FYS-STK4155 - Project 1

Helene Wold & Tiril A. Gjerstad

Autumn 2021

Abstract

In this project we analyse and compare the linear regression methods Ordinary Least Squares, Ridge and Lasso. All the three methods are combined with cross-validation and bootstrap resampling techniques. We look at both Franke function and some real terrain data to do this analysis. We used Mean Squared Errors to predict the performance of the models. We found Ridge regression using bootstrap resampling technique slightly better than other models. For Lasso we were not able to conclude due to problems with the scaling when implementing the method.

Contents

1 Introduction

Linear regression is a useful tool commonly used to predict models of data sets in machine learning. By implementing some linear regression methods, and combining this with different resampling techniques, we are able to decide on the best method to create the best fit for our model.
In this project, we are implementing and analyzing the linear regression methods Ordinary Least Squares, Ridge regression and Lasso regression. We will also analyse how resampling techniques such as bootstrapping and cross-validation influence the results of the methods. The data we are modelling is for the first part the Franke Function.
To analyse the fit and compare the models created by each linear regression method, we are looking at the bias-variance trade-off and the Mean Squared Error. The bias-variance trade-off shows us the correspondence between bias and variance, and is implemented when the model is created using the bootstrap resampling technique. If the bias decreases, the variance will most often increase. With this type of analysis, we can find the sweet spot where both variance and bias is low. The Mean Squared Error (MSE) is implemented when the model is created with the cross-validation resampling. The goal is to get MSE as low as possible.
When the methods are implemented and evaluated, we move on from the theoretical Franke Function, to a real data set gathered from Stavanger in Norway showing the terrain.
To validate our implemented linear regression methods, we are calculating the corresponding SciKit -methods as well and comparing. Even though we have not included this in our report. If wanted, the reader can enter our ***source code in GitHub***.

2 Theory

2.1 The Franke Function

For the first part of the project we will try to fit a model to predict the Franke Function by using different linear regression methods. *The Franke Function* is defined as

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

where $x, y \in [0, 1]$. This function is widely used when testing various interpolation and fitting algorithms. It is a weighted sum of four exponentials, with two peaks and one dip. A plot of the *Franke Function* is shown in Figure 1.

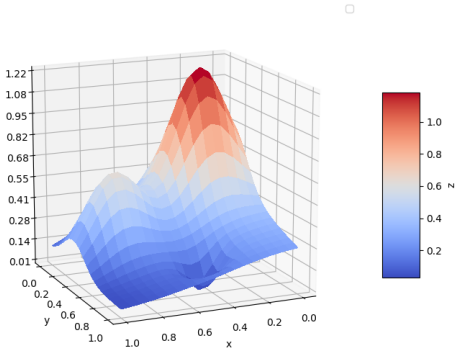

Figure 1: The surface of the Franke Function created by Equation (1), without noise.

2.2 Model accuracy

When we are training our models, we need something to measure how well the models performs. In this project we look at both MSE and R^2.

2.2.1 Mean Squared Error

The mean squared error is defined as

$$MSE\left(\hat{y}, \hat{\tilde{y}}\right) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{2}$$

where $\hat{y}$ is the predicted data and y is the real data. The smaller the value, the better the fit.

In our case, $\hat{y}$ is the output created from the different linear regression methods, and y is the output from the Franke Function.

The MSE is used to, as the name implies, find a mean squared error for the entire data set which can imply how well our model fits the true data. When training the data, we want to find the polynomial degree of the Linear Regression method used with the best (meaning the smallest) MSE.

However, the smallest MSE for the training data might not actually be the best fit for the testing data. This is because of a phenomenon called overfitting, meaning the model is fitted through every datapoint in the training data, leaving a very small MSE. Since the data is split into training and testing parts, the data points in the testing isn't necessarily similar to the training.

2.2.2 R2 score

The R2 score value is defined as

$$R^2\left(\hat{y}, \hat{\tilde{y}}\right) = 1 - \frac{\sum_{ix=0}^{n-1}(y_i - \tilde{y_i})^2}{\sum_{i=0}^{n-1}(y_i - \bar{y_i})^2} \tag{3}$$

where $\bar{y}$ is the mean value of y, defined as

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i \tag{4}$$

R^2 is also popularily named the *Coefficient of Determination*. It provides a measure of how well future samples are likely to be predicted by the model. The best possible score is 1.0, and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R2 score of 0.0.

2.2.3 Confidence Intervals

The confidence intervals are used as a type of estimation for our data. They calculate the range of parameters, and finds the interval where it is assumed that a certain percentage of the data will wind up in size of value.

Variance for β_j:

$$\sigma^2(\beta_j) = \sigma^2[(\mathbb{X}^T\mathbb{X})^{-1}]_{jj} \tag{5}$$

The confidence interval is given as [5]

$$\left(\mu_\beta \pm \frac{z\sigma_\beta}{\sqrt{n}}\right) \tag{6}$$

Here is z chosen to be $z = 1.96$ to get the 95% confidence interval.

2.3 Regression analysis

Regression analysis is a learning technique for supervised learning in machine learning, to predict an outcome variable $\tilde{y}$. There exists several different types of regression, we are using linear regression in this project.

To calculate the linear regression, we use the equation

$$\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{7}$$

as defined in the lecture notes [3].

By using the different regression methods, we find the parameter β that is minimizing the error ϵ. Throughout the project, we are using the regression methods *Ordinary Least Squares*, *Ridge Regression* and *Least Absolute Shrinkage and Selection Parameter*.

2.3.1 Ordinary Least Squares (OLS)

In *ordinary least squares* method, the optimal parameter β are given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{8}$$

as defined in the lecture notes [3]. This is derived from the *OLS* cost function for $C(\boldsymbol{\beta}) = 0$ [4],

$$\mathbf{C}(\boldsymbol{\beta}) = \sum_{i=0}^{p-1}(y_i - \beta_i)^2 \tag{9}$$

$\beta_{optimal}$ is found from

$$\beta_{optimal} = \underset{\beta}{argmin}\{C(\boldsymbol{\beta})\} \tag{10}$$

By combining Equation (7) and Equation (8), we get a general equation for the predicted model using *OLS*, namely;

$$\tilde{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{11}$$

This is an unique estimation of β for this method, other regression methods will have different estimations. By estimating the most optimal β, the β giving the smallest MSE, we can estimate the Franke Function the best way. Our main goal for the OLS linear regression is to estimate β to minimize the sum of the residuals. This means to minimize the error $\epsilon = y - \tilde{y} = y - X\beta$.

2.3.2 Ridge Regression

By adding a shrinkage parameter λ to our linear regression model (OLS) in Equation (9), we now have a new cost function [4], namely

$$\mathbf{C}(\boldsymbol{\beta}) = \sum_{i=0}^{p-1}(y_i - \beta_i)^2 + \lambda\sum_{i=0}^{p-1}\beta_i^2 \tag{12}$$

This is a regression method called Ridge regression. This method is used when the data set has independent variables that are very correlated to each other.

The consequence of this is that the coefficients β are shrunken, where the value of λ controls this shrinkage. $\lambda = 0$ gives OLS, and the larger λ is, the more shrinkage we get.

