Project 1

Morten T. Berg, Henrik H. Carlsen and Jonas Rønning

September 2021

Introduction

In this hand-in we performed a multivariable polynomial fit of the Franke function with the Ordinary Least Squares method as well as the Ridge and Lasso regression over a set of different hyper-parameters λ. For these methods we discuss how well the models fit the data by using the MSE and R^2 values as an estimate, and discuss what is the values of the polynomial degree and hyper-parameter that gives the best fit according to these measures. Moreover we also discuss the bootstrap and k-fold cross validation method and how they can be used to analyze our fitting methods.

Finally, in the last sections of the paper we apply the priorly discussed methods to a topographical map of a region near Stavanger.

1 Exercise 1

Linear Regression

The main theme of this exercise is to fit a polynomial to a two dimensional function, the Franke function, using linear regression. In a linear regression we assume that there is a linear relationship between some input parameters given by the design matrix X and the output y [1]. That is; we have a model of the form

$$y = X\beta = \beta_0 + \sum_i X_i \beta_i. \tag{1}$$

The sum is here over all relevant features. Here β_0 is the intercept. As is apparent from this expression the first column of the design matrix is $X_0^T = (1, 1, ..., 1)$. When doing the linear regression we want to pick the coefficients, β, that minimizes a cost function, $C(\beta)$. In this project we are discussing different choices of the cost function and design matrix. We are in particular interested in how to chose the design matrix and cost function that minimizes the mean square error(MSE). In all exercises we are considering a design matrix that describes a polynomial, and we vary its complexity.

1.1 The ordinary least squares

In the ordinary least square method we are trying to minimise the cost function [1]

$$C(\beta) = ||y - X\beta||_2^2. \tag{2}$$

Here X is our design matrix. The coefficients, β, that minimizes this is given by the analytic expression

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

In our code we replace the inverse with the pseudo-inverse. This is because the matrix $X^T X$ might be singular and therefore don't have any inverse and is generally computationally expensive to compute [2]. The pseudo-inverse, however, exists for any matrix. It can be calculated from the SVD of the matrix, where one take advantages of the fact that any $m \times n$ matrix A can be decomposed as [1, 3]

$$A = U\Sigma V^T. \tag{4}$$

Where U and V are respectively two $m \times m$ and $n \times n$ orthogonal matrices. Σ is a diagonal $m \times n$ matrix, with the values on the diagonal being the singular values of A. From this one find the pseudo-inverse of A as

$$A^\dagger = V\Sigma^\dagger U^T, \tag{5}$$

where $\Sigma^\dagger$ is a diagonal $n \times m$ matrix that are created by transposing Σ and replacing the non-zero singular values σ by their inverse σ^{-1} [4]. When calculating this numerically one typically remove small singular values to avoid problems. In numpy's pinv function this cut-off is by default set to $10^{-15} \cdot \sigma_{max}$, where σ_{max} is the largest singular value. Singular values smaller than the cut-off is set to zero (and not inverted).

1.2 Implementation

The aim of this task is to use The Ordinary Least Square (OLS) method on the Franke function with and without normally distributed noise. With noise the values for our function became

$$y_i' = y_i + \epsilon_i, \tag{6}$$

where ϵ is values taken from a standard distribution $\epsilon_i \sim \mathcal{N}(0,1)$. To ensure that the noise does not dominate it was multiplied by a factor 0.01.
Next the data was split into training and test sets with a $80\%, 20\%$ distribution of the data respectively. The test data was then used to perform an OLS regression on the Frank function of the form

$$\tilde{y}_i = \beta_{0,0} + x_{1,i}\beta_{1,0} + x_{2,i}\beta_{0,1} + x_{1,i}x_{2,i}\beta_{1,1} + x_{1,i}^2\beta_{2,0} + \dots . \tag{7}$$

Then we estimated how good the fifth order polynomial fit was for the train and test data by calculating the MSE and the R2 score using equation 8 and 9 respectively.

$$MSE(y, \tilde{y}) = \frac{1}{n}\sum_{i=1}^{n-1}(y_i - \tilde{y})^2 \,. \tag{8}$$

$$R2(y, \tilde{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y}_i)^2} \,. \tag{9}$$

The y_i's are values from the actual Franke function.The $\tilde{y}_i$'s are the values from the OLS method and $\bar{y}$ indicates that the mean value of all y_i's is taken.

