FYS-STK3155/4155 Project 2: Comparisons of machine learning methods on classification and regression analysis

Michael Bitney and Magnus Grøndalen

November 10, 2019

Abstract

The material for this project can be found in our GitHub-repository at:
https://github.com/michaesb/machine_learn_2.git

1 Introduction

Machine learning is today a growing field, with numerous breakthrough that makes a desirable method of handling big data in numerous industries. For example credit card data is big data sets, which previously done by employees manually. A bank needs to take a risk every time they make a loan and too many bad loans is what happened in economic crisis in 2008. This shows the importance of classifying the data properly and we look at how machine learning can tackle this problem.
Here we will look at Credit Card data and first look at it for Logistic Regression and then look how accurate this model. We then train a Neural Network with the same data and look at the difference in performance. We will look at how the Neural Network performs on a Regression on on the Franke function and compare to Linear Regression. Our goal is see how well these perform on a given data sets and see what is the best for the given case.
We have structured this paper in Theory, where we explain the theory behind our Data sets and Methods used, Results for where we explain our parameter values and display the scores for the different methods. We then discuss the results and compare in Discussion and then come with our Conclusion.

—-Just drafts—-

1. Write about machine learning in general
2. Write about Logistic Regression
   (a) Theory
      i. Cost function
      ii. Accuracy
      iii. Design matrix
      iv. Gradient descent
   (b) Results
   (c) Compare with Scikit-Learns logistic regression
3. Write about Neural Network implementation
   (a) Basic Theory - Neurons - Layers - Weights - Bias
   (b) Cost function
      i. Cross-Entropy
      ii. Mean Squared Error
   (c) Activation function
      i. Sigmoid

ii. ReLU
iii. Softmax

(d) Regularization

(e) Feed Forward

(f) Gradient Descent

(g) Backpropagation

—-End of drafts—-

2 Theory

Regression Methods

A regression method is a method using a statistical process to determine the relationship between one or multiple input parameters. Logistic Regression is a case where the output is binary, like win/lose or yes/no. Linear Regression is a case where we look for a linear function based on the inputs to give a result which it tries to find a linear relationship.

Linear Regression

For more information on how Linear Regression works, we invite the reader to check out the previous report on Linear Regression Methods [5]. Here we use OLS (Ordinary Least squares method) from the report as it performed the best.

Logistic Regression

Logistic Regression is a statistical model that lets you, unlike linear regression, perform classification on data sets. Here we have design matrix X with a β that will be our model 1.

$$f_{model}(\beta) = X\beta \tag{1}$$

In order to find our beta here we need a cost function, which is equation 6. With this error function, we can use Newtons iterative method 3 to find minimum in the cost function.

$$\beta_{i+1} = \beta_i - \frac{dC(\beta_i)}{d\beta} / \frac{d^2C(\beta_i)}{d\beta^2} \tag{2}$$

If we derive our the derivatives of the cost function we get:

$$\beta_{i+1} = \beta_i - (X^T W X)^{-1}(X^T(\hat{p} - \hat{y})).$$

Since $(X^T W X)^{-1}$ is a very heavy procedure for the computer, especially since it need be done for every iteration, we will replace it with a variable γ.

$$\beta_{i+1} = \beta_i - \gamma X^T(\hat{p} - \hat{y}) \tag{3}$$

Equation 3 demands that we find an optimal value for γ. If γ is too low, we would need a very high number of iterations to find the minimum, but if γ is to high, we might skip a minimum point and not get the optimal value for β.

Neural Network

A neural network is a machine learning technique with inspiration from the brain. It uses so-called neurons, or nodes, that is a weighted sum of its input neurons. We have an input layer of neurons, an output layer of neurons, and then we have the hidden layers of neurons. A neural network will always have one input layer and one output layer, but might have an arbitrary number of hidden layers. Each of the neurons in one layer is connected via weighted edges to the neurons in the next layer. The goal of the neural network is to find weights that produce a wanted output. In figure 1 we can see the behavior of a neuron. This is further explained in the paragraph below. Lets take a look at the components of a neural network.