Just as in OLS, we are now deriving the cost function where $C(\beta) = 0$, giving

$$\hat{\boldsymbol{\beta}}_{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{13}$$

where I is the identity matrix. Ridge regression is OLS with a modified diagonal term added to X^TX to reduce variance. We can say that Ridge Regression reduces the values of β_i as a function of λ.

2.3.3 Lasso Regression

Lasso Regression, ***L**east **A**bsolute **S**hrinkage and **S**election **O**perator*, is a linear regression method made to do variable selection and regularization, to enhance prediction. Lasso reduces the amount of known variables for the model. We can say that Lasso regression suppresses values of β_i for specific values of λ.

The cost function of the lasso regression is defined as [4]:

$$\begin{aligned}\mathbf{C}(\boldsymbol{\beta}) &= \sum_{i=0}^{p-1}(y_i - \beta_i)^2 + \lambda\sum_{i=0}^{p-1}|\beta_i| \\ &= \sum_{i=0}^{p-1}(y_i - \beta_i)^2 + \lambda\sum_{i=0}^{p-1}\sqrt{\beta_i^2}\end{aligned} \tag{14}$$

By deriving and minimising this sum, we find the optimal fit of parameters for this method. [4]

$$\begin{aligned}\beta_{optimal}(\lambda) &= \underset{\beta}{argmin}\{C(\beta)\} \\ &= sgn(\beta^{OLS})(|\beta^{OLS}| - \lambda)\end{aligned} \tag{15}$$

In this project, we are using the already implemented Lasso regression method in SciKit-learn's library.

2.4 Resampling techniques

Resampling is an important method in statistics, created to draw samples from the training set before refitting our model for each sample. This method may result in a decrease in bias, as the training data has a tendency to adapt more to the structures of the data set. As well as a decrease in bias, resampling may increase the variance of the test error slightly. [5]

Even though resampling may be the way to go to find the best model, it can be an expensive operation. As mentioned above, resampling uses different parts of the data, and fits the model several times. This will however give us information not possible to obtain through one run of normal linear regression.

For our project, we are applying the two resampling techniques Bootstrap and Cross-Validation.

2.4.1 The Bootstrap method

The bootstrap method is based on taking the averages of estimates from several subsets of data. The training data is split into k subsets containing n elements each, before the chosen linear regression analysis method is applied, and the average values from each subset are calculated. [1]

In this project, we are using the bootstrapping method to calculate the MSE, R^2 and bias-variance of the data set.

2.4.2 The *k*-fold Cross-Validation method

k-fold cross-validation shuffles the data randomly, before splitting it into k groups. Next, we iterate through all groups k times, but for each time (k) we choose one group to be the testing data. This is the foundation of the "k-fold"-name, i.e. the iteration k times. After each k-fold, we combine the folds and find the average of the model evaluation scores, such as R^2 and MSE averages. [5]

2.5 Bias-variance trade-off

As defined in the project description given in this course[2], we assume that the true data is generated from a noisy model,

$$\mathbf{y} = f(\mathbf{x}) + \boldsymbol{\epsilon} \tag{16}$$

where $\boldsymbol{\epsilon}$ is the noise normally distributed with mean zero and standard deviation σ^2.

We have the model $\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$, and found the parameters $\boldsymbol{\beta}$ by optimizing the cost function as described briefly throughout Section 2.3.

The cost function is defined as

$$C(\mathbf{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] \tag{17}$$

where the expected value $\mathbb{E}$ is the sample value. [2]

The next equation is proved in Section 6, but to sum it up:

$$\mathbb{E}\left[(\mathbf{y} - \tilde{\mathbf{y}})^2\right] = \frac{1}{n}\sum_i(f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{n}\sum_i(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2 \tag{18}$$

The first term in Equation (18) is the bias. The last term, σ^2 is the variance of the error $\boldsymbol{\epsilon}$. The second term is the variance of the model. The entire equation is a representation of the Mean Squared Error (Equation (2)).
Equation (18) tells us that in order to get a minimal total error, we need to minimize both the variance σ^2 and the bias-element. However, when the bias decreases, the variance has a tendency to increase, and vice versa. To be able to get the best result, we want to find a right balance between the two terms.

The variance is expected to increase when the model complexity increases. This is a result of the model not being able to adapt to the increasing amount of noise of the model. The bias is expected to decrease when the model gets more complex. However, when the model is starting to overfit, the bias has a tendency to increase again. By looking at the "dip" where both bias and variance is small, we may decide the best complexity of the model.

3 Methods

The methods we use are all to be found in our ***GitHub repository***, found at this link: `https://github.uio.no/helenewo/Project1`

Our functions created to calculate the equations defined in this report are found in *funcs.py*, and are applied throughout the other *.py*-files in the repository.

3.1 Briefly explained

We use two data sets in this project, the Franke function and a real data set collected from Stavanger in Norway. The Franke Function (Equation (1)) is defined for $x, y = [0, 1]$, with a portion of random noise applied. This is to avoid getting too perfect input data, as real data always has some sort of noise in the data set. We will come back to the real data later.

By applying the different Linear Regression methods defined in Section 2.3, we can fit our model to the data sets for different complexities before calculating the MSE and R^2 score value at each polynomial degree (i.e. the complexity). This is done through both normal linear regression using OLS, and for all linear regression methods in Section 2.3 through the resampling technique *bootstrapping*. For the resampling technique *cross-validation*, we are evaluating the bias-variance trade-off for all linear regression methods in Section 2.3. Through these evaluations, we can indicate which model and method is the best.

3.2 Prepping for Linear Regression

3.2.1 Design matrix

When the data from either the *Franke Function* or the data file is prepared, and the meshgrid is created, we apply the method *np.ravel* before creating our design matrix $\mathbf{X}$ to represent the data. The design matrix has dimensions $p \times n$, n being the number of data points, and p being the number of predictors, i.e. the polynomial degree.

3.2.2 Splitting the data

When the design matrix $\mathbf{X}$ is created, we split $\mathbf{X}$ and the real data z in two, one part for training our model and one for testing it. To do this, we used SciKit-learn's function *train_test_split()*-function, that takes in the arrays that are to be split, and floats defining the amount of data being split into training and testing data. We use 80% of the data to train our model, meaning 20% is left for testing.

Splitting our data is important to do, because we want to avoid training the model on our entire data set. By first training the model on the training data set before testing it on the testing set. To be able to see if our predicted model is the best fit we can get, we want to leave some data to test our model with after the algorithm is done. By doing this, we also set aside a part of our total data set, so that we can decide whether or not our model is a good general fit. We will also see if the model becomes overfitted easier, a term mentioned earlier in this report.

3.2.3 Scaling the data

In all our analysis, a standardization has been done on the design matrix. We used SciKit-learn's scaler called *StandardScaler* to scale our data. This scaler works by subtracting the mean(Equation (4)), before dividing it by the standard deviation.

The real terrain data. Before scaling the terrain data given, we downsample the image to reduce the number of data points. Then we pick out 20 by 20 data points to use as our data set. This is to be able to use a bigger part of the image, without it being of i.e. the size 300×300. By doing this, the data set will be more representative for the entire data set, and our model will be better suited for our analysis.