1.2.1 Values for R2 and MSE

Variable	Train	Test
MSE	0.00179	0.00181
R^2-Score	0.9746	0.9744

Table 1: Values for MSE and R^2 for a polynomial fit of degree 5 and 200 datapoints.

As we can see in Table 1 the MSE score for both the training and test data of the OLS fit has small values, the train data has as expected a smaller value as it is the data the fit was modelled after, moreover that the test MSE is so small means that the polynomial fit is quite good. The R^2 values also tell a similar tale as the training value is better than the test value, however they are both quite close to 1 which is another indication that the fit in question is very good.

1.3 Confidence intervals for the β parameters

The confidence intervals of the parameters β were found with the formula

$$var[\beta_{jj}] = \sigma^2[X^T X]^{-1}_{jj}, \tag{10}$$

where we set $\sigma^2 = 1$. Further the confidence interval was determined as 2 standard deviations, 2σ, where the standard deviation for the β's is defined as

$$\sigma = \sqrt{var[\beta_{jj}]} \,. \tag{11}$$

The confidence interval was calculated using

$$Conf(\beta_j) = 2\sqrt{var[\beta_{jj}]} \,. \tag{12}$$

This standard deviation is shown in Figure 1. We can see that it is quite large for some of the β's. A large confidence interval is an indication that the value is uncertain. In the figure we see that the constant term, and the linear, second and fifth order coefficients are quite certain, while the third and fourth order coefficients are quite uncertain.

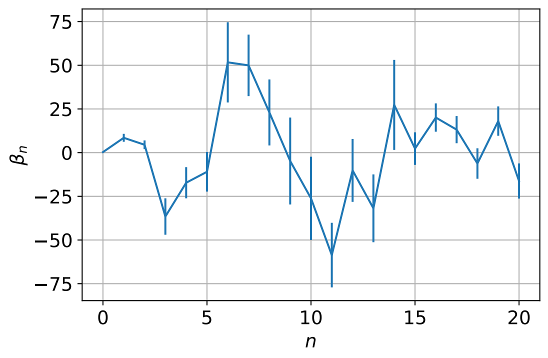

Figure 1: The confidence interval for the different β parameters. Here $n \in [0, 20]$ and indicates the β_n corresponding to the n'th column in the design matrix.

1.4 Scaling

Scaling data is an important tool in machine learning and data analysis [3]. It is used to inhibit certain features from dominating the training (in a multidimensional model). There exists a number of ways of scaling data. In our case we scale the data by subtracting the mean value of the sample from each data point and dividing by the standard deviation of the sample. When scaling it is important to scale the training set with its own mean and standard deviation to prevent data leakage. If the scaling was to be done on the whole data set before splitting the the model would have information about the test data which would give a better fit.

When using data which varies largely, by several orders of magnitude, then scaling the data is advantageous. The reason being that using unscaled data to train the model will produce large coefficients, which leaves them somewhat unstable. As we change the value of the input the output changes greatly and in turn produce a poorer fit. However our data-set has values ranging from 0 to 1 for both the input and the output, so in effect our data set is already "scaled". Thus we have chosen not to scale the data when looking at the Franke function.

2 Exercise 2

We want to show that

$$\mathcal{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \mathcal{E}[(f - \mathcal{E}[\tilde{\mathbf{y}}])^2] + \mathcal{E}[(\tilde{\mathbf{y}} - \mathcal{E}[\tilde{\mathbf{y}}])^2] + \sigma^2. \tag{13}$$

Following [5] we start by adding zero and rewrite to

$$\mathcal{E}[(\mathbf{y} - \tilde{\mathbf{y}} + \mathcal{E}[\tilde{\mathbf{y}}] - \mathcal{E}[\tilde{\mathbf{y}}])^2] = \mathcal{E}[(y - \mathcal{E}[\tilde{\mathbf{y}}])^2] + \mathcal{E}[\tilde{\mathbf{y}} - \mathcal{E}[\tilde{\mathbf{y}}])^2] + 2\mathcal{E}[(\mathbf{y} - \mathcal{E}[\tilde{\mathbf{y}}])(\mathcal{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})]. \tag{14}$$

