Project 1, FYS-STK4155

Sigurd Holmsen, Øystein Høistad Bruce

October 2021

Introduction to the code structure:

We have decided to create a file, **helper.py**, which will store all the functions that are not used for just one exercise. For simplicity, we have created 6 files for the 6 different exercises. The code will also provide some helpful docstrings, so if something are unclear please check them out.

We have created a helpful testing file, **test_project_1.py**, where you can reproduce the exercises with different parameters. The structure of this file is very simple, and it will explain (shortly) how to reproduce the exercises.

Check out the **README.md** file inside the **project1**-folder for further information about the code.

Exercise 1

Ordinary Least Square (OLS) on the Franke function

Nature of the problem

We want to find a linear model which approximates a continuous function using polynomials with two arguments (x, y) of high degrees. More precisely, we want to estimate parameters $\boldsymbol{\beta}$ such that

$$z_i = \beta_0 + \beta_1 x_i + \beta_2 y_i + \beta_3 x_i y_i + ... + \beta_{(k-1)} x_i y_i^{m-1} + \beta_k y_i^m + \epsilon_i \qquad (1)$$

for all $i \in 1, ..., n$ where n is the number of data/ observations, and for a polynomial of degree m, where the length of $\boldsymbol{\beta}$, k, depends on the polynomial degree m. ϵ_i is the error term for each approximation, and we are looking for a model where the error is as small as possible. More precisely, we want the "Mean Square Error" to be small. MSE is given as the mean square of the difference

between the (true) data $\boldsymbol{y}$ and the approximated data $\tilde{\boldsymbol{y}}$:

$$MSE = \frac{1}{n}\sum_i (y_i - \tilde{y}_i)^2 \tag{2}$$

A small error implies that our linear regression model is a good approximation to the function we want to approximate. In other words, a small error implies that the model is good at predicting values.

Generating the data

We have used the Franke function to generate the (output) data used in the regression, by selecting random pairs of x- and y-values from the uniform distribution on the interval 0 to 1. We have added a noise variable to each data point which is normally distributed as $N(0, 1)$. The code also includes a constant which can scale the noise variable, this constant can be changed by the user of the program.

Perform a standard least square regression analysis

To perform the standard least square regression, we first split the data into training and testing. The default testing size is 0.2 of the data. Then the data is scaled, before we compute the design matrix $\boldsymbol{X}$ (with the training data). The design matrix contains the dependent variables (used in the regression), and finally we use this along with the response variable $\boldsymbol{z}$ to compute the optimal regression parameters $\boldsymbol{\beta}$. The formula for calculating these parameters with ordinary least squares regression (OLS) :

$$\boldsymbol{\beta} = (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T y} \tag{3}$$

Confidence intervals of the parameters $\boldsymbol{\beta}$

We assume that the $\boldsymbol{\beta}$ values are normally distributed, and $\sigma^2(\beta_j) = \sigma^2[(\boldsymbol{X}^T\boldsymbol{X})^-1]_{jj}$. We also assume that the expected value of the $\boldsymbol{\beta}$ values are the observed $\boldsymbol{\beta}$. Using this, we can compute the confidence intervals of the $\boldsymbol{\beta}$ parameters (in the code).

Evaluating the Mean Squared error (MSE) and R^2 score

We calculate the Mean Square Error with (2): where $\tilde{y}_i$ are the predicted outputs using our model, and use sci-kitlearn to evaluate the R2-score.

Why do we scale our data?

When using ridge and lasso regression, we include a term in the cost function which penalizes extreme values in order to avoid poor approximations e.g. overfitting. However, some features are often given in a different unit than other

features, and the different units may differ greatly in magnitude. This will cause the penalizing team to penalize some features in an unfair amount simply by the magnitude of the unit it is given in. To fix this, we scale our data such that every feature may be assessed more equally. In our code, we have subtracted the mean of the data set for every feature as such:

$$x_j^{(i)} \rightarrow x_j^{(i)} - \bar{x}_j \tag{4}$$

In some cases, you may also want to divide by the standard deviation, but in this case we do not take it into account.

