FYS-STK 4155 Project 1

Kristoffer Langstad *
krilangs@uio.no

Abstract

In this project, we want to analyze different linear regression models on two different given data sets. First we will analyze what is called the Franke function with an added stochastic noise following a normal distribution, and then we will analyze real digital terrain data. The Franke function only contains about 400 data points, while the full digital terrain data contains about 6.5 million data points. Our computer can't handle such big data sets, so we have to reduce it to be able to analyze the data set. We will reduce the digital terrain data to about 7400 data points. To analyze these data sets, we will use three different regression models; the Ordinary Least Squares (OLS) regression, Ridge regression and Lasso regression. We will fit these models to the mentioned data sets as polynomials of varying degree. These regression methods are then combined with the k-fold cross-validation resampling method for splitting the data sets into a training set and a test set. Then we can evaluate the errors and the evaluation scores to see how well fitted our models are to the data sets depending on the model complexity and hyperparameters. We also look at the bias-variance trade-off we get as the complexity and hyperparameters change. From theory we could expect that the OLS would be the best fit model. This arises from the fact that Ridge and Lasso have a shrinking parameter, λ, that pushes the regression parameters, β, towards zero. As λ gets closer to zero, we would expect the solutions to go towards the same as for the OLS model. From our results on the Franke function with a noise term with standard deviation $\sigma = 0.2$, we can conclude that this happens. We get the smallest Mean Squared Error (MSE) and the best R^2 score from the OLS method for dimension 9 with $MSE = 0.040959$ and $R^2 = 0.680520$. The results for Ridge and Lasso are close to the OLS results for very low λ's ($\approx 10^{-7}$) with the same dimensionality (see Table 3 for all the best values). For the digital terrain data we get very similar best fit values for OLS and Ridge, with Ridge giving a slightly better fit to the terrain data. The Ridge give $R^2 = 0.77657$ for model complexity 16 and hyper parameter 10^{-5}, while the OLS give $R^2 = 0.773255$ for model complexity 15. Lasso on the other hand give $R^2 = 0.665718$ for the same parameters as for Ridge (full best fit values in

*https://github.com/krilangs/FYS-STK4155

Tabel 4). For these results we can conclude that Ridge give the best fit, while OLS is about equally good.

1 Introduction

A hot topic today is what we call machine learning. One thing we can do with machine learning is to analyze data (often a huge amount). There are now many different methods for doing this, where some are better than others in different cases. A way to study machine learning is to look at regression analysis and resampling methods of given data sets, which is split into what we call training and test data. Then we can study the data dependence and the effects of different model complexities and hyperparameters.

We will in this project study three linear regression methods: the Ordinary Least Squares (OLS), Ridge and Lasso. In linear regression, we want fit a polynomial to a given data set that we want to analyze. Linear regression let use look at relationships in data sets that are normally hidden under complexities and huge quantities of data. In our case we will first fit a polynomial to a two-dimensional function called Franke's function (which we will look more at later). After implementing the regression methods we will use resampling techniques, like cross-validation, to perform a more thorough assessment of the models we have by splitting the data sets into training and test data. Then we also look at the Bias-Variance tradeoff as we look at a varying model complexity (and hyperparameters) of the data set and models. Then we find which regression model fits the Franke function data best. Lastly we will use all the techniques we have mentioned to analyze real data terrain to see which model and degree of polynomial has the best fit to this data.

In the methods section we look at the theory of the different models, techniques and the implementations of the algorithms we use in this project. We start by looking at the Franke function with and without an additional normally distributed stochastic noise with mean $\mu = 0$ and standard deviation σ, to produce a dataset to be studied with OLS regression. Then we want to fit a polynomial up to fifth order, and study the confidence interval of β by computing the Mean Squared Error (MSE), R^2 score function and variance. Then we perform a resampling of the data where we split the data into training and test data. Then we construct the k-fold cross-validation algorithm and evaluate the MSE, R^2 score, variance and bias of the test data. For further studying machine learning, we study the bias and variance tradeoff as function of model complexity and number of data points. Now we have all the techniques we are using, so now we use the same techniques on the Ridge and Lasso method. For these two methods there is a hyperparameter dependence which we will also study. Then we want to introduce real terrain data to analyze. We use all three of the linear regression methods to evaluate which model fits the data best. In the Results section we look at the results and discuss all the methods and techniques used in this project. Here we look at data outputs and figures we get underway.

Lastly in the Conclusion section we summarize the results and come up our thoughts on which model is the best fit for the terrain data.

2 Theory

2.1 Linear regression

In linear regression we want to fit a model to a set of data. For fitting a model to a set of data $\mathbf{y} = [y_0, y_1, ..., y_{n-1}]$ of length n which are functions of some variables $\mathbf{x} = [x_0, x_1, ..., x_{n-1}]$. We can then define the prediction value y as

$$y(x_i) = y_i = f(\cdot) + \epsilon_i = \sum_{n-1}^{i=0} \beta_j x_i^j + \epsilon_i, \tag{1}$$

where β are regression parameters and ϵ_i is a normally distributed error with mean zero and variance σ^2. The goal is to find the optimal regression parameters β. This can be done with different methods like OLS, Ridge and Lasso.

2.2 Ordinary Least Square (OLS)

In OLS we find the β values that minimize the residual sum of squares (RSS), which here is a cost function dependent on the given data y_i (data) and the parametrized values $\tilde{\mathbf{y}}_i = \mathbf{X}\hat{\beta}$ (model):

$$C(\hat{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \mathbf{x}_{i*}\hat{\beta})^2 = \frac{1}{n}\{(\mathbf{y} - \mathbf{X}\hat{\beta})^T(\mathbf{y} - \mathbf{X}\hat{\beta})\} \tag{2}$$

$\mathbf{X}$ is our design matrix which contains the predictors. Then to find the minimum of the cost function we set $\frac{\partial C}{\partial \hat{\beta}} = 0$ and solve

$$\min_{\hat{\beta} \in \mathcal{R}^p} C(\hat{\beta}).$$

This should give us the optimal β values as

$$\hat{\beta}^{\text{OLS}}_{\text{optimal}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3}$$

These β values are the model values in the OLS model.

