Regression Analysis and Resampling Methods

Using Ordinary Least Squares, Ridge Regression and Lasso Regression on the Franke Function and Real Terrain Data

Bendik Nyheim, Marcus Moen, & Mira Mors

Link to GitHub page: https://github.com/benyhh/FYS-STK4155

Data Analysis and Machine Learning FYS-STK4155

Department of Physics
University of Oslo, Norway
September 2021

Contents

1 Ordinary Least Square on the Franke function

In this task we are going to use linear regression on data from the Franke function. We have randomly generated x_1-values and x_2-values and found the corresponding Franke function values, y-values. The goal of this task is to create a ordinary least square regression (OLS) on this data such that it can be used to find the other Franke function values without inserting them in the function itself. We also have an additional data set which is equal to the original except that we have added some additional noise to the y-values. This noise is generated using the normal distribution, $\mu = 0$ and $\sigma = 0.2$.
For ordinary least squares, the equation for the approximated $\tilde{y}$-values is

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta} \tag{1}$$

where the design matrix $\boldsymbol{X}$ is given by a two dimensional polynomial of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$. $\boldsymbol{\beta}$ are the parameters to be determined. The corresponding cost function is defined as

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \boldsymbol{X}^T \left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right) \tag{2}$$

Minimizing the cost function, it can be shown that the optimal parameters $\hat{\beta}$ is

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

Ref. [1] The data is split into two separate sets, training and test. We use the training set to train our OLS model and the test set to test how good the model is. The test set data is only used for testing the method and it is important that it is not used in making the model. We evaluate the methods by looking at the Mean Squared Error (MSE) and the coefficient of determination (R^2). A definition of these values can be found in equation (4) and (5).

$$MSE(y, \tilde{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{4}$$

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{5}$$

where the mean values are defined as

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

The OLS has been made using using polynomials in x_1 and x_2 up to fifth order. The implementation is explained by comments in the code.
We first looked at the original system without noise. To make our data set we generated $N = 500$ random x_1-values and x_2-values with the uniform distribution from 0 to 1, and found the corresponding y-values using the definition of the Franke function given in the assignment text. This data was then split into training and testing sets, we choose to use 80% of the data as training data and the rest as test data. After fitting an OLS to the training data we got the following MSE and R^2 values shown in Table 1.

Table 1: MSE and R^2 for train and test data after fitting ordinary least squares to the Franke function with and without added noise in the data set and with and without scaled data with the standard scaler.

	MSE Train	MSE Test	R^2 Train	R^2 Test
Without Noise (not scaled)	0.00174	0.00184	0.982	0.977
Without Noise (scaled)	0.00179	0.0189	0.982	0.977
With Noise (not scaled)	0.0489	0.0485	0.640	0.653
With Noise (scaled)	0.360	0.359	0.640	0.653

Here we can see that our model fits the data without noise pretty well. However, we can also see that there is an increase of the MSE and decrease of R^2 from the training to the test data. This is not unexpected as the model is built around the training set, and hence have found the optimal fit for these points. A high R^2 score, close to 1, means that our model fits the data very well, while a low score means there is a poorer fit. The MSE does increase but not by much. By looking at the R^2 values we can see that there is a slightly bigger gap here but both scores are good. If we go over to the data with noise we see a different picture. Here we have a much higher MSE then we got without noise. Again this is not unexpected as the system is not well behaved we can expect worse results. The MSE values here almost identical. This is a little unexpected since we assume the test data will be worse, but since we are in a noisy/random system this can happen. The noteworthy thing to look at here is the R^2 scores. We can see a big jump down from the training to the test score. This is a red flag and gives us clear information that we are not fitting the population well, just the training set.
We also found confidence intervals for the β-values. This is done using the following formula:

$$CI_{0.95}^{\beta_i} = [\beta_i - 1.96 \cdot \frac{VAR(\beta_i)}{\sqrt{n}}, \beta_i + 1.96 \cdot \frac{VAR(\beta_i)}{\sqrt{n}}] \tag{6}$$

where 0.95 means that it is the 95% confidence interval, $VAR(\beta_i)$ is the variance to β_i, and n is the number of data points. By using this formula we obtained 105 confidence intervals, one for each β_i. Since there are so many confidence intervals we decided to show the results in a plot. In figure 1, we have plotted the logarithm of the absolute value to beta as the blue dots and corresponding logarithm to the length of the interval is the red line. The reason for taking the logarithm is that some of the beta values are huge compared to the length of the interval. Hence, plotting the original values would not have shown the length of each interval.