Exercise 2

Bias-variance trade-off and resampling techniques

MSE vs the complexity of the model

Figure 1: MSE of the test- and training data

We can expect that the MSE-score of the training data will go to 0 as complexity increases, since the model is built on more information of the training data itself, which may lead to an overfitted model. For the testing data we will see the opposite: when the complexity increases, the model will be more accurate to the training data and the prediction will get worse (since the predictions are obviously not used in the model making).

The terms in (6), an extension of the MSE

We want to find $\boldsymbol{\beta}$ by optimizing the Mean Squared error, via the so-called cost function:

$$C(\boldsymbol{X}, \boldsymbol{B}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] \tag{5}$$

which we can rewrite to:

$$\begin{aligned}
\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[(\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^2] \\
&= \mathbb{E}[(\boldsymbol{y} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + 2\mathbb{E}[\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]] \cdot \mathbb{E}[\mathbb{E}[\tilde{\boldsymbol{y}}] - \boldsymbol{y}] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] \\
&=^1 \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] \\
&= \mathbb{E}[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + 2\mathbb{E}[\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}]] \cdot \mathbb{E}[\boldsymbol{\epsilon}] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] \\
&=^2 \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \sigma^2
\end{aligned}$$

Hence:

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \sigma^2 \tag{6}$$

Comments:

1. $\mathbb{E}[\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]] = \mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}] = 0$
2. $\mathbb{E}[\boldsymbol{\epsilon}] = 0$, and $\mathbb{E}[\boldsymbol{\epsilon}^2] = \mathbb{E}[\boldsymbol{\epsilon}]^2 + \mathbb{V}[\boldsymbol{\epsilon}] = \sigma^2$

The equation (6) shows that the Mean Squared error can be rewritten as the sum of bias, variance and σ^2. The first term is bias, and this is an expression of how much our model is expected to deviate from the actual function we are approximating. It has to be noted that when we estimate this, we do not know the true function we are estimating, and we have to replace f_i with our data y_i. The second term is the variation, and this shows how much our model varies from the expected computed model. This may become large if we have few data/observations to train our model. When deciding what complexity to utilize when creating a model, one has to take into consideration both how it will affect the bias and the variance, as bias often decreases with complexity, while variation usually increases.

Perform a bias-variance analysis of the Franke function

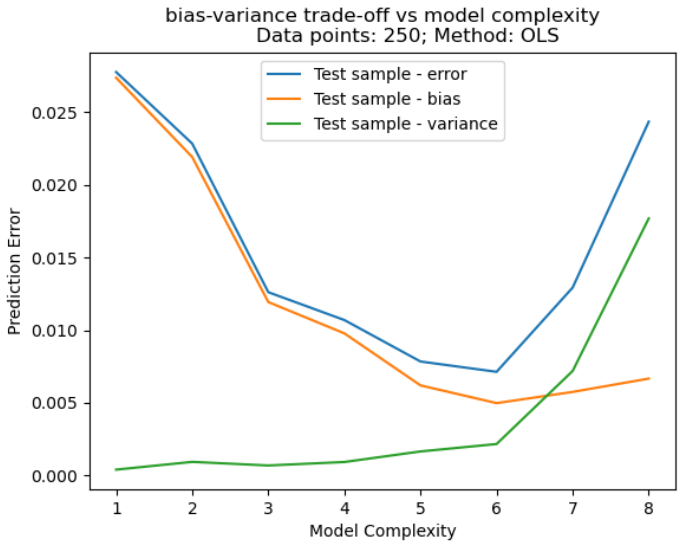

Figure 2: bias-variance figure

We plot the bias, variance and total MSE for the model as a function of complexity where we use the bootstrap resampling method for the beta values. In the plot, we recognize that the bias always decreases when the complexity increases. This makes sense, as the model will not always be able to fit a complicated function using polynomials of low degrees, the variance for such models is always low in the results. When the complexity increases, we usually see the variance increase as well. These two effects makes it such that the MSE will often have a minimum when the complexity is not too high or too low, and hence there is an optimal trade-off between bias and variance. However, if we increase the number of data points, we see that the variance will increase more slowly, and the optimal complexity will increase. Though it appears that you can always get rid of variance in the model by adding more data points, this will not necessarily be possible in real life scenarios, and this insures the importance of the bias-variance trade-off.