Taking the inverse of a matrix product $(\mathbf{X}^T\mathbf{X})^{-1}$ may be an expensive operation for bigger matrices. That is why we take use of the Singular Value Decomposition (SVD) to

prevent taking the inverse of the product of twp big matrices. From the SVD theorem we can expand our design matrix as:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

This is written in terms of the orthogonal transformation $\mathbf{U}$, the diagonal matrix $\mathbf{D}$ containing the eigenvalues and $\mathbf{V}^T$ which is also an orthogonal matrix. By inserting this into the OLS we get:

$$\tilde{\mathbf{y}} = \mathbf{X}\hat{\beta}^{\text{OLS}}_{\text{optimal}} = \mathbf{U}\mathbf{U}^T\mathbf{y} \tag{4}$$

This is now a more CPU friendly calculation of the OLS for finding the regression parameters $\hat{\beta}$.

2.3 Ridge regression

A problem with machine learning and regression is overfitting of the data when the model complexity increases. The Ridge regression tries to take this into account by modification of the OLS method by introducing a complexity parameter λ, also called the ad-hoc fix for singularity of $\mathbf{X}^T\mathbf{X}$ (van Wieringen [3]). The Ridge method shrinks the coefficients β to go to zero as λ increases. The complexity parameter is a parameter that controls the amount of shrinkage of the coefficients (Hastie et al. [2]). The cost function for Ridge can then be written as

$$C(\hat{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \mathbf{x}_{i*}\hat{\beta})^2 + \lambda\sum_{j=0}^{p-1}\beta_j^2 = \frac{1}{n}\{(\mathbf{y} - \mathbf{X}\hat{\beta})^T(\mathbf{y} - \mathbf{X}\hat{\beta})\} + \lambda\hat{\beta}^T\hat{\beta}. \tag{5}$$

The last term above is called the penalty term. Then to find the optimal β values we again take the minimum of the cost function to get

$$\hat{\beta}^{\text{Ridge}}_{\text{optimal}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \tag{6}$$

where $\mathbf{I}$ is the identity matrix with dimension $p \times p$ and $\lambda \geq 0$. For orthonormal matrix (which we get when using the SVD in the OLS method) we get $\mathbf{X}^T\mathbf{X}$=$\mathbf{I}$ and

$$\hat{\beta}^{\text{Ridge}} = \frac{1}{1+\lambda}\hat{\beta}^{\text{OLS}}$$

We see that for $\lambda = 0$ we end up with the expression for the OLS method, and since λ is defined to never be negative we get that the β values are smaller than for the OLS. This will make the model less sensitive to data set changes and making it less likely to overfit the training data. The model now should give a decrease in variance as the λ increases, but then in return give an increase in the bias.

2.4 Lasso regression

The Lasso method is similar to the Ridge method in that case it has a complexity parameter λ, but now we change the penalty term in the cost function equation 5:

$$C(\hat{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \mathbf{x}_{i*}\hat{\beta})^2 + \lambda\sum_{j=0}^{p-1}|\beta_j| \tag{7}$$

A difference with the Lasso is that it uses methods like gradient descent to find the $\hat{\beta}$ that minimize the cost function. For the Lasso there is no explicit expression for the model parameters like for the OLS and Ridge methods. In this project we will just use Scikit-Learn's Lasso function, instead of making our own which can be difficult. The equation for the regression parameters is:

$$\hat{\beta}^{\text{Lasso}}_{\text{optimal}} = \min_{\hat{\beta}\in\mathcal{R}^p}\left[\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \mathbf{x}_{i*}\hat{\beta})^2 + \lambda\sum_{j=0}^{p-1}|\beta_j|\right] \tag{8}$$

Like the Ridge method the task is to make the β values go to zero, but in a slightly different way. So the variance should decrease while the bias should increase with increasing λ also here.

2.5 Confidence interval

For the confidence interval of the parameters β we need to calculate the variance of the models. The variance of the OLS model can be calculated as

$$Var(\hat{\beta}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}, \tag{9}$$

where σ^2 is the variance of the normally distributed error ϵ_i in equation 1. This variance can be calculated as

$$\sigma^2 = \frac{1}{n-p-1}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2. \tag{10}$$

In Hastie et al. [2][1] it is shown that the confidence interval of β_j can be calculated as

$$(\hat{\beta}_j - z^{(1-\alpha)}v_j^{\frac{1}{2}}\hat{\sigma}, \hat{\beta}_j + z^{(1-\alpha)}v_j^{\frac{1}{2}}\hat{\sigma}) \tag{11}$$

For a 95% confidence interval we have $\alpha = 0.05$ such that we get $z^{(1-0.05)} = 1.96$. v_j is the j-th diagonal element of $(\mathbf{X}^T\mathbf{X})^{-1}$.

[1] Chapter 3 - equation (3.14)

2.6 Error evaluation

Two very useful tools for evaluating the error in predicted values are the Mean Squared Error (MSE) and the R^2 score function. The MSE is calculated as

$$MSE(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y})^2, \tag{12}$$

and the R^2 score function is calculated as

$$R^2(\mathbf{y}, \tilde{\mathbf{y}}) = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \tilde{y})^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2} \tag{13}$$

with the mean value of the data y as

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

$\mathbf{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value of the data. For the MSE we want the error to be as small as possible, close to zero. For the R^2 score, the best value we can get is 1. This can be described as how much variance in the data is accounted for by the model. For a score of 1 means that the prediction fits the data perfect. The score can lie in the interval $(-\infty, 1]$, meaning that the score can be negative. This can be interpreted such that the mean is a better fit than the model used.

2.7 Bias-variance tradeoff

To find the regression parameters $\hat{\beta}$ we have to look at the cost function again:

$$C(\mathbf{X}, \hat{\beta}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] \tag{14}$$

This can be rewritten as

$$\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \frac{1}{n} \sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{n} \sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2. \tag{15}$$

The full calculation can be found in appendix B.

From the rewritten cost function we now have three terms in the equation above. The first term is the bias squared

$$Bias = \frac{1}{n} \sum_{i=0}^{n-1} (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2. \tag{16}$$

The second term is the variance of the model

$$Var = \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 \tag{17}$$

The last term σ^2 is the irreducible error of a normally distributed noise.