3.3 Implementation of the Linear Regression Methods

As mentioned earlier, we are implementing the OLS, Ridge and Lasso regression methods. Our OLS and Ridge method are implemented in the file *linear_regression.py*.

3.3.1 Ordinary Least Square method (OLS)

In funcs.py we implement a function *calc_OLS* that takes in the design matrix $\mathbf{X}$ and the real data z and calculates the correct β_{OLS} presented in Equation (8). We use the *numpy.linalg.pinv*-function to invert the $(\mathbf{X}^T\mathbf{X})^{-1}$-term. To confirm our implementation, we calculate the MSE and R^2 score value for our method, and compares it to SciKit-learn's method. The OLS method is applied in *ols.py*, *cross_validation.py* and

bootstrap.py. In *ols.py* we also calculate the confidence intervals of the β. The code to do this is implemented as a function in *funcs.py*.

3.3.2 Ridge Regression

Just as for the OLS method, we implement a function named *calc_Ridge* to calculate β_{Ridge} from Equation (13). We send in different values of λ in the range $[0.00001, 10]$ to find the best fit of our model.

The function takes in X, z and our value λ, and is applied in *cross_validation.py* and *bootstrap.py*. Most of the algorithm for the analysis is done in the file *ridge.py*

3.3.3 LASSO Regression

For Lasso regression, we use SciKit-learn's method, since the method is rather heavy to implement correctly. This method takes in the value $\alpha > 0$, which is our λ in Equation (14). This constant is important to dampen some of the noise when we increase the complexity. This linear regression method is applied in both *cross_validation.py* and *bootstrap.py*. Most of the analysis used for the analysis is done in *lasso.py*

3.4 Resampling techniques

As mentioned in Section 2.4, we use resampling techniques such as *cross_validation* and *bootstrapping* to be able to find the best linear regression model.

3.4.1 Bootstrapping

By implementing the bootstrap resampling technique described in Section 2.4.1, we analyse the bias-variance trade-off (Section 2.5) for different polynomials and parameters for each linear regression method.

3.4.2 Cross-validation

We also implement the cross-validation resampling technique (Section 2.4.2) to evaluate MSE and R^2 score value for the different polynomial degrees and other parameters. For Ridge and Lasso regression, we also study the dependence on the parameter λ.

4 Results and Discussion

4.1 Notes about the plots

In this section, the plots has a varying type of y-axis. Some might have normal values, others will be with a logarithmic y-axis. This is because in some plots, it was difficult to see the changes without a logarithmic axis. However, in other plots, we want to look at the trend as well as the exact values and changes where the plots are stable. Therefore, we have chosen to use logarithmic axes where we want to see how the small changes in values affects the MSE.

4.2 Scaling the data

As mentioned in Section 3.2.3, we are scaling our design matrix and real data with SciKit's *StandardScaler*-function. This is to avoid unnecessary errors when applying the linear regression methods that are sensitive to the magnitudes of the features, such as Ridge regression and Lasso regression. Those sensitive algorithms depends on scaled data to end up without great variance in the output.

The linear regression method OLS is not as dependent of scaling. This is because OLS does not depend on any parameter λ etc. However, when we want to compare our methods using the MSE values and bias-variance trade-off, it is impractical to work with both scaled and unscaled data. We therefore scale the data the same way in all our analysis, so the results we are comparing and analyzing are of the same magnitudes.

Since we only are in the need of scaling the data to be of the same magnitude, and to avoid unnecessary faults when applying Ridge and Lasso regression, the StandardScaler-method is sufficient. As explained in Section 3.2.3, this method divides the mean value of the model by the standard deviation of the model.

4.3 MSE and bias-variance trade-off for OLS

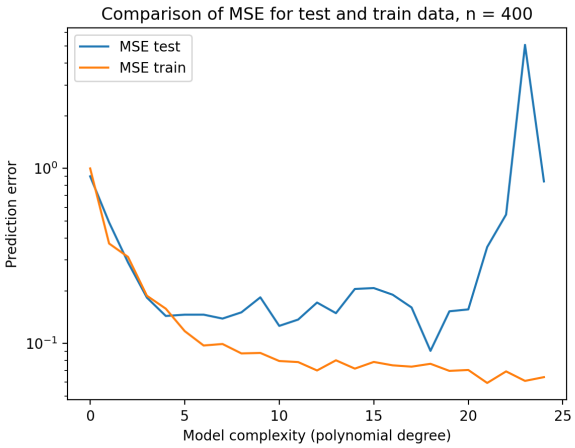

Figure 2: Mean Squared Error calculated from Equation (2) with the OLS method. Number of data points: $20 \times 20 = 400$, with noise.

In Figure 2 we can see the different mean squared errors for the ordinary least squares method at different polynomial degrees. As seen in the figure, the MSE of the testing data stabilises at a complexity between

4 and 7. As expected, the training data MSE decreases as the complexity increases. This is caused by 'overfitting' our predictions to our training data set.

We want to avoid choosing a complexity where the test data is much worse than the training data. The best fit of our model is then with a complexity 4 and 7, i.e. the point of crossing between our training data and testing data, and while the testing data MSE is still stable at $MSE = 0.13$.

As seen in Figure 2, there is also a dip in the MSE value of our testing data. Technically this is the point where we have the best fit for this model. However, we are randomly adding noise to our Franke Function, and the splitting of data is not equal for each complexity. By "splitting of the data is not equal", it is meant that when the data is being split at each complexity, and the data points put into each part is not the exact same data points for each complexity.

We also create noise at random seeds, meaning we get some random noise when we generate this, but it is the same random noise each time we run the algorithm. This indicates that our results will be the same for each run, but still have such a random amount of noise that it can not be recreated in any other way than the method we have used.

The dip at complexity 18 is not a part of the trend of the MSE values, and is therefore a result of the noise in our data and how the data is split. Therefore, it is not unexpected that certain values in our model accuracy analysis seem to be the best fit, while it does not necessarily fit into the ideal trend of the analysis.

This is a good example on why we always should look at trends and plots in a bigger scale, and not choose the best value without looking at the whole picture. For most data we would see a much higher MSE value at complexity 18, so it will not be a good fit generally.

4.3.1 Bias-Variance trade-off for OLS using bootstrap

As seen in Figure 3, the variance increases along with the complexity, and the MSE of the testing data is greatly affected by this. By using the bootstrap resampling technique, see Section 2.4.1, we try to optimize our model.

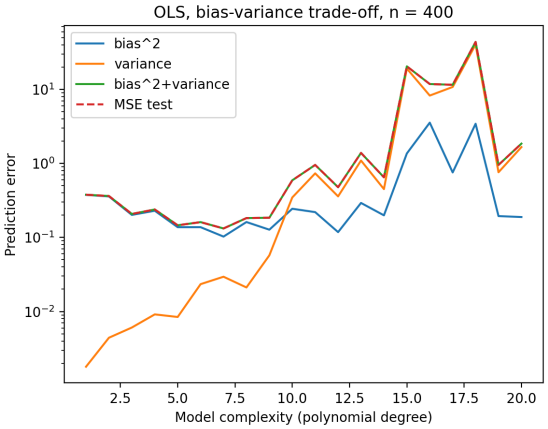

Figure 3: Bias-variance trade-off OLS using the bootstrapping resampling technique, with 20×20 data points with same seed as in Figure 2.