Inserting the ansatz $\mathbf{y} = f(\mathbf{x}) + \epsilon$ the last term becomes

$$2\mathcal{E}[(\mathcal{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})f + (\mathcal{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})\epsilon - (\mathcal{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})\mathcal{E}[\tilde{\mathbf{y}}]) = 0. \tag{15}$$

So this term vanish. We therefore have

$$\mathcal{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \mathcal{E}[(f(x) + \epsilon - \mathcal{E}[\tilde{\mathbf{y}}])^2] + \mathcal{E}[\tilde{\mathbf{y}} - \mathcal{E}[\tilde{\mathbf{y}}])^2] \tag{16}$$

$$= \mathcal{E}[(f - \mathcal{E}[\tilde{\mathbf{y}}])^2] + \mathcal{E}[(\tilde{\mathbf{y}} - \mathcal{E}[\tilde{\mathbf{y}}])^2] + \sigma^2. \tag{17}$$

Where we have used that $\mathcal{E}[\epsilon^2] = \sigma$ and that terms linear in ϵ vanish when taking the average.

The first term, $\mathcal{E}[(f - \mathcal{E}[\tilde{\mathbf{y}}])^2]$ is the square of the bias and the second term, $\mathcal{E}[(\tilde{\mathbf{y}} - \mathcal{E}[\tilde{\mathbf{y}}])^2]$, is the variance. The Bias can be interpreted as the errors stemming from the assumptions and simplifications the method is built on. The variance is the spread of the data points generated by the method. We use 2 to more intuitively explain bias and variance. The red center is the aim; the perfect model.

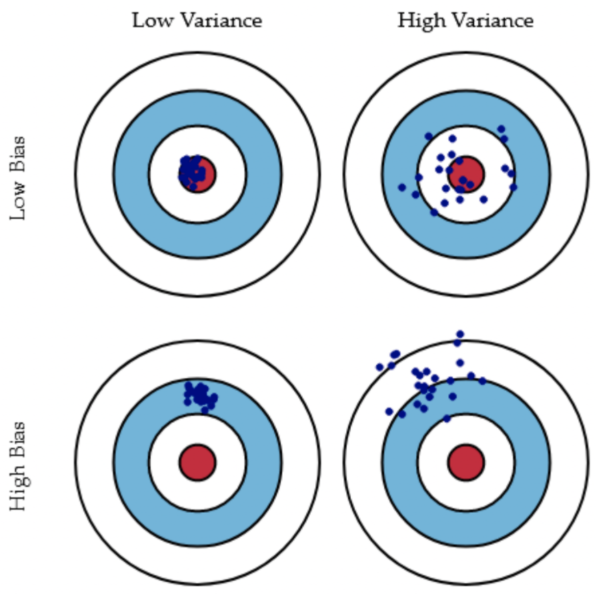

Figure 2: Illustration lecture of high and low bias and variance. Being close to the red center indicates an accurate model. The figure is from [6].

2.1 Bootstrap

The bootstrap algorithm is an re-sampling technique used to best extrapolate the underlying statistics of a limited data set. The algorithm creates a ”new” data set by sampling a random data point from the original data set and putting it back so the same data point can be chosen again. This is done until the bootstrap sample has as many data points as the original data set. As many bootstrap sets as wanted can be created. Then one runs the statistical analysis of choice on all these bootstrap samples and calculates the averages to find the best values [1].

2.2 Recreation of Figure 2.11 from Hastie et al.

In figure 3 and 4 we find the mean squared error (MSE) and the variance/bias as a function of the complexity of the polynomial we try to fit. Here our best estimates of these values are found by using the bootstrap resampling method, with 30 re-sampling cycles. From figure 3 we can see that the train and test error falls until the polynomial degree reaches 7. After that the test error increases and eventually becomes unstable. This is due to overfitting. That is a phenomena that occurs if we try to fit a high order polynomial to too few points. As an extreme we can assume that we use a polynomial of degree $N_t + 1$, where N_t is the number of training samples. This polynomial would pass through all the training data. If we tried to test this polynomial on points that where not in the training sett, it would fail.