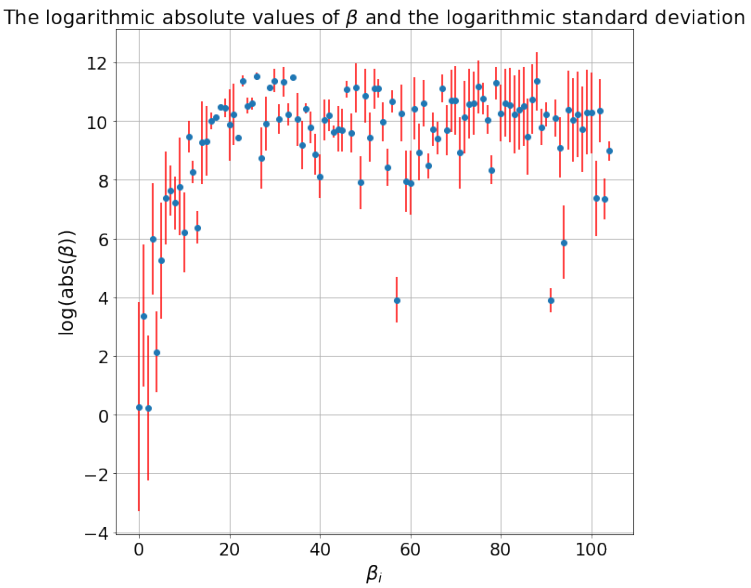


Figure 1: Confidence intervals for beta values after running OLS on the Franke function with added noise

In this project we have decided to not scale the data. In exercise 2-5 we will only use the noisy data. This is because almost all real-world problems, and especially problems where we want to use machine learning, will have noisy data. Data in physics will often come from measurements, and here we know that there are uncertainty. Hence, using noisy data in the rest of the task will make more sense. Two common ways to scale the data is using standard scaler and min-max scaler. The standard scaler subtracts the mean of the data and divides by the standard deviation. This makes the data have a mean $\mu = 0$ and standard deviation $\sigma = 1$. The min-max scaler works by subtracting the minimum value and dividing by the range, which is the difference between the maximum value and the minimum value. This makes all the data lie between 0 and 1. The main reason of scaling data, is to reduce the number of outliers, and with the data we have, both of these scaler will increase the number of outliers. This is because our data have $\sigma = 0.2$, so the standard scaler will just increase it. Our data is also already uniformly distributed between 0 and 1, so the min-max scaler will either increase the range of the data, or in the best case do nothing.

2 Bias-variance trade-off and resampling techniques

Now we look at how resampling techniques relate to the bias-variance trade-off. Resampling means taking a sample and then taking another sample from the sample. In this way, we can simulate repeated statistical experiments. This is done since the amount of data available to us is usually limited.
In this exercise we will use the bootstrap method as our resampling method. One bootstrapping cycle implies drawing data from the original design matrix of the training data while the test data set remains unchanged. The selection is done randomly and with replacement until the newly drawn sample has the size of the original training data. Just as in exercise 1, linear regression is run on the newly obtained design matrix. Then, the desired sample statistic, for example the mean squared error,

can be calculated.
By repeating this procedure, thus performing many bootstrap cycles, the average of the sample statistics can be calculated. In doing so, it is possible to estimate a precise uncertainty of the mean of the data's distribution.
Let us consider a data set $\mathcal{L}$ consisting of the data $\mathbf{X}_{\mathcal{L}} = \{(y_j, \boldsymbol{x}_j), j = 0 \dots n-1\}$. We assume that the true data is generated from a noisy model

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon} \tag{7}$$

where ϵ is normally distributed with mean zero and standard deviation σ^2. The predicted values are given by the design matrix of the training data and the parameters $\boldsymbol{\beta}$

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta} \tag{8}$$

where $\boldsymbol{\beta}$ are found by optimizing the mean squared error via the so-called cost function

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] \tag{9}$$

Note that the expected value $\mathbb{E}$ refers to the sample value.
Let the bias be defined as the difference between the expectation value of the prediction , $\mathbb{E}[\tilde{y}]$ and the true value $f(x)$

$$Bias = \mathbb{E}[\tilde{y}] - f(x) \tag{10}$$

As an example, a small bias implies that the the difference between the true value and the predicted value, on average, is small. The average corresponds to the expectation of the different training sets and not for the examples in the training set itself.
Similarly, the variance is defined as the mean squared deviation of $\tilde{y}(x)$ from its expected value $\mathbb{E}[\tilde{y}(x)]$

$$Var(\tilde{y}(x)) = \mathbb{E}[\tilde{y}^2] - (\mathbb{E}[\tilde{y}])^2 = \mathbb{E}\left[(\mathbb{E}[\tilde{y}] - \tilde{y})^2\right] \tag{11}$$

The variance is the spread of the predicted values for a given true value.