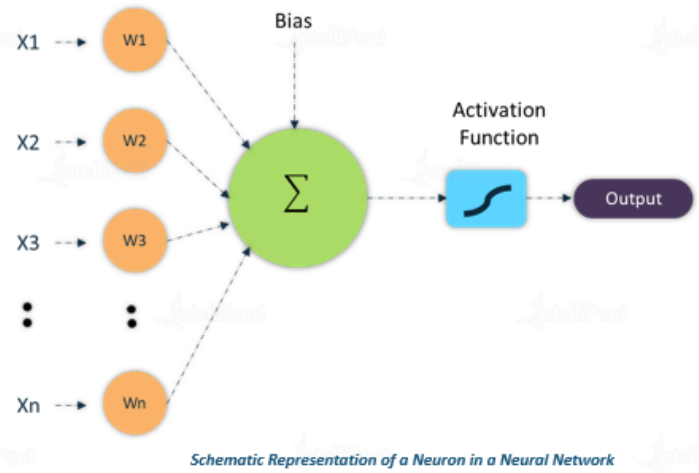

Figure 1: Figure showing the basics of a neuron in a neural network. Credit: https://intellipaat.com/

As stated above, a neural network has one input and one output layer, and an arbitrary number of hidden layers with an arbitrary number of neurons in each layer. Each neuron in one layer receives signals from all the neurons in the previous layer, and sends a signal to all the neurons in the next layer, much like a biological system. For a specific neuron y we have that

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) \tag{4}$$

, where n is the amount of neurons in the previous layer, x_i is the value of the i-th neuron in the previous layer, w_i is the weight of the 'wire' between neuron x_i and itself. At last, the $f()$ denotes the activation function of the neuron. This is the output of a neuron, and there exists multiple different activation functions for machine learning. Examples are the sigmoid function, the Rectified Linear Unit (ReLU) and the softmax function, and they are normally used in different scenarios. We will only look at the sigmoid function in this project. You can see the equation for the sigmoid function in (5). In figure 2 you can see the sigmoid function plotted with the step function. The step function is what we would like to approximate, but because we need the activation functions derivative, we choose the sigmoid.

$$f(t) = \frac{e^t}{1+e^t} = \frac{1}{1+e^{-t}}. \tag{5}$$

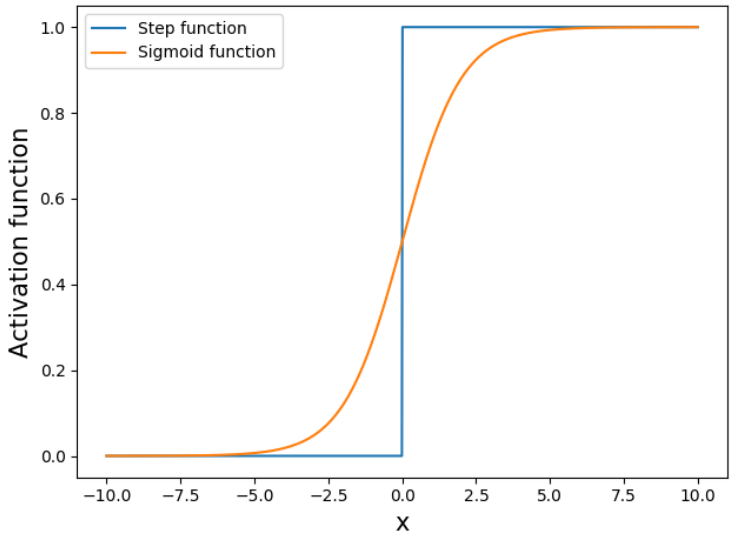

Figure 2: Figure showing the step function and the sigmoid function.

The method that takes the input variables and produces an output is called the feed forward method. It is responsible for adding the weighted sum forward through the network for each layer and neuron, through the activation functions, finally producing an output. In figure 1 you can see a visualization of this process for one neuron.

To validate whether our neural network is performing well, we need some kind of function that evaluates the fitness, or the error of the output from our neural network. This function is called the cost/loss function, and this is the function that we are trying to minimize. I.e. we want to optimize the weights and biases to minimize the cost function. There exists a variety of cost functions to choose from, but the simplest function is the Mean Squared Error (MSE) as you can see in equation (29). This cost function is probably the most popular and used cost function, but there is another cost function that is especially good in binary classification problems. This cost function is called the cross-entropy loss function, or the log likelihood, and the equation can be seen in (6).

$$C(y, \hat{y}) = -\sum_{i=1}^{n} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right). \tag{6}$$

Where y is the wanted output data, and $\hat{y}$ is the output predicted by our neural network.

When we have all these components as described above, i.e. a cost function, activation functions and feed forward method, we need to find a way to improve the performance of the neural network. This is done with something called backpropagation. It uses the errors at the output neurons, and "propagates" the error backwards through the network. To do this we need some kind of method that lets us change the weights to minimize the error from the cost function. One optimizer is the gradient descent, it lets you move the weights in the direction that decreases the cost function. First we look at the derivative of the cost function, using the cross-entropy loss in equation (6).

$$\frac{\partial C}{\partial \hat{y}} = -\sum_{i=1}^{n} \left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i} \right). \tag{7}$$

To explain the backpropagation, we first make some abbreviations:

- $\hat{y}$: Output layer after activation
- z_o: Output layer before activation
- w_o: Weights for output layer
- b_o: Bias for output layer
- a_h: Hidden layer after activation
- z_h: Hidden layer before activation

