Regression models and resampling methods

Kamilla Ida Julie Sulebakk, Marcus Berget and Sebastian Amundsen

September 2021

Exercise 1: Ordinary Least Square (OLS) on the Franke function

We want to explore regression models and resampling methods using random generated data for directions x and y in the interval $x, y \in [0, 1]$. These values are to be used in the second order Franke function:

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

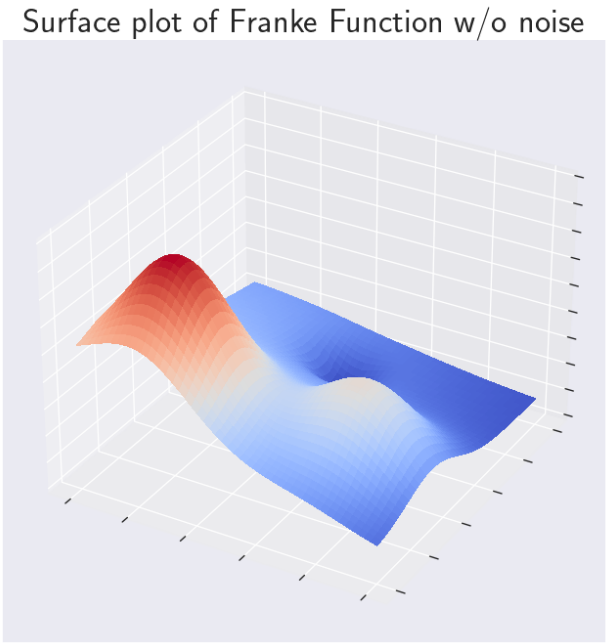

Figure 1: The surface plot of the Franke function without any noise.

This function takes both x and y as inputs, which means that we get a 3 dimensional system. We begin by plotting the terrain from the function, to get a clear picture of what we are trying to model. We see a "landscape" with a couple of "hills" and a "sink" (Figure 1). It is important to note that this is the Franke function without any noise.

We implement the most basic regression model as a starting point. This is the ordinary least squares method. We will try a polynomial fit with x and y dependence on the form $[x, y, x^2, y^2, xy, \dots]$. The fit is found using the method of least squares using polynomials x and y up to the eighth order. The mean square error (MSE) can be used to evaluate how well

the model fit our data, which is given by:

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

Where $\tilde{y}_i$ is the predicted value and y_i is the corresponding true value. Having a MSE of zero, would mean that the estimator $\hat{y}$ predicts observations of the parameter $\hat{\tilde{y}}$ with perfect accuracy. This is obviously ideal, but is however, not typically possible.

The predicted value $\tilde{y}_i$ can be rewritten as $\tilde{y}_i = x_i^T\beta$, which we can use to find the residual sum of the squares:

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - x_i^T\beta)^2 \tag{2}$$

The ordinary least squares method (OLS) wants to minimize the sum of the residuals, which is given by the RSS value (see equation 2). We want to find the β which minimize this function. If we define a $N \times p$ matrix $\mathbf{X}$ with each row as an input vector and a N-vector $\mathbf{y}$ of the true values, we can represent our solution $\hat{\beta}$ as a 1-dimensional array of size p using matrix notation:

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

where $\mathbf{X}^T\mathbf{X}$ is non singular [1]. The size of $\hat{\beta}$ will depend on the degree of our polynomial fit. For example polynomial degree 5 we will give us a $\hat{\beta}$ length of p=21. The length N is given by the number of datapoints.

Just like MSE, the coefficient of determination R^2 is a measure for how precise a model is. To be more precise, it provides a measure of how well future samples are likely to be predicted by the model. The score normally ranges between 0 and 1, where 1 is the best possible score. In general, the higher the R-squared, the better the model fits the data. The coefficient of determination R^2 is given as follows:

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

Where the mean value of y_i is defined by $\bar{y}$:

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

Creating a perfect model for a set of data, can easily be done by using a high polynomial degree when fitting. However, creating a foolproof model that only fits one specific data set is not very beneficial. We want our model to predict correlated data, not slavishly follow the path of least square error. With this in mind we should split our data into training and testing sets in order to make the model compatible with multiple data sets. By splitting our data we can test how well our model predicts. In this exercise we have chosen to train on 80% of our data, and test on the remaining 20%. By training, in this context, we mean to find the β that minimize the cost function, using the training data. Having obtained $\hat{\beta}$ we test our model on both sets, expecting that it fits well for both the training and test data.

A much used approach before starting to train the data is to scale our data. The data may be very sensitive to extreme values, and scaling the data could render our inputs more suitable for the algorithms we want to use.

There are several methods for scaling data sets. In this project we have scaled our data by calculating the mean and standard deviation for each feature. For each observed value of the feature, we subtract the mean and divide by the standard deviation. The first feature (first column in our design matrix) was kept untouched, in order to avoid singularities when calculating β. This sort of scaling is called "Standardizing". When standardizing data one obtain a "standard normal" random variable with mean 0 and standard deviation 1 [3].

If any, there are very few data sets in the real world which have ideal properties without noise. When generating our own data set for the Franke function we therefore explore the addition of an added normal distributed $N(0,1)$ stochastic noise, in order to make the data set more realistic.

The OLS Algorithm

Define Franke Function with noice
Define the number N of x and y values
Create linspace for x and y values
Define meshgrid with x and y
Call Franke Function with x and y. This is our z.
for *degree* $\in \{1, 8\}$ **do**
    create design matrix X
    split data into training and testing set
    scale the data
    $\beta = (X_{\text{train}}^T X_{\text{train}})^{-1} X_{\text{train}}^T z_{\text{train}}$

$\tilde{z}_{\text{test}} = X_{\text{test}} \cdot \beta$
$\tilde{z}_{\text{train}} = X_{\text{train}} \cdot \beta$
calculate mean squared error, bias and variance
end for

Confidence interval of the regression coefficients, β

Having that the noise is normally distributed and unbiased, we can assume that each of the estimators also will have a normal distribution. Thus, we will never know if the estimators have the true value of the model we are trying to fit. However, we can calculate the confidence interval of the estimators so we know in what degree we can trust them. To do this we first have to calculate the variance of the estimate of the j-th regression coefficient, which is given by:

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

Where σ^2_{noise} is the error of the noise. Now that we know the variance, we can define the confidence interval of each of the estimators:

$$CI = \mu_\beta \pm \frac{z\sigma_\beta}{\sqrt{n}}, \tag{6}$$

Where μ_β is the mean value, σ_β is the standard deviation of each beta and z defines the level of confidence we want to choose for the confidence interval. For example, $z = 1.96$ corresponds to a confidence of 95%.

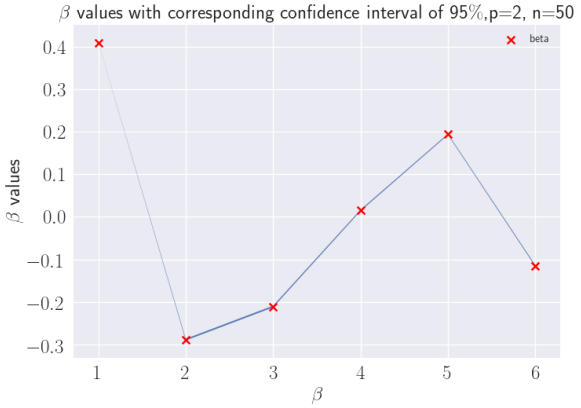

Figure 2: Plot showing the values of the β's as function of the number of features, with the corresponding confidence interval. As we can see, the number of features is 6 for a polynomial of degree 2. The thick blue line between the points indicate the confidence interval.

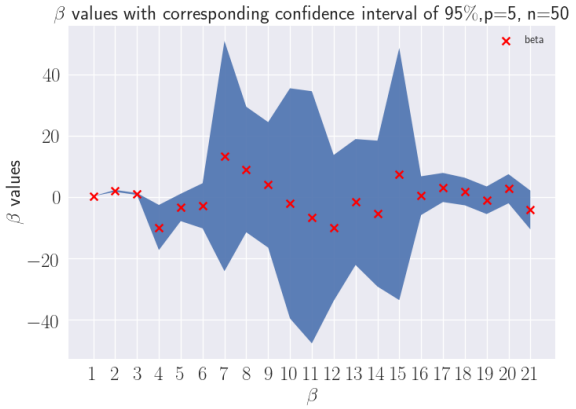

Figure 3: Plot showing the values of the β's as function of the number of features, with the corresponding confidence interval. As we can see, the number of features is 21 for a polynomial of degree 5. The thick blue line between the points indicate the confidence interval.

Comparing Figure 2 and Figure 3, we see a staggering difference in the confidence intervals. The confidence interval for the large polynomial of degree 5 is in general much larger than that of degree 2, which

we can barely even see. However, if we take a look at the confidence interval for only the first three data points of $p = 5$, it is comparably much smaller than the rest. We also notice that towards the last features, the confidence interval starts shrinking. We believe that the reason for the small confidence interval of the first three β's is that the first three features has less "wiggle room" to vary in, because they in large define the main characteristics of the function we are fitting. Then as we move towards the β's in the middle, change in one β can be compensated by a change in the neighbouring β's, while maintaining a good fit to the data set. As we increase the number of data points to $n = 200$, we can see that the the respective confidence intervals for the β's shrink (Figure 4). This makes sense. As we increase the data points associated to each β, the noise will statistically have less of an impact when estimating the β's.