Let us return the expression in equation (9) which defines the mean squared error for a given data set. For a given point, the mean squared error can be written as (when substituting for $y = f(x) + \epsilon$)

$$\begin{aligned}
\mathbb{E}\left[(y - \tilde{y})^2\right] &= \mathbb{E}\left[(f(x) + \epsilon - \tilde{y})^2\right] \\
&= \mathbb{E}\left[(f(x) - \tilde{y})^2\right] + \mathbb{E}\left[\epsilon^2\right] + 2\mathbb{E}\left[(f(x) - \tilde{y})\epsilon\right] \\
&= \mathbb{E}\left[(f(x) - \tilde{y})^2\right] + \underbrace{\mathbb{E}\left[\epsilon^2\right]}_{\sigma_\epsilon^2} + 2\mathbb{E}\left[(f(x) - \tilde{y})\right]\underbrace{\mathbb{E}\left[\epsilon\right]}_{=0} \\
&= \mathbb{E}\left[(f(x) - \tilde{y})^2\right] + \sigma_\epsilon^2
\end{aligned}$$

square expansion, linear property of expectation

independence of random variables ϵ and $\tilde{y}$

Looking solely at $\mathbb{E}\left[(f(x)-\tilde{y})^2\right]$ we add and subtract $\mathbb{E}[\tilde{y}]$ and get

$$\begin{aligned}
\mathbb{E}\left[(f(x)-\tilde{y})^2\right] &= \mathbb{E}\left[((f(x)-\mathbb{E}[\tilde{y}])-(\tilde{y}-\mathbb{E}[\tilde{y}]))^2\right] \\
&= \mathbb{E}\left[(\mathbb{E}[\tilde{y}]-f(x))^2\right]+\mathbb{E}\left[(\tilde{y}-\mathbb{E}[\tilde{y}])^2\right] \\
&\quad -2\mathbb{E}\left[(f(x)-\mathbb{E}[\tilde{y}])(\tilde{y}-\mathbb{E}[\tilde{y}])\right] \\
&= \underbrace{(\mathbb{E}[\tilde{y}]-f(x))^2}_{bias(\tilde{y})^2}+\underbrace{\mathbb{E}\left[(\tilde{y}-\mathbb{E}[\tilde{y}])^2\right]}_{var(\tilde{y})} \\
&\quad -2(f(x)-\mathbb{E}[\tilde{y}])\mathbb{E}\left[(\tilde{y}-\mathbb{E}[\tilde{y}])\right] \\
&= bias(\tilde{y})^2+var(\tilde{y}) \\
&\quad -2(f(x)-\mathbb{E}[\tilde{y}])\underbrace{(\mathbb{E}[\tilde{y}]-\mathbb{E}[\tilde{y}])}_{=0} \\
&= bias(\tilde{y})^2+var(\tilde{y})
\end{aligned}$$

expanding terms inside the square

expectation of bias which is constant has no effect

linearity of expectation

Substituting this expression back we finally get

$$\mathbb{E}\left[(y-\tilde{y})^2\right] = bias(\tilde{y})^2+var(\tilde{y})+\sigma_\epsilon^2 \tag{12}$$

Equation (12) only holds for one given data point x. However, we normally have test a data set, such as $\mathcal{L}$ consisting of the data $\mathbf{X}_\mathcal{L}=\{(y_j,\boldsymbol{x}_j), j=0\ldots n-1\}$, and therefore we have to add an additional expectation in order to consider the distribution of test points x. This gives (using the definitions found above)

$$\mathbb{E}_n[\mathbb{E}\left[(y-\tilde{y})^2\right]] = \mathbb{E}_n[bias(\tilde{y})^2]+\mathbb{E}_x[var(\tilde{y})]+\mathbb{E}_n[\sigma_\epsilon^2] \tag{13}$$

$$\mathbb{E}_n[\mathbb{E}\left[(y-\tilde{y})^2\right]] = \mathbb{E}_n[bias(\tilde{y})^2]+var(\tilde{y})+\sigma_\epsilon^2 \tag{14}$$

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \underbrace{\frac{1}{n}\sum_i\left[(\mathbb{E}[\tilde{\boldsymbol{y}}]-f(x_i))^2\right]}_{\text{average bias squared}}+\underbrace{\frac{1}{n}\sum_i\left[(\tilde{y}_i-\mathbb{E}[\tilde{\boldsymbol{y}}])^2\right]}_{\text{variance}}+\underbrace{\sigma_\epsilon^2}_{\text{irreducible error}} \tag{15}$$

where $\mathbb{E}_n$ denotes the expectation over the distribution of different test points x. The above formula connects the test mean squared error to bias, variance and irreducible error. Ref. [4]

The bias depends on the model chosen. Logically, the worse the model, the greater the error, i.e., the bias. Oversimplification is usually the cause of high bias and is referred to as "underfitting." A more complex model, on the other hand, minimizes the bias. However, if the model is too sensitive to small variations in the training data, the variance increases. This is called "overfitting" because the model is not sufficiently generalized. The irreducible error, as the name implies, is irreducible and should ideally have no effect on the underlying model. It follows that the mean MSE for the training data cannot be

smaller than the irreducible error. The aim is therefore to find a model that keeps both the bias and the variance small.
Similar to exercise one, the following results are based on linear regression on data from the Franke function. The implenetation is explained by comments in the code. Again, the x_1- and x_2-values are uniformly distirbuted from 0 to 1 and we added a noise with $\mu = 0$ and standard deviation of $\sigma = 0.2$.