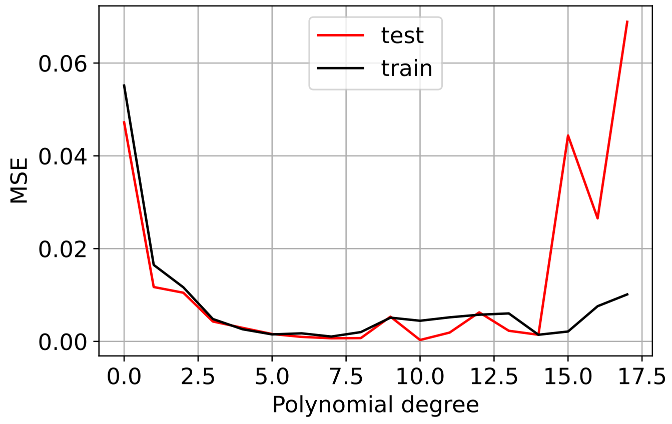

Figure 3: Plot of the MSE as a function of polynomial degree. As we can see here as the polynomial degree increases the MSE of the test data becomes unstable. This is most likely because of overfitting. The data have been resampled using the bootstrap method and 17 data points.

In figure 4 we see that the variance and bias are quite low for all the values of the polynomial degree that we consider. The variance grow with the polynomial degree, while the bias are more or less stable. This means that for higher degree the estimates are still centered near the target, but they are spreding out.

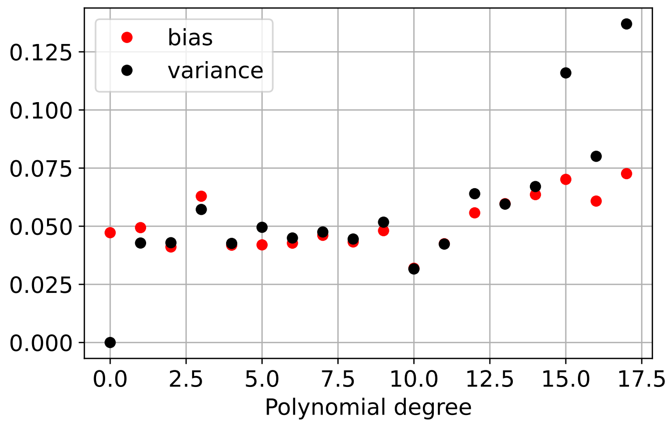

Figure 4: The figure shows the bias vs variance for polynomials of increasing degrees from 0 to 17.

3 Exercise 3

In this exercise we write our own code for cross-validation. The exercise says that we should scale our date, but we did not see any big differences when doing this, neither for our own code nor when using sklearn's built in function.

3.1 Cross-validation

The cross-validation algorithm works as follows. We start by dividing the data into K equally sized parts. One of the parts, say i, is taken out and the model is trained on the remaining parts. One then test the model on the part i that were left out of the training and get an estimate for the error. One repeats this so that all the parts will play the roll of the test, that is $i = 1, 2, ..., K$, and take the error to be the average of the errors estimated by the parts [1]. How many parts, or folds, one want to divide the data into is problem dependent. If the data set is small one should use a larger number of folds. The typical sizes are between $K = 5$ and $K = 10$. The special case $K = N$ is named leave one out, since one train the model on all data points except one [1]. Cross-validation is widely used in practise since it gives a good trade off between accuracy and computation time.

When preforming the cross-validation we use an function from scikit-learn to preform the split for us. Our homemade cross validation is tested against the one that is provided by scikit-learn and our bootstrap method. The results are produced with the number of folds $k = 10$, as it gave the best results.

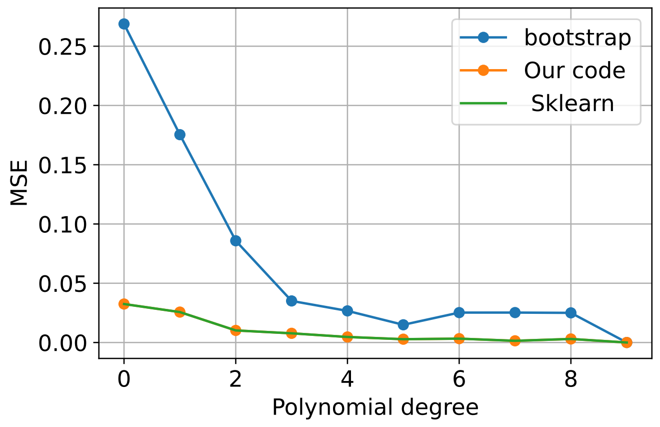

Figure 5: In this figure we see the comparison of the SKlearn method of k-strap cross validation and our own K-fold as well as our own bootstrap method. As we can see the bootstrap gives a higher MSE for small polynomials. Our own k-fold, denoted by "our code", is exactly equal to Sklearns version. The plot was produced by using 40 data points.