- w_h: Weights for hidden layer
- b_h: Bias for hidden layer
- X: Input layer

We then propagate backwards using the chain rule. We take one step backwards until we find the weights and biases. First we look at the activated output layer $\hat{y}$, which is equated using the activation function, such as the sigmoid function.

$$\hat{y} = \text{sigmoid}(z_o) = f(z_o). \tag{8}$$

Since we now are seeing the z_o, we want to describe the cost function as a derivative of this quantity, and we do so by derivating $\hat{y}$ by z_o and then using the chain rule.

$$\frac{\partial \hat{y}}{\partial z_o} = f(z_o) \cdot (1 - f(z_o)) = \hat{y} \cdot (1 - \hat{y}). \tag{9}$$

$$\frac{\partial C}{\partial z_o} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_o}. \tag{10}$$

Then we look deeper into z_o, by seeing how we equate it.

$$z_o = w_o \cdot a_h + b_o \tag{11}$$

We already now see the weights and biases for the output layer, and we can find the derivatives of the cost-function with respect to them. So we have $\partial C/\partial z_o$, so we need the $\partial z_o/\partial w_o$ which is just a_h, so we get

$$\frac{\partial C}{\partial w_o} = \frac{\partial C}{\partial z_o} \cdot a_h \tag{12}$$

For the bias we just need the $\partial z_o/\partial b_o$, which is 1, and we get

$$\frac{\partial C}{\partial b_o} = \frac{\partial C}{\partial z_o} \cdot 1 \tag{13}$$

Now we have the derivatives for both the weights and the biases in the output layer, so we now need to go deeper to find the derivatives for weights and bias in the hidden layer. We then need to look into a_h, and what it's made of.

$$\frac{\partial z_o}{\partial a_h} = w_o \tag{14}$$

We then again use the chain rule to find the impact a_h has on the cost function, and do the same as we did in the previous steps for $\hat{y}$

$$\frac{\partial C}{\partial a_h} = \frac{\partial C}{\partial z_o} \cdot w_o \tag{15}$$

$$a_h = f(z_h) \tag{16}$$

$$\frac{\partial a_h}{\partial z_h} = f'(z_h) = a_h \cdot (1 - a_h) \tag{17}$$

Using the chain rule once again to find the impact of z_h on the cost-function.

$$\frac{\partial C}{\partial z_h} = \frac{\partial C}{\partial a_h} \cdot \frac{\partial a_h}{\partial z_h} \tag{18}$$

$$z_h = w_h \cdot X + b_h \tag{19}$$

Finally, we found the weights and biases for the hidden layer, and we just repeat what we did at the output layer for the weights and bias.

$$\frac{\partial z_h}{\partial w_h} = X \tag{20}$$

$$\frac{\partial C}{\partial w_h} = \frac{\partial C}{\partial z_h} \cdot X \tag{21}$$

$$\frac{\partial C}{\partial b_o} = \frac{\partial C}{\partial z_o} \cdot 1 \tag{22}$$

Now we have found all the derivatives we need to update the weights and biases for the neural network. We also need to decide how much we want to change the weights and biases based on its derivatives, and we do that by using a learning rate γ. The final equations for the optimizing is

$$w_o = w_o - \gamma \cdot \frac{\partial C}{\partial w_o} \tag{23}$$

$$b_o = b_o - \gamma \cdot \frac{\partial C}{\partial b_o} \tag{24}$$

$$w_h = w_h - \gamma \cdot \frac{\partial C}{\partial w_h} \tag{25}$$

$$b_h = b_h - \gamma \cdot \frac{\partial C}{\partial b_h} \tag{26}$$

$$\tag{27}$$

Neural networks are often used for classification problems, but can also be used for regression analysis. Depending on whether you're solving classification or regression problems, there are some differences on how to set up your neural network. Lets take a look at the differences.

Classification vs. Regression

Using a neural network for classification problems means that you want the neural network to classify inputs as the correct category. This means for example classifying whether an image is showing a dog, a cat or a horse. If the output from your data says that dog is equal to 0, cat is equal to 1 and the horse is equal to 2, it is often favored to create three output neurons that is either 0 or 1, instead of one output neuron between 0 and 2.

Neural network for regression means you want to fit a function, or at least something linear or sort of continuous with your neural network. This means having just one output neuron. For example you could estimate real-estate housing prices with different attributes like size, location, number of bedrooms and so forth.

