FYS-STK4155 - Applied data analysis and machine learning
Project 2 - Classification and Regression, from linear and logistic regression to neural networks

Mäntysalo A. Visa, Nyberg Fredrik and Zariouh Osama

November 2019

The material for project 1 can all be found at: `https://github.com/OsamaZa/Project2`

Abstract

This project is a study on both classification and regression. Classification will be done by implementing both logistic regression and a back propagation network to Taiwanese credit card data provided by UCI Machine Learning Repository [4], however, the regression analysis will only be done with the artificial neural network with data generated by a two-dimensional Franke's function. Benchmarking of the algorithms will be done by comparing with scikit-learn. Finally, the methods will be evaluated and the results will be compared with previous studies.

Introduction

Studies of complex systems may produce enormous amounts of data. There is therefore a need for reliable and efficient methods for extracting useful information, which then can be transformed into knowledge such that meaningful forecasts can be made. This translates into classification and regression problems within the discipline of computational machine learning.

Classification problems deal with making predictions that are discrete e.g. whether or not a credit card holder defaults their next payment. The data being dealt with in these kinds of problems consists of several continuous or discrete features e.g. amount of credit and gender. Regression analysis on the other hand deals with creating a curve of best fit as an output for a set of points. In this study classification will be done on credit card data, and the regression analysis will be done on data generated from Franke's function with added noise.

The classification of the credit card data will be done by implementing both logistic regression and a feed forward neural network with back propagation. Optimal parameters for the logistic regression will be found with a gradient descent algorithm. In order to find the optimal weights and biases for the neural network, the stochastic gradient descent is used. The information is fed in the forward direction of the network, and updates to the weights and biases are made in the backward direction accordingly to a cost function minimization problem. These methods will be benchmarked with algorithms found in scikit-learn. The performance of each method will be evaluated based on their accuracy score and confusion matrix.

For the regression analysis of the Franke's function, the same algorithm for the neural network will be applied with a new cost function and expression for the output. The R2-score and MSE will determine the quality of the fitting. This will also be benchmarked with algorithms found in scikit-learn. Finally, these results will be compared with previously obtained results done with ordinary least squares, ridge and lasso.

Before going into details, note that "defaulting" corresponds to predicting 1 and that "not defaulting" corresponds to predicting 0. This will be interchangeably used where it accommodates best.

Theory

Accuracy score

In a binary classification problem, the performance of the predictions made by a given algorithm can be measured with a accuracy score.

$$Accuracy = \frac{\Sigma_{i=0}^{n} I(t_i = y_i)}{n} \tag{1}$$

where I is an indicator function. I yields 1 if the target t_i is equal to the prediction y_i and 0 otherwise. The sum is normalized by dividing with the total number of targets n, hence if all the predictions equal their corresponding target, $Accuracy = 1$, the algorithm will be a perfect classifier for the given set of data.

Confusion matrix

A confusion matrix is a table that sums up how the trained model performs on test data which the true values are known. This gives a better indication of how well the model predicts the outcome. The output of a binary confusion matrix are how many 1's and 0's that are correctly predicted, and how many 1's and 0's that are wrongly predicted.

Precision of the different predictions can be defined as the amount of correct positive predictions divided on the total amount of true positive targets. Two precisions can be calculated, on for the positive predictions 1 and on for the negatively predictions 0's.

One hot encoder

Scikit learn's OneHotEncoder takes features which have discrete values as input, and makes a binary column for each discrete value. That means that if a feature has discrete values from 1 to 6, OneHotEncoder will split this column into 6 columns with values where the elements have values 0 or 1. And each column correspond to one of the discrete values. For each element in the columns, the column that corresponds to the original value in the original column, will be 1. The others will be 0.[3]

Logistic regression

Logistic regression is most commonly applied for systems with binary outcomes in classification problems. Given a set of discrete or continuous input data, the model is hence only going to assign the data into two categories with output 0 or 1, or a true or false. A function for mapping these input data is therefore needed.

There is choice between a hard or soft classifier when choosing this function, and it depends on the input data which type is more favourable for a given system [1]. An example of a hard classifier could be the sign function where $sign(x_i) = 0$ if $x_i < 0$ [1]. However, it is often useful to have a soft classifier where a function maps the probability of a data point being assigned to each output [1]. One such function is the logit function, also known as the Sigmoid function $p(x_i)$ [1].

$$p(x_i) = \frac{exp(x_i)}{1 + exp(x_i)} \tag{2}$$

This function transforms all inputs x_i into probabilities from 0 to 1. When fitting a Sigmoid function to a given classification problem, fitting parameters β are used as weights for the data x_i with labels y_i which are 0 or 1 [1].

$$p(y_i = 1|x_i, \beta) = \frac{exp(\beta_0 + \beta_1 x_i)}{1 + exp(\beta_0 + \beta_1 x_i)} \tag{3}$$

$$p(y_i = 0|x_i, \beta) = 1 - p(y_i = 1|x_i, \beta) \tag{4}$$

Utilising the Maximum Likelihood Estimation, one obtains an estimation for the total likelihood for all possible outcomes for y_i labeled x_1.

$$P((x_i, y_i)|\beta) = \Pi_{i=1}^{n} [p(y_i = 1|x_i, \beta)]^{y_i} [1 - p(y_i = 1|x_i, \beta)]^{1-y_i} \tag{5}$$

From $P((x_i, y_i)|\beta)$ the log-likelihood is obtained, which is the cost function $C(\beta)$ for logistic regression.