4 Exercise 4

4.1 Ridge regression

In ridge regression we want to minimize the cost function [1]

$$C(\beta) = ||y - X\beta||_2^2 + \lambda||\beta||_2^2. \tag{18}$$

Here $\lambda > 0$ is a tunable parameter that shrinks the size of the regression parameters. It is therefore often referred to as the shrinkage parameter. Similarly to the OLS method there exists an analytical expression for the best choices of β given as

$$\beta = (X^T X + \lambda I)^{-1} X^T y. \tag{19}$$

The addition of λI to $X^T X$ before inverting ensures that the inverse always exists[1]. This is one of the standard ways of regularising a matrix before taking the inverse. Even thought the inverse is guaranteed to exist we will use the pseudo-inverse to avoid problems with matrices that are nearly singular.

To study the bias-variance trade-off for different values of λ and polynomial degree we use Ridge regression with 30 bootstrap samples to produce the results. The optimal polynomial degree was found to be 13, and the results for a polynomial of that degree is what is discussed bellow. Note that we have increased the number of data points to 100. Figure 6 shows the bias and variance for $\lambda \in [10^{-2}, 10^3]$. Notice that the variance tends to shrink when λ is increases, while the bias grows. This indicates that the spread of our estimates becomes less, but they miss the target (see Figure 2).

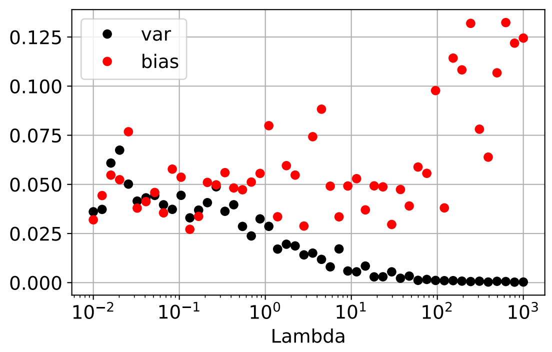

Figure 6: The bias and variance is plotted for different values of λ. At around $\lambda = 10^2$ the bias increases exponentially as the variance converges to zero. The number of training samples are 30.

The hyperparameter λ typically shrinks the least important features. One can see this from inserting the SVD of X in to $\tilde{y} = X\beta$. Inserting Equation (19) and using the SVD one find [4]

$$\tilde{y}_i = \sum_{j=0}^{p-1} u_j u_j^T \frac{\sigma_j^2}{\sigma_j^2 + \lambda} y_i. \tag{20}$$

Where σ_j is the singular values and u_j are the column vectors of the U matrix in the SVD (see equation (4)). The presence of λ shrinks all features, but the ones related to a low singular value are shrunken the most. When $\lambda \to \infty$ the predictions goes to zero. This is why we get such a high bias at the same time as the variance disappears for a high λ. The MSE as a function of lambda for the train and test data is shown in figure 7. From the figure we see that the lowest values of λ gives a smaller MSE. When λ becomes small this method approaches the OLS and this indicates that the OLS method is better suited than Ridge for this problem. The reason that the MSE increases with λ is the same as discussed above for the bias and variance. Important features are shrunken so that the model just puts out zero (in the $\lambda \to \infty$ limit).

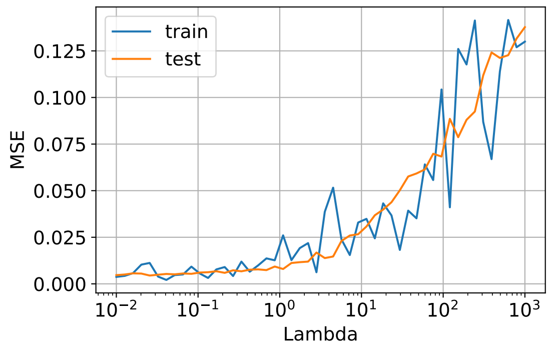

Figure 7: The MSE found with the bootstrap resampling technique for Ridge as a function of the hyperparameter λ. The number of training samples are 30. A polynomial of degree 13 was used.