Here we can see that if the variance is decreased then the bias is increased, and the opposite for increased bias. For increasing complexity/degree of polynomial, we would expect the variance to increase and the bias decrease. This is the bias-variance tradeoff. The bias of the model can be seen as the measure of the tendency the model has to either overestimate or underestimate the predictions. When we have many β parameters then we may get something called overfitting. This is when we get too high variance/low bias of the model. This comes from that the model may start to fit the noise as well a the training data. One precaution is to split the data into training and test data. This is to avoid the noise being take with the test data. Another method is to add a term to reduce the variance even though that will increase the bias. Often the increase in the bias is so small, such that the tradeoff we get is acceptable. On the other end we can have underfitting for high bias and low variance. This happens when the complexity is very small so that the model can't pick up features of the data set.

2.8 Resampling

We don't always have enough data to be able to fit our model. To solve this problem we can use resampling of the data. A way to split our data set is to use cross-validation techniques like the k-fold cross-validation resampling. This splits the data set into training and test data. The main idea of resampling techniques is to gain more data by drawing arbitrary samples from the training data and refitting a model on each sample set. These new sets can then be evaluated to see how well they correspond to each other. The k-fold splits the data into k sets/folds of data. Normally this is between 5 and 10 folds, while normally we split the data into 70-80% training data and the remaining test data. One of the folds is set as a validation (test) set, while the $k-1$ other folds are trained on our model for given hyperparameters. Then we test our fitted training model on the validation set. Then we calculate the evaluation score for each iteration from the test data. This is then used as the expected prediction error. This procedure is repeated until all the folds have been validated.

2.9 Data

The first data set in this project is the data we get from fitting polynomials to the two-dimensional Franke function:

$$\begin{aligned} f(x,y) =& \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\right)+\frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49}-\frac{(9y+1)^2}{10}\right) \\ &+\frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\right)-\frac{1}{5}\exp\left(-(9x-4)^2-(9y-7)^2\right) \end{aligned} \tag{18}$$

This function is defined for $x, y \in [0,1]$, and is a widely used function for testing various interpolation and fitting algorithms. With this function we also add a normally distributed noise.

The second set is (real) digital terrain data that we download from the U.S. Geological Survey EarthExplorer website [1]. The chosen digital terrain data we use is of Hardangervidda in Norway.

3 Methods

For us to study the three regression methods, we start by looking at Franke's function in equation 18 with the added stochastic noise. This data with added noise will act more as real data, since real data is most likely not as smooth as the original Franke function. We choose a noise with standard deviation $\sigma = 1$ to test the error functions. For this noise we would expect poor results since the noise is of the same order as the data. After this test we choose $\sigma = 0.2$ in the rest of the project. With this functions we can study how well the regression methods fit for the data we get from the function. First we fit the OLS regression method to the data. Equation 4 is solved in Python using NumPy. We construct first a matrix with degree of polynomial 5 and with $N = 50$ number of data points. With this matrix we calculate the optimal β values using the SVD method. Then we use these regression parameters to calculate the variance (eq.9), MSE (eq.12), the R^2 score (eq.13) and bias (eq.16). Now we can calculate the confidence interval of the regression parameters of the OLS. There we use the Franke function with and without the noise to see the reliability of the OLS method on the Franke function, and plot for the 98% confidence interval case. Then we apply the k-fold cross-validation method on the OLS method to perform a resampling. We will use 5 folds in the k-fold resampling. We these resampled data, we again calculate the error measurements and compare for the Franke function with and without noise. Next, we use the k-fold to calculate the bias-variance tradeoff (basically the same as the last calculation). Now we will plot the error measurements as the figure in Hastie et al. [2] (Fig. 2.11) for the training and test data. In that figure we don't see how the variance and bias change with increasing complexity. So

we also plot the variance and bias separately for the test data. From the OLS analysis we can come up with an optimal matrix dimension that fits the model to the data best.

Next up is to do the same analysis for the Ridge and Lasso regression as we did for the the OLS. Since the Ridge and Lasso are dependent on a hyperparameter λ, we have to run through and test for the best hyperparameter as well. For the Ridge method we solve equation 6 again using NumPy. For the Lasso we have to use SciKit-Learns's built in function to solve equation 8. Since we now have to dependency-parameters, we choose to make a heatmap using the python function seaborn. With this function we look at the R^2 score function, which gives us how well fitted the model is, using the different combination of dimensions and hyperparameters. Using this heatmap we can conclude best fit parameters for the two regression methods.

Now with the best fit parameters for all the three regression methods, we can conclude which model fits the data the best.

With the analysis of the Franke function completed, we can analyze real terrain data using the three regression models. The terrain data we are analyzing are of an area of Hardangervidda in Norway [1]. The data set is a two-dimensional data set which contain almost 6.5 million data points. Our computer cannot analyze the full data set, since this requires to much memory usage. So we will only use every 30th data points in both directions, giving us about 7400 data points to analyze. For the k-fold cross-validation we change the implementation a little to more suit the terrain data, and choose 75% as training data and 25% as test data. Then we use the three regression models on the reduced terrain data to see which model fits the terrain data the best.

4 Results

4.1 Franke's function

For the Franke function we started by having a stochastic noise with standard deviation $\sigma = 1$. The results of this can be seen in Table 1. There we see that the MSE is high and the R^2 score is low. This is as we suspected in the method section, poor results. Again, this raises from that the noise if of the same order as the data set from the Franke function. We get this results for all three regression methods. The reason is that the models will try to fit a polynomial to the stochastic noise. So for this order of noise, we conclude that none of the regression methods are sufficient for modeling this situation. That is why we choose a much lower standard deviation as $\sigma = 0.2$ through the rest of the project with Franke's function. In Table 2 we see that with lower standard deviation in the noise, we get much better results for all the three regression models.

For the OLS regression of the Franke function we calculate the confidence intervals of the regression parameters β. This is seen in Figure 1, and there we can see that the confidence intervals with noise can vary from being small to be a lot bigger than without

	Without noise	With $\sigma = 1$	With $\sigma = 0.2$
MSE	0.002136	1.038669	0.043992
R^2 score	0.974316	0.092237	0.084214

Table 1: Table for the MSE and R^2 score of the Franke function for complexity 5. For the Franke function without noise we see that the MSE is low and that the R^2 score is close to 1. With increasing standard deviation σ we see that the values are getting more bad. For $\sigma = 1$ the noise is so high that the results we would get wouldn't give anything useful. That is why we use $\sigma = 0.2$ for the rest of the Franke's function analysis.