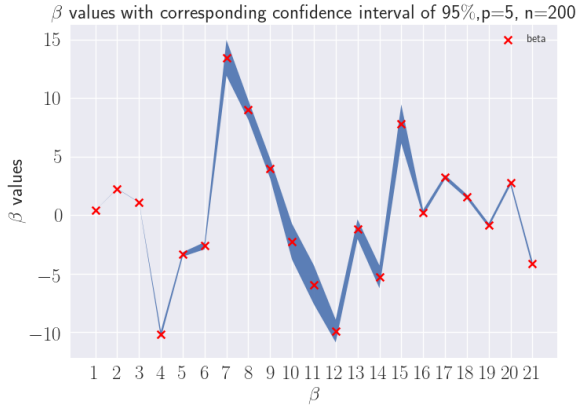

Figure 4: Plot showing the values of the β's as function of the number of features, with the corresponding confidence interval. Here we have changed the number of data points for the x and y arrays to $n = 200$. The thick blue line between the points indicate the confidence interval.

Mean Squared error (MSE) and R^2 score function

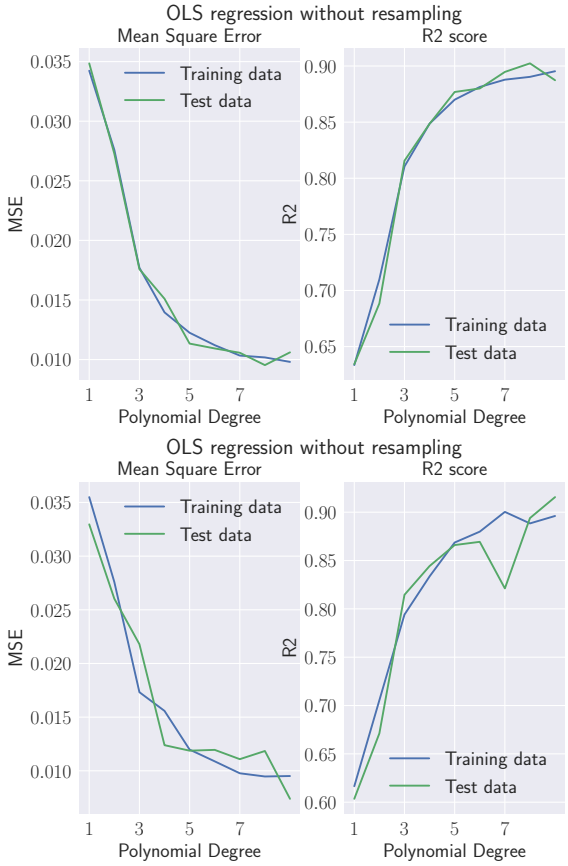

Figure 5: Plots of the mean squared error and R2 score for the test and training data calculated with $N = 50$ and $N = 20$, respectively.

When analyzing our data, we observed that scaling did not affect the results when performing ordinary least squares. This is in alignment with our expectations, as we were dealing with data with equal units, which means that they would contribute equally to the analysis. In other words, there was no need to scale our data for the OLS method. However, this conclusion war not necessarily true for the later methods we implemented.

We plotted the mean squared error and the R2 score as a function of polynomial degree. The results are provided in figure 5, for 50×50 and 20×20 data points, respectively. Firstly, wee see that the the

smoothness of the curves increases when the number of data points rises. This is not a surprise, as the number of data points naturally affects the precision of the model. We would expect the MSE value from the training set to be higher than the test data for all polynomial degrees. Surprisingly, this is not the case in our results. We can see that the "test curve" fluctuates around the training curve for the MSE values. For the R2 score we should ideally have that the test data always give a smaller number than the testing data. This is not the case in our case. Also here, we see that the "testing curve" fluctuates around the "training curve". These observations indicates that there could be something wrong with our implementation. We explore this in the **Further exploration** section.

We do however observe that the MSE value seems to converge towards zero, which we would expect (the test set should converge towards $\sigma^2_{error} = 0.01$). The R2 score seems to converge towards 0.9, while we would expect it to converge towards 1. We would perhaps have seen this limit of sequence if we had tested for higher polynomial degrees.

Exercise 2: Bias-variance trade-off and resampling techniques

We now want to implement the bootstrap resampling technique so that we can study the bias variance trade off.

Bias-Variance Tradeoff

Lets assume a data set generated from a noisy model: $\mathbf{y} = f(\mathbf{x}) + \epsilon$, where ϵ is normally distributed and have standard deviation σ^2. We have that an approximation for the function f is defined as $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$, where $\boldsymbol{\beta}$ are the optimal parameters and $\boldsymbol{X}$ is the design matrix. The $\boldsymbol{\beta}$ parameters are in found by optimizing the mean squared error using the cost function, which is given by:

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

Where $\mathbb{E}$ is the sample value. We can rewrite this equation as:

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

We have that $\mathbb{E}[f] = f$, $\mathbb{E}[\epsilon] = 0$, $Var(y_i) = \mathbb{E}([y_i - \mathbb{E}(y_i)]^2) = \mathbb{E}(y_i^2) - [\mathbb{E}(y_i)]^2$. We can now let the expectation value work on all the objects inside the bracket:

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

Where all the terms with $\mathbb{E}[\epsilon]$ disappears:

$$\begin{aligned}
&\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] \\
&+ \mathbb{E}[(-1)^2(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\epsilon^2] \\
&\boxed{= \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2}
\end{aligned} \tag{7}$$

The first term in the final expression is the bias, the second is the variance and the third is the variance of the noise. $\mathbb{E}[\tilde{y}]$ is the mean value of $\tilde{\mathbf{y}}$.

Bootstrapping

The first resampling technique we will look at is the bootstrapping method. This method is used to estimate various statistics of a dataset. In principal the bootstrap method works by picking out one value at a time with replacement, until you have created a new dataset of equal size as the original one. You then

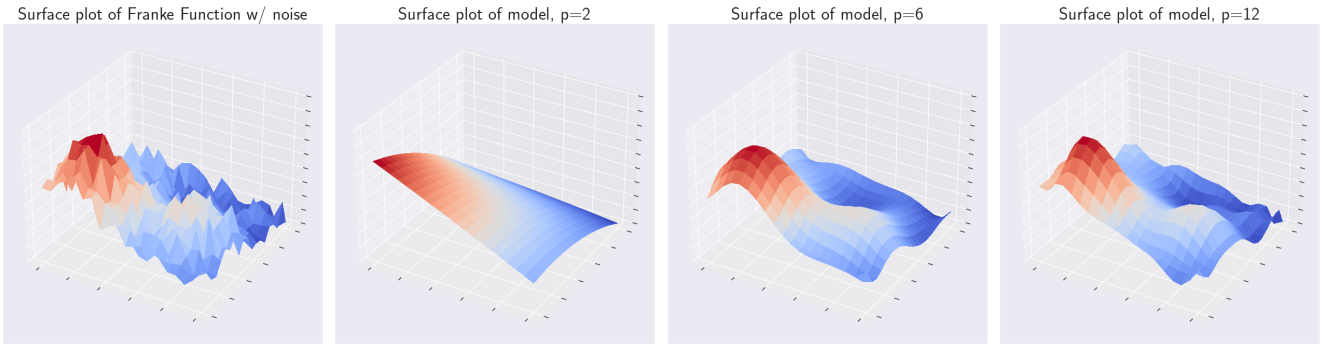

Figure 6: Comparison of the surface of the Franke function with noise, with the regression models of varying polynomial degree.

store that dataset and repeat the process a reasonable number of times. Finally, you can for each dataset take the estimate of whatever statistic you are interested in and make a sampling distribution over all the bootstrap samples (or do the estimations in each iteration to avoid unnecessary memory usage). The bootstrap method will hopefully reduce the weight of outliers in our dataset, given by the fact that there is always a higher probability of picking the values which are more abundant in the set. However, the method does not provide more information than what is already in the dataset. For this reason it doesn't always work well with small samples.

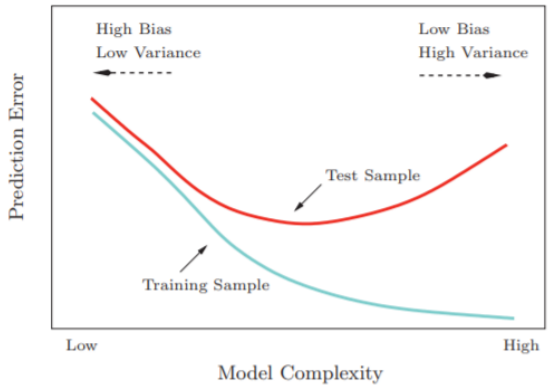

Figure 7: Plot of the bias-variance trade-off for ordinary least squared regression using the bootstrapping resampling technique to estimate the bias and variances [1].