The same trend for the MSE is found when we use cross validation. This can be seen in Figure 8 where it is apparent that the MSE increases with λ. Generally the MSE for the bootstrap method is smaller than the one provided by the k-fold on all values of λ for a polynomial of degree 13.

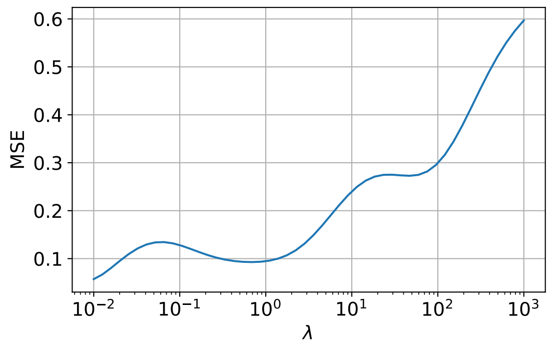

Figure 8: The MSE for Ridge regression with 10 k-folds is plotted as a function of λ for a polynomial of degree 13.

5 Exercise 5

5.1 Lasso regression

In Lasso regression we want to minimise the cost function [1]

$$C(\beta) = ||y - X\beta||_2^2 + \lambda||\beta||_1. \tag{21}$$

Here we have replaced the l_2-norm in the Ridge regression with the l_1-norm. Contrary to the two previous methods there is in general no analytic expression for the optimal β's for this cost functions. There does however exist fast iterative algorithms to minimize this cost function[1].

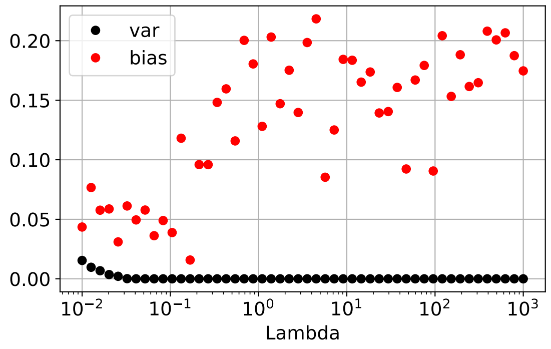

Figure 9: The figure shows the bias-variance trade-off as function on the hyper parameter λ. The polynomial degree is 14.

We have looped over different polynomial degrees and values for the hyper-parameter. We found that the lowest MSE was obtained for a polynomial of degree 14. The variance and bias for this polinomial degree is shown as a function of λ in figure 9. and we see the same trend as for the ridge regression. That is that the bias grow with λ, while the variance shrinks and eventually vanish. That the variance decrease is due to the fact that higher values of the parameter λ kills off the less important features. If λ becomes to large the lasso will output just zeros like ridge.

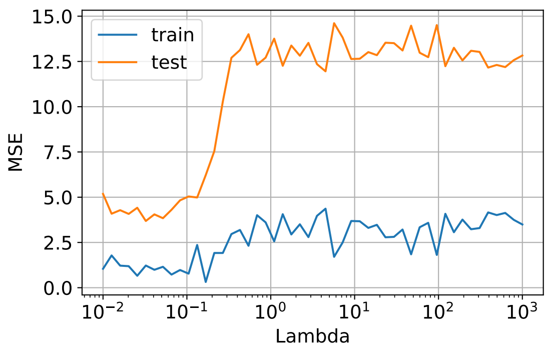

Figure 10: For both train and test data the MSE is calculated as a function of λ for a polynomial of degree 14. The values are obtained by using the bootstrap re-sampling method with 30 cycles.

Figure 10 shows the MSE for the train and test data using bootstrap, and Figure 11 shows the MSE obtained by cross validation for those same values of λ. The bootstrap MSE is found in the same loop as the bias and variance above, while the cross validation is unrelated. The fact that the MSE is less for smaller λ indicates that the OLS method is preferred. Also note that the error is roughly 100 times larger than what is found with Ridge and OLS. This is the same as the number of sample points that where used. There is therefore very probable that a factor $1/n$ is missing somewhere in our code. This factor is not seen when using other of scikit-learn library functions (See Figure 5), so Lasso is acting up. The polynomial degree that gave least MSE was 7 for the cross validation.