Figure 3 confirms our expectations in regards to $MSE = bias^2 + variance$ proved in Section 6, and in regards to the expected variance. We would however expect a bigger bias at smaller complexities, and to see it increase when the model is being overfitted, as mentioned in Section 2.5.

This is due to the amount of data points used to create our model. The more data points, the higher complexity is needed to replicate and adapt to our data set. Figure 3 and Figure 4 are visualizing the same results, namely bias, variance, bias^2 + variance and MSE for our testing data. The only difference between these results is the amount of data points. Figure 3 is created with $20 \times 20 = 400$ data points, and Figure 4 is created with $60 \times 60 = 3600$ data points.

When the amount of data points increases, our model is easier to train and predict. The variance is as expected, as our values are rather small (10^{-3}). It it expected that the variance will increase, however slower when the data set is bigger, as it is more noise to adapt to in our model. We also have more data points to train our model up against, which gives a better model all together. If we were to increase the amount of polynomials we are looking at, the variance would be fitted better. In Figure 3, the variance is starting to overfit after a complexity of $p = 10$. This is possible because of the "small" amount of data points.

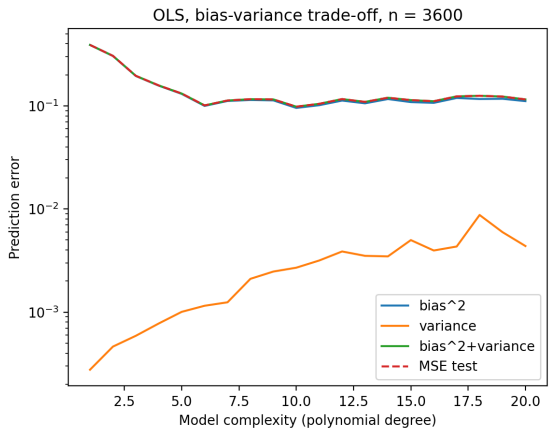

Figure 4: Bias-variance trade-off for the OLS method, using bootstrapping resampling technique. This visualises Figure 3, but with $60 \times 60 = 3600$ data points, also with noise.

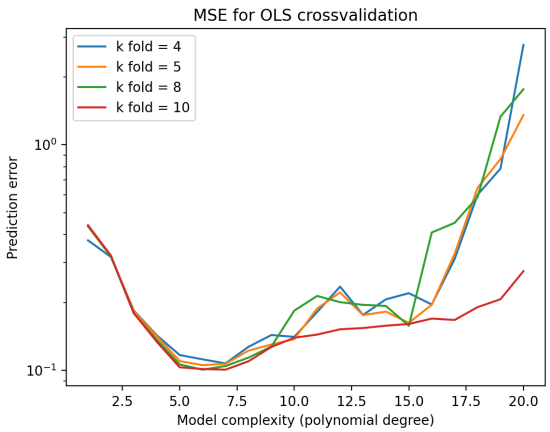

Figure 5: A plot comparing n folds with k-fold cross-validation of the OLS.

As mentioned, we found from Figure 2 that the best fit of our model is for a complexity between 4 and 7. From the complexity 6, the variance starts to increase. The best choice of complexity might be $p = 5$, while the variance is still small, and a MSE of the testing data still being in the low region. This will result in an MSE of $MSE = 0.14$ for the OLS linear regression, sampled normally.

On the other side, for the OLS calculated with bootstrapping, we can see the MSE test starts to increase at the polynomial degree $p = 9$, while being stable from $p = 4$ to $p = 9$. Our best choice is then the complexity $p = 6$, where the variance is at its best while the MSE testing value is stable.

4.3.2 Cross-validation resampling at OLS

Moving on to the cross-validation resampling technique used on the OLS linear regression method, we can look at Figure 6 showing us the MSE for both the testing data and the training data. We see that both the testing and the training data is increasing when the model gets more complex. This is again expected for the testing data, however not for the training data.

The more complex polynomials is better represented with the biggest amount of folds, but from Figure 5 we can see that the best complexities is still in the range of $p = 5$ to $p = 8$. To get the simplest model, we want to choose $k = 5$ over $k = 10$, as the higher amount of folds will return a more complex and computing-costly model.

The k-fold cross-validation is introduced in Section 2.4.2. We used 5 folds for Figure 6 and Figure 7, since this was the amount of folds giving us the best fit with this method without having a too big of a computational cost.

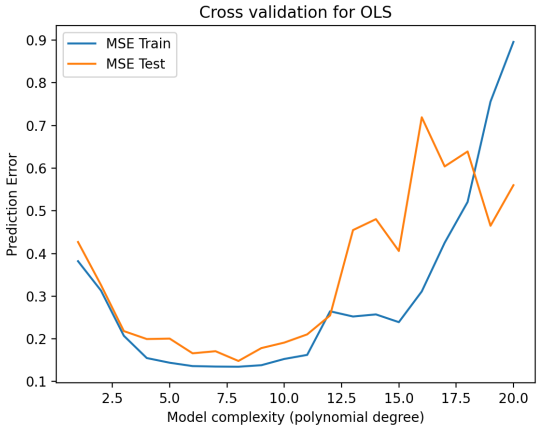

Figure 6: MSE test and MSE train for the OLS linear regression using k-fold cross-validation with 5 folds, and 400 data points.

From Figure 6, we see the MSE test and MSE train being low and stable until a complexity of $p = 10$. After this, the values increase, as a result of overfitting and the bias increasing along with complexity (Section 2.5). A complexity of $p = 8$ at fold $k = 5$, we get an test MSE of $MSE = 0.11$.

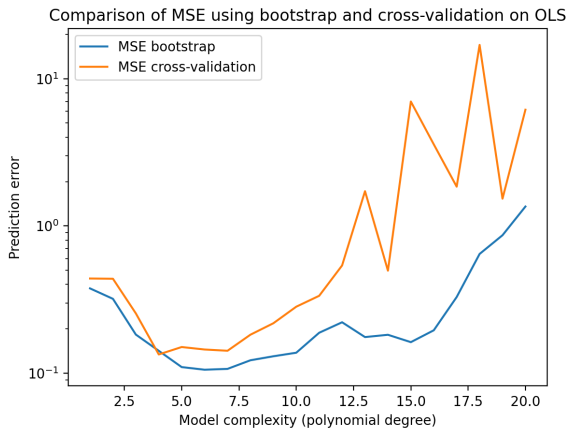

Figure 7: Bootstrap VS. cross-validation in regards to the testing MSE values.

We do have the opportunity to decide which resampling technique is the best for the OLS linear regression. By looking at Figure 7, it is clear that the bootstrap resampling technique will give the best results. To remind ourselves, the chosen best complexity for the bootstrap and the cross-validation was set to be $p_{bootstrap} = 6$ and $p_{cross-validation} = 8$. When looking at those points in Figure 7, the bootstrap technique is proven best in regards to the MSE of the testing data for our model.

4.4 MSE and bias-variance trade-off for the Ridge Regression

Next, we are implementing the Ridge Linear Regression method. The Ridge Regression (Section 3.3.2) add a new parameter λ to the Ordinary Least Squares. Let's begin with looking at how λ affects the MSE for the different polynomial degrees.