Exercise 3

Cross-validation as resampling techniques

Cross-validation

We wrote our own code for the cross-validation technique, and are able to change the parameter for number of folds. We compute the MSE we get from the cross-validation technique between 5 and 10 folds.

Comparison with bootstrap

When we compare the cross-validation and bootstrap techniques, we generally observe that cross-validation gives slightly lower MSE than the bootstrap-technique. This may be because the cross-validation technique does not use the same training-data for every iteration, but uses all the data both for training and testing when building the model. When the number of observations is high, both the bootstrap and cross-validation techniques give somewhat similar results, but this is because the variance in the model parameters is low. It is also worth noting that this technique generally requires less computing power, since the bootstrap method often needs many iterations to approximate an accurate MSE.

Increasing amount of folds

We check the MSE using cross-validation, increasing the amount of folds from 5 to 10, and we see that the MSE generally decreases. This may be because we split the data more times, and put less emphasis on train-test splits which may cause an unusually high MSE.

Exercise 4

Ridge regression on the Franke function with resampling

Performing ridge regression

To perform the ridge regression, we use matrix inversion as with OLS. We let our program take different values for λ.

Comparison with OLS

We perform the same bootstrap and cross-validation methods from exercise 2 and 3. In the bootstrap analysis, we compare the model complexity with the MSE and use the same number of data points and the same polynomial degrees. We observe that the variance for the ridge method is lower, especially when the degree is high as seen in the plot below. This makes sense as the goal

of ridge regression is to penalize extreme values of $\boldsymbol{\beta}$, which should give a lower variance. The MSE seems to increase for very small values of λ as this causes more variance.

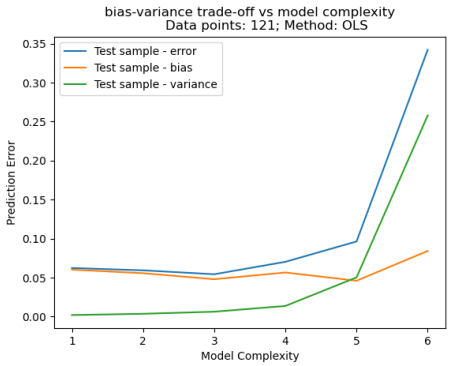

Figure 3: OLS

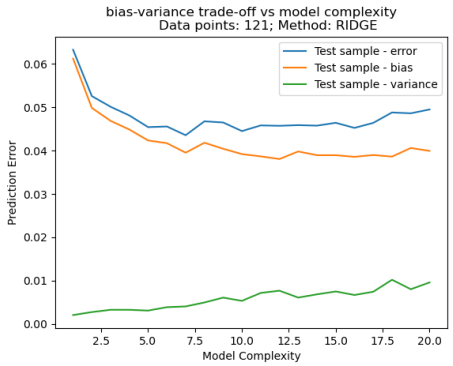

Figure 4: RIDGE (lambda = 0.1)

When comparing the cross-validation results, we observe less MSE from ridge than OLS. All signs indicate that ridge is a better fit than OLS.