Figure 2 shows the mean squared error, given by equation (14), after 100 bootstrap cycles for the testing and training data. The number of data points are $N = 500$ whereof 80% are used for training. As described above, for low polynomial degrees, the difference between the true value and the predicted vales on average is high. In other words, underfitting causes the MSE to be high for both the train and test data. The higher the complexity, the better the model fits the training input data to the true training values. The training MSE decreases accordingly. However, the higher the complexity, the more sensitive the model becomes to fluctuations in the training data. In other words, this makes the model more and more specific to the training data. If the test data are now applied to the model, the variance increases and so does the MSE of the training data. At complexity four (which equals a polynomial degree of fifteen), the training MSE has its minimum. Here there is the best trade-off between bias and variance.

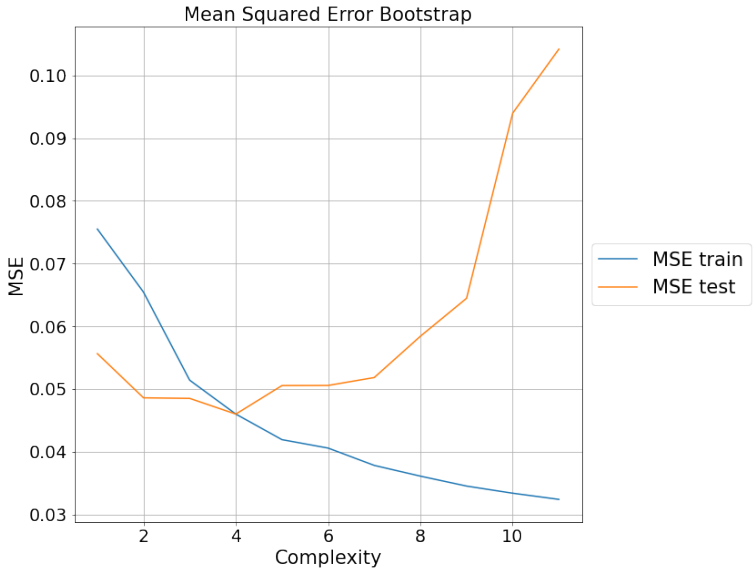


Figure 2: Mean MSE for train and test data after 100 bootstrap cycles as a function of the complexity of the model after running OLS on the Franke function

Figure 3 shows how the number of data points, thus complexity four as it gave the best result in figure 2. If the training data set is small, only a few points must be fitted and hence the training MSE is small. However, the data set is too small to generate a generalized model and therefore the test MSE is high. In the figure, this corresponds to where $N \approx 250$. The larger the data set, the more data there is for finding the best β parameters. These parameters are thus more universal and the MSE for train and test begin to align and eventually fluctuate between $MSE = 0.04$ and $MSE = 0.045$. This value

corresponds to the MSE in figure 2 at complexity four. Fluctuations are always present because the data itself contains random noise.

Figure 3: Mean MSE for train and test data after 100 boostrap cycles as a function of the size of the data set after running OLS on the Franke function

3 Cross-validation as resampling techniques, adding more complexity

For this exercise we are implementing the k-fold cross validation algorithm. The algorithm splits the data set into k equally sized folds. One of the folds is used as the test data while the other $k-1$ folds are used for training. After a model is fit on the training set and evaluated on the test set, we rotate the folds such that a new fold is used for testing. This is repeated k times, so that all the folds are used for testing once. We then take the mean of all the evaluations, in this case the MSE and R^2, and we compare these values for different numbers of folds. Ref. [3]

We used $k \in [5, 10]$ and evaluated the OLS method for complexities up to 13th degree where the data was set up as in the previous exercise. The results are presented in figure 4. It is clear that a polynomial of degree 5 yields the best results. As for which number number of folds k is optimal, there is little difference for a 4th order polynomial, although $k = 9$ seems to give the best results overall. An important consideration in our implementation of k-fold is the way we split the data into folds. We split the first $k-1$ folds in equal size, and the last fold will be either equal in size or bigger. A better way to do this would be splitting the folds into as equal sizes as possible. Though we didn't implement it in the best possible way, it is likely that it makes a little difference. At most, the last fold has $k-1$ more data points than the other folds. So with 500 data points and $k = 10$, all the folds will have the same size. With $k = 9$, the first $k-1$ folds will have 55 data points, while the last will have 60. This is insignificant.