$$C(\beta) = \Sigma_{i=1}^{n} y_i log p(y_i|x_i, \beta) + (1 - y_i) log[1 - p(y_i = 1|x_i, \beta)] \tag{6}$$

$$C(\beta) = \Sigma_{i=1}^{n} y_i(\beta_0 + \beta_1 x_i) - log[1 + exp(beta_0 + beta_1 x_i)] \tag{7}$$

The expression of $C(\beta)$ becomes the cross entropy if one inserts a minus in front of the sum. This is then the expression to be minimized for optimizing the weights β for the best Sigmoid fit. In more compact notation the derivative can be expressed as follows.

$$\frac{\delta C(\beta)}{\delta \beta} = -X^T(y - p) \tag{8}$$

Here X contains the x_i values, y is a vector containing the y_i values, and lastly p is a vector containing the probabilities $p(y_i|x_i, \beta)$ .The equations are form Hjorth-Jensen [1]

Gradient descent method

When it comes to most problems in machine learning, there is almost always a cost function, $C(X, g(\beta))$, that needs to be minimized, hence an expression for the gradient of the cost function, $\nabla_\beta C(X, g(\beta))$, is needed [1]. However, as the cost function usually exist in high dimensions, an analytical expression for the parameters β for the minima are therefore challenging to obtain [1]. One such way to bypass this issue is by implementing gradient descent methods.

Gradient descent is an iterative process where $-\nabla_\beta C(X, g(\beta))$ is computed for some random initial β [1]. With this, the direction where $C(X, g(\beta))$ decreases the most in de multi dimensional surface is obtained [1]. This is then used to estimate the next β which the gradient of the cost function will be computed.

$$\beta_{k+1} = \beta_k - \eta \nabla_\beta C(X, g(\beta_k)) \tag{9}$$

A learning rate, η, is introduced to control how large each step in the desired direction is for each iteration, hence, it dictates the convergence of β. A sufficiently small γ will almost always result in $C(X, g(\beta_{k+1})) < C(X, g(\beta_k))$ [1]. A smaller learning rate increases computation time, too large on the other hand might not converge to a true result. One downside of this method is that the found minima can either be local or global.

The stochastic gradient descent method reduces the chance of getting stuck in a local minimum. When the dataset X contains a large amount of data it is useful to randomize and then split the data into so-called minibatches. Then the gradient can be approximated as the gradient computed with the data contained in each randomly picked minibatch, which again contains a randomised selection of the total number of data. An iteration over the number of minibatches is referred to as an epoch. It is normal to iterate over many epochs as well. [1]

$$\beta_{k+1} = \beta_j - \eta \Sigma_{i \epsilon B_k}^{n} \nabla_\beta C(x_i, \beta_j) \tag{10}$$

This does not result in gradients with the steepest descents, but drastically reduces the computational time for convergence, and the introduction of randomness also reduces the chance of landing in a local minima. [1]

Neural network

Artificial neural networks attempt to mimic the human brain. [1] A human brain is composed of billions of neurons that communicate with each other via electric signals. [1] It is believed that repeated patterns of signals between neurons result in stronger connections between the said neurons, and hence promotes learning or memory associated to what induced the signals in the first place. [1] Artificial neural networks are therefore able to learn without being programed explicitly to solve specific problems. The first and the simplest artificial network devised was the feed-forward neural network. [1]

Feed-forward neural networks have a layered design, input layer and output layer separated with one or more hidden layers, and information moves only in the forward direction through the layers. [1] The hidden layers consist of nodes which act as artificial neurons, and are connected to the nodes in the subsequent layers. [1] These artificial neurons activate and output accordingly to what is fed into them, the most basic model for these is called the perceptron [1]

$$Output = \begin{cases} 1, & \text{if } w \cdot x + b > 0 \\ 0, & \text{if } w \cdot x + b \leq 0 \end{cases}$$

Here w can describe the importance of the connections to the preceding layer and is often referred to as the weight, and b is the bias and is a measure on how easy it is for the perceptron to activate. As mentioned above a neural network often consists of several layers with several neurons, which means that there are a set of weights and biases between each layer in the network. Initially all the weights and biases are set to some random numbers. [1]

The training of a neural network is a form of supervised learning and can be done with a backpropagation algorithm. [1] When feeding in training data, the networks performance is evaluated by computing the error between the outputs from the output layer and the actual output. [1] This error is the cost function $C(w, b, x)$ of the network, which is to be minimized with respect to the preceding weights and biases. This minimization problem makes suggestions, with gradient descent methods, to what the weights and biases of the preceding layers should be changed to in order to minimize the output error. Hence, updating the weights and biases repeatedly with different data will produce better and better output, and the network will appear to be learning.

With the perceptron model a small change to certain weights or biases can totally flip whether or not a neuron is activated, and therefore change the ripple through the network in some complicated way, and hence also have a large impact on the final output. [1] It is therefore desirable to have the activations of neurons be in the span of 0 and 1 rather than strictly being 0 or 1. This way small changes in the weights and biases only cause small changes to the output.

This is done with so-called activation functions which squish the input of a weighted sum to some value between 0 and 1. [1] The Sigmoid function is one such function and was mentioned in the section about logistic regression. Very positive values of the weighted sum are transformed to be close to 1 and very negative weighted sums are transformed to be close to 0. The threshold for what should be considered a positive and negative weighted sum is controlled by the bias.

This type of transformations between the neuron inputs and outputs makes the interplay between layers more nuanced and subtle changes to the parameters w and b cause subtle changes to the output layer. When the network is exposed to sufficient amounts of training data for a given task, it will have optimized the weights and biases between the layers. The network will therefore appear to be learning, or rather memorising, how to treat given input data.

