Report Project 1

Erlend Øydvin
(Dated: November 20, 2021)

Abstract
Ordinary least square regression, Ridge regression and Lasso regression has been fitted to terrain data. According to the results Ridge regression is best suited to predict terrain data.

I. INTRODUCTION

In this report we will study a regression problem and a classification problem using a neural network. The main objective is to develop and test a neural network. The regression neural network are compared with the OLS and Ridge regression techniques used in project 1 and the classification neural network are compared with a logistic regression technique. The different methods are applied against Franke's function and the Wisconsin breast Cancer dataset. When analysing the regression problem the main challenge is to distinguish noise from the function that created the dataset, thus noise has been added to Franke's function. For the classification problem the main challenge is to find the right weights that fits the dataset best. Both the regression problem and the classification problem has been ranked using k-fold cross validation and R2 score. The MSE score was omitted in this report as it gives the same information as the R2 score, lowest MSE corresponds to the highest R2 score.

The pyhton code can be found at the following git-repository: `https://github.com/erlnwind/FYS-STK4155/tree/main/Project_2`

II. THEORY

The main goal for the regression problem is to minimize the following cost-function

$$C(\vec{\beta}) = \frac{1}{n}\sum_{i=1}^{n-1}(y_i - \tilde{y_i})^2 + \lambda\beta^2 \tag{1}$$

under certain constrains. $\vec{\beta}$ is the parameters to be minimized, y contains reference data and $\tilde{y}$ contains an approximation to y. λ is a regularization parameter and determines the constraint on β. Equation 23 is also called the mean sum of error squared. The main goal for the classification problem is to minimize the cross entropy given by

$$\begin{aligned}\mathcal{C}(\vec{\beta}) = -\sum_{i=1}^{n} y_i \ln[P(y_i = 0)] + \\ (1 - y_i)\ln[1 - P(y_i = 0)] + \lambda\frac{1}{2}\beta^2,\end{aligned} \tag{2}$$

where the logistic function is defined by

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

OLS and Ridge

In project 1 it was shown that an analytical solution to the OLS and Ridge regression problem could be found by solving the following equation

$$\vec{\beta} = (A^T A - \lambda I)^{-1} A^T \vec{y} \tag{4}$$

where the new parameter λ is a parameter that determines the shrinking effect on β. Setting λ equal zero amounts to finding the OLS solution. I is the identity matrix.

Logistic regression

In OLS and Ridge regression we seek to find a continuous linear function $f(\beta)$ that best describe our data. In many problems the goal is to classify observations into groups. This can be done with logistic regression. The main goal in logistic regression is to minimize Equation 2 with the logistic function defined by Equation 3. By taking the derivative of Equation 2 with respect to its parameters β yields

$$\frac{\partial \mathcal{C}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -\boldsymbol{X}^T(\boldsymbol{y} - \boldsymbol{p}) + +\lambda\boldsymbol{\beta} \tag{5}$$

This expression can be used to minimize the cost function by using for instance an gradient descent algorithm. In this hand-in we have developed an Stochastic gradient descent algorithm that were used for this purpose.

Stochastic gradient descent - SGD

In many cases it is too much data or simply not possible to find an analytic expression that minimizes Equation 23 and Equation 2. Stocastich gradient descent is an iterative method for finding the optimal value for β in Equation 23 and Equation 2. It works by computing the derivative of the cost function based on randomly drawn data from the total dataset, called a minibatch. SGD has the advantage that we reduce the chance of getting stuck in a local minimum and it can increase the speed at which we converge towards the optimum. The SGD algorithm is given by

$$\vec{\beta}_{new} = \vec{\beta}_{old} - \gamma\nabla C(\vec{\beta}_{old}), \tag{6}$$

where γ is the learning rate and $\nabla C(\vec{\beta}_{old})$ is the derivative of the cost function. The derivative of the cost function can sometimes be hard to find an analytical expression for. In such cases it can be found using an numerical differentiation package, for instance autograd. Adding a momentum term

to Equation 6 can increase the performance of the minimization

$$\vec{v}_{new} = \vec{v}_{old}\alpha - \gamma \nabla C(\vec{\beta}_{old}) \tag{7}$$

$$\vec{\beta}_{new} = \vec{\beta}_{old} + \vec{v}_{new}, \tag{8}$$

where the added parameter α is a number between 0 and 1 and determines the momentum effect. The momentum effect serves as a memory of the direction in witch we are moving, it helps the gradient descent gain momentum in a persistent direction, not being too affected by oscillations caused by changing curvatures.

