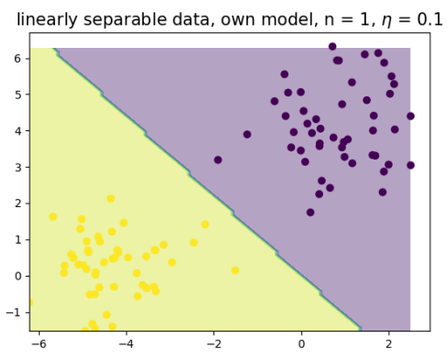


(a) Linearly separable data

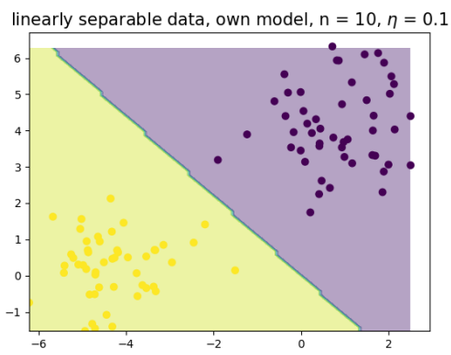


(b) Non-linearly separable data

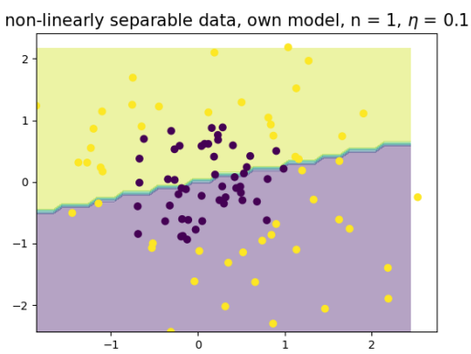
Variation of iterations



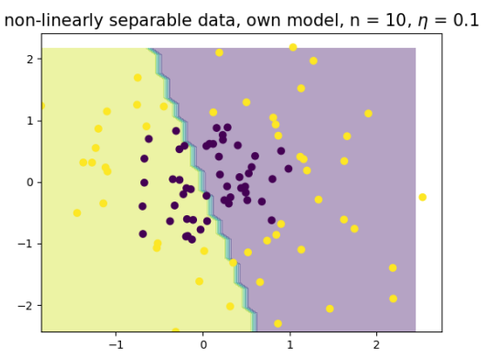
(a) Number of iterations = 1



(b) Number of iterations = 10

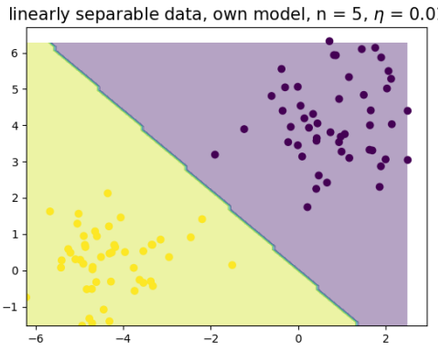


(c) Number of iterations = 1

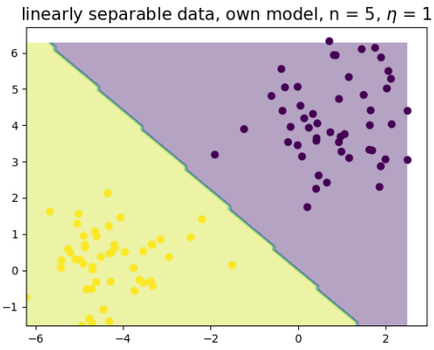


(d) Number of iterations = 10