Model	MSE	R^2 score
OLS	0.043992	0.656863
Ridge	0.043993	0.656861
Lasso	0.067861	0.470688

Table 2: This is a table for the MSE and the R^2 score with complexity 5, hyperparameter 10^{-3} and stochastic noise $\sigma = 0.2$ for the three regression methods. Now we see that the values are much better than for in Table 1 for high noise.

the noise. For the confidence intervals of Ridge and Lasso, we look at their dependence of the hyperparameter λ as well a the complexity. These plots can be seen in the GitHub repository A. The plots behave in a similar way like Figure 1 for the OLS. For the Lasso method in Figure 2 for hyperparameter $\lambda = 10^{-7}$, we see that the blue dots will be more or less on zero. This is most likely because of the way that Lasso pushes the regression parameters to zero, while Ridge only pushes them towards zero.

Now we look at the results we get when we introduce the k-fold cross-validation resampling on the regression methods. In figure 3 we see the bias-variance tradeoff for the OLS model with resampling for 5 folds. There we see the prediction error (MSE) that we get from the resampling. For low model complexities we see that the training and test curves are close to each other. For higher complexity we see the test curve rising. This comes from overfitting the data. We have included the areas for the bias and the variance of the data, and plotted them separately in Figure 4. There, it is easier to see how they evolve for increasing model complexity. As stated earlier we expected the variance to be low and increase with higher complexity, and the opposite for the bias. This conclusion is correct from the results we get in the mentioned figures. One thing to notice is that the bias is not as smooth as the variance. The curve goes up and down, but seems to decrease overall. The reason for this sporadic behavior is unknown. For the bias-variance tradeoff with the Ridge model in Figure 5, we see that the prediction error is low for very small hyperparameters. When we start getting closer to zero, the prediction error rises

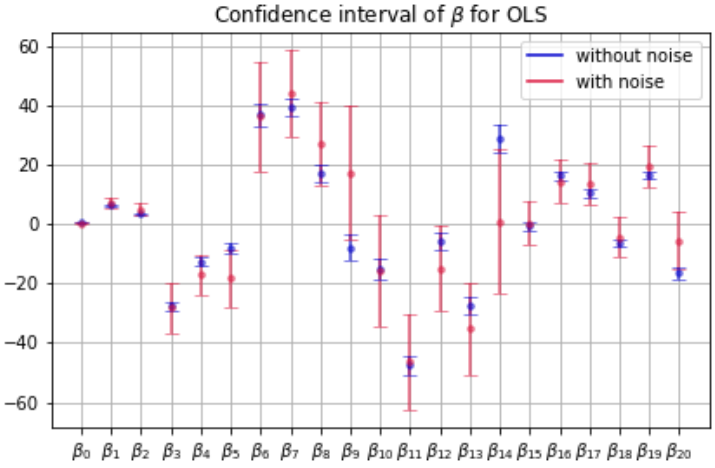


Figure 1: This is a plot of the 98% confidence intervals of the regression parameters β for the OLS model. The blue dots/lines indicate the confidence intervals for the Franke function without noise, while the red indicates the confidence intervals with noise ($\sigma = 0.2$). As we see even with a small noise added, the confidence intervals can be quite large. By following the link in Appendix A to GitHub, we can see a plot for the confidence intervals for $\sigma = 1$, and they are much much bigger than what we got here.

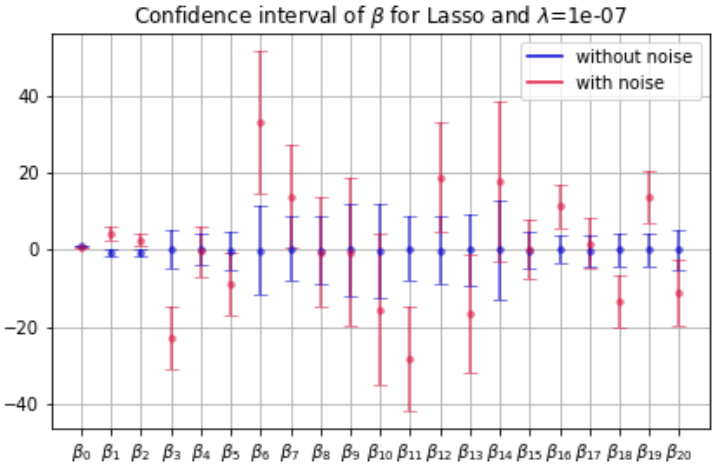


Figure 2: This is a plot of the 98% confidence intervals of the regression parameters β for the Lasso model with $\lambda = 10^{-7}$. Here we see the blue for the Franke function without noise laying more ore less on zero, while the red noise is a little more scattered above and belove zero.

quickly. In this scenario (and for Lasso), we have low bias and high variance for small values of λ. For changing model complexity for both the Ridge and Lasso, can be seen in the GitHub repository A (also with and without noise in the Franke function). The bias-variance tradeoff is very similar for Ridge and Lasso, so it is redundant to include many figures that look and behave the same in this report. That includes also the variances and biases for the two regression models.