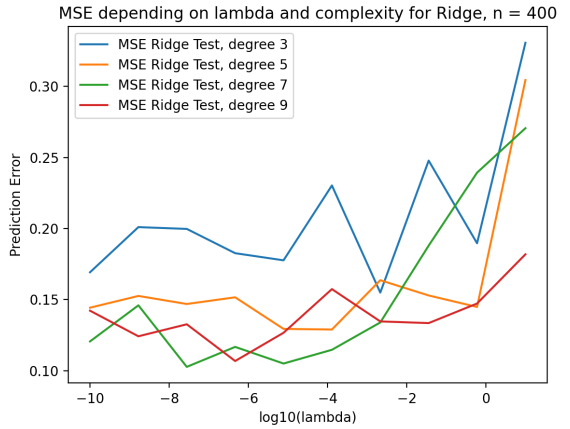

Figure 8: MSE evaluation of ordinary Ridge Regression showing the dependence of λ for the different polynomial degrees.

Figure 8 shows us that the lower values of λ for a higher complexity gives a better MSE value for our testing data. The λ value goes from 0.00001 to 10, while Figure 8 tells us that we should choose a λ between 0.00001 and 0.02 , i.e. somewhere between -5 and -2 on the x-axis in our figure. This is the point of all lowest MSE values. To get the best fit to our model by using the Ridge Regression method with ordinary sampling, we might want to choose a complexity of $p = 7$, with a λ of $\lambda \approx 0.001$, giving the MSE testing value $MSE = 0.13$. We assume that the dip at the bigger λ value is a result of noise.

We avoid using a complexity of $p = 9$ with the smallest lambda of $\lambda < 0.0001$, since $\lambda = 0$ in the Ridge Regression will return the Ordinary Least Squares method. We therefore choose a slightly higher value for λ, do be able to differentiate between OLS and Ridge. The λ value we choose will determine how much the variance is affected, which is why a $\lambda = 0$ will make Equation (13) become Equation (8). A bigger λ will give less variance in our model, but the bias will still increase since it is not dependent on λ. This will give a higher MSE. Therefore it is as expected that the MSE value might grow even though λ grows.

Let us now do different resampling techniques to see if we can find a better fit for our model.

4.4.1 Bias-variance trade-off using Bootstrap resampling

For this resampling technique, we use $\lambda = 0.001$ and the number of bootstrap resamples $nboot = 50$. By using 50 resamples, we get the effect desired by using bootstrap resampling, without an excessively big computational cost and time duration. To give an example, 3 resamples would not give us enough data to show how the resampling technique works, and 1000 resamples is an excessive amount of resamples for a data set of 400 data points.

Figure 9 shows a bias-variance similar to Figure 3, with an increasing variance when the complexity increases. Even though the bias, variance and test MSE shown in Figure 9 has a peak at complexity $p = 17$, it is still a smaller peak than for complexity $p = 15$ to $p = 18$ in Figure 3 which exceeds $MSE = 1$ after a complexity of $p = 10$. This shows that by using bootstrapping on the Ridge Regression, we get better results when the complexity increases than with OLS regression. Figure 9 shows the best MSE at a complexity $p = 7$ when $\lambda = 0.001$, with an $MSE = 0.09$. This might not be a significantly better MSE value than for OLS, but bootstrapping using Ridge Regression is a more stable resampling technique compared to OLS using the same resampling technique.

For Figure 9, we chose a λ of $\lambda = 0.001$. To be able to decide at this parameter value, we look at the prediction error as a function of λ.

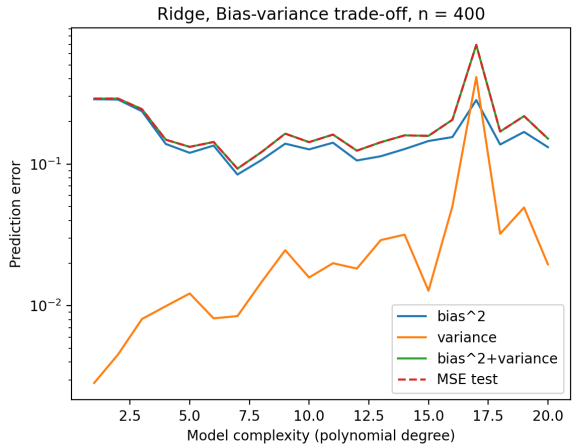

Figure 9: Bias-variance trade-off for the Ridge Regression using $\lambda = 0.001$ and 50 resamples showing the effect of the complexity.

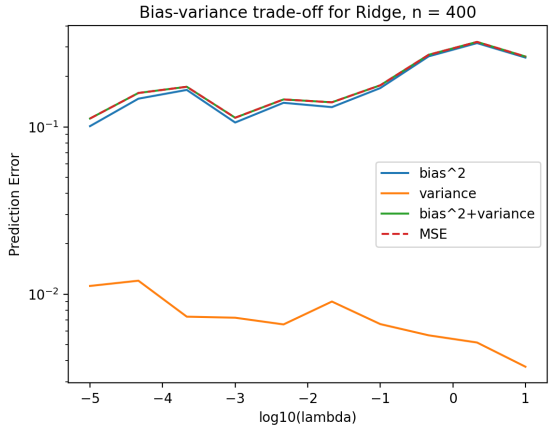

Figure 10: Bias-variance trade-off for the Ridge Regression using $p = 5$ and 50 resamples showing the effect of the parameter λ.

Just as in Figure 9, the total MSE increases when λ increases, because of the bias increase. The variance, which is dependent on λ, decreases when λ grows. As mentioned in Section 3.3.2, λ decides the effect of the variance, meaning when λ increases, the variance caused by noise is being counteracted by the λ parameter. The best value for our λ should still be at $\lambda = 0.001$, which is where the MSE value is in a stable and small trend without λ closing up on 0.

4.4.2 Cross-validation resampling of Ridge Regression

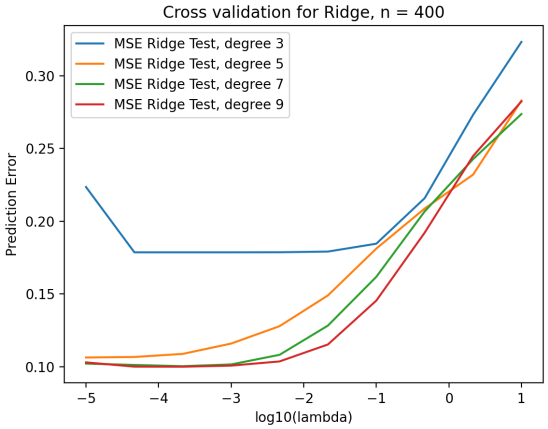

Figure 11: Cross-validation of Ridge regression using 400 data points and 8 folds, showing MSE dependent of both lambda and polynomial degree.

Figure 11 shows the MSE values of Ridge Regression using k-fold cross-validation with 8 folds. By doing a similar comparison such as in Figure 5, we found 8 folds to be the best choice. 10 folds gives the best result for higher complexities, but it is a heavier computation with small improvement.

Just as in Figure 8, we see that the smaller values of λ is preferred, i.e. about 0.00001 to 0.01. We therefore choose a $\lambda = 0.001$, the point before several of the MSE values starts to increase for the different complexities.