Neural network

A central part of this project was to develop a Feed Forward Neural Network. The FFNN was used to solve the regression and the logistic problem. The first step in a FFNN is to compute the weighted sum of the input features for each neuron in the input layer as follows

$$z_j^l = \sum_{i=1}^{F} w_{ij}^l x_i + b_j^l. \tag{9}$$

Then each input z_j is passed through a activation function

$$a_j^l = f(z_j^l). \tag{10}$$

Then this process is repeated for every layer l in the network up to the output layer denoted by L. In the output layer z_j^L is calculated for each neuron and then passed to the output function. For the regression problem the output function is given by the linear function

$$a_j^L = f(z_j^L) = z_j^L, \tag{11}$$

while for the classification problem the output function is given by the softmax function

$$a_j^L = \frac{\exp\{(z_j^L)\}}{\sum_{c=0}^{C-1} \exp\{(z_c^L)\}} \tag{12}$$

After doing doing the feed forward we compute the predicted error and update the weights accordingly. This is done with the back-propagation algorithm. The first step in back-propagation is to calculate

$$\delta_j^L = f'(z_j^L)\frac{\partial \mathcal{C}}{\partial (a_j^L)} \tag{13}$$

for the output layer where the cost function C is given by Equation 23 and $f'(z_j^L)$ is the derivative of the activation function. Then the next steps is to compute

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l) \tag{14}$$

for all layers back to the input layer. Back-propagation constructs new weights and we can update the weigths and biases as follows

$$\begin{aligned} w_{jk}^l &\leftarrow= w_{jk}^l - \eta \delta_j^l a_k^{l-1}, \\ b_j^l &\leftarrow b_j^l - \eta \frac{\partial \mathcal{C}}{\partial b_j^l} = b_j^l - \eta \delta_j^l, \end{aligned} \tag{15}$$

III. METHOD

Error measurement

In this report we measure the performance of the regression techniques using R2 score. The R2 score can be computed by

$$R^2(\vec{\beta}) = 1 - \frac{\sum_{i=1}^{n-1} (y_i - \tilde{y}_i)^2}{\sum_{i=1}^{n-1} (y_i - \bar{y}_i)^2} \tag{16}$$

where y_i is the real value for the data i, $\tilde{y}_i$ is the model predicted value and $\bar{y}_i$ is the mean value of all y_i in test. The performance of the classification problem was measured using

$$accuracy = \frac{\sum_{i=1}^{n-1} I\left(t_i = y_i\right)}{n} \tag{17}$$

where t_i is the predicted classification and y_i is the correct classification. The MSE was not computed in this hand-in at it indicates the same optimum as the R2 score.

The regression problem

The Stochastic gradient descent algorithm was tested with OLS and Ridge regression fitting a p-order polynomial on the form

$$z(x,y) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 y^2 + \beta_5 xy ... \tag{18}$$

This can be written as

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & \dots \\ 1 & x_1 & y_2 & x_1^2 & x_1y_2 & y_2^2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots \\ 1 & x_2 & y_1 & x_2^2 & x_2y_1 & y_1^2 & \dots \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots \\ 1 & x_N & y_N & x_N^2 & x_Ny_N & y_N^2 & \dots \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ \vdots \end{bmatrix} \tag{19}$$

Solving this equation boils down to finding the β coefficients that best describe the observations in z thus minimizing Equation (23). The regression methods was tested against a two dimensional function called Franke's function. Frankes function is a weighted sum of exponentials and goes as follows

$$\begin{aligned} f(x,y) = &\frac{3}{4} exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) \\ &+ \frac{3}{4} exp\left(-\frac{(9x+2)^2}{49} - \frac{(9y+1)^2}{10}\right) \\ &+ \frac{1}{2} exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) \\ &- \frac{1}{5} exp\left(-(9x-4)^2 - (9y-7)^2\right). \end{aligned} \tag{20}$$

The results from OLS and Ridge was then compared to the results from the neural network. The neural network

does not fit a polynomial to the surface, but is connected directly to each data i from the total dataset and is then trained to best reproduce the data. The hope is that it will, given the right architecture, converge towards the exact Frankes Function, not replicating noise. In the output layer the output function was set to the linear function given by Equation 11.