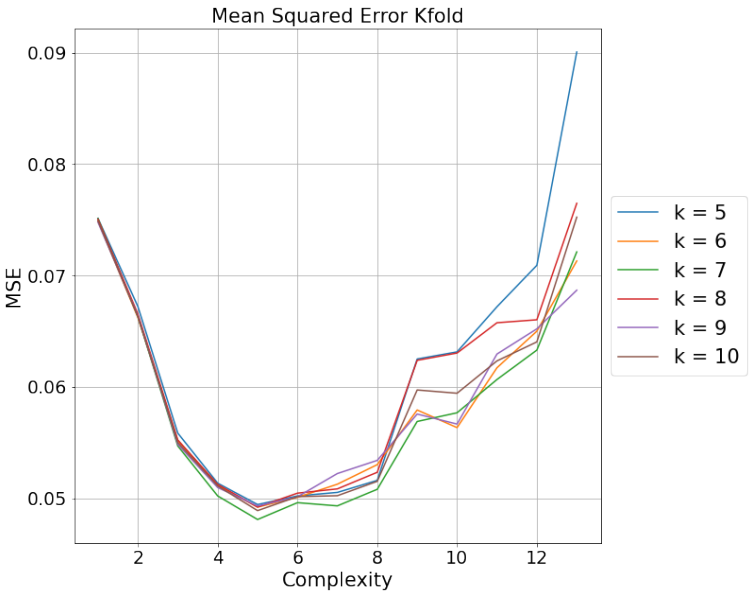


Figure 4: The mean squared error for different complexities of the model and number of folds k after running OLS on the Franke function.

As we can see the MSE we got using k-fold is approximately equal to the MSE we got using bootstrap. We can see that they have the same shape as well. This make sense since they are both methods for evaluating the true MSE for the population using the given methods.

4 Ridge Regression on the Franke function with resampling

In Ridge regression, we add a new parameter λ to the cost function, that is

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{n}||\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_2^2 \tag{16}$$

It can be shown that the optimal β is now

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{17}$$

Compared to OLS there is little difference when calculating the optimal β. There is just one added term inside the matrix inversion, which also fixes the singularity problem. The parameter λ has the effect of decreasing variance in the β values by pushing them towards 0. This leads to both low variance and low bias. Ridge regression is good for decreasing the complexity of the model, but it does however not reduce the number of features, as the coefficients can't be reduced to zero. Ref. [2]

Figure 5 shows the MSE for different model complexities and values of λ. Again, we see that a 4th degree complexity is optimal, and the optimal lambda is zero, which leads to OLS. For other complexities however, it seems like Ridge regression is an improvement from OLS.

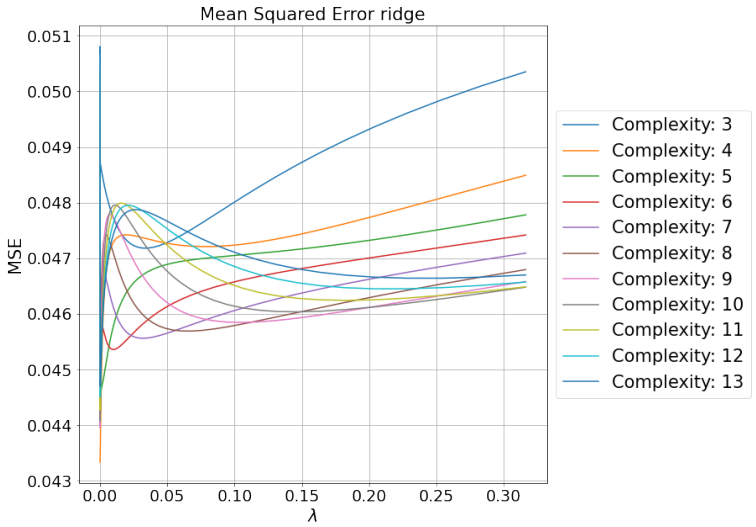


Figure 5: The mean squared error for different complexities of the model and corresponding optimal λ values after running Ridge regression on the Franke function.

Similiary to figure 2 figure 6 shows the the mean squared error after 100 bootstrap cycles for the testing and training data. For each complexity the optimal λ is found. The reasoning of the bias-variance trade-off corresponds to the one in exercise 2. For low complexities, the discrepancy between the predicted values and the true values on average is high. Consequently, the MSE is high for both the train and test data. However, the MSE values are lower for Ridge regression than for OLS. Higher complexities allow better fitting of the training data and hence the training MSE decreases. Overfitting causes the variance to increase and thus the raining MSE rises. At complexity four, there is the best bias-variance tradeoff and the MSE value of ~ 0.043 corresponds to the one in figure 2.