As already stated, the choice of cost function often vary between classification and regression problems. An all-round good cost-function is the Mean Squared Error in equation (29), and is used in both classification and regression. A popular cost-function for classification is the cross-entropy cost-function as you can see in equation (6). The backpropagation also needs to adapt to the cost-function of choice. To help decide whether the neural network is performing well, we have something called a score. For classification the score is calculated using the accuracy-score function

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(\hat{y}_i = y_i)}{n}. \tag{28}$$

Where $\hat{y}$ is our predicted output, y is our targets, and I is the indicator function, 1 if $\hat{y}_i = y_i$ and 0 otherwise. As you can imagine, this score-function doesn't apply to regression analysis, where we rather use something called the R^2 score function, as described in the equation (30). More on this topic in the section below.

Testing and modeling error

Classification accuracy score and cross-entropy loss

Wriiiite

MSE and R^2-score

In order to test our regression models, we will use multiple methods to determine how well our models fit the data. Here we use an expression of the Mean Squared Error. This tells us the average of all the errors squared, which gives us an idea on how good a model is.

$$MSE(\hat{y}, \hat{\tilde{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \mathbb{E}[(y - \tilde{y})^2] \tag{29}$$

Here the $\tilde{y}_i$ is the model, y_i is the data, and $\mathbb{E}[]$ is the expectation value of an expression.

We will use an R^2-score function, which tells us how well the fitted line is to the data. The best score is 1 and the worst score is $-\infty$. An R^2 score of 0 is a straight line at the mean, which is considered noise. See equation (30) for the formula.

$$R^2(\hat{y},\tilde{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{30}$$

Franke function

The Franke function is a common test for regression analysis with two inputs.

$$f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \tag{31}$$

$$+ \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \tag{32}$$

$$+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \tag{33}$$

$$- \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right). \tag{34}$$

This function is a common test for regression analysis and you can see the shape function in plot 3

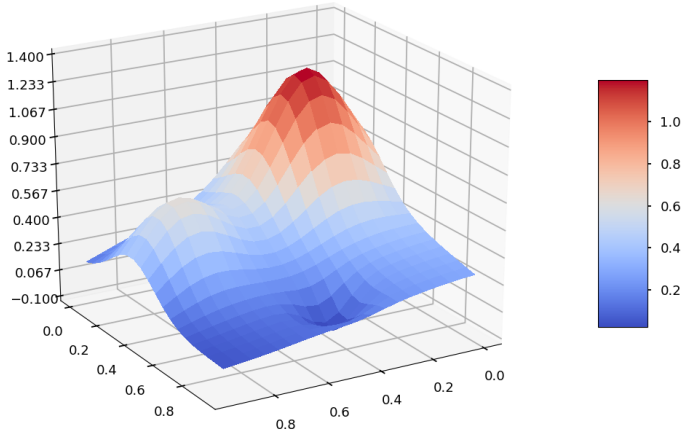

Figure 3: This plot shows the Franke function in a 3D plot

Credit Card data

Credit card data is used in economics to determine if an individual is suited to receive a loan from a bank. A loan is at a risk for the bank, although the bank is will demand more than the loaned amount, because might not be able to pay back the loaned amount, because of anything from economic succession to purchasing habits to to. A failure to pay back the loan is called a default. Credit Card data can provide a background to whether a person resources to determine if a person has the necessary resources to pay the loan back. Before employees had to go through this information, but now we can use machine learning to solve this problem.

3 Results

Data processing: Credit Card data

For our logistic Regression and Neural net we have used the Credit Card data. This Credit card data is credit card information from individuals from Taiwan and it shows information about an individual and whether they pay their bill or default. This data was retrieved from the site `https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients#` and is a common example for Logistic Machine Learning Algorithms. We have a description of the data given on the site and in our repository. We saw however that

in the data that the were certain classifications that was not mentioned in the description. We decided to remove these individuals from the the data set and focus on the ones we can identify. We then have roughly 23 000 individual to process. The data contain attributes like gender, payment history, education and more. These attributes har the type of information given to the function.

Logistic Regression

We constructed our Logistic Regression function using Gradient descent, but we needed adjust the parameters learning rate and number of iterations to find good model for the credit card data. We plotted multiple learning rates to try find the best one in graph 4

Figure 4: Here we have plotted the accuracy vs the learning rate with 20 iterations in order to plot for many learning rates. Note the spike between 0 and 5

We saw a spike between 0 and 5 and decided to zoom on this area in graph 5

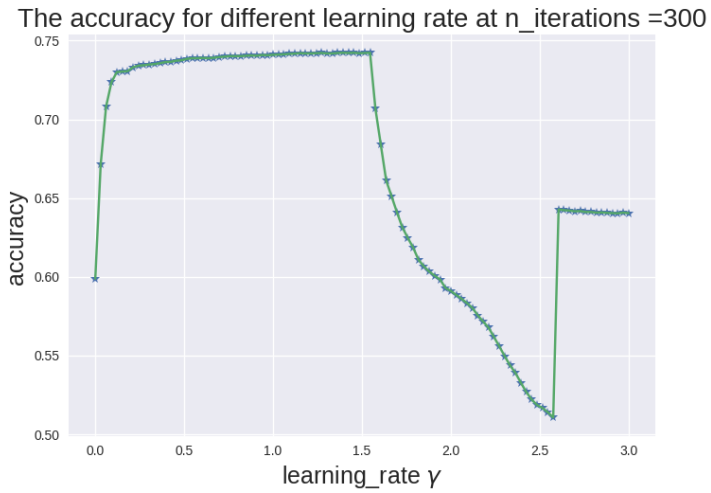

Figure 5: Here we see the a zoomed up version of graph 4 but with a higher iterations at 200.