Using the Bias Variance tradeoff to find the optimal model complexity

Figure 7 is a classical example of the bias-variance tradeoff. By increasing the model complexity you will lower the bias, but in turn increase the variance.

However, increasing the variance isn't necessarily a bad thing as long as your model matches the complexity of the data. The problem is that you seldom know the complexity of the data you are trying to fit. A reasonable strategy for finding the optimal complexity is to start low and then increase the complexity of the model, while monitoring the bias and variance. While doing this you will eventually reach a point where you start fitting according to the noise. At this point you have started to "overfit" your model.

In Figure 8 we have provided a plot of the bias-variance trade-off for OLS regression, with 20 and 50 bootstrapping iterations, respectively. We notice that the differences between performing 20 and 50 boot-

straps are not substantial, at least not for the polynomials up to eighth order. As the CPU time increases with the number of iterations, we therefore choose to use 20 bootstraps in all further analysis for Franke's function. Analysing the results obtained with 20 bootstraps iterations, we see that at low complexity the MSE and bias have a steep decrease until $p = 5$. Then the bias and MSE flattens out before the MSE start increasing again at about $p = 9$. Also at $p = 9$ the variance begins to increase even faster, while at the same time the bias stays more or less flat for the rest of the plot. The increase in the MSE is suspicious, and it might be a sign of overfitting. The observations make it seem that the optimal complexity is somewhere in the interval between $p = 5$ and $p = 9$ due to the low values of MSE and bias, while the variance still is low.

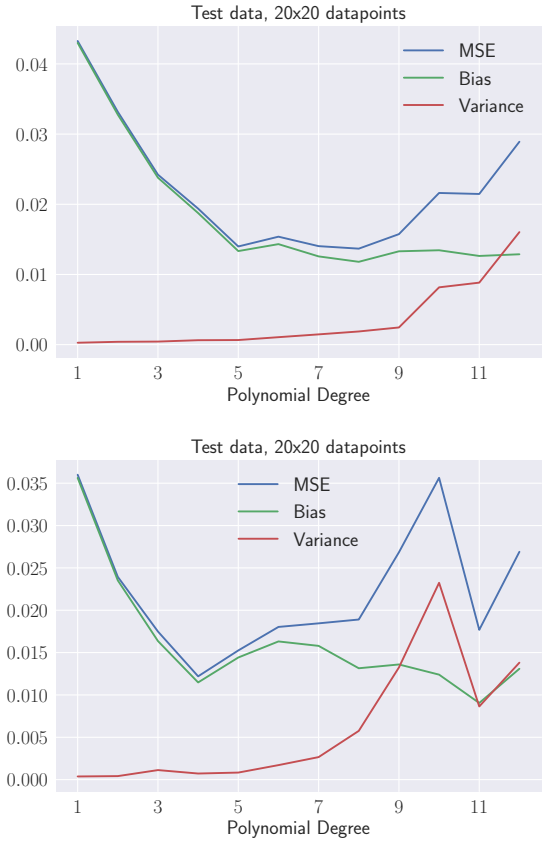

Figure 8: Plot of the bias-variance trade-off for ordinary least squared regression for N = 20 and 20 and 100 bootstrap iterations, respectively. This result can be utilized to estimate the bias and variances.

This is also seen in figure 6. We see that for $p = 2$ the complexity isn't high enough to capture all the characteristics of the original function. For $p = 6$ we see that the model matches most of the Franke function's main characteristics. For $p = 12$ we see that also here most of the characteristics are captured, but now it seems like the model has fitted according to some of the noise in the data. Thus, finding the optimal complexity by studying the Bias- Variance trade off seemed to work pretty well.

Effect of changing number of datapoints

In Figure 9 we again look at the bias variance trade-off, but now the number of data points has increased from $20x20$ to $50x50$, and we notice that for a polynomial degree of $p = 12$ there are no signs of overfitting. This makes sense. With a larger data set the noise will have less effect on each feature which makes it harder for the model to overfit.

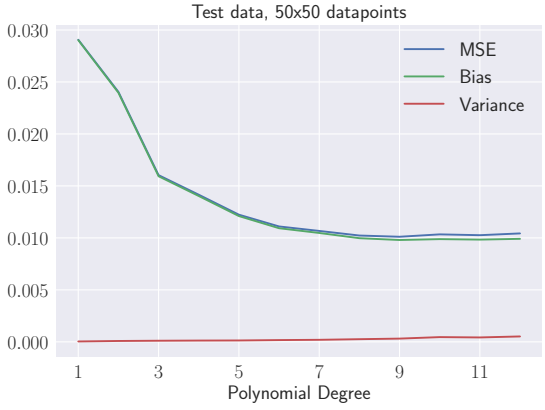

Figure 9: Plot of the bias-variance trade-off for ordinary least squared regression with 20 bootstraps iterations and N = 50. This result can be utilized to estimate the bias and variances.

Exercise 3: Cross-validation as resampling techniques, adding more complexity

Cross Validation K-Folding

Ideally, when having enough data, one would set aside a validation set and use it to assess the performance of the prediction model. However, since data are often scarce, this is usually not possible. As a solution to this problem, K-fold cross-validation uses part of the available data to fit the model, and a different part to test it. In this exercise, we will study k-fold as resampling technique. This method revolves around splitting our data into k number of groups, which are equal in size. We pick one group to test on for the first iteration through our bins. We then choose the next bin for our second iteration. This is repeated a total of k times. This is done so that we perform testing on each section of our data. The rest of the $k-1$ folds will be used to train our model. This process is visualized in Figure 10. It is often common to shuffle the data before one split it into folds. If the dataset is structured in an order dependent way, the learning process can become biased, if one abstain from shuffling the data. However, this is unnecessary in our case, since the metric we are utilizing does not dependent on information associated with the ordering of our data. In our specific case, shuffling the data would only lead to additional CPU time.

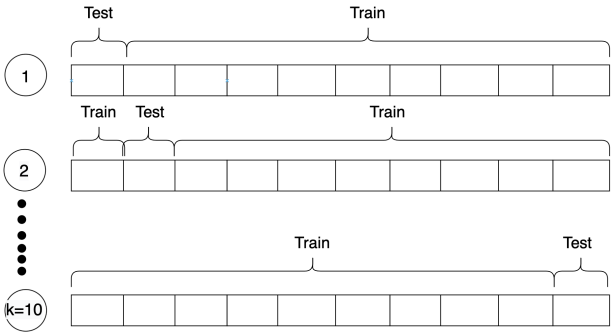

Figure 10: Illustration of the k-fold cross validation method for k=10.

Finding the optimal number of folders

We test our cross-validation code with 5-10 folds. We can then decide the most suitable value for k by studying our results. Figure 11 contains some rather interesting results. They all start with approximately the same MSE at $p = 1$, but then as the complexity increases the MSE has an extreme increase (thus the logarithmic scale). We can see that at $p = 7$ the MSE is already at a value of almost 10^2. This does not match the results of the Bias-Variance analysis in Exercise 2. At first we believed that the high MSE was due to overfitting, but that doesn't really make sense, due to the data being normalized with a noise of $\sigma^2_{noise} = 0.01$. Thus, as far as we understand, a model which is overfit shouldn't have MSE's that many magnitudes of order higher than 0.01. We're not sure of why k-folding produce such high MSE's, but it could be interesting to see the consequence of increasing the number of folds. It looks like as we increase the number of folds from 5 to 10 the mean MSE seems to go down a bit. Thus when choosing between $k = 5$, $k = 8$ and $k = 10$, we would say that the optimal number of folds is $k = 10$.

Further, when comparing our cross validation method with the one which is provided with Scikit-Learn, manufactured in figure 12, we get some interesting results. When setting $N = 50$, we can virtually observe that the MSE results obtained with our own algorithm is in alignment with the results obtained using Scikit-Learn. However, changing N to 20, we see how our code, after reached polynomial degree 8, produces a less steep MSE curve than the one from Scikit-Learn, which we find disturbing. To which extent this can tell us wether our model produces reasonable results or not is not easy to say, but we can not exclude the fact that the model may be affected by numerical errors in our algorithm.

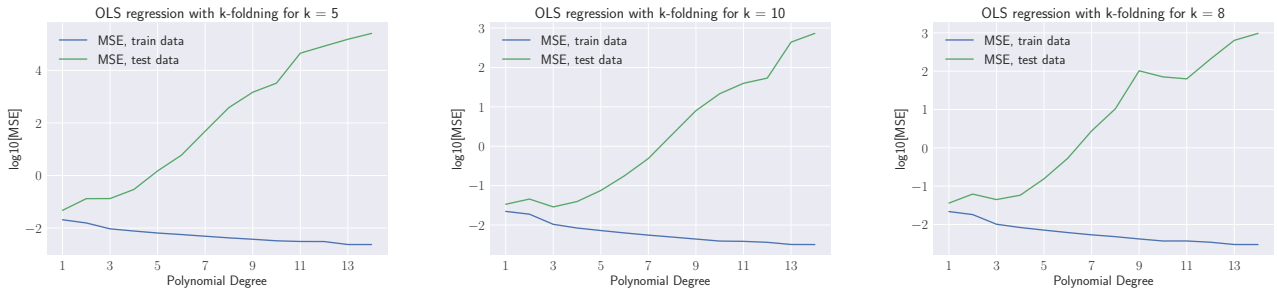

Figure 11: Plot of the mean squared error for ordinary least squared regression with the k-fold resampling method for $N = 50$, with $k = 5$, $k = 8$ and $k = 10$, respectively.