The classification problem

For the classification problem I have studied the Wisconsin Breast Cancer data set. It is a binary classification problem that lists 9 attributes of breast cancer cells and then an output value 0 for benign (not cancer) and malignant (cancer). The dataset was first analysed using logistic regression by finding the weights that minimize Equation 5. Afterwards the dataset was analysed using the neural network. Since this is a binary problem the output function was set to the soft-max function described by Equation 12. The logistic function poses a simple and intuitive weighting of the features, thus if there exits a good fit for the dataset we can (hopefully) understand more or less directly which features that were important and witch features that were less important in the decision. This comes under the assumption that there exits such an relationship. The neural network on the other hand have the ability to mimic and predict complex relationships between the features, which are not possible to do using logistic regression. However, if the neural network becomes too complex it also becomes too complex to make an intuitive understanding of the solution.

Hidden layers and activation functions

In the regression and classification problem we can experiment with different hyper-parameters. In regression and logistic regression we can experiment with the learning rate, the regularization term and the polynomial degree. In the neural network we can in addition experiment with the choice of activation function, number of epochs and the architecture of the neural network. In this project I have experimented with the sigmoid activation function given by

$$f(x) = f(x) = \frac{1}{1+e^{-x}} \tag{21}$$

the rectified linear unit (RELU) function

$$ReLU(x) = \begin{cases} 0 & z < 0, \\ z & z \geq 0. \end{cases} \tag{22}$$

and the leaky ReLU, adding a small gradient α for the negative terms

$$LeakyReLU(x) = \begin{cases} \alpha z & z < 0, \\ z & z \geq 0. \end{cases} \tag{23}$$

K-fold cross-validation

In machine learning it is common to split the dataset into a train group and a test group. The algorithm is first trained against the train group and the tested against the test group. This routine can expose over-fitting of data. In this report we have used k-fold cross validation. k-fold cross validation works by shuffling the data-set and dividing it into into k groups. Then one group is taken out as test group and the remaining groups are used to train the algorithm. This is repeated for all groups. The resulting R2 is obtained by taking the mean R2 of all groups.

	SGD	skl	Analytical
R2	0,90	0,85	0,96

TABLE I. The best R^2 scores for the regression problem using my own implemented SGD algorithm, the scikit-learn test and the analytical matrix inversion for the regression problem.

IV. RESULTS

In this section I only present some of the results from the analysis. More details can be found in the Jupyter Notebooks that are uploaded to GitHub.

The regression problem

The Ridge and OLS regression methods was used with SGD to fit a polynomial to Frankes function with added noise. A grid search was used to find the optimal learning rate and regularization. Increasing the polynomial degree above 7 led to my own implemented SGD producing less good results. The same was true for the test algorithm implemented by scikit-learn. The analytical matrix inversion method was able to produce a better score for the same polynomial-degree (7) and produces an slightly better fit for larger polynomials (0,97 max). Increasing the number of epochs increases the R2 score in general, but at the expense of increasing the computational time. The highest R2 scores (computationally viable for my computer) with its corresponding learning rate are listed in table I. Increasing the Ridge regularization term seems to decrease the R2 score, as expected. The neural network was used with SGD to learn and predict Frankes Function with added noise. A grid search was useed to find the optimal learning rate and regularization terms. The neural network was also tested with different activation functions and architectures. Too few hidden layers and neurons caused a low R2 test score. Many hidden layers also caused a low R2 test score. An optimal balance was found to be [30, 30, 30]. Scikit learn was able to handle larger architectures. Increased regularization made the neural network produce a less good fit, except for the Leaky RELU implementation where regularization made a better fit. The best fit R2 scores for the different activation functions are listed in Table II. Here we can observe that the sigmoid activation function are able to generate slightly better results than the ReLu and Leaky ReLu activation function. We can also observe that my own neural network under-performs some when using the ReLu and Leaky ReLu activation function.