We saw that the optimal value for the learning rate is in the interval [1.3,1.5]

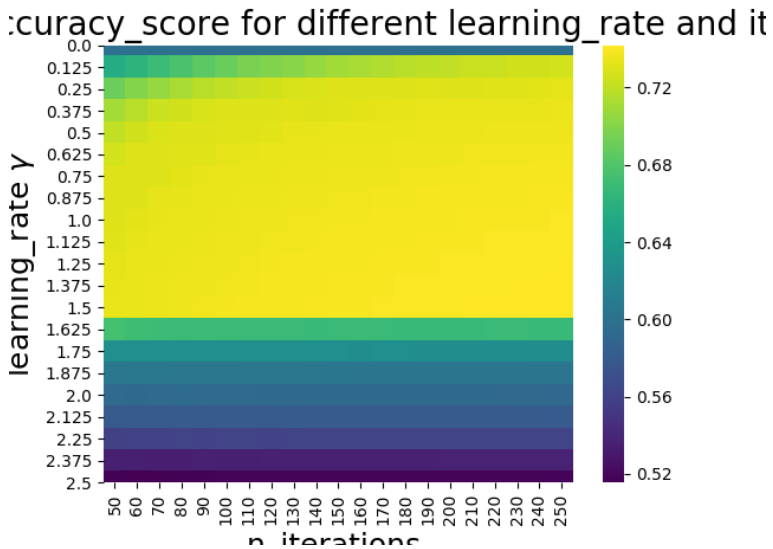

Figure 6: This is a heatmap of the accuracy where the x-axis is the learning rate and y-axis is number of iterations.

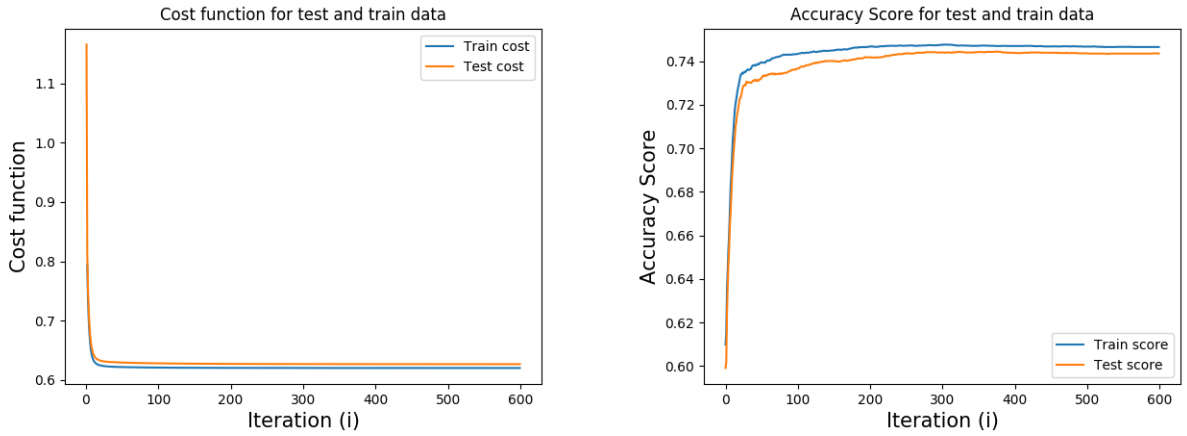

Figure 7: Figure to the left shows the cost/loss through the iterations of the gradient descent, and the figure to the right shows the accuracy scores throughout the gradient descent

Neural Network for Classification

We made our neural network with only one layer. We used the gradient descent method to find the minimum point using the cross-entropy cost function. We found it difficult to find the optimal parameter values, since they all impact it in a different way and it didn't appeared to have any obvious pattern. After trial and error we found the best learning rate to be at 0.0001 and number of neurons/features at 80.

In figure 8 we used a learning rate that was way to big, but in figure 9 we see the loss is gradually getting better, as well as the accuracy score.