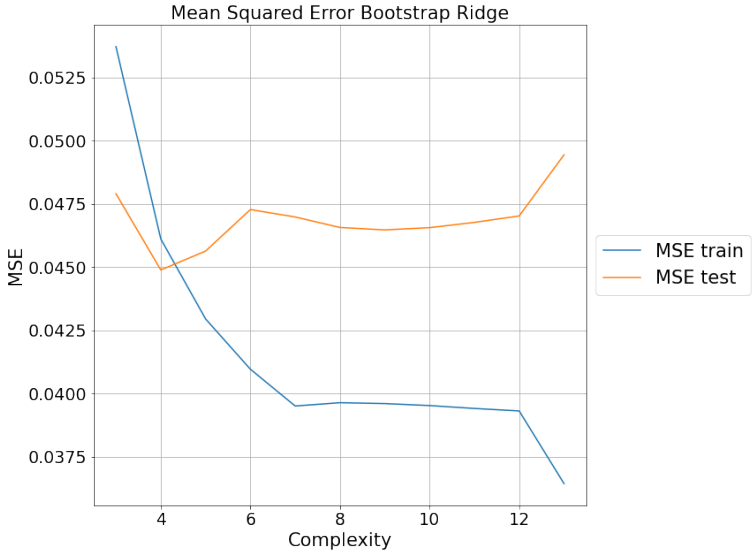


Figure 6: Mean MSE for train and test data after 100 bootstrap cycles as a function of the complexity of the model after running Ridge on the Franke function

5 Lasso Regression on the Franke function with resampling

Lasso regression stands for Least Absolute Shrinkage and Selection Operator. The cost function to be minimized in Lasso regression is

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \frac{1}{n}||\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_1 \tag{18}$$

It is similar Ridge, but the main difference is that the coefficients in this case can be reduced all the way to zero. Thus, the number of features in the model can be reduced as well. The limitation of Lasso is that it might remove important predictors. If the number of features p is greater than the number of samples n , Lasso will remove features such that it has at most $p = n$. This method might also remove important variables. If two or more variables are highly collinear, it can completely remove some of them, which might be bad for interpretation of the data. Ref [2]

Figure 7 shows the MSE for different model complexities and values of λ. Lasso actually chose a model with complexity 8 and $\lambda \approx 0.000144$ as the optimal model. The MSE is however increased, which means the model is not better than OLS. The values are given in table 2. If we used a function (such as grid search from scikit-learn) for finding the optimal λ, we would expect to find a lambda that is either zero, or one that yields a lower MSE than OLS.

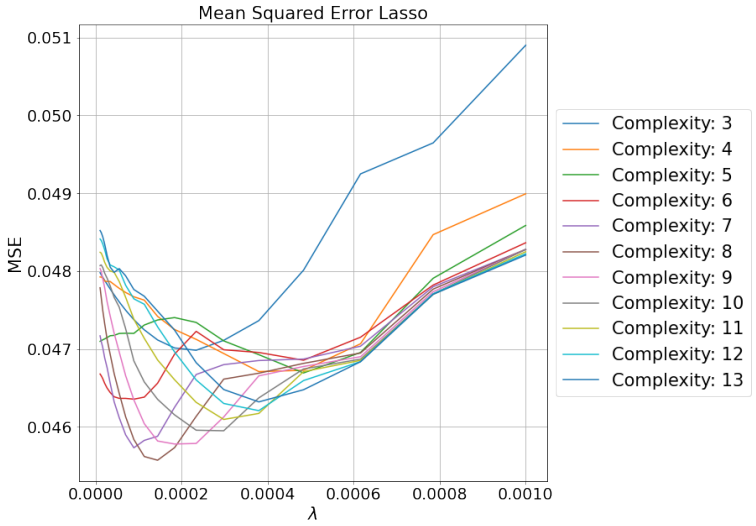


Figure 7: The mean squared error for different complexities and λ values using Lasso regression.

Figure 8 displays (just as figures 2 and 6) the mean MSE for train and test data after 100 bootstrap cycles using Lasso regression. The method has been implemented using the scikit learn package. The training MSE start out high due to the already mentioned underfitting. As expected the training MSE decreases with increasing complexity. For complexities higher than six, the training MSE starts to rise again. However, we would have expected the trainig MSE to decrease as a function of complexity. Additionally, the testing MSE is always lower than the training MSE. This is not supposed to happen for higher complexities. Overfitting should result in the testing MSE being higher than the training MSE. It is difficult to find the real reason why this happens as we did not implement the Lasso function ourselves.

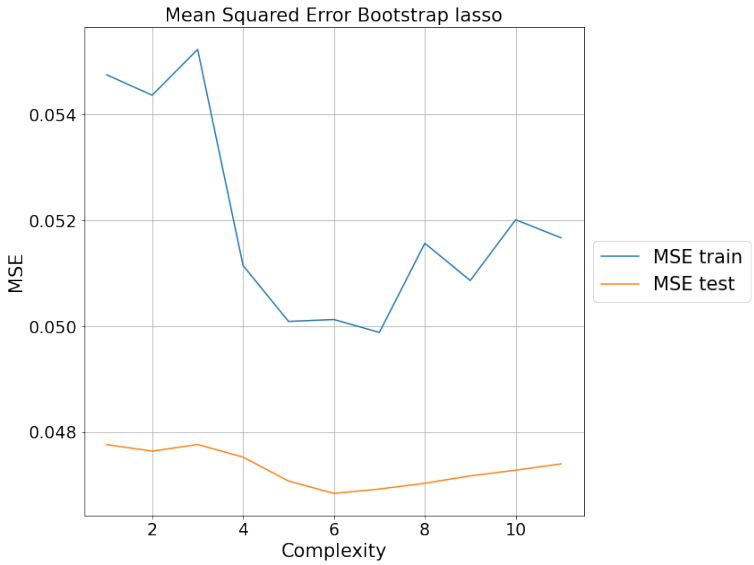


Figure 8: Mean MSE for train and test data after 1000 bootstrap cycles as a function of the complexity of the model after running Ridge on the Franke function