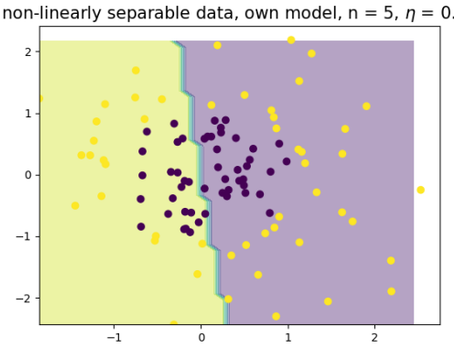
Variation of learning rates



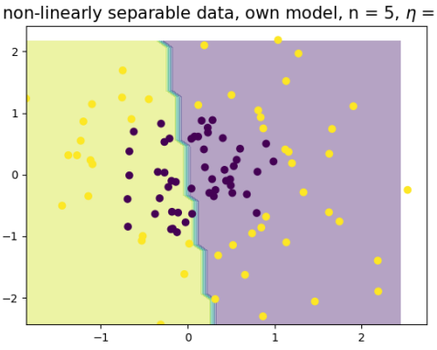
(a) Learning rate = 0.01



(b) Learning rate = 1



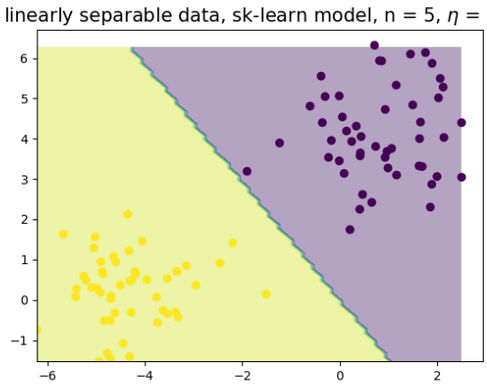
(c) Learning rate = 0.01



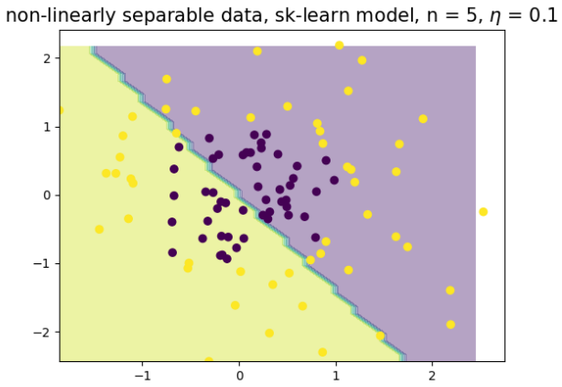
(d) Learning rate = 1

3.3.2 Results of perceptron from scikit learn

The same variations are also implemented with the perceptron model from scikit learn. The baseline can be again seen in the follwing figures:

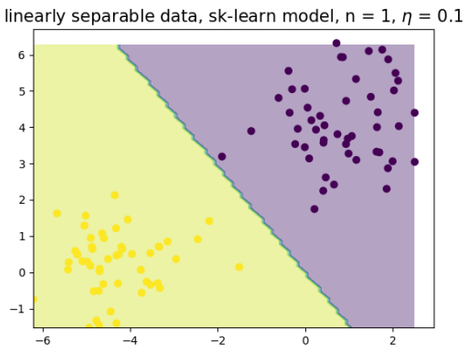


(a) Linearly separable data

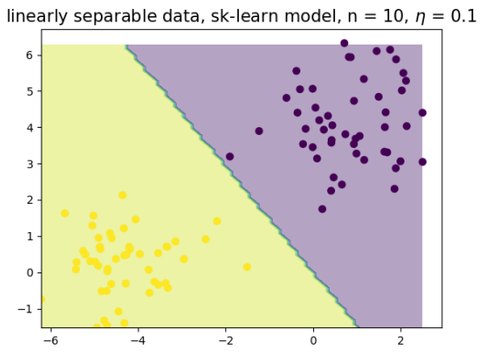


(b) Non-linearly separable data

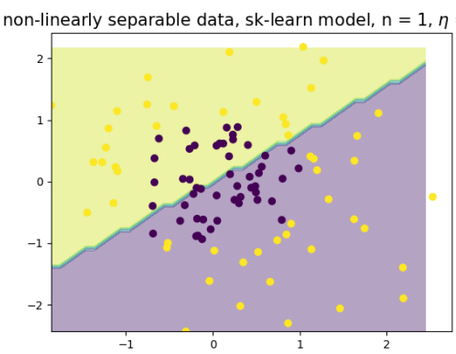
Variation of iterations



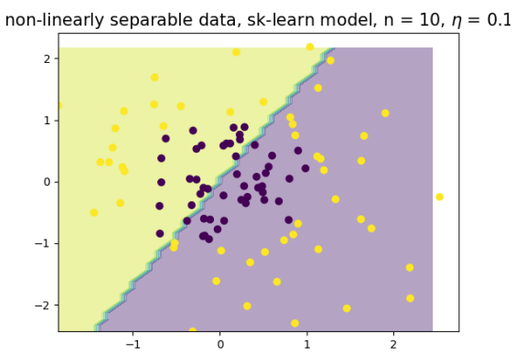
(a) Number of iterations = 1



(b) Number of iterations = 10

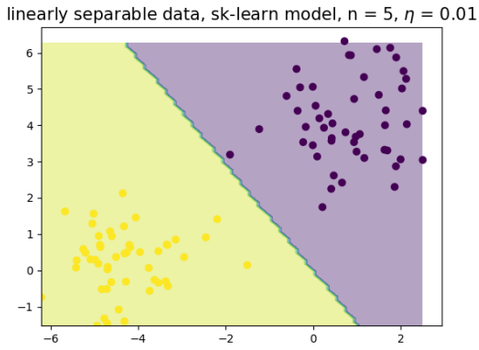


(c) Number of iterations = 1

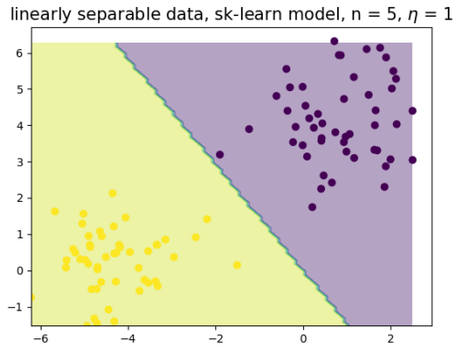


(d) Number of iterations = 10

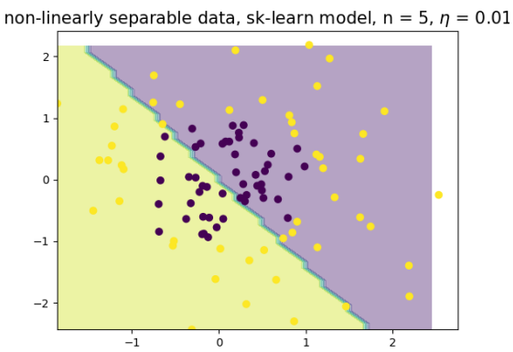
Variation of learning rates



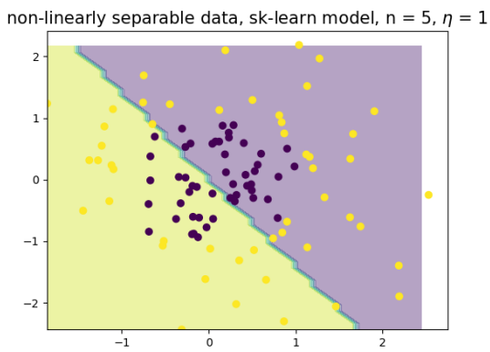
(a) Learning rate = 0.01



(b) Learning rate = 1



(c) Learning rate = 0.01



(d) Learning rate = 1

3.3.3 Discussion

Between the self-implemented and sk-learn model there is not much of a difference. The slight variation is probably caused by the sk-learn model randomly initializing the weights, while our model simply starts with a vector of zeros.

For the linearly separable dataset all variations of iterations, learning rate and perceptron model lead to a perfect classification, independently from any parameters. Even the absolute minimum of one iteration is sufficient. Therefore, the mean square error is always 0 and the classification accuracy always 100%.
For the non-linearly separable data the mean square error and the accuracy score always lie between 40% and 60%, no matter how many iterations are used. Exact values will not be given, because the graphs are sufficient to see that no useful 2D linear boundary can be found for this data, and about half of it will always be misclassified.

The non-linearly separable dataset cannot be separated with a linear decision boundary in this way. However, one solution would be to first transform the data into a 3-dimensional space, so it becomes linearly separable again. Then a 2-dimensional hyperplane can be determined with this kind of single perceptron model do find a decision boundary.