Cross-Validation

When evaluating the cross-validation method, the MSE is also dependent on the parameter λ. The MSE achieves a minimum when λ has a value around 0.1, as shown in the produced plot:

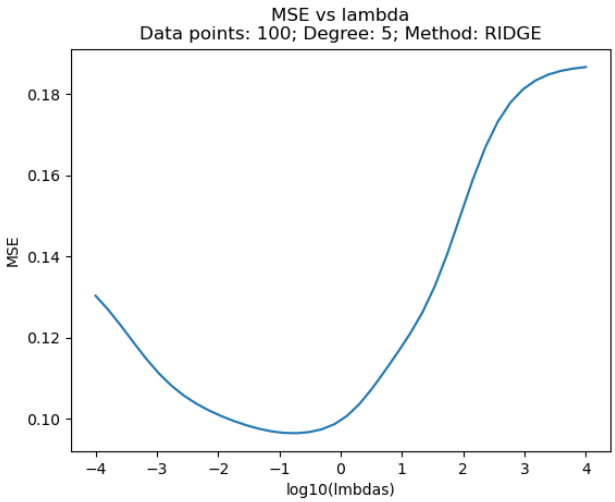

Figure 5: MSE depends on the lambda parameter in ridge regression

The minimum becomes more apparent when the model has significant noise. This is because the model will have more variance, and the penalizing term should not be too small. If the model were to have no noise, it seems it would be optimal to choose as low of a λ as possible, but this is not realistic.

Bias-variance trade-off with Bootstrap

Here, we study the bias, variance and total MSE as a function of λ.

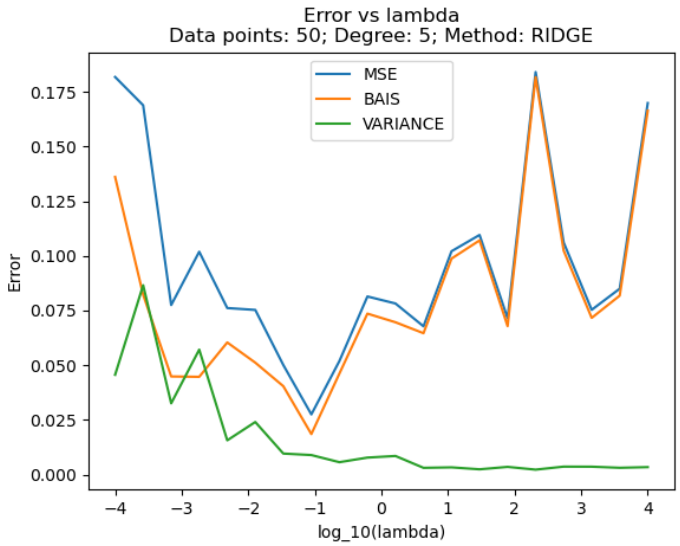

Figure 6: MSE, bias and variance depends on the lambda parameter in ridge regression

Since we are drawing a new training set for each λ, the figure is not very smooth, yet we clearly see the same tendencies as in the cross-validation analysis. The variance decreases, and the bias increases with λ, which yields a minimum in the MSE where λ is chosen optimally.

Exercise 5

Lasso regression on the Franke function with resampling

Performing lasso regression

We decided to use Sci-kitlearn's functionalities to perform the lasso regression instead of computing our own.

Cross-Validation

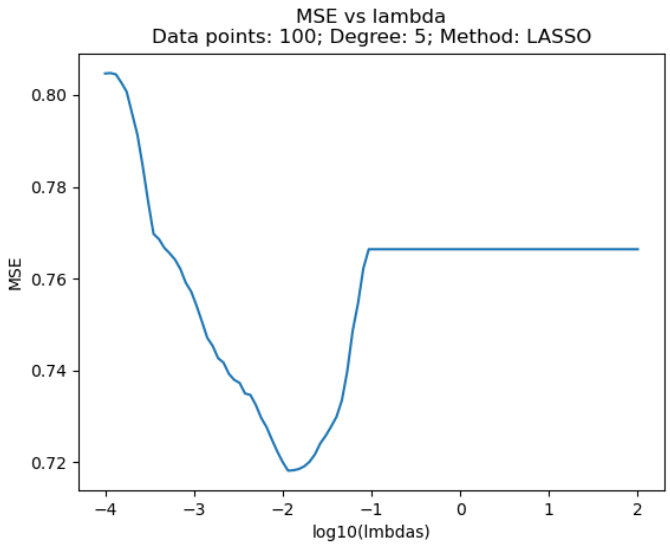

Figure 7: MSE depends on the lambda parameter in lasso regression

As for ridge regression, the MSE is dependent on the value of λ. We see that we again achieve an optimal λ with a value around 0.01 as seen in the plot. We also see that the error reaches a maximum value when λ increases.