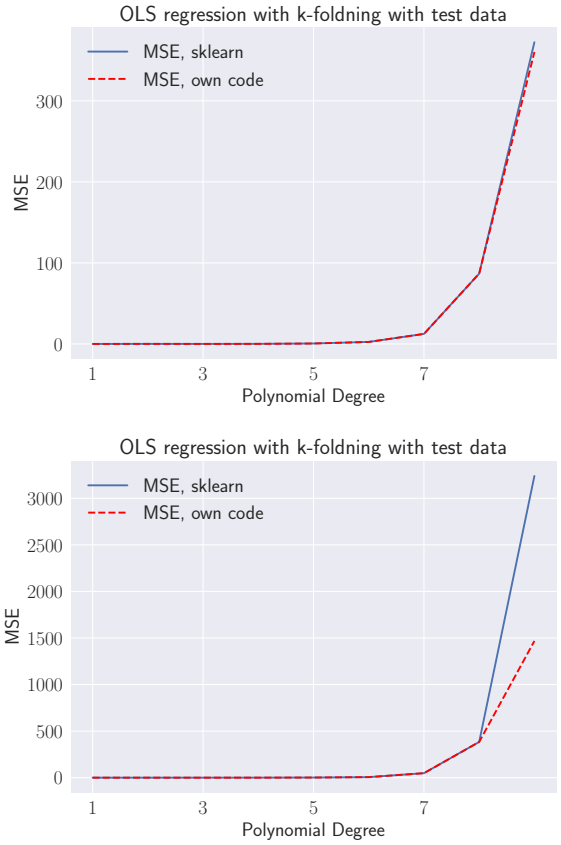

Figure 12: Plots of the mean squared error for OLS regression with k-folding for $k = 5$ and $N = 50$ and $N = 20$. In the figure the result obtained with with our own code is compared to the result produced with Scikit-Learn.

Exercise 4: Ridge Regression on the Franke function with resampling

We will now look at another regression model, namely the Ridge Regression method. This is a shrinkage method, which shrinks the regression coefficients β by imposing a penalty on their size [1]. Ideally, this should result in less overfit models. The OLS method have a tendency to create too large β parameters. We want to circumvent this in the Ridge Regression method. Instead of only minimizing the sum of the residuals, as we did in the OLS method. We now want to add a penalty term, which should optimize our model parameters β. In Ridge regression we want to minimize the sum:

$$\beta^{Ridge} = \text{argmin}\left\{ \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{8}$$

This is also known as the penalized residual sum of squares, which we can rewrite in matrix form as:

$$RSS_{Ridge}(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^T\beta$$

This is the sum we want to minimize in the Ridge Regression method, where $\lambda\beta^T\beta$ is the penalty term. The solution of this equation is given by:

$$\beta^{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{9}$$

Where $\mathbf{I}$ is an identity matrix with dimensions $p \times p$. We see that the Ridge Regression solution is a linear function of $\mathbf{y}$ [2]. The parameter λ is introduced to make our algorithm more flexible. In some cases it is beneficial to weigh the penalty term differently.

We could for example want our model to emphasize the minimization of the MSE term, which is done by making the λ parameter smaller. We want to study the dependence on λ by looking at the bias-variance trade-off as a function of various values for λ.

It is important that our data is standardized when we use the Ridge model. The ridge penalty prioritize minimizing the largest parameters β, due to the fact that the ridge penalty squares the model parameters. This means that large values "weigh more" then smaller values when we calculate RRS_{Ridge}. Since the Ridge method discourage large model parameters, the size of the values in the dataset will impact how our model preforms the fit. We want our data to be normalized so that they are on the same scale. The shrinkage of the β parameters is dependent on the sum of squares of all the coefficients. This means that two independent variables with different scales will contribute differently to the penalized terms, which could result in a bad fit.

The algorithm for Ridge regression is equal to the one for OLS, 0, except from the extra term in the expression for β, see equation 9.

Bootstrapping

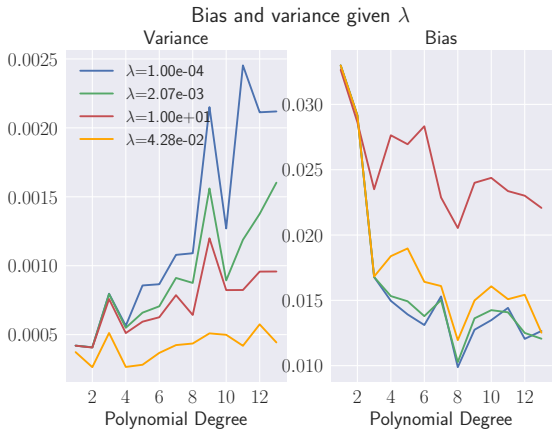

Figure 14: Plots of the bias and variance as function of the model complexity for Ridge regression with 20 bootstraps iterations and $N = 20$ for $\lambda_1 = 2.07x10^{-3}$, $\lambda_2 = 10$, $\lambda_3 = 4.28x10^{-2}$ and $\lambda_4 = 1x10^{-4}$.

In Figure 14 we have plotted the bias and variance as a function of polynomial degree, with $\lambda = 2.07 * 10^{-3}$, $\lambda_2 = 10$, $\lambda_3 = 4.28x10^{-2}$ and $\lambda_4 = 1x10^{-4}$. The bias and MSE is lower when we decrease our λ. This is expected since a small λ have similar properties to the OLS method, which cares more about minimizing the bias and MSE. When we increase our λ the penalty term $\lambda\beta^T\beta$ also increases, which means that our MSE values have to become larger. This doesn't necessarily mean that our model with the higher λ term is worse. This loss in bias precision can be "mediated" by the decrease in variance, which higher λ values often provide. We can see this trend in Figure 14. Generally, the variance is lower and the bias is higher for the larger λ values (with one exception). This observation supports the notion of an inverse relationship between variance and bias. We want to find the λ values which minimizes both variance and bias. In our case this seems to be λ_3, which surprisingly gives lower variance than λ_2 and has a reasonably low bias. By looking at the plot qualitatively, the collective minimum value for both variance and bias appears to be at polynomial degree 8.

In Figure 13 we have plotted the surface of our model, for $p = 2$, $p = 8$ and $p = 12$, using Ridge regression without resampling. It is visually obvious that the model for $p = 2$ does not provide a high enough complexity to capture the characteristics of the target function. We see that $p = 8$ matches some of the characteristics of the Franke function. However, $p = 12$, seems to give the best fit to the target function. By studying figure 14, we see that choosing polynomial degree 12, would not have led to overfitting, as the variance remains low for this complexity. We did not plot the surface for the model when using the bootstrap resampling method. So it is hard to predict how this model would have looked like compared to the target function.

What this result might tell us is that choosing a polynomial degree $p = 12$, when performing Ridge regression with bootstrapping, would have given a "better" match to the Franke function, despite the fact that this would have resulted in a slightly higher variance and bias.

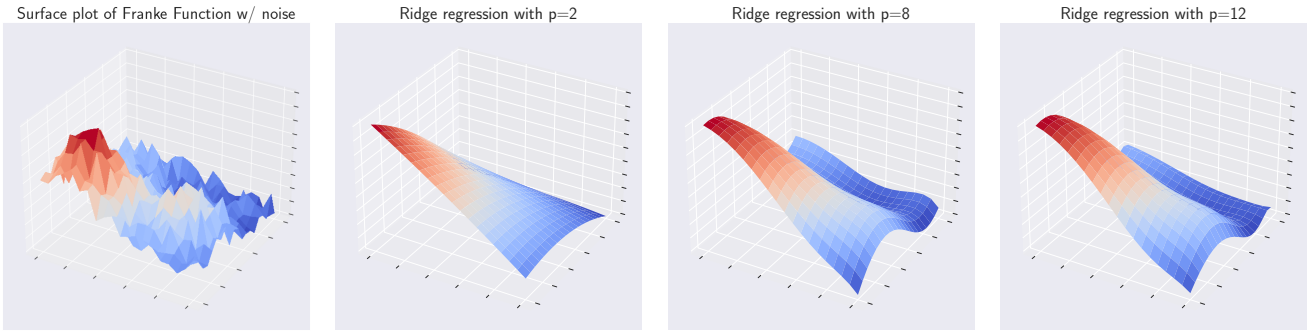

Figure 13: Comparison of the surface of the Franke function with noise, with Ridge models of varying polynomial degree.