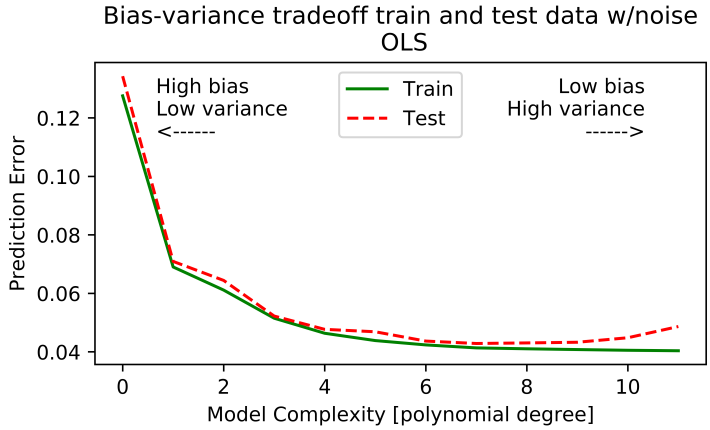
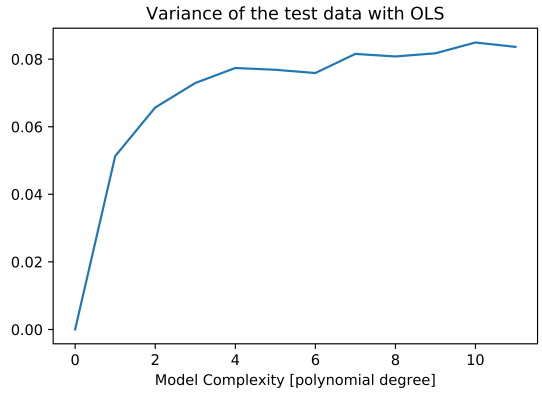
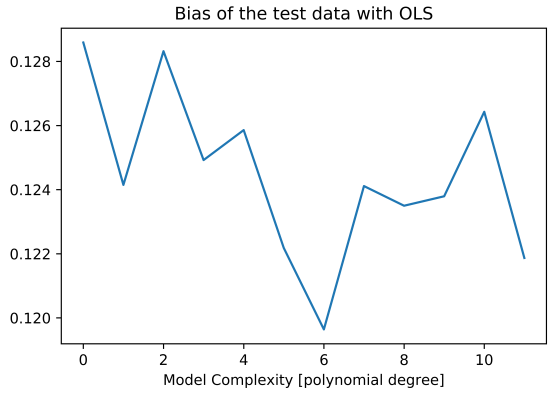


Figure 3: In this figure we see the bias-variance tradeoff for increasing model complexity for the prediction error (MSE) for training and test data of the Franke function with noise. This is for the OLS model with k-fold cross-validation resampling with 5 folds. When the model complexity is getting higher, around 8-9, we see that the test curve starts to increase more. This comes from overfitting. In this figure we have included the regions of the variance and the bias as well, but these can be individually seen in Figure 4.

Now we have almost everything we need to determine which model fits the Franke function data best. We only need to find the best values of the regression parameters and hyperparameters. For the OLS method we ran through the model complexities and calculated the MSE and R^2 score (and also variance and bias). These values were stored in a text file to easier see the values, for easier implementation into LaTeX. From the calculated MSE and R^2 score for the OLS with resampling, we find that for model complexity $n = 9$ we get the best fit. The calculated values can be seen in Table 3. To find the best fit values for Ridge and Lasso we used a Python function called seaborn to make a mapping of the model complexities and the hyperparmeters. In Figure 6 we see the mapping for the tested parameters for Ridge. The mapping for Lasso looks almost the same and give the same parameters as the best fit (see GitHub A). From Table 3 we see that best parameters for the models that fit them best to the Franke function with noise. We see that we get low MSE and somewhat high R^2 score. This seems reasonable since we have a noise term added to the data. From the results we get in that table, we can conclued that the best fit



(a) Plot of the variance of the data with OLS and resampling



(b) Plot of the bias of the data with OLS and resampling

Figure 4: Here we see how the variance and bias behave for increasing model complexity for the Franke function with the OLS model and resampling. As expected for the OLS model, the variance increases with the model complexity, while the bias (somewhat) decreases. This is the core of bias-variance tradeoff where they are dependent on each other, so if one increases then the other decreases. We also see that there is a much higher change in the variance than in the bias.

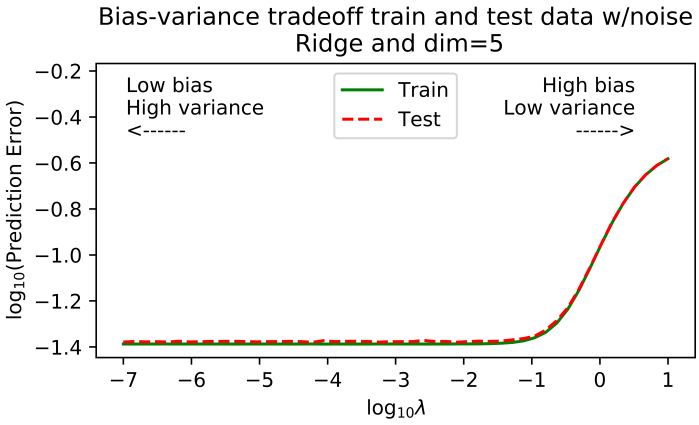


Figure 5: In this figure we see the bias-variance tradeoff for increasing model complexity for the log of the prediction error (MSE) for training and test data of the Franke function with noise. This is for the Ridge model with k-fold cross-validation resampling with 5 folds and model complexity 5. Here we have low prediction error for low hyperparameters. When the hyperparmeters are getting close to zero, the data rise quickly. We also see the areas for the variance and bias. For other model complexities, the variance, bias and for the Lasso model, go to the GitHub repository A.

model is the OLS method. The OLS has the best wanted parameters, where we see that the Ridge gives the same values for a very small hyperparameter. This fits well with that we expect Ridge and Lasso to converge to OLS for very small hyperparameters. To illustrate how the model fits the data, we make a 3-dimensional plot of the best OLS model and the generated data in Figure 7. The 3D plots of the Ridge and Lasso can be seen in the GitHub repository, and they look very similar to the figure for the best OLS model. In that figure we can see that it looks like the fitted OLS model seems to correspond well with the generated Franke function data.

Model	MSE	R^2 score	Model complexity n	Hyperparameter λ
OLS	0.040959	0.680520	9	-
Ridge	0.04059	0.680520	9	10^{-7}
Lasso	0.042758	0.666487	9	10^{-7}

Table 3: This is a table for the lowest MSE and the best R^2 score for the three regression methods for the best fit model complexities and hyperparameters. Here we see that the best fit model to the Franke function is the OLS method. The Ridge is very close for a very small hyperparameter. As we mentioned in the methods section, we see that when the hyperparameter goes closer towards zero, then Ridge and Lasso go towards OLS.