Method

A specific random seed(95) was used throughout the analysis. The first dataset used in the classification part of this project was credit card data provided by the UCI Machine Learning Repository. All the data that was used can be imported from `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`[4]. There are a total of 30000 observations with the binary targets “defaulting” or “not defaulting”, with approximately 22.12% corresponding to default payments. Each observation contains 2 explanatory variables, or features, such as the amount of the given credit, gender, education etc. which needed to be set up before analysis.

The data was read and set into a data frame with Pandas functionality. Not all observations had a complete set of features, but if all incomplete data was removed, only approximately 4000 would remain. It was therefore decided not to neglect any data based on feature completeness. The correlation matrix was plotted as a heatmap. Also the correlations between features and the target were plotted as a horizontal bar plot. Features with low correlation with the target, such as the 16th and 17th, were neglected. These features corresponded to the amount of bill statements in may and april respectively. The data was scaled and SciKit Learn’s OneHotEncoder was then implemented on the categorical features to ensure that the data was transformed to a binary column for each category. Since the majority of the data corresponds to non-defaulting cardholders, it was decided that the data should be balanced when creating the models. So the training data was balanced so that it consisted of equal amount of 0’s and 1’s. This was done to ensure that the model was trained for predicting both outcomes equally. This resulted in 8062 observations being used as training data. The correlation between all the features and the target is shown in figure 1 and the correlation matrix is shown in figure 2.

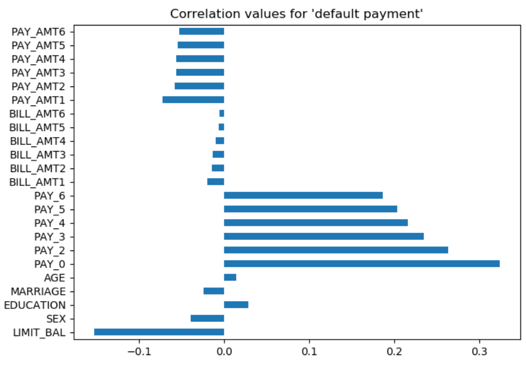

Figure 1: Horizontal bar-plot of the correlations between the features and the target.

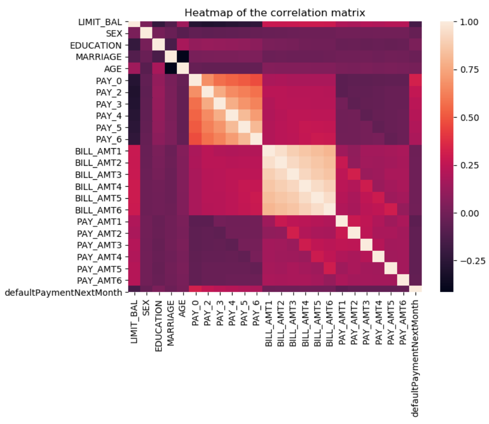

Figure 2: Heatmap of the correlation matrix.

The second dataset used in the regression part of this project was artificial data generated by the Franke's function. The Franke's function is given by the following expression

$$\begin{aligned} f(x,y) = {} & \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \\ & + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ & + \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \\ & - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) \qquad (11) \end{aligned}$$

and is defined for x,y $\in [0,1]$. The z values were computed with a grid of x and y values ranging from 0 to 1 with steps of 0.015 giving a total of 4489 data points.

Logistic regression

For logistic regression on the credit card data, the training data was used to find the optimal beta using the gradient descent method with 500 iterations, which gave a well converged beta. This beta was used to calculate probabilities for the test data. If the probability for a target was bigger than or equal to 0.5, the predicted target would be 1. If the probability was smaller than 0.5, the predicted target would be 0. This was done for different learning rates and penalty parameters and evaluated. The evaluation was done through accuracy score and confusion matrix. The confusion matrix is used just to make sure that the predictions is a mix of numbers. For example, around 77% of the targets is 0. So a prediction that only gives zeros will have an accuracy score of around 77%, even though it is not good. From this evaluation the optimal learning rates and penalty parameters were decided through a heatmap of the accuracy score. Logistic regression using scikit learn's logistic regression was also used for comparison.

Neural network

For classification using neural networks, the training data was as usual used to find the optimal weights and biases. This was done using a stochastic gradient descent with 50 epochs and mini batches at 100 data points. For each mini batch a feed forward neural network calculation was done as described in theory with the sigmoid function as the activation function. Then the back propagation algorithm was used to find updated weights and biases for each layer of nodes. After the neural network was trained, it was used on the test data to predict probabilities for the targets. 1 or 0 were decided as in the logistic regression case from the probabilities. The accuracy score and the confusion matrix were evaluated as in the logistic regression case to find the optimal penalty parameter and learning rate. Also for this method, the scikit learn's neural network (Miltil-layer-Perceptron-model package) with stochastic gradient descent was used for classification too, and the results were compared with results from the trained neural network. The neural network was altered for regression. The main changes are in the output layer and in the gradient of the output weight. In the output layer the outgoing signals are given raw, in the sense of not being evaluated by the sigmoid function. The gradient of the output layer has a change in the error between the training data and the calculated prediction. The new error is just the difference between the training data and the calculated prediction. Optimization of the regularization parameters and learning rates to find the highest R2 scores and lowest MSEs was done with a grid analysis. A heatmap for the accuracy score for classification, and R2-score for regression was plotted for several intervals for penalty parameter, λ, and learning rate, η, by trial and error.[2]

Results

Classification

The accuracy scores for the classification analysis, both with and without L1-regularization, for both logistic regression and the neural network can be found in table 1 to 8. Corresponding confusion matrices for all models, along with their precisions, can also be found in the prementioned tables. In figure 3 to 5 the error for each epoch of training is plotted. Other results can be found on `https://github.com/OsamaZa/Project2`.