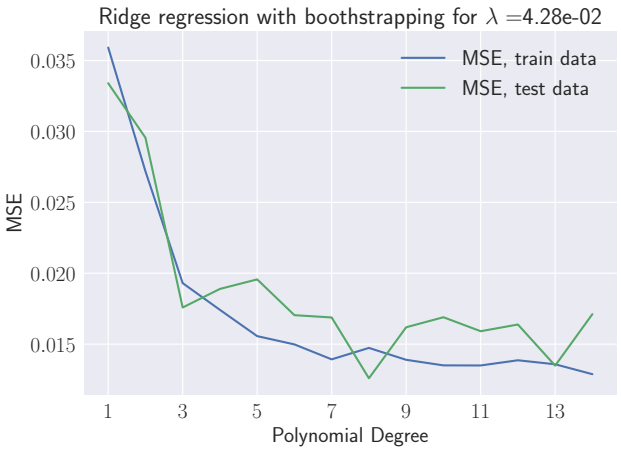

Figure 15: Plots of the MSE function as function of polynomial degree for Ridge regression with 20 bootstraps iterations with $N = 20$ and $\lambda_3 = 4.28x10^{-2}$.

We see that Ridge with the bootstrap resampling method results in slightly higher MSE values for the test function compared to ordinary least squared with the same resampling method. This is in line with our expectations, as we have an extra contribution in the cost function for Ridge regression compared to the one for OLS, which naturally will increase the MSE for Ridge regression.

K-folding

In figure 16 the MSE functions for multiple values of λ are provided. To start with, we see that the mean squared error decreases when increasing the number of folders from 5 to 10. This is the same result as observed in exercise 2. When studying the figure where $k = 10$, we see that the MSE is lowest when $\lambda = 2.07x10^{-2}$, for $p = 4$.

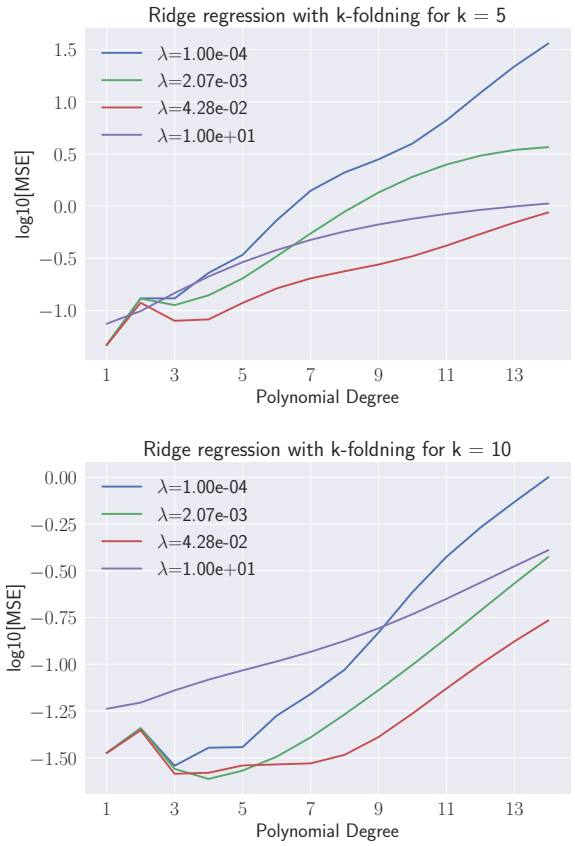

Figure 16: Plot of the mean squared error as function of the model complexity for Ridge regression with k-folding for $N = 20$ and $k = 5$ and $k = 10$, respectively.

For OLS regression we met some problems when comparing our own code with Scikit-Learn. However when performing the same comparison with Ridge regression, see figure 17, we observed that our own code fully overlapped with the result Scikit-Learn provided for all values of N, which is very fulfilling.

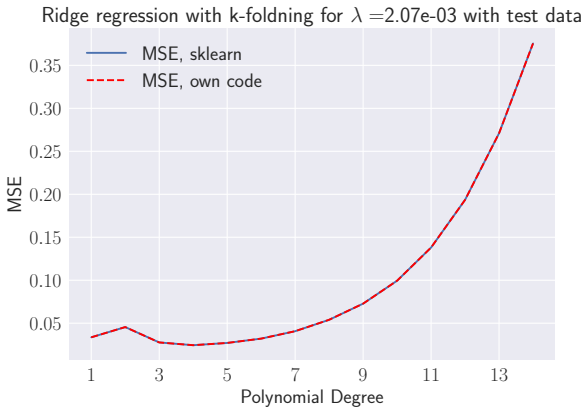

Figure 17: Plot of the mean squared error as function of the model complexity for Ridge regression with k-folding for $k = 10$, $\lambda = 2.07x10^{-3}$ and $N = 20$. In the figure the result obtained with with our own code is compared to the result produced with Scikit-Learn.

Furthermore, we have shown the mean squared error as function of λ in figure 18. From this result, we see that the MSE values for the test data is larger than for the train data, which is what one would expect. The MSE value increases as a function of larger λ values. This is due to the fact that the "weighting" of the penalty term becomes larger as we increase λ, which will subsequently increase our MSE. We see that the derivative of the MSE curve is zero from λs between 1 and 10. This could be where the λ term is so large that the algorithm is "forced" to set the β parameters to zero to minimize the sum β^{lasso} (equation 10). In the figure we have also provided the approximate minimum value for the mean squared error. What we see is that $\lambda = 0.0021$ gives the smallest value for the MSE. This supports the previous result for the λ (if we round of) giving the lowest mean squared error, when modeling with polynomial degree 4.

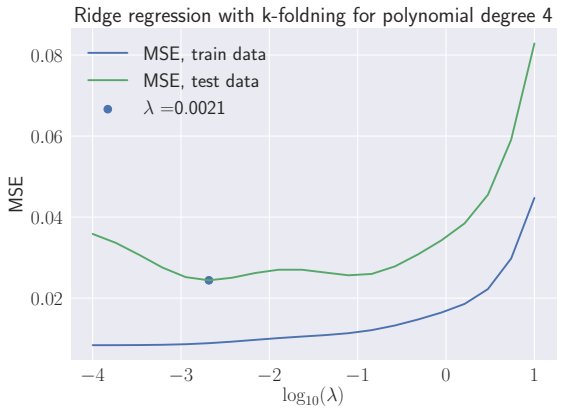

Figure 18: Plot of the mean squared error as function of λ for Ridge regression with k-folding for k = 10, polynomial degree 8 and N = 20.

Taking a closer look at the values for the mean squared error we see that Ridge with the k-folding cross-validation resampling method results in higher MSE values for the test function compared to ordinary least squared with the same resampling method. Again, this is in compliance with our expectations, since we have an extra contribution in the cost function for the Ridge regression method compared to the OLS method.

Moreover, we want to compare the MSE function when using k-folding for $\lambda = 2.07x10^{-3}$ (Figure 17) with the MSE function when using bootstrapping as a resampling technique (Figure 15). We immediately observe that the cross-validation resampling method gives larger MSE values than the bootstraping method. Whereas the lowest MSE value when bootstrapping is at approximately 0.0125 for p = 8. It does not get smaller than approximately 0.0250, for p = 2, while performing cross-validation resampling.

Exercise 5: Lasso Regression on the Franke function with resampling

The last regression model we will look at is the Lasso Regression method. This algorithm is very similar to the Ridge Regression model. The only difference being the penalty term of the RSS value which we want to minimize. We have that the ridge penalty contains the term $\sum_1^p \beta_j^2$ and the lasso penalty contains the term $\sum_1^p |\beta_j|$. We want to investigate the differences between the Lasso and the Ridge regression method. The Lasso cost function in Lagrangian form is defined by:

$$\beta^{lasso} = \text{argmin}\left\{\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|\right\} \tag{10}$$

One difference between Lasso and Ridge is that the Lasso penalty does not prioritize which parameters to minimize based on size, which the Ridge method certainly does (see exercise 4). In contrast to the Ridge method, the Lasso penalty does not have this attribute, since it only contains the absolute model parameters. This means that all values are "weighted" equally for the Lasso method.

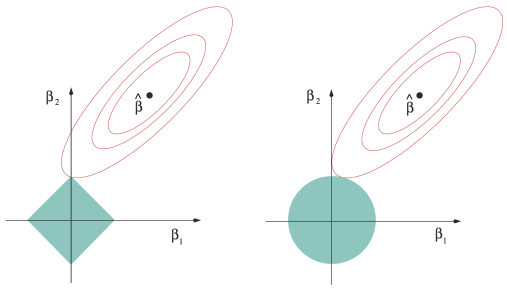

Figure 19: The blue areas are the constraint regions, where the circle is for Ridge and the diamond is for Lasso. The red ellipses are the contours of the least squares error function for both models, respectively. [1]

The Lasso regression method is capable of setting our β parameters to zero, since the shrinking of the loss term is constant. Lasso translates each coefficient with a constant λ, truncating at zero.