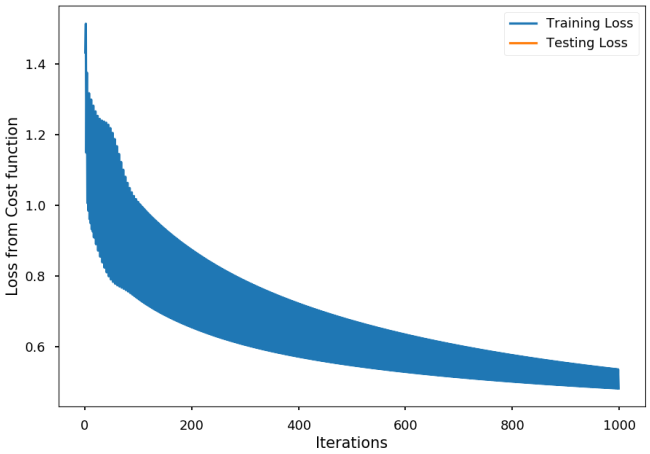

Figure 8: In this figure we see the Cost function through the gradient descent with a learning rate being too large.

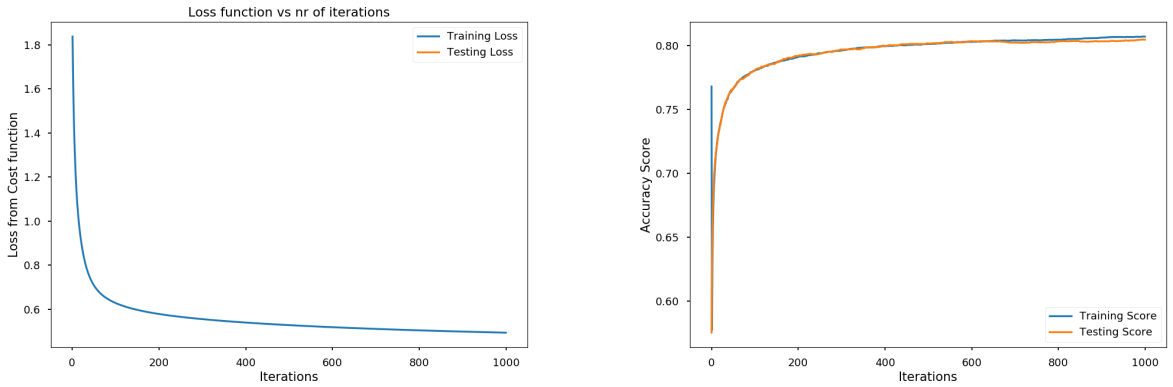

Figure 9: This is the performance for the Neural Network on the credit card data. Figure to the left shows the cost/loss through the iterations of the gradient descent, and the figure to the right shows the accuracy scores throughout the gradient descent.

Neural Network for Regression

In the regression analysis using our Neural Network, we only used one hidden layer, and varied the number of neurons, the L2 penalty and the learning rate. We used the Mean Squared Error as our cost-function, and the sigmoid function again as the activation functions for all the layers. We used the regular gradient descent also in this neural network, but would probably benefit from using a stochastic gradient descent or mini-batches.

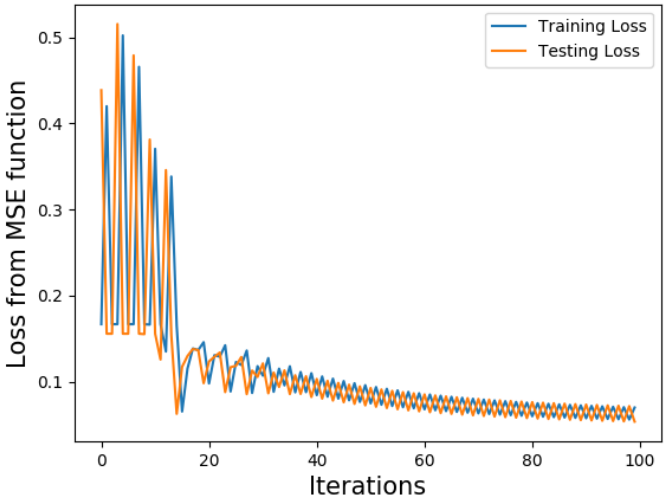

Figure 10: In this figure we see the learning process for our neural network for regression. Here we are using a high learning rate.