We have now made models using OLS, Ridge, and Lasso. After doing an analysis for all the models we found the best model for each method. For OLS it was with complexity 4, Ridge also had complexity 4 as its best with $\lambda = 0$, while Lasso gave us the best result using complexity 8 with a very small lambda. The results are in summarized in table 2:

Table 2: Mean squared error for test data after running OLS, Ridge and Lasso on the Franke function and using the bootstrap method

	OLS	Ridge	Lasso
MSE	0.04356	0.04334	0.04557

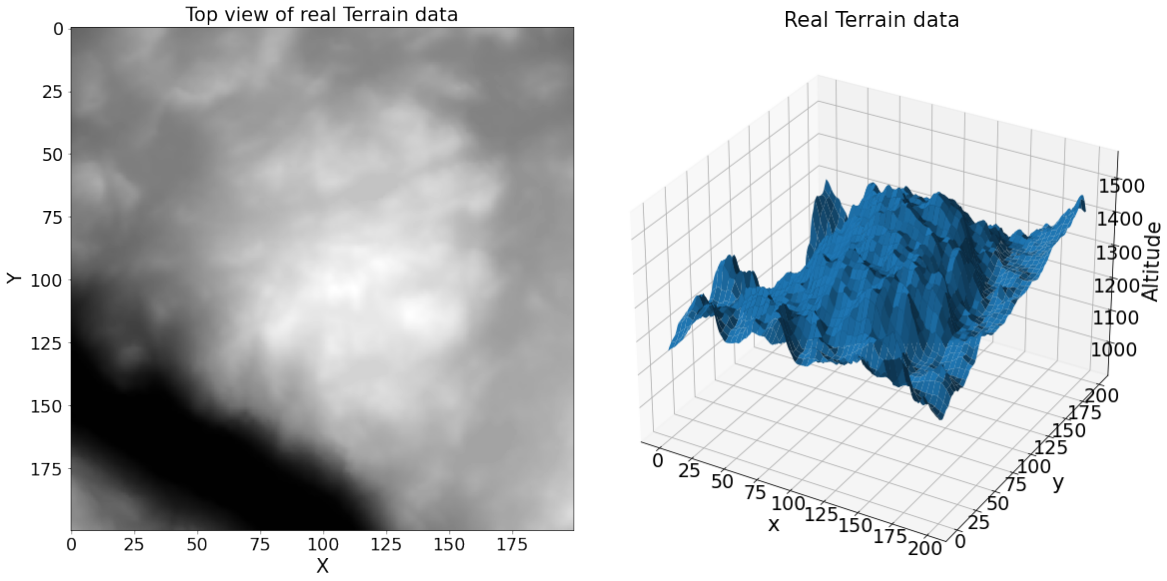
This table shows us that the best model to use on this data is the OLS method with complexity 4. The reason for why we get different results for OLS and Ridge, even though they are the same since $\lambda = 0$, is that they are evaluated using bootstrap and this can give us some randomness.

6 Analysis of real data

In this task we analyse real-world terrain data. Here we have given a set of coordinates, x- and y-values, and the goal is to find the altitude at the given coordinates. We are using all the methods we have discussed in previous exercises to find the best model for this data. We have again chosen not to scale the data, even though the arguments for not scaling in the previous exercises does not apply for this data set. The reason we choose to not scale in this exercise as well, is because there are not many outliers in the terrain data. In addition, we want to include abrupt changes in the terrain, and not smooth out the contours. However, we might need much higher complexities in our models in order to capture the complexities of the terrain. This is a problem for our computers, as very high complexities

are very computationally heavy to calculate.

The data we have chosen to use is the "Terrain over Norway 1" which is in the project description. If we were to use the whole picture we would need an extremely high complexity to fit the data correctly. Hence, we have chosen to cut down the image and only look at the top left corner. The picture we are studying is this:



(a) Top View

(b) 3D Terrain

Figure 9: Plots of real terrain data

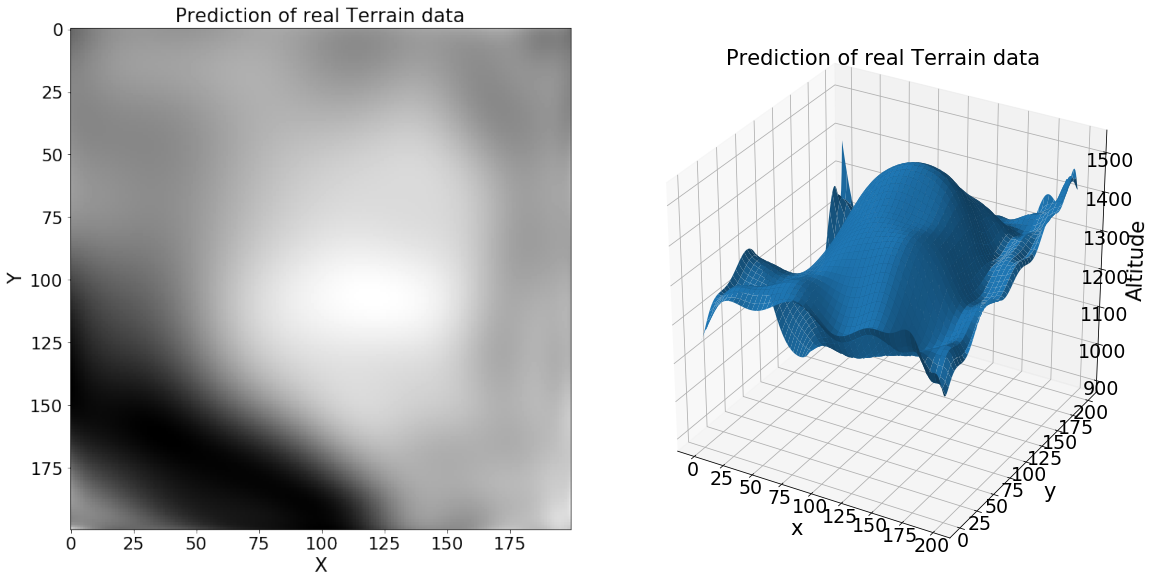
Let's first take a look at how our OLS model performs. First we try to find the most optimal complexity degree. We ran the same analysis as in exercise 2, but here we looked at complexities starting from 5 and up to 35. We can go even further, but because of the time issues this will have we decided to stop at 35. 35 is enough for us to get a pretty good result in this case. In table 3 you can see that the MSE_{OLS} to the best complexity tested, which was 35, is 390.80.

Now we can move over to the Ridge model and see how it performs on the same data as OLS. We used the same lambda values as in exercise 4, and the same complexities as for OLS. The best result was obtained with a complexity of 35 and a lambda value roughly equal to $1.13 \cdot 10^{-10}$. The corresponding MSE value can be found in table 3 and was slightly better then the OLS method, namely $MSE_{Ridge} = 371.28$.

Lastly we fitted a Lasso model to the same data. This gave us much worse results and is clearly not as good as Ridge and OLS for this problem. We are a little unsure about why this happens.

Table 3: Mean squared error for test data after running OLS, Ridge and Lasso on real terrain data. The complexity is 35 for all models. Ridge has $\lambda = 1.1288 \cdot 10^{-10}$ and Lasso has $\lambda = 3.3598 \cdot 10^{-4}$

	OLS	Ridge	Lasso
MSE	390.80	371.28	1308.63



(a) Top View

(b) 3D Terrain

Figure 10: Prediction of terrain using Ridge regression with $\lambda = 1.1288 \cdot 10^{-10}$ and complexity 35

In figure 10 we can see the predicted values using the Ridge method. Given that we only used complexity 35 the results are quite good. We can clearly see the similarities between the predicted results and the true values. The model does a good job with finding the curves and knowing the terrain.

So why do we want to do such an analysis? Especially exercise 6 makes it clear why regression can be useful. Let's say that we only have an unclear picture (due to limited data) of an unknown terrain and we want to find out the whole pattern. We can run a regression analysis like we have done here to find an estimate of the unknown terrain based on the limited data we already have acquired. From the results in figure 10 we can see that using regression can give us a pretty good approximation about how the terrain looks like.

References

[1] Morten Hjorth-Jensen. *Week 35: From Ordinary Linear Regression to Ridge and Lasso Regression*. Sept. 2021. URL: https://compphysics.github.io/MachineLearning/doc/pub/week35/html/week35.html. (accessed: 06.10.2021).

[2] Morten Hjorth-Jensen. *Week 36: Statistical interpretation of Linear Regression and Resampling techniques*. Sept. 2021. URL: https://compphysics.github.io/MachineLearning/doc/pub/week36/html/week36.html. (accessed: 06.10.2021).

[3] Morten Hjorth-Jensen. *Week 37: Summary of Ridge and Lasso Regression and Resampling Methods*. Sept. 2021. URL: https://compphysics.github.io/MachineLearning/doc/pub/week37/html/week37.html. (accessed: 06.10.2021).

[4] Giorgos Papachristoudis. *The Bias-Variance Tradeoff*. Nov. 2019. URL: https://towardsdatascience.com/the-bias-variance-tradeoff-8818f41e39e9. (accessed: 06.10.2021).