This is not the case for the Ridge method, where the algorithm preforms proportional shrinkage. For ridge regression we have a disk of convergence given by $\beta_1^2 + \beta_2^2 \leq t$, where t is constant. The constraint region for ridge regression is $|\beta_1| + |\beta_2| \leq t$ [1]. The contours of error and constraint functions are given in Figure 19. This illustrates why the Lasso method is capable of setting parameters to zero, while the Ridge method is not. This is shown by where the contours of least square error intersects with the constraint re-

gions. It is physically impossible for the error contour to meet the constraint regions at zero for the Ridge case, due to the smooth surface of the circle. On the contrary, the error contour can intersect the constraint region for the Lasso method at zero, since the constraint region for Lasso has sharp corners. This intersect at zero is shown in Figure 19.

As for Ridge, it is a good idea to normalize the data when using Lasso regression. We have that the magnitude of the coefficients β are dependent on the magnitude of each variable in the dataset. Irregularities in the scale of the data could result in wrongly estimated β values, which could lead to underfitting.

The algorithm for Lasso regression is equal to the one for OLS, 0, except from the differences in the expression for β, see equation 10, which can be modified. However, in our algorithm we have used the functionalities of Scikit-Learn when performing Lasso regression, as writing our own code turned out to be very difficult.

MSE for Lasso and OLS

Comparing the values for the mean squared error we see that Lasso and OLS without resampling results in pretty similar MSE values for the test function. However, taking a closer look, we see that Lasso regression method in fact produces higher MSE values (see figure 5 and 20). This is not surprising, as we know that the cost function for Lasso regression has a penalty term, and one would therefor expect the MSE to be larger for Lasso then for ordinary least squared regression.

Figure 20: Plots of the MSE function as function of polynomial degree for Ridge regression without bootstrapping with $N = 20$ and $\lambda_3 = 4.28x10^{-2}$.

Bootstrapping

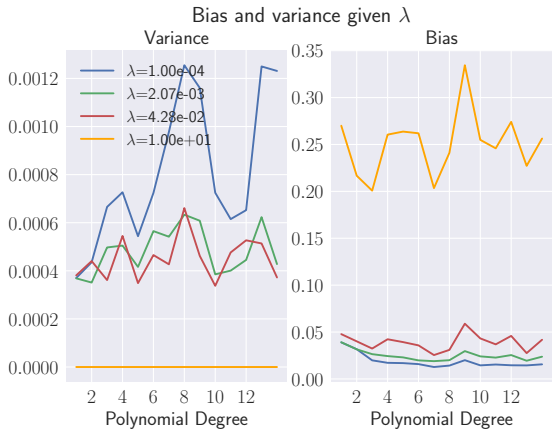

Figure 21: Plots of the bias and variance as function of the model complexity for Lasso regression with 20 bootstraps iterations with $N = 20$ for $\lambda_1 = 2.07x10^{-3}$, $\lambda_2 = 10$, $\lambda_3 = 4.28x10^{-2}$ and $\lambda_4 = 1x10^{-4}$.

In Figure 21 we have plotted the variance and bias for values $\lambda_1 = 2.07x10^{-3}$, $\lambda_2 = 10$, $\lambda_3 = 4.28x10^{-2}$ and $\lambda_4 = 1x10^{-4}$. We can see that λ_2 gives low variance and really high bias, which is not ideal. This indicates that λ values which are one or more orders of magnitudes greater than our data is not suitable as

penalty parameters. The lowest value λ_4, on the other hand, gives high variance and low bias. This is not ideal either, which suggest that we should use λ values which are closer to the same order of magnitude as the data.

We want values which have both low bias and variance, which none of the two previously mentioned values have (λ_2 and λ_4). It is more interesting to look at the values λ_1 and λ_3. The corresponding functions appear to give us a reasonable "middleground", where both the variance and bias are low. It is really up to preference which one of these λ- values you want to pick. λ_1 gives slightly lower bias, while λ_3 generally gives lower variance.

Generally, we see that the higher λ values result in higher bias, while lower λ values gives higher variance. Our results could indicate that we should use a λ value which have a lower or equal order of magnitude as our dataset. The absolute best λ parameter is hard to determine, but you can qualitatively estimate values which give good results. Another factor which could impact the choice of λ values is the amount of noise which is present. We could perhaps choose a high λ value if we had really noisy data, so that we can "weed out" most of the unwanted data.

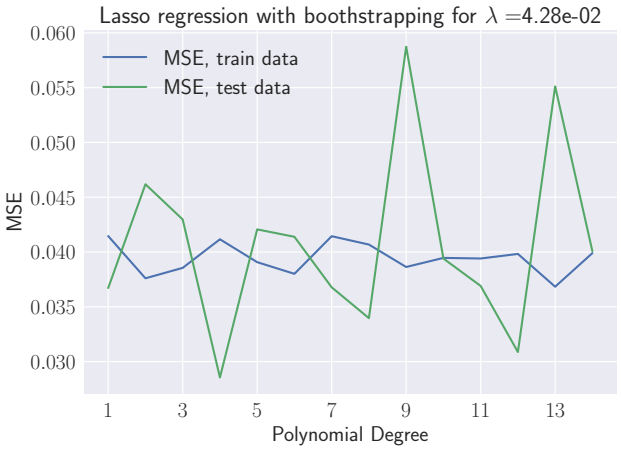

Figure 22: Plots of the MSE function as function of polynomial degree for Ridge regression with 20 bootstraps iterations for $N = 20$ and $\lambda_3 = 4.28x10^{-2}$.

We wanted to compare the values for the mean squared error obtained with Lasso and OLS regression with bootstrapping as resampling technique. In figure 22 we have provided the MSE function for Lasso regression for N=20 and $\lambda = 10$. We immediately notice that MSE function for the test data is very volatile, which is not the trend we seek. The reason for this surprising result is not easy to tell, but, it might be due to numerical or systematical errors in our algorithm. However, when performing a comparison of the values for the mean squared error we see that Lasso with bootstrap resampling method results in significantly higher MSE values for the test function compared to ordinary least squared with the same resampling method. This is in line with our expectations, as we have an extra contribution in the cost function for Lasso regression compared to the one for OLS, which increases the MSE for Lasso regression. Nevertheless, does the mentioned volatility in the MSE function for the test data, lower the validity of the result to a great extent.

K-Folding

In figure 23 we see the MSE functions for Lasso regression with k-folding for multiple values of λ. To start with, we see that the mean squared error decreases when increasing the number of folders from 5 to 10. This is the same result as observed for OLS and Ridge regression. We expect the MSE to be larger for Ridge and Lasso with k-folding, than for OLS with k-folding, due to the extra added term(s) in the cost functions. However, what we can observe is that the MSE for Lasso to a large extent is greater than for Ridge regression. Whereas Ridge regression with cross-validation resampling provides a MSE at approximately equal to 0.023 for $\lambda = 2.07x10^{-3}$ and $p = 4$ (see figure 16), Lasso gives a MSE approximately equal to 0.032 when studying the same value of λ and $p = 7$. This might tell us, that choosing Ridge as regression method, in our case, gives the best fit to Franke's function.

Further, when studying the figure where $k = 10$, we see that the MSE is lowest when $= 1.00x10^{-4}$, for $p = 3$, which is not that surprising, as such a small λ, makes the Lasso regression algorithm approximately equal to ordinary least squared regression. However, knowing that there might be some numerical and/or systematical errors in our algorithm, we will in the further analyze use $= 2.07x10^{-3}$, which gives the smallest value for the MSE when the polynomial degree is equal to 7. This value for λ provides a somewhat flat MSE curve from p=7, which might indicate that we can use a high model complexity without hav-

ing to worry about overfitting (as MSE and variance most often is inversely proportional). As a result, this might allow us to produce a model that provides a great fit to the target function.

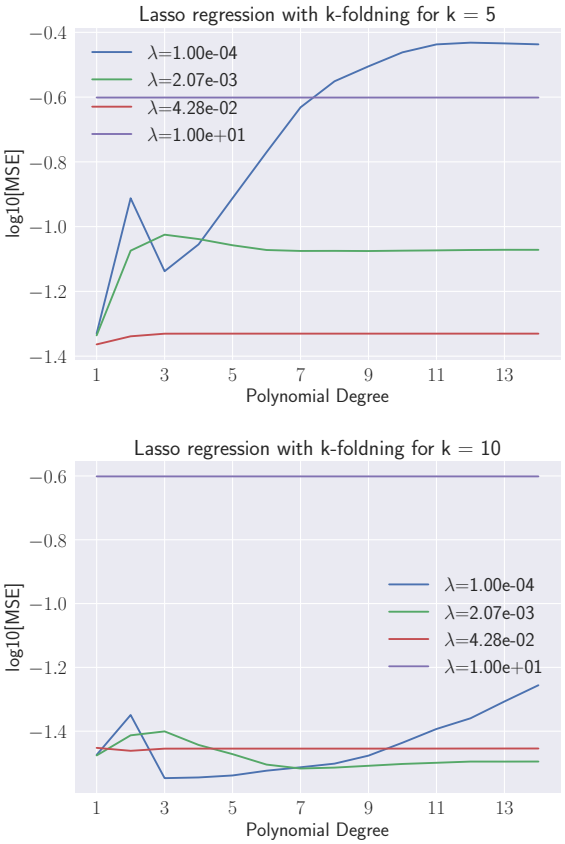

Figure 23: Plot of the mean squared error as function of the model complexity for Ridge regression with k-folding for $N = 20$ and $k = 5$ and $k = 10$, respectively.