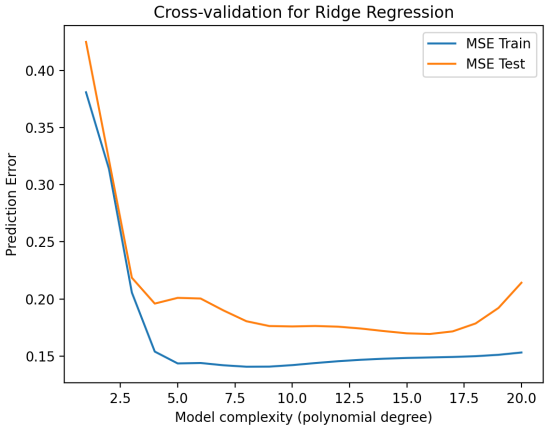

Figure 12: MSE values of Ridge regression calculated with cross-validation resampling technique. This plot uses 400 data points, $\lambda = 0.001$ and 8 folds.

Just as earlier figures, we see the MSE error values decrease along with the complexity as expected. It is a bump at complexity 5 to 7 for the test MSE.

However, this is probably just an effect of the random noise, just as the dip at $p = 18$ in Figure 2. We therefore choose a complexity of $p = 9$ when $\lambda = 0.001$, which gives an MSE of $MSE = 0.17$. This value is rather high as opposed to earlier MSE values for other combinations of resampling techniques and linear regression methods.

4.5 MSE and bias-variance trade-off for Lasso regression

We will also look at Lasso Regression, and test how well Lasso performs with our data. We might have implemented SciKit's Lasso regression function in a wrong way, because the results we get is not what we would expect.

First we will take a look at Figure 13 and Figure 14. These two figures illustrate the same thing, Lasso regression without any resampling techniques implemented with *sklearn.linear_model.Lasso*(). The only difference is how the scaling of the data is implemented. In Figure 13 we are using the *StandardScaler* from *sklearn*, in the same way as for OLS and Ridge. This gives us way too high MSE-values, which makes no sense in this case. In Figure 14, the same analysis is done, but here we let Lasso normalize the data by giving the argument $normalize = True$ to the Lasso-function. These results is more in line with what we expect to see. We therefore assume that something is happening with the scaling of the data. We want to do the same analysis for all our methods, and also scale the data in the same way. We have chosen to use our own scaling method with *StandardScaler*, also for the analysis in Lasso regression, although this gives unexpected results.

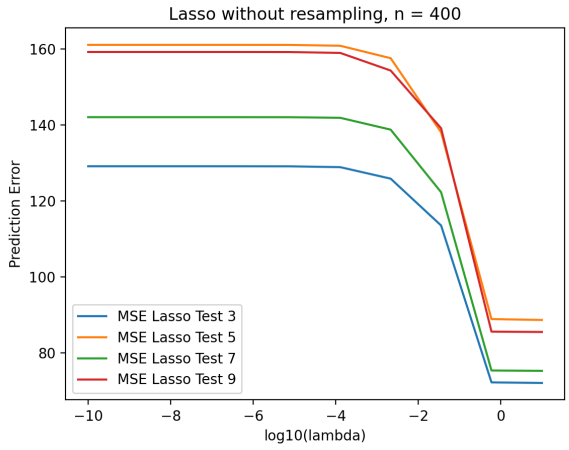

Figure 13: Lasso regression using Standard scaler for normalizing the data

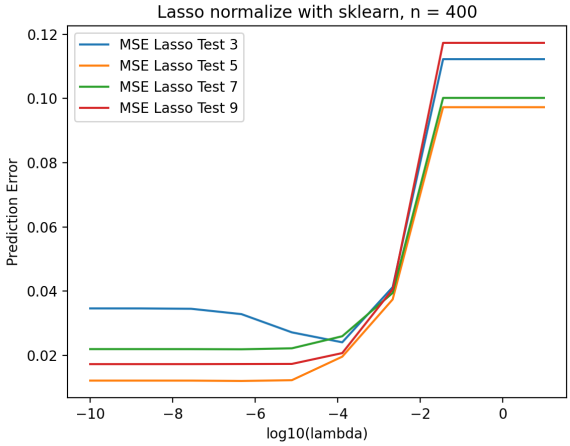

Figure 14: Lasso regression using normalizer in sklearn

The trend we see in Figure 14 is the trend we also would expect, small values of λ is preferred, and we see MSE increase when λ also increase. This is not what happens in Figure 13.

Since we are scaling the data with *StandardScaler*, which did not give us good results, it is therefore not a great starting point for further analysis. Looking at bootstrap analysis for Lasso, we do not get these high values for MSE, so we will try to do some analysis of the outcome. Keep in mind that this might not be the correct results.

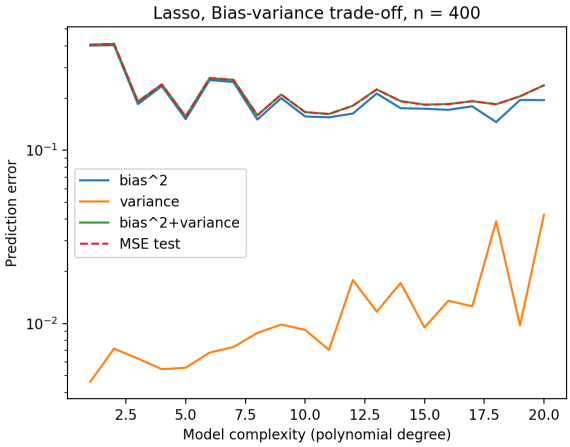

Figure 15: Study of bias-variance trade-off with Lasso regression. $nboot = 50$, $\lambda = 0.001$

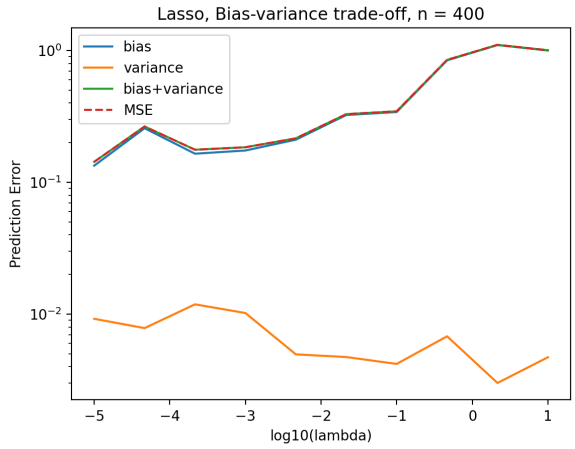

Figure 16: Study of bias-variance trade-off as a function of various values of λ using Lasso regression. $nboot = 50$, complexity $p = 8$.

Figure 15 and Figure 16 shows bias-variance trade-off for Lasso regression based on respectively complexity and λ. In both figures, the variance is rather low.

When we look at complexity in Figure 15, we see that bias decreases from start. This make sense because we do not get a good model with too low complexity. At complexity $p = 5$ we have a dip, and the bias stabilizes at a complexity of $p = 7$. This indicates that a complexity of $p = 5$ is a good choice, because we see the variance starts to increase at this point.