4.2 Digital terrain data

For the terrain data of an are of Hardangervidda in Norway, we had to reduce the amount of data points drastically. The total amount of data points are around 6.5 million, while the reduced terrain data only has around 7400 data points. For more points than that, the calculations take a very long time. Then we fitted the OLS model with k-fold resampling to the digital terrain data. We then ran through several model complexities to find the best fit dimension for the OLS model, which we saved to a text file (this can be seen in the GitHub repository) for easier readout. We also plotted the bias-variance tradeoff for this case, which can be seen in Figure 8. There we see that for higher model complexities, the prediction error slope starts to even out. So for polynomial around 15 we see that the prediction error seem to stabilize. So we can then conclude that polynomial 15 is the best fit for the OLS model on the terrain data, since the prediction error seem to be about the same for higher polynomials. The plots for the variance and the bias for the OLS model can be seen in the GitHub repository. We have not included them since we have included their areas in Figure 8, and they have the same form as the variance and bias for the Franke function in Figure 3. The only difference is that the values are much higher in this digital terrain data case. This comes from that we have a lot more data that acts like a noise term to a function, such that we do not have smooth curves like the original

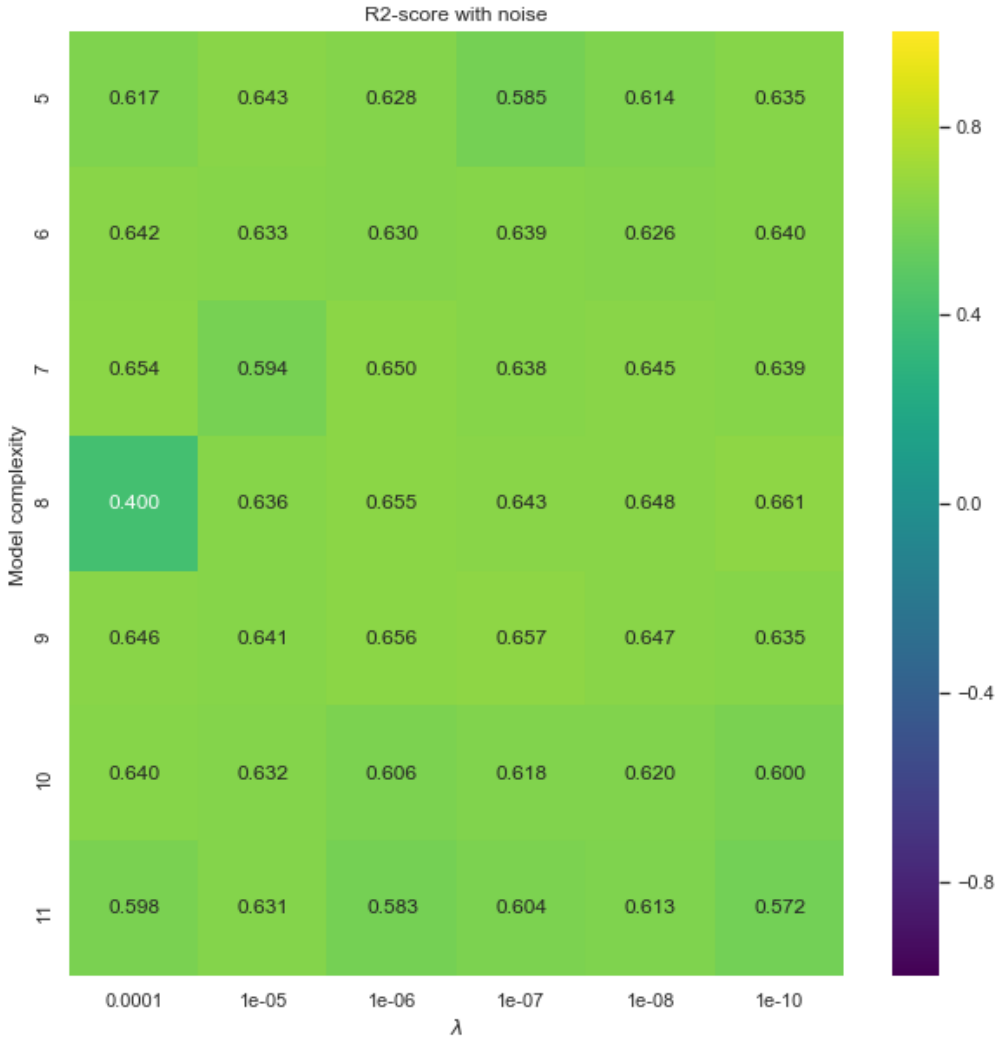


Figure 6: In this figure we see a heatmap of the R^2 score of tested model complexities and hyperparameters for Ridge with k-fold CV. Here we see the correlation between the model complexities and hyperparmeters for Ridge. From this figure we see that the best score is for $n = 9$ and $\lambda = 10^{-7}$. We use these values as the best since we get the same parameters for Lasso. It is not the best, but the differences are so small and the best we choose fit well for both methods. For the Lasso heatmap, go to the GitHub repository A.

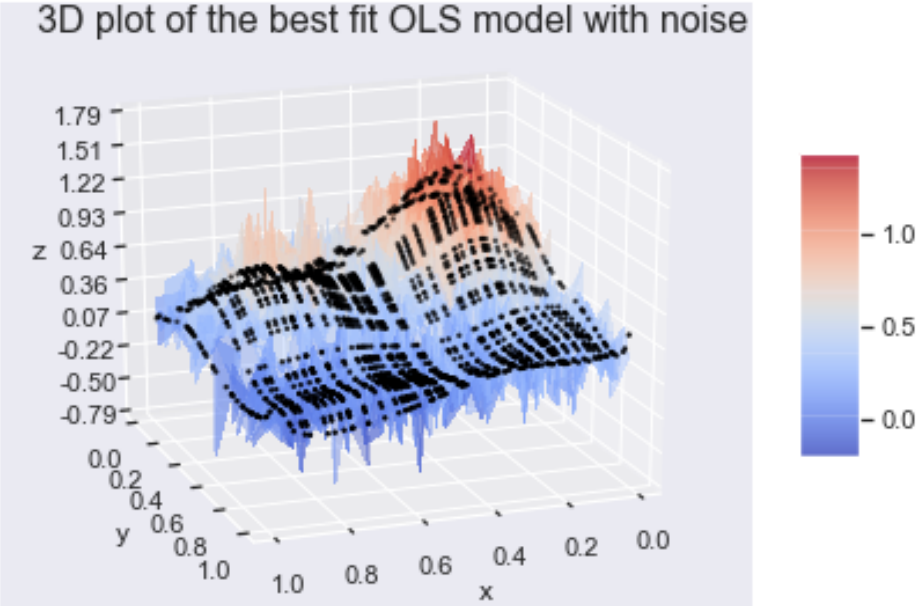


Figure 7: In this figure we see a 3D plot of the best fit OLS model, which is plotted as the black dots on the Franke function data. Here it looks like the model fit seems to fit the Franke data quite well. By looking at the colorbar to the right and compare with the Franke data, we can see where the curve goes up and down, which seems to be the same for the fitted OLS model. For the Ridge and Lasso, go to the GitHub repository A.