When comparing our own cross-validation code with the one provided by Scikit-Learn, in figure 24, we see how they fully overlap, as also seen for Ridge regression. This result is very satisfying, and might tell us and helps to strengthen the validity of the results.

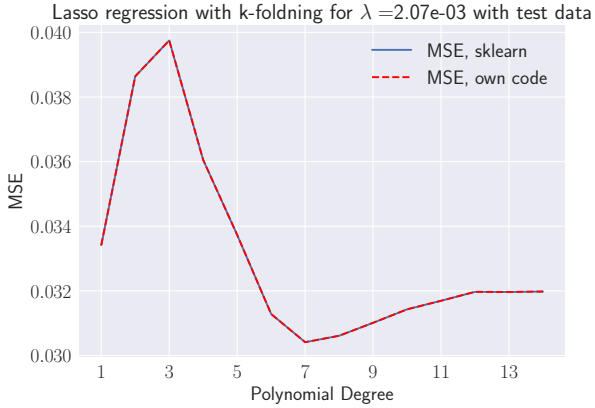

Figure 24: Plot of the mean squared error as function of the model complexity for Lasso regression with k-folding for $k = 10$, $\lambda = 2.07x10^{-3}$ and $N = 20$. In the figure the result obtained with with our own code is compared to the result produced with Scikit-Learn.

Furthermore, we have shown the mean squared error as function of λ in Figure 25. From this result, we see that the MSE values for the test data is larger than for the train data, which again is in compliance with our expectations. In the figure we have also provided the approximate minimum value for the mean squared error. What we see is that $\lambda = 0.0021$ gives the smallest value for the MSE, which is the same result as obtained using Ridge regression. This result supports that modeling with $\lambda = 2.07x10^{-3}$ gives the lowest mean squared error (at least for the λs that we have analysed), when modeling with polynomial degree 7.

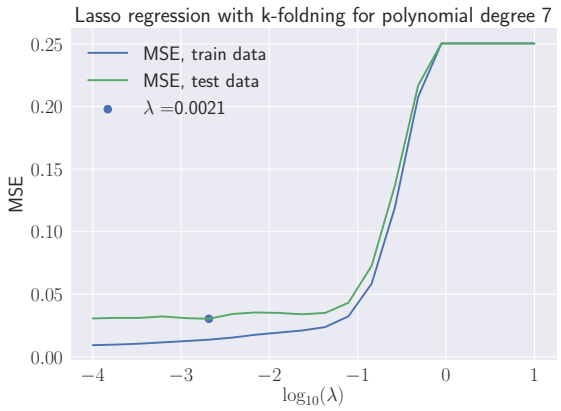

Figure 25: Plot of the mean squared error as function of λ for Lasso regression with k-folding for k = 10, polynomial degree 7 and N = 20.

If we compare the MSE function when using k-folding for $\lambda = 2.07x10^{-3}$, see figure 17, with the MSE function when using bootstrapping as resampling technique, see figure 15, we observe that cross-validation resampling method gives larger MSE than bootstraping method. Whereas the MSE value when bootstrapping, has it's lowest value at approximately 0.025 for p = 7, it is the smallest at 0.0305, for the same polynomial degree, while performing cross-validation resampling. An intersting result for Lasso regression however, might be that both bootstrapping and k-folding provides the same complexity when searching for the the optimal model. This compliance is not to be found for Ridge regression method.

Further exploration

Run time of the different methods

One aspect which we didn't look at was the runtime of the different algorithms. We expect the OLS method to take the shortest time, as it contains the least number of calculations. Ridge should be the fastest regularized model, since it has the least complicated penalty term in the penalized residual sum of squares. We also expect the k-fold resampling method to be faster than the bootstrap method, given that you have to sample the data more times when bootstrapping. We could have implemented timers in our code to figure out if these assumptions are correct.

Using the same design matrix for each polynomial

One aspect which we didn't explore in our numerical computations was to use the same design matrix for each polynomial degree when studying the bias and variance. Taking a look at the algorithm of the OLS (0) we see that for each polynomial degree we create a new design matrix which defines a new test set for each iteration. Ideally, we would like to use the same test set for all polynomial degrees to avoid randomness. We believe that doing this could have resulted in less fluctuating curves than those seen in for example figure 21.

Other improvements

In hindsight we should have created test functions to validate various steps of the different algorithms. By doing this we could potentially pick up at the problems with k-folding, or recognize that we create a new test set for each polynomial. We should also have printed more values to avoid value-comparison by looking at plots, which isn't very accurate. We also could have found better λ values in the penalty term for both regularized models, by studying more plots of the bias and variance for different λ values.

Exercise 6: Analysis of real data

Finally, we will use our methods and insights on a real dataset. We will look at a landscape in a region close to Stavanger in Norway.

Finding the optimal model complexity for OLS

We start by inspecting the MSE and R2 score as function of the model complexity, in order to estimate the optimal model complexity. In figure 27 we can see that both metrics start flattening out from about $p = 17$. The same is seen in figure 28. Thus, we will continue using a model complexity of order $p = 17$. We don't see the point of choosing a higher complexity as it would prolong the computation time as well as increase the risk of overfitting. Though we should note that we see no signs of overfitting in either figures, due to the more or less constant MSE. In figure 26 we can by "eye-test" see that for $p = 17$ we have found a good model which captures the characteristics of the terrain well.

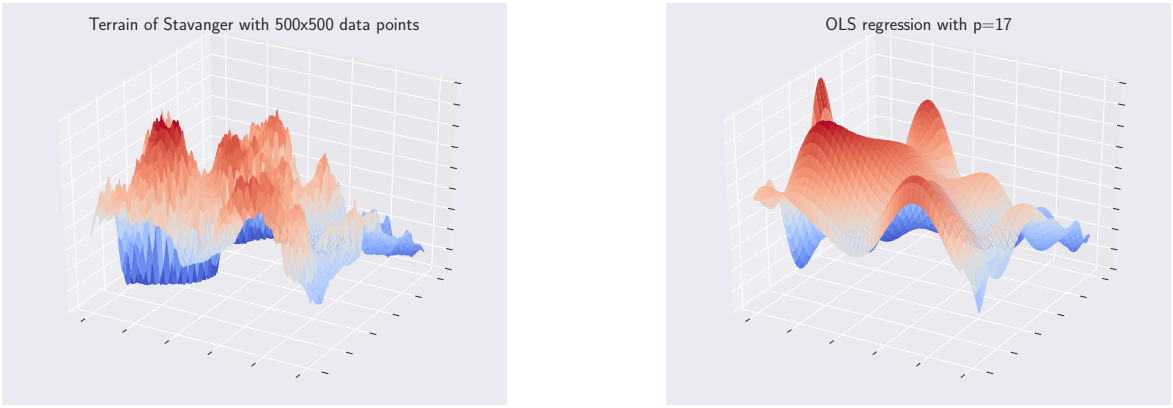

Figure 26: Surface plot of the target function (left) versus the OLS model fit of complexity $p = 17$.(right)

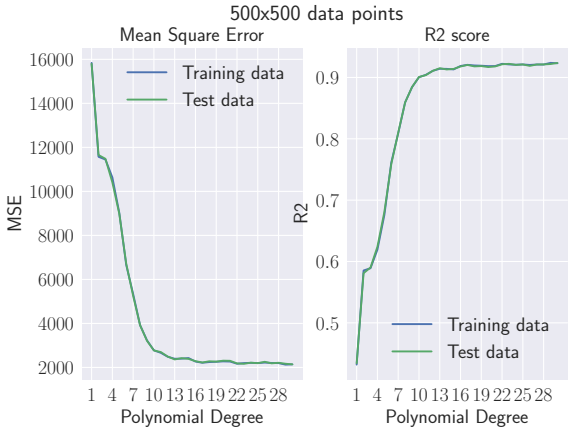

Figure 27: MSE and R2 score as function of the model complexity with OLS regression without resampling, for $N = 500$. The data is scaled.

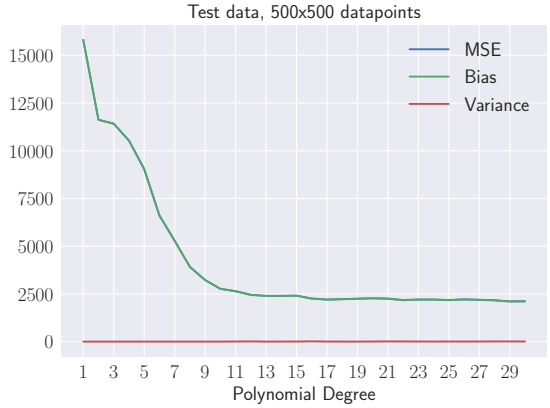

Figure 28: Bias variance tradeoff analysis of the model using the test data for OLS regression with 100 bootstraps iterations and $N = 500$. The data is scaled.