Table 1: Result from model A_{NN} the Neural network classification with L1 penalty

Neural network with L1 penalty		
Highest accuracy with learningrate = 0.01 and penalty = 1e-8	SciKit	Developed algorithm
Accuracy	0.7553	0.7793
Correctly predicted 0's	3753	3988
Correctly predicted 1's	779	688
Wrongly predicted 0's	930	695
Wrongly predicted 1's	538	629
Prediction precision for 0's	0.8014	0.8516
Prediction precision for 1's	0.5914	0.5224

Table 2: Result from model B_{NN} the Neural network classification with L1 penalty

Neural network with L1 penalty		
Highest accuracy with learningrate = 1e-8 and penalty = 1e-5	SciKit	Developed algorithm
Accuracy	0.7553	0.7792
Correctly predicted 0's	3753	4674
Correctly predicted 1's	779	1
Wrongly predicted 0's	930	9
Wrongly predicted 1's	538	1316
Prediction precision for 0's	0.8014	0.9981
Prediction precision for 1's	0.5914	7.59e-4

Table 3: Result from model C_{NN} the Neural network classification without penalty

Neural network without penalty		
Highest accuracy with learningrate = 2.154e-7	SciKit	Developed algorithm
Accuracy	0.7543	0.7805
Correctly predicted 0's	3759	4683
Correctly predicted 1's	767	0
Wrongly predicted 0's	924	0
Wrongly predicted 1's	550	1317
Prediction precision for 0's	0.8027	1
Prediction precision for 1's	0.5824	0

Table 4: Result from model D_{NN} the Neural network classification without penalty

Neural network without penalty		
Highest accuracy with learningrate = 0.002154	SciKit	Developed algorithm
Accuracy	0.7543	0.7433
Correctly predicted 0's	3759	3723
Correctly predicted 1's	767	737
Wrongly predicted 0's	924	960
Wrongly predicted 1's	550	580
Prediction precision for 0's	0.8027	0.7950
Prediction precision for 1's	0.5824	0.5596

Table 5: Result from model A_{Log} the logistical classification with L1 penalty

Logistical regression with L1 penalty		
Highest accuracy with learningrate = 0.003162 and penalty = 0.1	SciKit	Developed algorithm
Accuracy	0.7642	0.812
Correctly predicted 0's	3849	4363
Correctly predicted 1's	736	509
Wrongly predicted 0's	834	320
Wrongly predicted 1's	581	808
Prediction precision for 0's	0.8219	0.9317
Prediction precision for 1's	0.5588	0.3865

Table 6: Result from model B_{Log} the logistical classification with L1 penalty

Logistical regression with L1 penalty		
Highest accuracy with learningrate = 0.0001 and penalty = 0.0001	SciKit	Developed algorithm
Accuracy	0.7642	0.7697
Correctly predicted 0's	3849	3875
Correctly predicted 1's	736	743
Wrongly predicted 0's	834	834
Wrongly predicted 1's	581	574
Prediction precision for 0's	0.8219	0.8275
Prediction precision for 1's	0.5588	0.5642

Table 7: Result from model C_{Log} the logistical classification without penalty

Logistical regression without penalty		
Highest accuracy with learningrate = 0.002154	SciKit-learn	Developed algorithm
Accuracy	0.7642	0.8048
Correctly predicted 0's	3849	4570
Correctly predicted 1's	736	259
Wrongly predicted 0's	834	113
Wrongly predicted 1's	581	1058
Prediction precision for 0's	0.8219	0.9759
Prediction precision for 1's	0.5588	0.1967

Table 8: Result from model D_{Log} the logistical classification without penalty

Logistical regression without penalty		
Highest accuracy with learningrate = 0.0001	SciKit	Developed algorithm
Accuracy	0.7642	0.765
Correctly predicted 0's	3849	3848
Correctly predicted 1's	736	742
Wrongly predicted 0's	834	835
Wrongly predicted 1's	581	575
Prediction precision for 0's	0.8219	0.8217
Prediction precision for 1's	0.5588	0.5634

Table 9: Result from regression with neural network.

Neural network regression		
	Learning rate = 1e-6 Penalty parameter = 0.01	Learning rate = 0.01
MSE	0.1628	0.1607
R2_score	0.8536	0.8512
Scikit-learn neural network regression		
MSE	0.02174	
R2_score	0.7176	

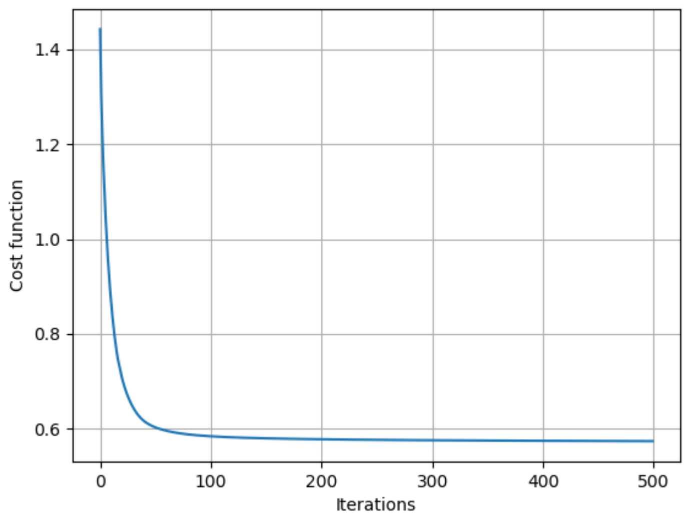

Figure 3: Cost function as a function of number of iterations for logistic regression.