Bias-variance trade-off with Bootstrap

Again, we study the bias, variance and total MSE as a function of λ:

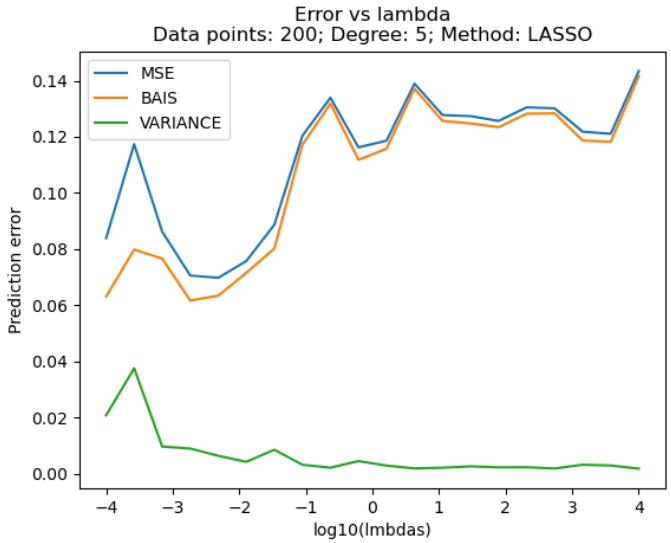

Figure 8: MSE, bias and variance depends on the lambda parameter in lasso regression

We confirm that λ has an optimal value since the bias and variance increase and decrease respectively with λ as seen in the plot.

Critical discussion of the three methods

We will discuss what regression method gives the best fit for our data.

It is clear the the ridge and lasso methods are more robust towards variance in the model, and can better handle a high model complexity without giving a high MSE. We compare the bias-variance trade-off for all three methods, and try to identify the optimal complexity which gives the minimum MSE.

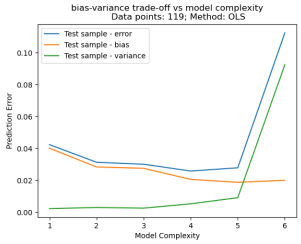

Figure 9: OLS

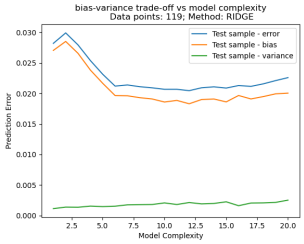

Figure 10: RIDGE

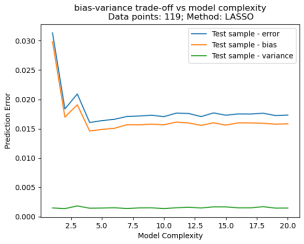

Figure 11: LASSO

First, we observe that the Ridge and Lasso regressions trade off a lower variance for higher bias. The Ridge vs OLS MSE are about the same as before. The minimum point for Ridge is lower than for OLS, indicating that Ridge regression better fits the data. Lasso gives a lower MSE than both OLS and Ridge, meaning it is the best fit for the data. Lasso regression trades off lower variance for higher bias, which appears to be optimal for the given data set. This conclusion may of course depend on several factors, e.g. the amount of data. If the amount of data had been increased, every model would include less variance, and this might have made OLS the best fit as this generally has lower bias.

Exercise 6

Analysis of real data

We now repeat exercise 1-5, but instead of generating the data ourselves, we import real terrain-data. We will not discuss and interpret the results in the same detail, because both results and interpretation are similar as before. We let the x- and y-coordinates correspond to the explanatory variables while the data for each coordinate is the response variable. After reading the data, we perform the regression in the same way as before. We do not always use all the data given because a large amount of data would be computationally expensive (we have restricted us to 250×300 datapoints)

Exercise 1 (repeat)

Ordinary Least Square (OLS) on the real data

Confidence intervals, MSE and R2 score

We compute the confidence intervals as before, the code can be ran in the python file "exercise 6". We also produce the MSE and R2 score. With a polynomial degree of 6, we get an R2 score of around 0.82, which means that 0.82 of the variance in the data is explained by our model.