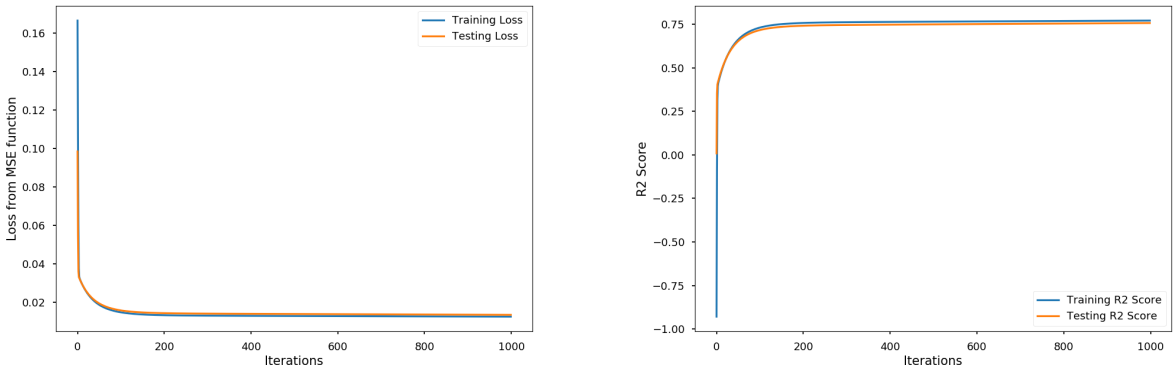

Figure 11: In this figure we can see the learning process of using gradient descent in our neural network, with the cost-function being on the left, and the accuracy score on the right.

Comparison

In order to compare our neural network, we utilized the scikit-learn package to create a neural network. The difference between the package we used from scikitlearn and our own neuralnet code, is that they have you are not limited to one layer and can use an adaptive learning rate. We will compare the scikitlearn package with one layer and multiple layers.

Classifying

Method	Accuracy
Logistic Regression	0.743
Logistic Regression Scikit-Learn	0.817
Neural Network	0.819
Neural Network Scikit-Learn (single layer)	0.823
Neural Network Scikit-Learn (4 layer)	0.822

Table 1: Here's the accuracy and score for the Credit card data for multiple methods. Note that Scikit-Learn has only an initial learning rate and will be adjusted as the program progress.

To decide how well our classification methods really behaved, and for comparing the different methods, we decided to look into the cumulative gain charts. We made use of Scikit-Plot's method `plot_cumulative_gain` to find these figures in 12. By finding the area between the baseline graph and our "Default"-line, relative to the area between the baseline graph and the "best" graph, we can compare the different classification techniques.

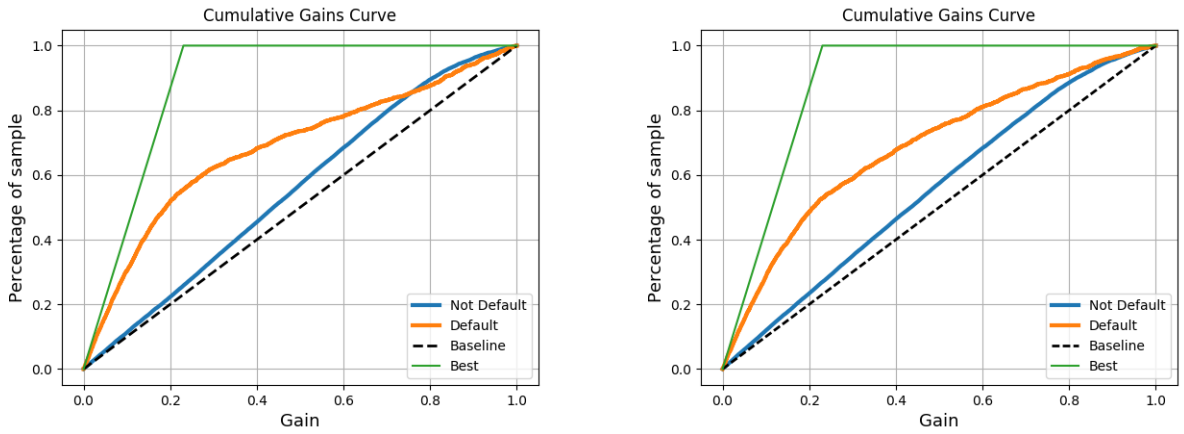

Figure 12: These figures show us the cumulative gain graph of Logistic Regression and Neural Network respectively. The area for the Logistic Regression was 0.478 and the area for our Neural Network was 0.559

Regression

Method	R^2-Score	MSE
OLS	0.978	1.98e-3
Neural Network	0.821	0.040
Neural Network Scikit-Learn (Single layer)	0.974	0.022
Neural Network Scikit-Learn (four layers)	0.9997	1.25e-3

Table 2: Here the mean squared error and the accuracy for different methods on the Franke Function. They were each given a mesh grid of 50 x 50 points and all have been spent time to find the best score by adjusting the values. Note that the Neural network code that we designed has only one hidden layer and scikit-learn has the option to use one or multiple layers. Note that the four layer Neural Network needed a lot of time to run. A seed value was used to get the same values for all methods