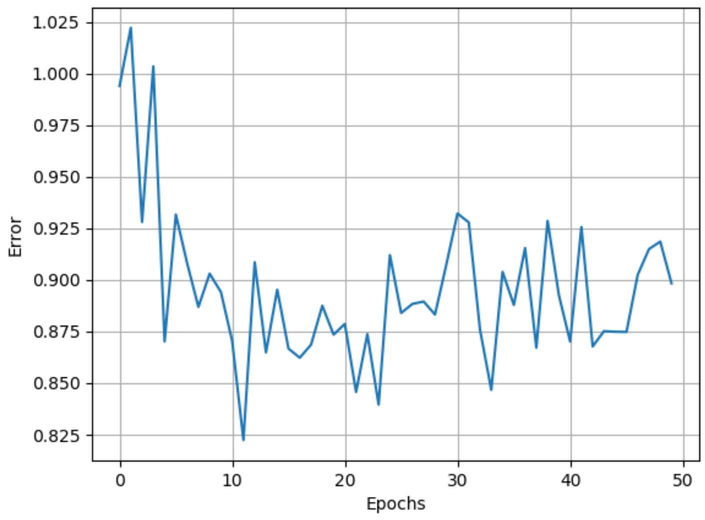

Figure 4: Error as a function epochs for neural networks on classification.

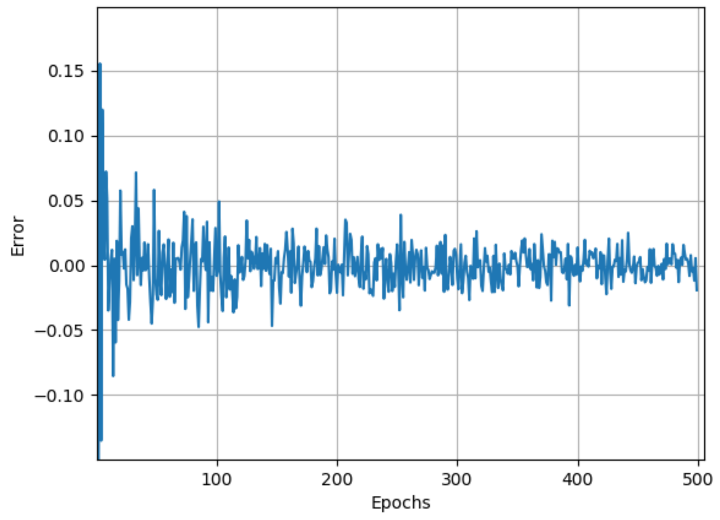

Figure 5: Error as a function of epochs for neural networks on regression.

Table 10: Accuracy scores from the best models and from article

	Article accuracy	Model A_{NN}	Model D_{NN}	Model B_{Log}	model D_{Log}
Logistical regression	0.82			0.7697	0.765
Neural network	0.83	0.7793	0.7433		

Table 11: MSE and R2-score for data made with Franke's function from project 1 where $\lambda_{Ridge} = 3.73 * 10^{-6}$ and $\lambda_{Lasso} = 3.16 * 10^{-8}$.

Franke's function	OLS	Ridge	Lasso
R2-score	0.946	0.969	0.965
MSE	0.00431	0.00247	0.00283

Discussion

Cost function

Figure 3 shows how the cost function reaches a minimum after a certain amount of iterations for logistic regression. The gradient is the true gradient, so it will always move towards lower values for small enough learning rates.

Figure 4 shows how the error goes as each epoch is finished for neural network on classification. It shows that the minimum error actually happens after a little above 10 epochs, and it is not getting better as more epochs is used. That is because the gradient is only an approximation using the data from each minibatch. This approximated gradient does obviously not always move the cost function towards lower values, but it converges around a certain limit.

The error as a function of epochs in the regression case using neural networks is shown in figure 5. It varies a lot less than in the classification case, and converges more towards a certain limit close to 0. That may be because the regression case is a lot less complex than the classification case, were there is 21 features used (two is neglected). For the regression case there is only two features, x and y. Also for regression, there is a smooth function that is being fitted. Therefore the gradient is probably less dependent of the points picked in each minibatch.

Classification

The models A_{NN} and B_{NN} from table 1 and 2 achieved the best accuracy scores for the neural network with regularization. At first glance, there appears to be no significant increase in performance by implementing a penalty parameter when comparing the accuracy scores. However, the confusion matrix for both models reveals that A_{NN} has a better precision in predicting 1's than B_{NN}. Almost all predictions done by model B_{NN} are 0. This means that model B_{NN} is faulty and is therefore discarded.

The models for neural network without regularization, C_{NN} and D_{NN} summed up in table 3 and 4, have the same pattern in the sense of that one model, C_{NN} only predicts "not defaulting". Therefore having a precision 0 for "defaulting". This model is also discarded. Therefore model, D_{NN}, was chosen for representing neural network without penalty. Model D_{NN} has a better precision for "defaulting" than A_{NN} and a lower precision for "not defaulting", but with a decrease in overall accuracy. It is therefore observed a increase in performance when implementing L1-regularization.

By comparison with scikit-learn neural network model A_{NN} gets a better accuracy. This improvement of accuracy comes from a higher precision on predicting "not defaulting", but the precision of predicting "defaulting" is lower. This seems not to be the case for the models without penalty parameter where the scikit-learn neural network is superior in both precisions.