Exercise 2 (repeat)

Bias-variance trade-off and resampling techniques

MSE vs the complexity of the model

In this plot, both the training- and testing-error are decreasing, and coincide even for models with high complexity. This is not the same result we got for the Franke function, but this can be explained by the number of data points used. If we use a 250×300 image as data, we will get $7.5 \cdot 10^5$ data points, and the model will have enough training data to avoid and overfit even at a degree of 40. If we computed an even higher polynomial, we would eventually get an increasing test error, but this would require a lot of computing power.

Perform a bias-variance analysis of the real data

We plot the bias, variance and total MSE for the model as a function of complexity, and as seen in the plot. We recognize that the bias is decreasing with the complexity, yet the variance is constant near zero. Again, since we have so much data, the variance in the model is very low. In other words, we are able to compute polynomials of a high degree without resulting in much variance in the model.

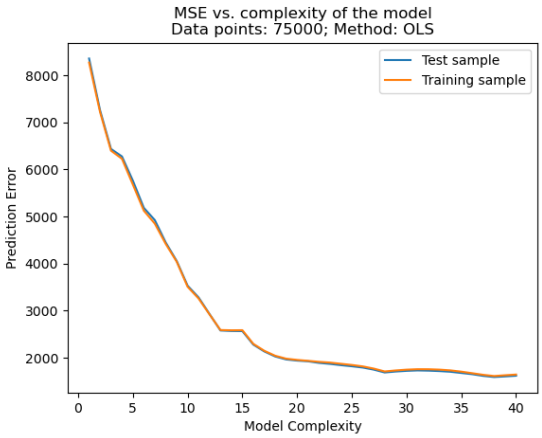

Figure 12: MSE of the test- and training data

Exercise 3 (repeat)

Cross-validation as resampling techniques

We reuse our code for cross-validation.

Comparison with bootstrap

We observe that for the real data, the bootstrap method gives a lower MSE than cross-validation, even when we increase number of folds to 10. This may be because we have less variance in the model.

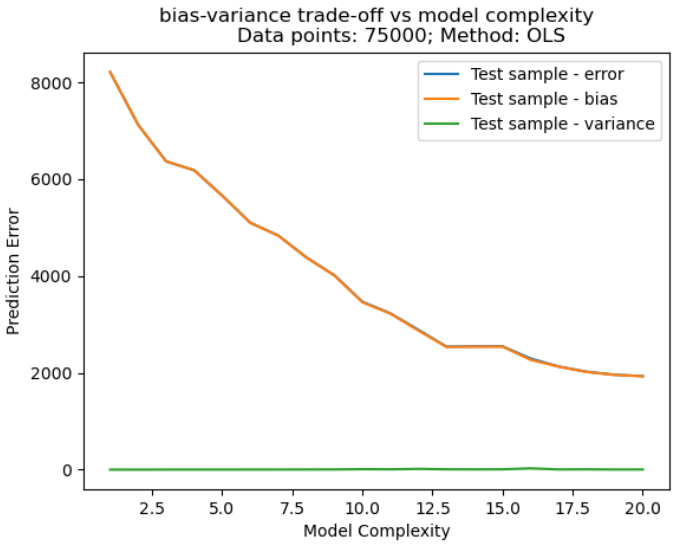

Figure 13: bias-variance figure

Exercise 4 (repeat)

Ridge regression on the real data with resampling

Comparison with OLS

We observe that the variance is low for both ridge and OLS regression. The bias for the ridge method is higher than for OLS, meaning ridge regression gives a higher MSE for polynomials of a degree up to 20. This may be because the penalizing term in ridge is for limiting variance. With a model with low variance, the term is of no use, and OLS is a better fit.