4 Discussion

Logistic Regression

We can see that our Logistic Regression works based on plots 7 . Here we can see as the program progress the cost function increases and the loss function decreases. This indicates that the program is adjusting the β -values to minimize the cost function, which is the desired value. We can also see that the test and train

values are very close to each other on both graphs. We can see on the loss function function plotted that train is below test, which is due to the test data being a whole new data source. We can also see that the accuracy is better for train for test as well. We might get some overfitting, since test has a higher accuracy, but it seems to be very low in this case, since test and train are very close to each other. It could worth looking into doing cross-validation on the data, to avoid overfitting in the data, especially if one intends to run more iterations with.
We used our learning rate from the heatmap 6 and gives us a model at a accuracy at roughly 0.75. It is possible that we have missed a better value for the learning rate somewhere, because we did a very low-resolution sweep. So it's likely that this result can be improved upon with a better a machine with better computational power. One thing to consider is that our iterative method might be stuck in a local minima. In order to avoid this we can use mini-batches and a stochastic gradient to avoid this local minima trap. It's difficult if not impossible to see with a regular gradient if this happens, so this shows more potential to use a stochastic gradient descent instead to find a better model.

For deciding the error of our Logistic Regression classification, we used the cross-entropy as our cost-function. We used the cross-entropy without any regularization terms, such as L_1 and L_2 penalties that is used in Ridge and Lasso regression. We would suggest adding regularization terms to the cost-function, when we would have probably increased our accuracy by doing so.

Neural network for classification

what component we uses, why we used them,
should have u

Neural net for regression

Comparison of methods

In table 1 we have displayed the accuracy score each of the method. We can see that Logistic Regression performed worst out of the methods. Scikit-learns Logistic Regression performed much better. We couldn't specify the learning rate in the scikit-learn package, which might infer it might have an adpaptive learning method or some more advanced method underneath. Unfortunately it is a bit of a black box, but it gave us a better result than our own package.
Our Neural Network preformed however much better than the logistic Regression. This was with one single layer and shows that with more improvements it be better than logistic regression. The Scikit-learn package performed better then our own code by a little. Here we could see that adding more layers to the network did not improve the performance. However this was not thoroughly looked into, because of the sheer number of permutations, we might have missed a good parameter value here and doing a systematic grid search could possibly gives us a better model.

We can see at cumulative gain graph 12 that Logistic regression has a higher increase in the beginning. This could indicate that Logistic Regression is better at identifying the clear cases with higher probability. However Neural Network seemed to better at the cases with lower probabilities and was all around better for all cases. However this difference is hard to see in the graph, but is an interesting effect that is worth looking into. We calculated the area under the curve as well, with Neural Network being higher the Logistic Regressor and this fits well with what the accuracy score implies.

In the previous report [5] we have looked at Regression method OLS and you can see the comparison between them in 2. Here we can see that OLS performs better than the Neural Network, than our Neural Network and Scikit-learn Network (single layer). OLS gives a very good model with the points given, but Neural Network might need more data points of the Franke function. However the four layer Neural Network get score of 0.9997. This took a long time, but this is a very accurate model. Unfortunatly we could not increase due to computational constraints, but shows for our case with the Franke function is very accurate.

5 Conclusion

We created both a Logistic Regressor and two neural networks one for classifying data and for handling terrain data and compared it to scikit-learns package for these method. On the Credit card data we found that the neural network performed better than the Logistic Regression. The scikit-learn version of the Neural Network performed the best here with an accuracy score of 0.823.

On the Franke function Ordinary least squared performed better than the regular neural network, except a four layer scikit-learn Neural Network which gave us an accuracy score of 0.9997, although this took a

large amount of time. We saw that Neural Networks generally needs more data points than linear regression and OLS is concluded to be generally a better method in this case, despite having a slightly lower score due computational strength needed and simplicity.

We recommend to try a stochastic gradient with mini-batches on the neural network and using cross-validation to check for overfitting on iterative method. We also recommend looking into our Neural Network with high performance computers in order to find better parameters for all methods, to improve the models more.

References

[1] Lecture Notes made by Morten Hjorth-Jensen in class Fys-Stk4155/3155 at uio
`https://compphysics.github.io/MachineLearning/doc/web/course.html`

[2] Hastie Tibshiran Friedman; The Elements of Statistical Learning; Second edition; Springer 2009

[3] Géron, Aurélien; Hands-On Machine-Learning with Scikit-Learn & TensorFlow; First edition; O'Reilly 2017

[4] Yeh, Lien; The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients;

[5] Grøndalen Bitney; Project Regression for the subject FYS-STK3155/4155
`https://github.uio.no/michaesb/ml_project1_mms/blob/master/report_fysstk_magnubgr_michaesb.pdf`

Acknowledgements

Thanks to Morten Hjorth-Jensen and student teachers in FYS-STK-4155/3155 for help during this project.