For the logistic regression models, A_{Log} and B_{Log} from table 5 and 6 achieves the best accuracy scores with regularization. But as in the discussion for neural network classification the models have different precisions for both "defaulting" and "not defaulting". Model A_{Log} has an accuracy score of 0.812 and model B_{Log} an accuracy score of 0.7679. Model A_{Log} has a precision of 0.9317 for predicting 0's and a precision of 0.3865 for predicting 1's. Model B_{Log} has a precision of 0.8275 for predicting 0's and a precision of 0.5642 for predicting 1's. The choice of discarding model A_{Log} is not easy to make since the line between keeping a model and discarding it is vague.

The models C_{Log} and D_{Log}, which is tabulated in table 7 and 8, are logistical regression models without regularization. The accuracy for model C_{Log} is 0.8048. However, the confusion matrix reveals again that this model is not necessarily the most ideal. Model C_{Log} has a precision of 0.9759 for predicting 0's and a mere 0.1967 of predicting 1's. Model C_{Log} is therefore a model that should be discarded.

Comparing with scikit-learn's model, the regularized model A_{Log} and B_{Log} both outperforms scikit-learn's model when it comes to accuracy where model A_{Log} is worse in predicting 1's. But B_{Log} beats scikit-learn in precision, both for predicting "defaulting" and "not defaulting". Over all performance is increased when implementing L1-regularization.

From the results for the accuracy score there was little difference in performance between code made from scratch and code based on scikit-learn when it came to classifying the credit card data, where the code made from scratch gave better accuracy score. This was true for both logistic regression and feed forward neural network. The reason for this outperformance is probably due to that the scikit-learn model didn't get to converge properly.

Data comparison

The article has a better accuracy score in both the logistic regression and neural network. This can be seen in table 10. However the article don't use a confusion matrix, but an area ratio which is supposed to pick up some of the same points as an confusion matrix. The article also states that because of the high number of zeros, the accuracy score is not an reliable estimator of the model alone. Predictions done in this study also got an accuracy score at 81%, but was not used because of low precision of 1's. Since the article does not state what penalty parameter and learning rate it used, it is hard to compare too accurate.[5]

Also for the neural network case it is hard to compare too accurate since the article does not state a lot about how the neural network used is built up. It does not state how many layers is used, the number of nodes per layer, learning rate, penalty parameter or activation function. Also the method from the article splits the data into two equally sized sets, training data and test data.

Regression

The neural network falls short when comparing with the other regression methods such as OLS, ridge and lasso, as seen in table 11. This might be because the grid search was not fine enough to pick the parameters that would give a better fitting. The computations took way to much time for the neuron network. So a decision was made to make a rougher grid search. If there are parameters that makes the neural network better and competitive with the OLS, ridge or lasso methods from project 1, the time required to compute with a neural network will still weaken its competitive position.

Conclusion

For classification neural network was the algorithm that gave the best accuracy, but it was not by much. They both got an accuracy score of around 77, which is not great but okey. For regression, OLS, ridge and lasso all were a lot better than neural network, even though neural network was used on a Franke function without noise. Of these three the ridge method was the best. The pros of the logistic regression is that it is relatively fast, gives an well enough prediction and is relatively simple and intuitive compared to neural network. The cons about it, is that it wrongly predicts 23% of the targets and can't be used for regression. The pros about neural networks is that it can be used for both classification and regression, and that it predicted little bit better than logistical regression. The cons is that the calculations are time expensive, and do not predict a lot better than logistic regression in this classification case. And it predicts a lot weaker for the regression analysis compared to the linear regression methods. But maybe more complicated neural networks would have performed better.

Further work that could have been interesting to play out in this project is to vary the learning rate. The result from neural network might have been better if the learning rate began with a relative large value and shrink for each iteration in each epoch. Try to make the scikit-learn algorithms to completely converge or let them iterate more so a better convergence is reached. Adding a converges criteria in the developed code to make sure that it also converges.

Appendix

Heatmap of the grid searches.