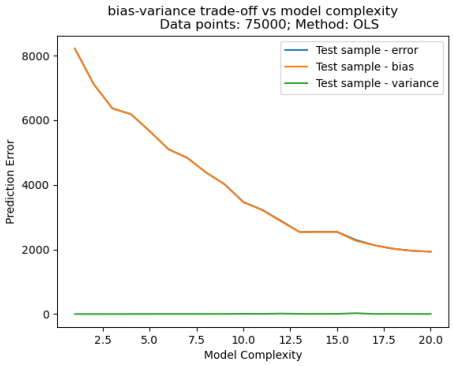

Figure 14: OLS

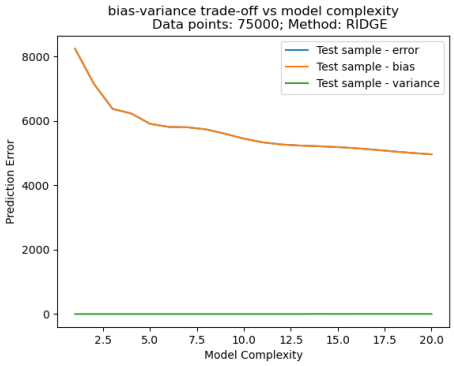

Figure 15: RIDGE (lambda = 0.1)

Cross-Validation

We observe that the cross-validation has a global minimum around $\lambda = 10$.

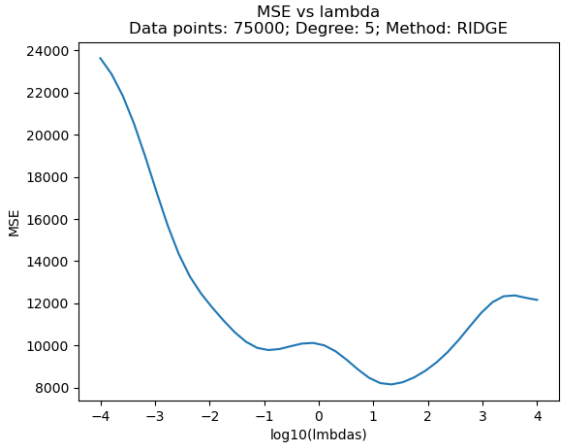

Figure 16: MSE depends on the lambda parameter in ridge regression

Bias-variance trade-off with Bootstrap

Here, we study the bias, variance and total MSE as a function of λ.

We observe that the MSE increases as a function of λ because of the increasing bias. Again, we have very little variance for any λ.

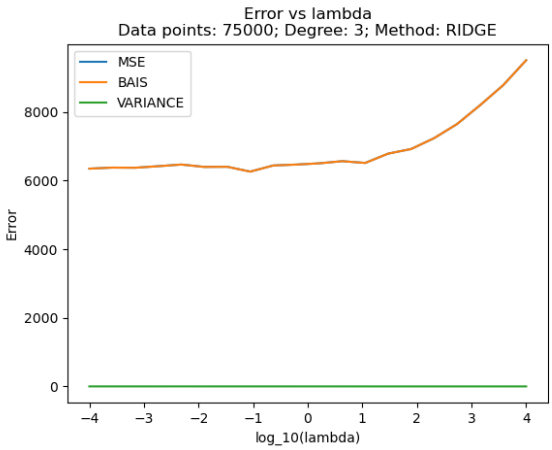

Figure 17: MSE, bias and variance depends on the lambda parameter in ridge regression

Exercise 5 (repeat)

Lasso regression on the real data with resampling

Cross-Validation

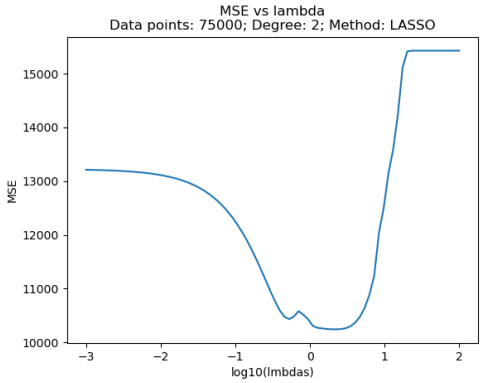

Figure 18: MSE depends on the lambda parameter in lasso regression

We observe that the cross-validation has a global minimum for ca. $\lambda = 1$.