Looking at the dependence of λ in Figure 16, we see the bias starts to increase for bigger values. This is also where variance decreases. The best trade-off will be for λ values on about $\lambda = 0.001$.

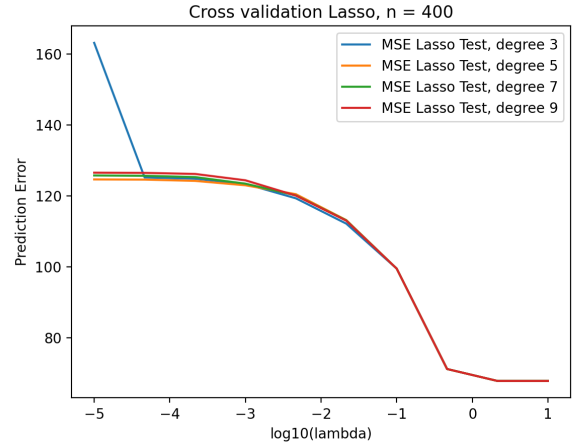

Figure 17: Cross-validation with Lasso. This results does not make any sense.

Next, we have the plot for cross-validation, shown in Figure 17. We will not do any analysis here, because out data does not make any sense. This is most likely of the same reason with scaling, as explained earlier.

4.6 Comparing OLS, Ridge and Lasso for the Franke Function

As seen in Section 4.5, the results from our implementation of Lasso is not as expected. We will therefore disregard the Lasso linear regression method in our comparisons.

By using combinations the different resampling techniques and linear regression methods, we get the results presented in Table 1 found in in Section 7. These values are the ones chosen to be the best throughout our report. Our goal is to find the method resulting in the lowest MSE value.

Let us begin by looking at the different MSE values of the Ordinary Least Squares method. Here, the MSE decreases from ordinary sampling to bootstrap resampling, and then to the k-fold cross-validation resampling technique. We stay below a complexity of $p = 10$, which gives minimal overfitting of our model. If we were to fit our model to the Franke Function (Equation (1)) using ordinary least squares, the best way to do so would be by using the k-fold cross-validation with 5 folds, which gives $MSE = 0.11$ for a complexity $p = 8$.

For the Ridge Regression, we see that the k-fold cross-validation gives a higher MSE value of $MSE = 0.17$. The best MSE is for the bootstrap resampling technique with a λ of $\lambda = 0.001$, with a complexity $p = 7$ gives $MSE = 0.09$.

The MSE for Ridge Regression using bootstrap resampling with $\lambda = 0.001$ and complexity $p = 7$ is the best we got from our analysis of fitting the model to the Franke Function.

4.7 Analysis of real data

Moving on to the analysis of the real terrain data, gathered from Stavanger in Norway. We have chosen to not include an analysis of the Lasso Regression for this part, since the results are just as bad as in Section 4.5. If wanted, the results can be found through our algorithm in GitHub.

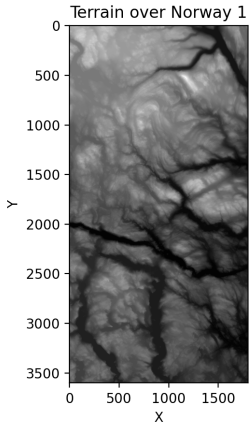

Figure 18: Original terrain data

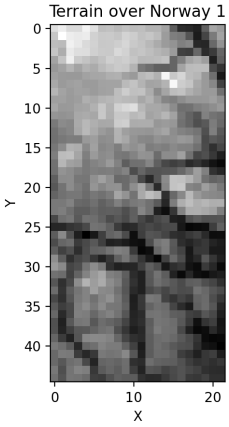

Figure 19: Terrain data downsampled

As mentioned in Section 3.2.3, we want to pick an amount of data suited to represent as much of the data set as possible. The downsampled and selected data is shown in Figure 20.

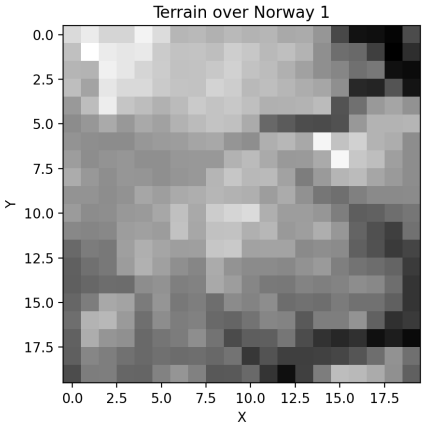

Figure 20: Slice of downsampled terrain data. We will use linear regression and try to recreate this part of the image.

Now we are set to recreate a model to represent this amount of data.

4.7.1 Ordinary Least Squares

By applying the same methods and analysis strategy used in Section 4.3, we found the best MSE possible for the OLS to be with the resampling technique cross-validation.

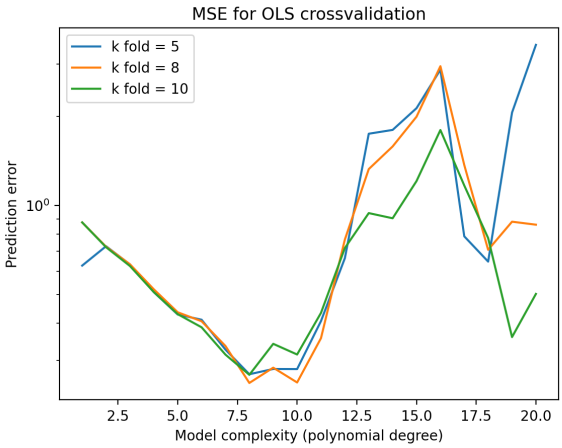

Figure 21: k-fold cross-validation of OLS linear regression on the real terrain data with 400 data points.

Figure 21 shows the k-fold cross-validation on the OLS linear regression. We found the best model to be with 8 folds at a complexity $p = 8$, which gives an MSE of $MSE = 0.25$.

4.7.2 Ridge

Now, we repeat the methods used in Section 4.4. Here, we found the best resampling technique to be bootstrap resampling.

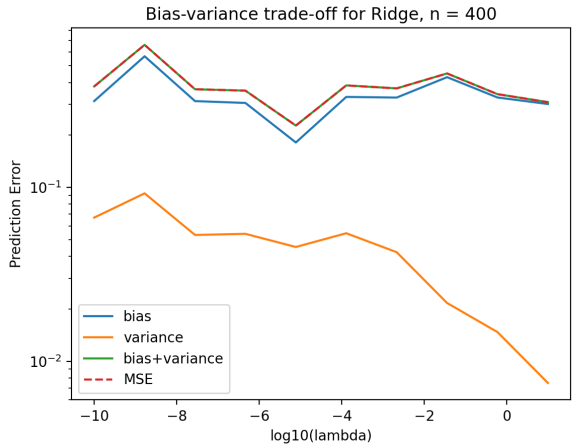

Figure 22: Bias-variance trade-off using bootstrap resampling on Ridge Regression. 50 bootstraps, complexity $p = 9$.

By doing the same analysis as earlier to find the best complexity, we found $p = 9$ to be the best with a $\lambda = 0.00001$, giving an MSE of $MSE = 0.19$.