Franke function. As we see in Figure 8, we see that the MSE is also very high. So it also makes sense to have a higher polynomial fit for since we have much more data to fit for the OLS. Now with the best fit polynomial for the OLS model, we made a 3-dimensional plot of the best fit OLS and the digital terrain data to illustrate how well fitted the OLS model is to this data. This is seen in Figure 9. There we see that the OLS model seems to fit the digital terrain date well, at least for low areas in the terrain (low on the z-axis). In those areas we have many dots close to each other, while higher up there are not so many dots as close. We get similar results for the best fit Ridge and Lasso models. In Figure 10, we have the 3D plot of the best fit values for Lasso. This plot is not as good as for OLS and Ridge, which look almost the same. For Lasso we don't seem to get as much of the shapes as for OLS and Ridge. OLS and Ridge tend to give more ups and downs which fits the terrain data better. The Lasso only seems to give smooth curves (of dots) in the 3D figure, which does not totally fit the expected shape of the terrain data. In Table 4 we see the best fit values for the MSE, R^2 score, variance and bias for the three best fit models of the terrain data. So from this data we see that Ridge has a slightly better MSE and R^2 than OLS, and they are both better than Lasso. This fits well with the 3D plots of the best fitted values for the models, where OLS and Ridge give more accurate descriptions of the terrain data.

A factor to take notice of in our programming is that if we run several tasks at once (or especially several plotting tasks), then we may end up with different results than if we had run it one task at a time. So along the way, some of our results may have been affected by other implementations in our program to not give the exact best fit values. The k-fold cross-validation also had to be altered when we started to analyze the terrain data. So it is possible that these functions are not as optimal as they could be. So there are things in our programs that may alter the results in a different way than they really should, like when running several tasks at the same time.

Model	MSE	R^2 score	Variance	Bias	Model complexity	Hyperparameter
OLS	4061	0.773255	13404	17920	15	-
Ridge	4002	0.776567	13415	17918	16	10^{-5}
Lasso	5988	0.665718	11085	17928	16	10^{-5}

Table 4: This is a table for the best fit values of the three regression models for the digital terrain data. Here we see that the Ridge model is slightly better than the OLS model, and they are both better that the Lasso model. We expected the OLS model to be the best fit with Ridge to converge to OLS for small hyperparameters, and that is almost what we got. The difference between the OLS and Ridge is very small, so it may be that the best fit model is the OLS and that we could have chosen more accurate values.

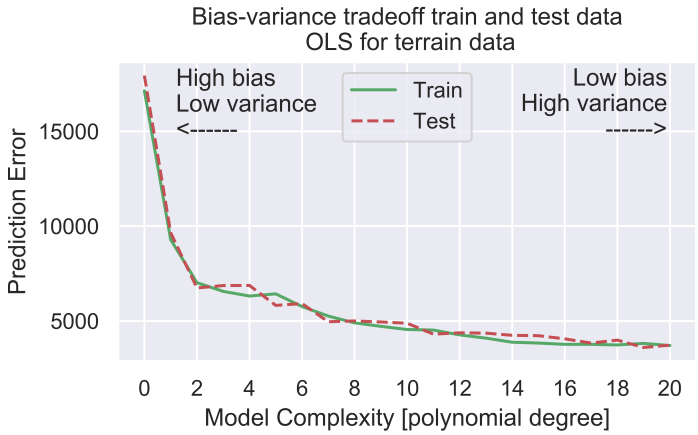


Figure 8: In this figure we see a plot of the bias-variance tradeoff with the OLS method using resampling on the digital terrain data. Here we see that the prediction error decreases fast for small polynomial in the beginning. Then the slope starts to even out. For model complexity around 15, we see that the prediction error starts to become stable around a minimum. This may indicate that after polynomial 15 we don't get any better error. So we may conclude that polynomial 15 is the best fit for OLS to the terrain data. Ridge and Lasso give similar results as for the Franke function, so we have just put the bias-variance tradeoff figures at the GitHub repository A.

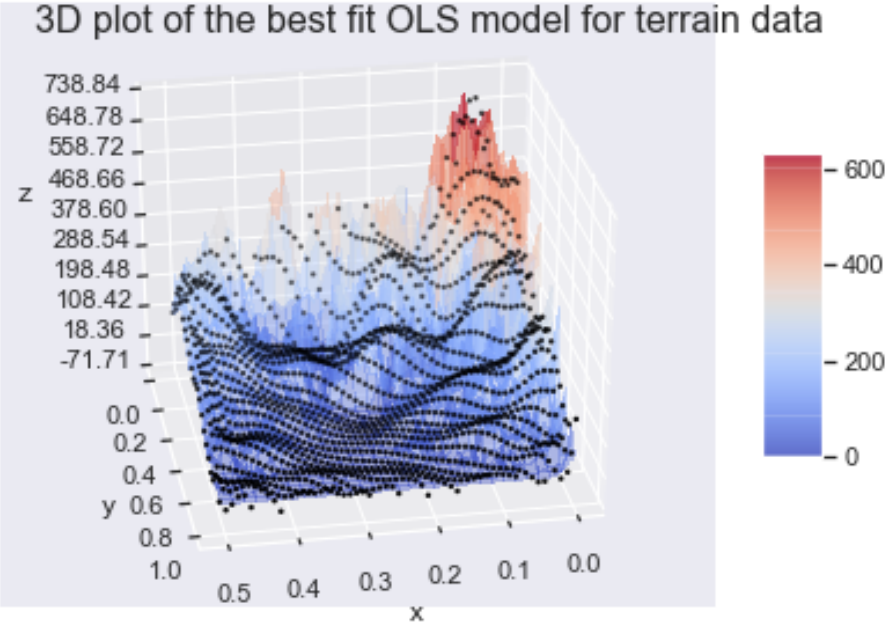


Figure 9: In this figure we see a plot 3-dimensional plot of the best fit OLS model with polynomial degree 15, as the black dots, with the digital terrain data. Here we see that it looks like the OLS model seems to fit, at least for the low values in the colorbar (low in the terrain). This is seen by the increased number of dots for the lower terrain field. Ridge give very similar result so we have just put the 3D figure at the GitHub repository A.