K-folding

When writing for this last exercise we discovered that the Scikit k-fold function had an option to shuffle the data, which we tried to do. When the data was shuffled we produced results which actually makes sense. This is seen in figure 29 where we basically have reproduced the results MSE as done in figure 27, but if the k-folding works as it should, with even better estimations. We see that when using k-folding the MSE

starts flattening out at an noticable lower MSE (below 2000 vs above 2000). Furthermore, it seems like the MSE from k-folding converges a bit slower towards its lowest value. We do find it a bit strange that the value of "convergence" is that much lower when using k-folding (looks like over 2 times as low). We would expect a value closer to that of figure 27. The large difference hints of a systematic error, rather than randomness. Nevertheless, figure 29 supports the proposal of $p = 17$ as a good fit. The MSE flattens out a good period before $p = 17$ and shows no signs of overfitting past.

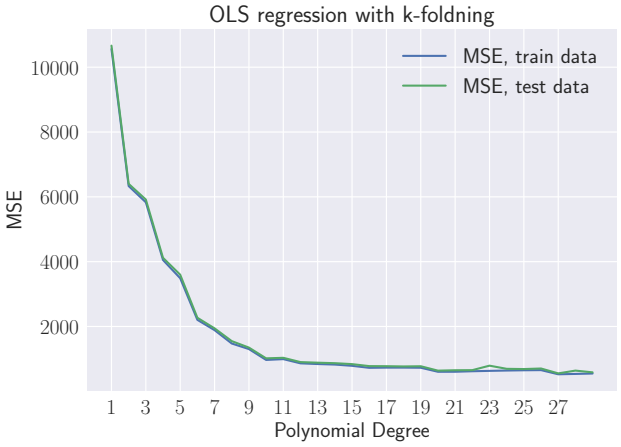

Figure 29: MSE as function of the model complexity for OLS regression resampled using cross-validation with k = 10 and N = 500. The data is scaled.

Ridge Regression

By increasing the λ's in figure 30 the overall MSE increases, as expected. We also notice that for $\lambda = 10$ the MSE is still relatively low which might tell us that because of the scale of the data, we are using too small λ-values. This is something we would have investigated if we had more time. The results of figure 31 is a bit interesting, as it tells us that the model has the lowest MSE for $\lambda = 0$, which basically is OLS regression. However this doesn't convince us that it means OLS regression necessarily is better than Ridge for this data set. If we had more time we could have plotted the variance as well, both as function of complexity and as function of λ. This could have provided more useful information. For now, we guess that we can say we are biased towards the OLS regression method (pun intended). In figure 32 we can see that there might be some truth to our bias. Though it isn't super obvious, it seems like by using a quick "eye-test", that the OLS fit in figure 26 seems like a better fit than for Ridge with $\lambda = 17$. Noted, we do acknowledge that the "eye-test" is far from bulletproof and not applicable to other types of data-fitting.

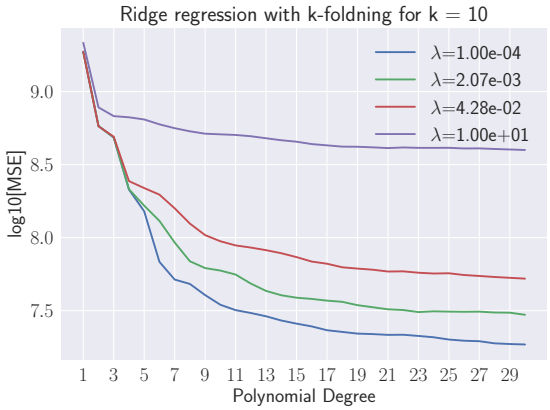

Figure 30: Comparison of various -values using Ridge regression with k-folding as resampling method, for k = 10 and N = 500. The data is scaled.

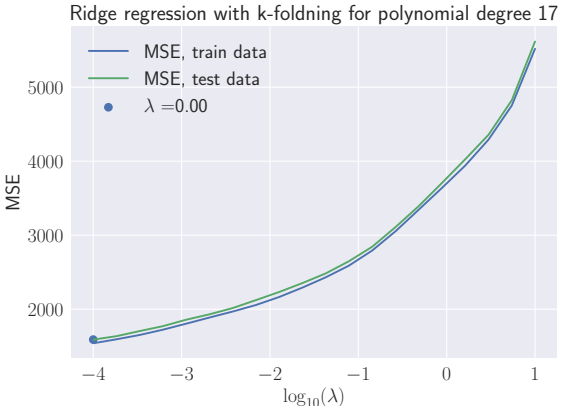

Figure 31: Plot of the mean squared error as function of λ for Ridge regression with k-folding for k = 10, polynomial degree 17 and N = 500. The data is scaled.

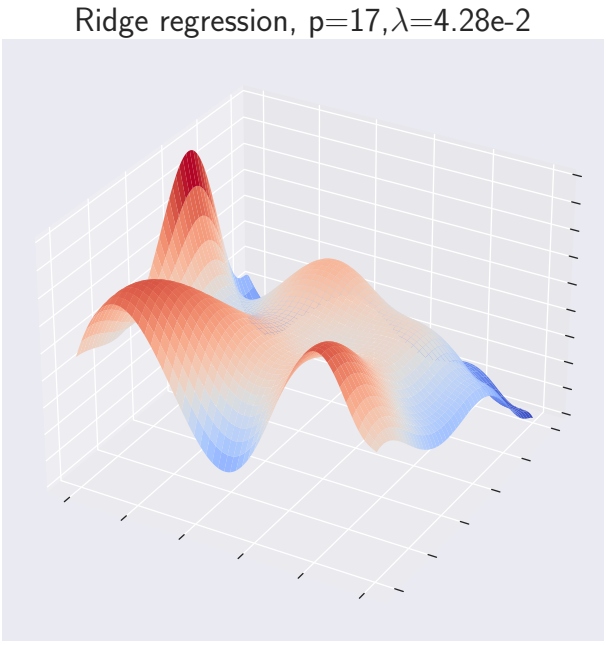

Figure 32: Surface plot of the Ridge regression fit using $p = 17$, $\lambda = 4.28 \cdot 10^{-2}$ and $N = 500$. The data is scaled.

Lasso Regression

The results for using Lasso regression (figure 33 and 34) are pretty similar to that of Ridge regression, but there are some key differences. Firstly, the MSE when using Lasso regression is in general higher than when using Ridge. This is most likely due to the structure of the penalty terms for the different regression models. For Ridge we have that the penalty term goes as β^2 and for Lasso we have that it goes as $|\beta|$. This will impact how we need to adjust our β values to compensate for the increase in λ, in conjunction with minimizing the cost functions. For Lasso we need to make our β values smaller than for Ridge to make the RSS as small as possible with the same increase in λ, which means that the MSE will generally be larger for Lasso than for Ridge.

Another observation is that the three lowest λ-values in figure 33 are much closer to each other, which also can be seen from Figure 34. This is probably also caused by the differences in penalty terms for the two algorithms. The consequence of this observation could mean that it's potentially possible to lower the variance by increasing the λ's, without increasing the MSE. This is definitely something we would investigate given the time. We also tried to plot the surface of the Lasso model, but weren't able to due to differences in the model code.

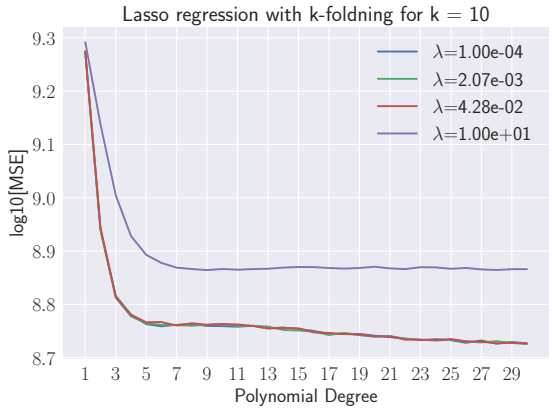

Figure 33: Comparison of various -values using Lasso regression with cross-validation, for $k = 10$ and $N = 500$. The data is scaled.

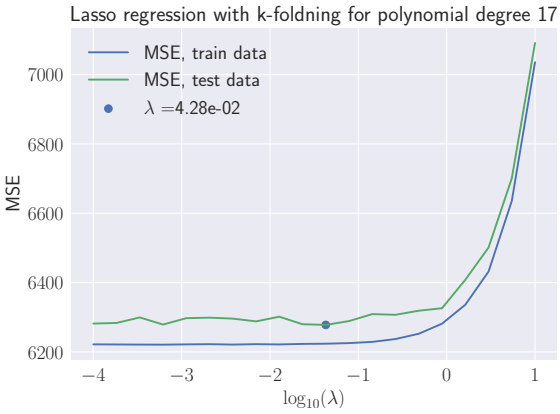

Figure 34: Plot of the mean squared error as function of λ for Lasso regression with k-folding for $k = 10$, polynomial degree 17 and $N = 500$. The data is scaled.

References

[1] Hastie, T, et al.,2016, The Elements of Statistical Learning

[2] Giba ,Boris. 2021, Ridge Regression Explained, Step by Step

[3] Lakshmanan, Swetha. 2019, How, When, and Why Should You Normalize / Standardize / Rescale Your Data?