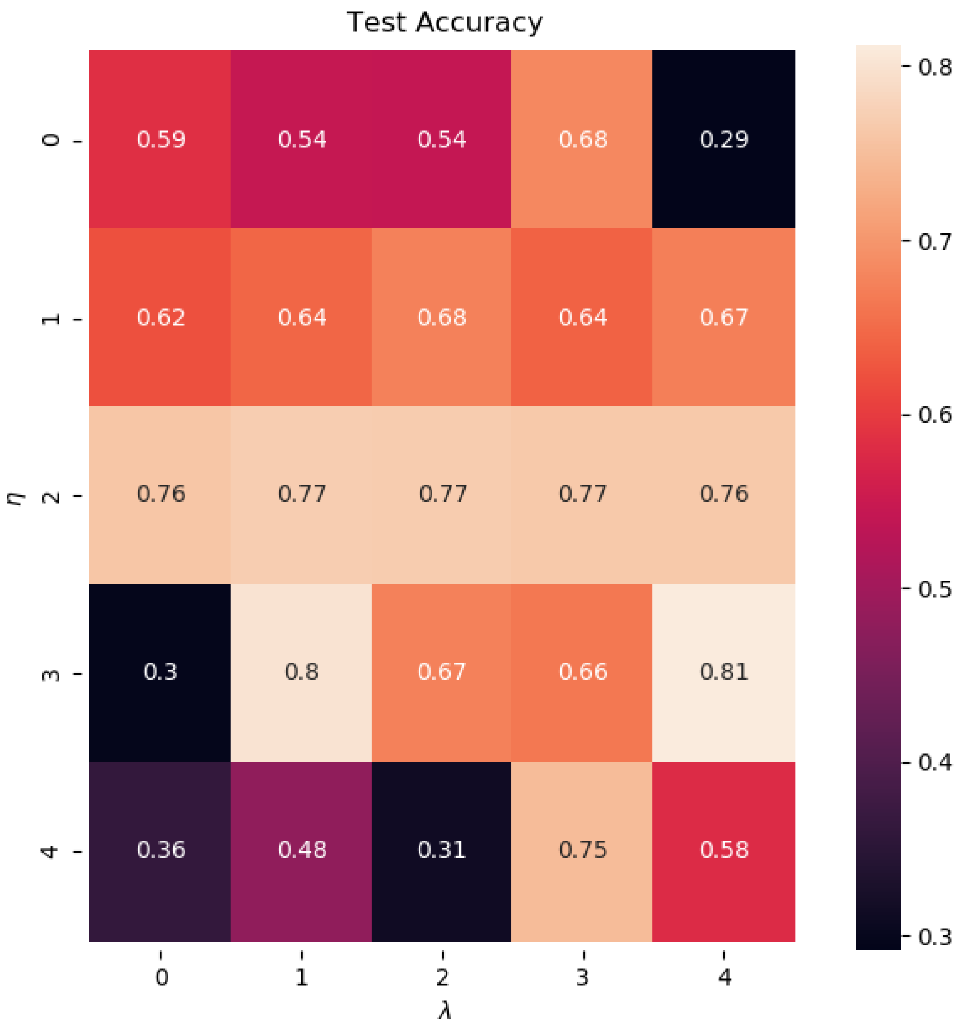

Figure 6: Heatmap of grid search for parameters in logistical regression.

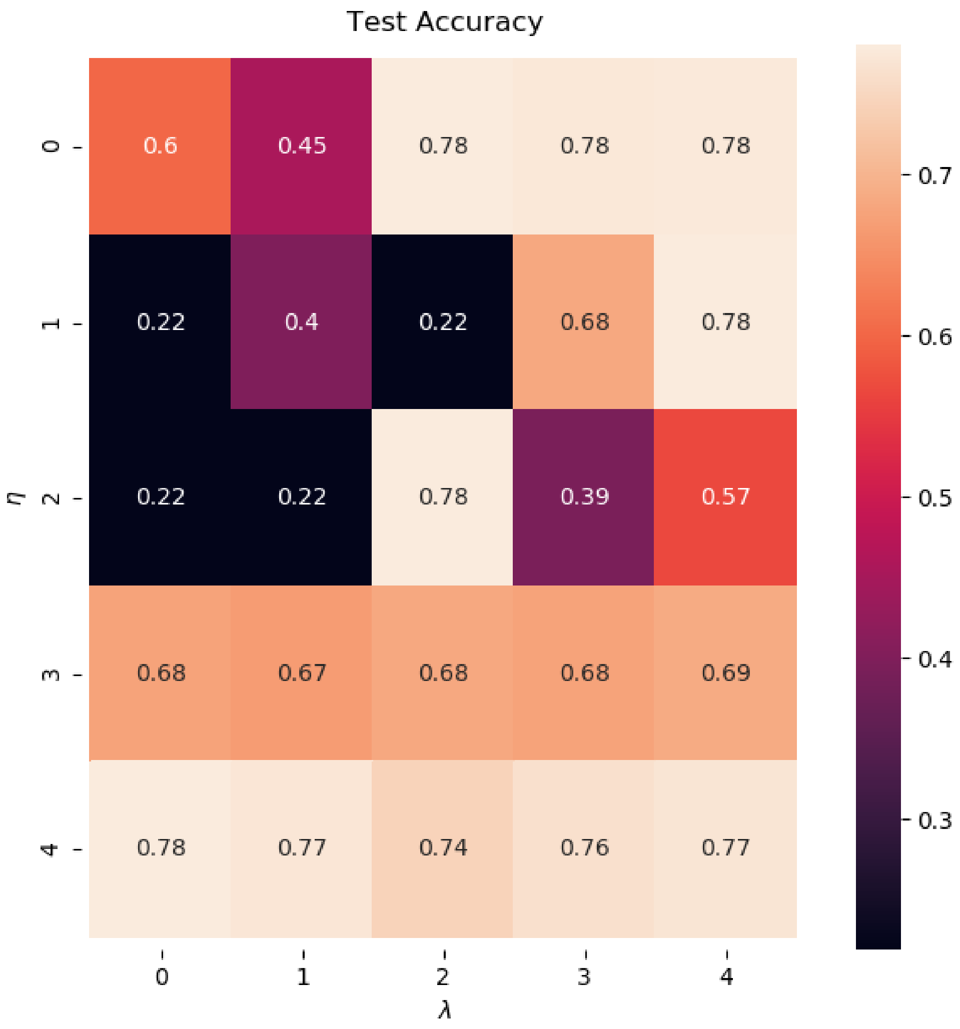

Figure 7: Heatmap of grid search for parameters in neural network classification.