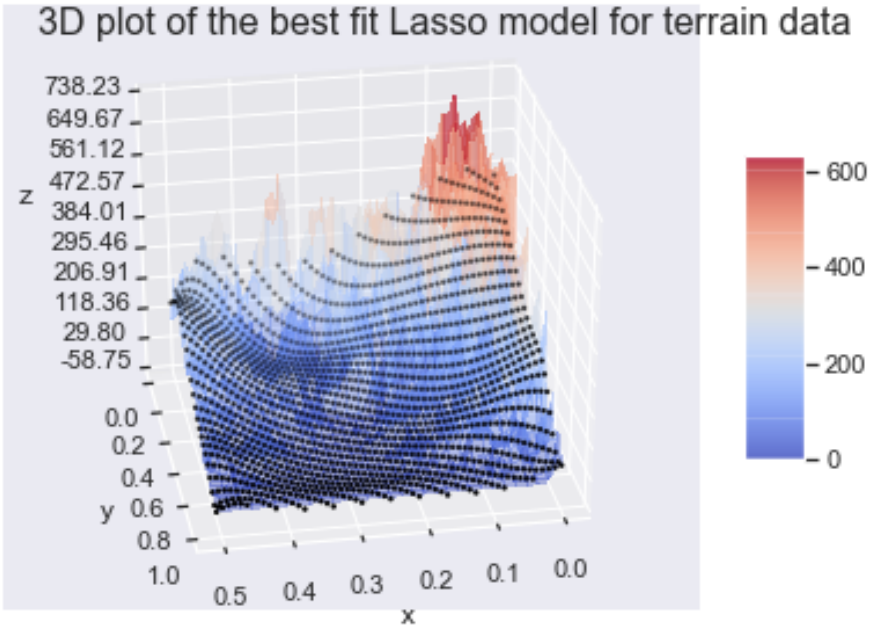


Figure 10: In this figure we see a plot 3-dimensional plot of the best fit Lasso model with polynomial degree 16 and hyperparameter 10^{-5}, as the black dots, with the digital terrain data. If we compare this with the best fit for the OLS in Figure 9. We can see that the OLS seems to fit better as the dots seems create more curves in the plot, which the Lasso doesn't for the low parts in the terrain. This seems reasonable by looking at the best fit values in Table 4, where the values for the Lasso is not as good as for OLS and Ridge. This gives us not as equally good fit in this figure as for OLS.

5 Conclusion

In this project we have analyzed two data sets of different sizes by using three different linear regression models to see how well the models can fit the data sets. To see this we have calculated the MSE and the R^2 scores to evaluate the performance of the three methods. The three methods we used were the OLS, Ridge and Lasso regression methods. These were first used to analyze the data sets we got for Franke's function with added noise. Then we analyzed a real digital terrain data set from Hardangervidda in Norway. With the regression methods we have used k-fold cross-validation resampling with 5 folds to find the optimal polynomial degrees and hyperparameters. We found that the OLS fits both the data sets quite well for polynomial degree 9, for the Franke data, and degree 15 for the digital terrain data. The Ridge model were equally as good or even slightly better than the OLS for polynomial degree 9 and 16, and hyperparameters 10^{-7} and 10^{-5}. We earlier assumed that for very small hyperparameters, then Ridge would converge to the OLS. When evaluating the data sets we did not go fully in depth into the different parameters. Had we done this then we might have found that the OLS would have the upper hand on Ridge. As we saw in some of the figures, they would stabilize around some parameters, but not seem to give one absolute best parameter. For the Franke function, we only had 400 data points, and for the digital terrain data we had to decrease the number of data points by a lot. If we could evaluate more of the terrain data, then we might have got better fits for other parameters. But all in all, Ridge and OLS gave us satisfactory model fits to these data, and seem to be better than the Lasso in these cases.

For the future, we could try out other evaluation models to see if they are better fits to the data sets. We could also try to improve our numerics such that we could evaluate more data points from the digital terrain data.

A Appendix

Link to GitHub repository:
https://github.com/krilangs/FYS-STK4155/tree/master/Project1

B Appendix

The full calculation of the rewritten cost function from equation 14 to the bias-variance tradeoff equation 15:

We start by using that $\mathbf{y} = f(\mathbf{x}) + \hat{\epsilon}$ and $\tilde{\mathbf{y}} = \mathbf{X}\hat{\beta}$.

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

Here we have used that the expectation value of $\hat{\epsilon}$ is $\mathbb{E}[\hat{\epsilon}] = 0$, and that the expectation value of $\tilde{\mathbf{y}}$ is $\mathbb{E}[\tilde{\mathbf{y}}] = \mathbf{X}\hat{\beta}$ [2][2]. Then we split the equation such that we get:

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

Now we use that the expectation value of $\hat{\epsilon}^2$ is $\mathbb{E}[\hat{\epsilon}^2] = Var[\mathbf{y}] = \sigma^2$ since $\mu = 0$. The other expectation values can be discretized in terms of sums as:

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

References

[1] **U.S. Geological Survey EarthExplorer**. URL https://earthexplorer.usgs.gov/.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. **The elements of statistical learning**: data mining, inference and prediction. *Springer-Verlag, New York*, 2009. ISSN 0172-7397. doi: 10.1007/b94608. URL https://doi.org/10.1007/978-0-387-84858-7.

[3] Wessel N. van Wieringen. **Lecture notes on ridge regression**, 2015.

[2] Chapter 7.3 - equation (7.9)