4.8 Comparison of the OLS and Ridge for Terrain data

Again, we find all MSE values for the different combinations of resampling techniques and linear regression methods, and gather them in a table, namely Table 2 found in in Section 7.

As explained in Section 4.7.1 and Section 4.7.2, we found the best methods for OLS and Ridge to be cross-validation and bootstrap. By using cross-validation on OLS with 8 folds at a complexity of $p = 8$, we get an $MSE = 0.25$. Bootstrapping Ridge Regression with a $\lambda = 0.00001$ gives the best fit at a complexity $p = 9$ with $MSE = 0.19$.

The best method to use when fitting our model to the real terrain data is therefore bootstrap resampling of the Ridge Regression.

5 Conclusion

Throughout this report, we have applied and discussed the different combinations of linear regression methods and resampling techniques. When fitting our model to the Franke function, we found the Ridge Regression using bootstrap resampling with the parameter $\lambda = 0.001$ and complexity $p = 7$ to be the best fit. This was also the result for our terrain data, however with a smaller value of $\lambda = 0.00001$. Neither of the methods gives an overfitted model, while maintaining a rather low MSE value.

The implementation of the Lasso Regression did not work, giving values off chart compared to the Ordinary Least Squares and Ridge Regression for both data sets.

References

[1] A. C. Davison and D. V. Hinkley. *Bootstrap Methods and their Application*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997. DOI: 10.1017/CBO9780511802843.

[2] Morten Hjorth-Jensen. *Project 1*. https://github.com/CompPhysics/MachineLearning/blob/master/doc/Projects/2021/Project1/pdf/Project1.pdf. Accessed: 07-10-2021. 2021.

[3] Morten Hjorth-Jensen. *Week 35 - From Ordinary Linear Regression to Ridge and Lasso Regression*. https://compphysics.github.io/MachineLearning/doc/pub/week35/html/week35.html. Accessed: 07-10-2021. 2021.

[4] Morten Hjorth-Jensen. *Week 36: Statistical interpretation of Linear Regression and Resampling techniques*. https://compphysics.github.io/MachineLearning/doc/pub/week36/html/week36.html. Accessed: 08-10-2021. 2021.

[5] Morten Hjorth-Jensen. *Week 37: Summary of Ridge and Lasso Regression and Resampling Methods*. https://compphysics.github.io/MachineLearning/doc/pub/week37/html/week37.html. Accessed: 07-10-2021. 2021.

6 Appendix A

Let us rewrite Equation (17);

$$\mathbb{E}[(\mathbf{y}-\tilde{\mathbf{y}})^2] = \mathbb{E}[(f(\mathbf{x})+\boldsymbol{\epsilon}-\tilde{\mathbf{y}})^2]$$

$$= \mathbb{E}[(f(\mathbf{x})+\boldsymbol{\epsilon}-\tilde{\mathbf{y}}+\mathbb{E}(\tilde{\mathbf{y}})-\mathbb{E}(\tilde{\mathbf{y}}))^2]$$

$$= \mathbb{E}[((f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))+(\boldsymbol{\epsilon}+\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}}))^2]$$

$$= \mathbb{E}[(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))^2+2(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}})(\boldsymbol{\epsilon}+\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}}))+(\boldsymbol{\epsilon}+\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})^2]$$

$$= \mathbb{E}[(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))^2+2\boldsymbol{\epsilon}(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))+2(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})+(\boldsymbol{\epsilon}+(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})^2)]$$

$$= \mathbb{E}[(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))^2+2\boldsymbol{\epsilon}(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))+2(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})+(\boldsymbol{\epsilon}^2+2\boldsymbol{\epsilon}(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})+(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})^2)]$$

$$\begin{aligned}
&= \mathbb{E}[(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))^2] \\
&+ \mathbb{E}[2\boldsymbol{\epsilon}(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))] \\
&+ \mathbb{E}[2(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})] \\
&+ \mathbb{E}[\boldsymbol{\epsilon}^2] \\
&+ \mathbb{E}[2\boldsymbol{\epsilon}(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})] \\
&+ \mathbb{E}[(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})^2)]
\end{aligned}$$

We know that $\mathbb{E}[\boldsymbol{\epsilon}] = 0$, therefore will the second and fifth part of the expression above disappear.

$$\begin{aligned}
&= \mathbb{E}[(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))^2] \\
&+ \mathbb{E}[2(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})] \\
&+ \mathbb{E}[\boldsymbol{\epsilon}^2] \\
&+ \mathbb{E}[(\mathbb{E}(\tilde{\mathbf{y}})-\tilde{\mathbf{y}})^2)]
\end{aligned}$$

The first and fourth term is at the same form as the cost function shown in Equation (17). The second term will turn into 0. The third term comes from $\sigma^2 = \mathbb{E}[\boldsymbol{\epsilon}^2] - \mathbb{E}[\boldsymbol{\epsilon}]^2 = \mathbb{E}[\boldsymbol{\epsilon}^2] - 0 = \mathbb{E}[\boldsymbol{\epsilon}^2]$. We know that $\mathbb{E}[\boldsymbol{\epsilon}] = 0$, therefore will the second and fifth part of the expression above disappear.

$$= \frac{1}{n}\sum_{i=0}^{n-1}(f(\mathbf{x})-\mathbb{E}(\tilde{\mathbf{y}}))^2 + 0 + \boldsymbol{\sigma}^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\mathbb{E})-\tilde{\mathbf{y}})^2$$

$$\begin{aligned}
&= C(\mathbf{x},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 \\
&= \mathbb{E}[(\mathbf{y}-\tilde{\mathbf{y}})^2] \\
&= \frac{1}{n}\sum_i(f_i-\mathbb{E}(\tilde{\mathbf{y}}))^2 + \frac{1}{n}\sum_i(\tilde{y}_i-\mathbb{E}[\tilde{y}])^2 + \boldsymbol{\sigma}^2
\end{aligned}$$

We have now proven the relationship

$$\mathbb{E}[(\mathbf{y}-\tilde{\mathbf{y}})^2] = \frac{1}{n}\sum_i(f_i-\mathbb{E}(\tilde{\mathbf{y}}))^2 + \frac{1}{n}\sum_i(\tilde{y}_i-\mathbb{E}[\tilde{y}])^2 + \boldsymbol{\sigma}^2$$

7 Appendix B

This appendix contains tables with a summary of the calculated Mean Squared Error values of the different combinations of linear regression methods and resampling techniques. All values are discussed in Section 4

	Resampling techniques									
	Ordinary			Bootstrapping			Cross-validation			
	MSE	p	λ	MSE	p	λ	MSE	p	λ	k
OLS	0.13	5	-	0.12	7	-	0.11	8	-	5
Ridge	0.13	8	0.001	0.09	7	0.001	0.17	9	0.001	8

Table 1: The different values and parameters used to get the best fit to our model. Lasso Regression is ignored.

	Resampling techniques									
	Ordinary			Bootstrapping			Cross-validation			
	MSE	p	λ	MSE	p	λ	MSE	p	λ	k
OLS	0.40	5	-	0.32	5	-	0.25	8	-	8
Ridge	0.26	9	0.000001	0.19	7	0.00001	0.24	9	0.0000001	8

Table 2: MSE values for the different combinations of resampling techniques and linear regression methods. Lasso Regression is ignored.