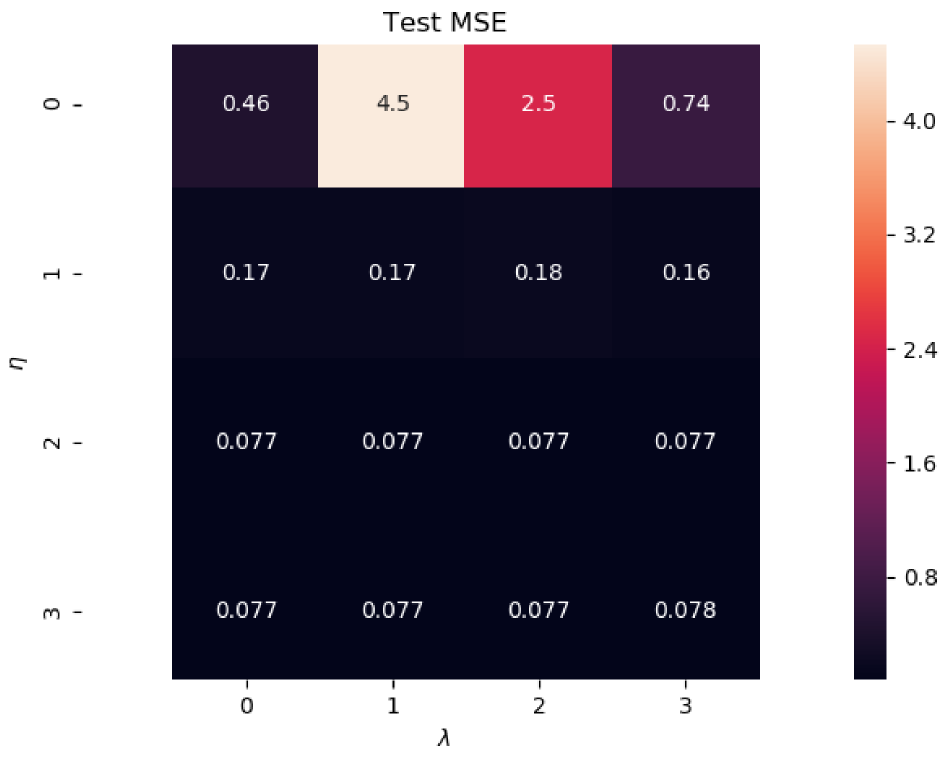

Figure 8: Heatmap of grid search for MSE in neural network regression.

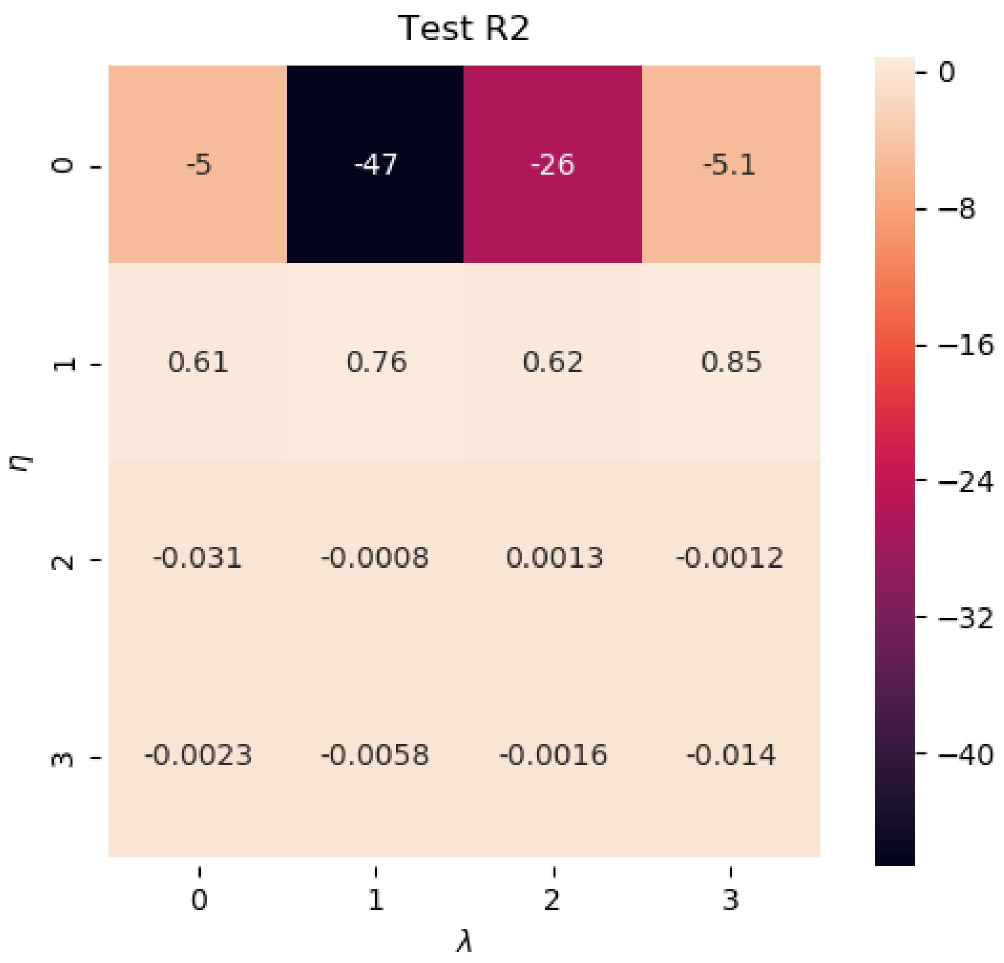

Figure 9: Heatmap of grid search for R2 in neural network regression.

References

[1] Morten Hjorth-Jensen. *Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis.* Sept. 2019. URL: `https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/Regression.html`.

[2] *MLP.* URL: `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`.

[3] *OneHotEncoder.* URL: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn-preprocessing-onehotencoder`.

[4] I-Cheng Yeh. *default of credit card clients Data Set.* 2009. URL: `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`.

[5] I-Cheng Yeh. “The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients”. In: *Expert Systems with Applications 36(2) 2473-2480* (2019).