Bias-variance trade-off with Bootstrap

Again, we study the bias, variance and total MSE as a function of λ:

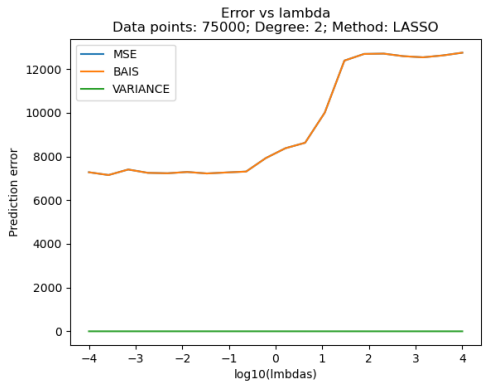

Figure 19: MSE, bias and variance depends on the lambda parameter in lasso regression

We observe that the MSE increases as a function of λ because of the increasing bias. Again, we have very little variance for any λ.

Critical discussion of the three methods

We will discuss what regression method gives the best fit for our data. Here, we plot the MSE as a function of complexity for all three regression methods.

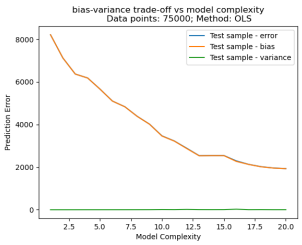

Figure 20: OLS

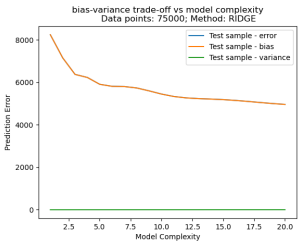

Figure 21: RIDGE

Figure 22: LASSO

We observe that if we use a large amount of data, there is close to no variance in the models. Since Ridge and Lasso regression have higher bias, they also have a higher MSE score, and it looks like OLS is the better fit. We could still increase the model complexity and get even lower scores, but this would require more computing power than we have available. Therefore we decided to look at fewer data points in hopes of finding some optimal complexity and minimum MSE for the models:

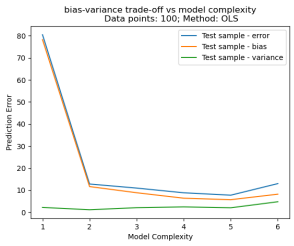

Figure 23: OLS

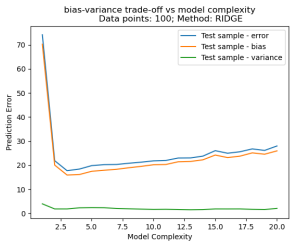

Figure 24: RIDGE

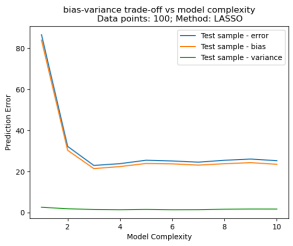

Figure 25: LASSO

Here we can more clearly see the minimum MSE for each model, and it appears that OLS is still the best fit. All models have low variance, and the Lasso and Ridge regression have more bias than OLS.

Image of our predicting models:

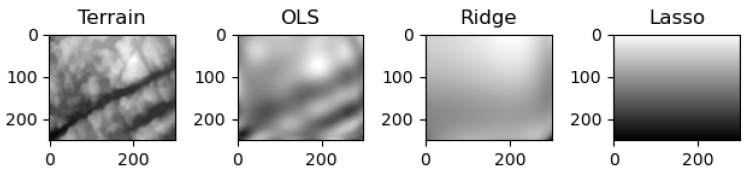

Figure 26: The different regression methods trying to tell the terrain