	Sigmoid	RELU	Leaky RELU
R2 (own)	0,97	0,83	0,83
R2 (skl)	0,97	0,95	-

TABLE II. The best R2 scores for the regression problem using the neural network with different activation functions.

	Logistic Regression	Logistic Regression - skl
Accuracy score	0,96	0,96

TABLE III. The best R2 test scores for the classification problem using Logistic Regression.

The classification problem

The Logistic Regression method was used with SGD to perform an analysis of the breast cancer dataset. A grid search was used to find the optimal learning and regularization rates. Regularization seems to generally decrease the accuracy score. Increasing the number of epochs was essential in order to make to optimizer converge. The result was compared with scikit-learn. The optimal accuracy scores are listed in III. The neural network was used with SGD to analyse and predict the breast cancer dataset. A grid search was performed with different learning rates and regularization rates. The neural network was also tested with different activation functions and architectures. Too few hidden layers and neurons caused a low accuracy test score and too many layers and neurons caused the optimizer to not converge. A fine balance was found using layers of size [7, 7]. Sckikit learn was able to handle larger architectures increasing the performance slightly with layer sizes of at least up to [30, 30, 30]. Using an sufficient number of epochs was essential for the neural networks to converge. The best fit accuracy scores for the different activation functions are listed in Table **??**. Here we can observe that sckikit, learn performed slightly better than the other optimizers, altough it must be noted that this was a case with more complex network architecture. Since Sckikit learn did not have a Leaky ReLu implementation I was not able to run a test with it.

V. DISCUSSION

The regression problem

All regression techniques was able to produce a decent fit to Frankes Function. The stochastic gradient descent method with OLS and Ridge regression showed the poorest results. This could maybe have been improved by increasing the number of epochs. The analytical implementation gave a larger score, for the same polynomial degree, indicating that doing more iterations with the SGD approach could improve the R2 score. The neural network and the analytical expression (with polynomial order 22) shared the best fit with a R2 score of 0,97. Assuming the neural network was able to learn the Frankes function perfectly this indicates that higher order polynomials are, at least, a good approximation to Frankes Function. There was no sign of overfitting for large neural network architectures or large polynomial degrees. This could be due to the fact that the added noise was too low. Too large neural network architectures and polynomial degrees caused the SGD to not converge. In general sckikit learn produced better or just as good results as my implementations. This could be due to the fact that sckikit learn is more efficient and thus allows more epochs or that it has a better optimization routine. For the SGD regression with Ridge and OLS sckikit learn produced less good fits than my implementation, this could be due to the fact that my implementation used a momentum based approach while sckikit learn used a constant learning rate.

	Sigmoid	RELU	Leaky RELU
Accuracy score (own)	0,97	0,97	0,93
Accuracy score (skl)	0,97	0,98	-

TABLE IV. The best accuracy scores from the classification problem using a neural network with different activation functions.

The classification problem

All classification techniques was able to produce a a fit to the breast cancer dataset that was better than a random guess. The neural network seems to produce a slightly better fit than the logistic regression techniques. This indicates that there is a slightly more complicated relationship between the different cancer features than a linear one.

-

VI. CONCLUSION

The regression problem was modeled using an analytical and an SGD implementation of Ridge and OLS regression and compared with a neural network. The results show that the neural network performs slightly better than the analytical expression and much better than the Ridge and OLS polynomial fit, although this could change by increasing the number of epochs in the SGD optimizer. For larger datasets SGD could outperform the analytical expression as it becomes increasingly hard to solve the design-matrix for large datasets with many degrees of freedom. The classification problem was solved using an implementation of logistic regression and a neural network. The neural network performed slightly better than the logistic regression method, indicating that there is a nonlinear relationship between the features in the breast cancer dataset. Neural networks seems to be able to sometimes provide faster and more flexible way of approximating a function. This comes at the expense that the solution quickly become non-intuitive to interpret as the size of the network increases.