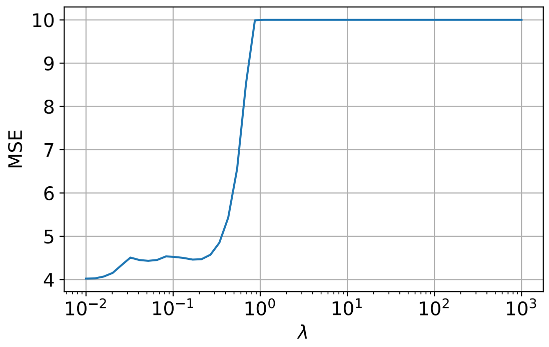

Figure 11: The MSE as function of λ is plotted using Lasso regression with k-fold cross-validation with $k = 10$ and a degree of order 7 was found to be the optimal polynomial for k-fold cross validation.

6 Exercise 6

In this exercise we chose to take a closer look at the given data set $SRTM - data - Norway - 1.tif$ which is a topological map of the region close to Stavanger. We remove every third data point from the data set to make it less time expensive to work with. Also, in real life it is not uncommon to have an incomplete data set.

6.1 OLS regression

Further we ran an ordinary least square regression for the data set over the polynomial range [0,20] to see which polynomial degree would be best suited to the landscape. The data was partitioned into training and test sets with the same distribution as used earlier, namely training data being 80% of the data and test being 20%. Then the training and test sets were scaled with the formula

$$x' = \frac{X - \mu}{\sigma}\,, \tag{22}$$

where σ is the standard deviation of x and μ is the mean value.We also implemented the bootstrap method as we did the OLS method to get "more" training data and a better fit. Moreover from this we got values for the train and test MSE for each polynomial degree as shown in 12.

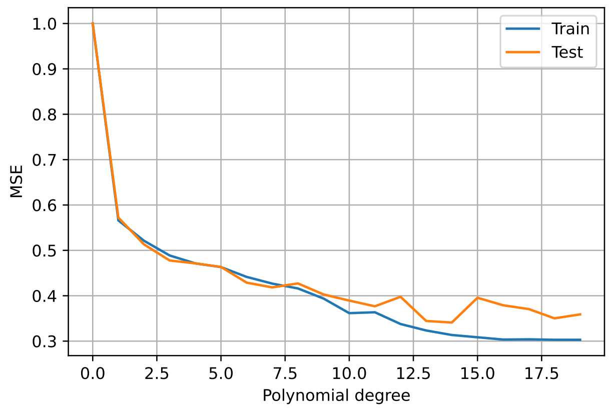

Figure 12: The MSE is plotted against polynomial degree for both train and test data.

6.2 Ridge regression

For the ridge regression we did mostly the same as we did for the OLS regression in terms of scaling, splitting and running over the same polynomial range, however this time we also ran over an additional set of hyper parameters λ to try and single out the best polynomial degree as well as the best λ value. The best polynomial degree is found to be 17. In Figure 13 we plot the MSE for both train and test data as a function of λ for a polynomial of the best found degree. The best hyperparameter is found to be $\lambda = 10^{-2}$, which is the smallest we consider. This is likely due to the fact that regular OLS preform best for this type of fitting.

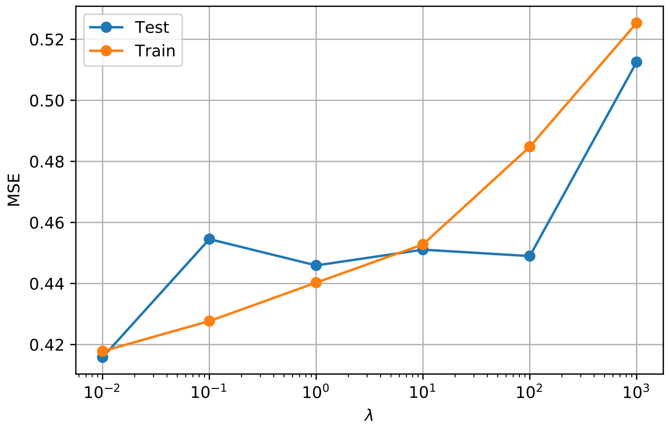

Figure 13: The MSE is plotted as a function of the hyper parameter λ for the best polynomial degree, 17.

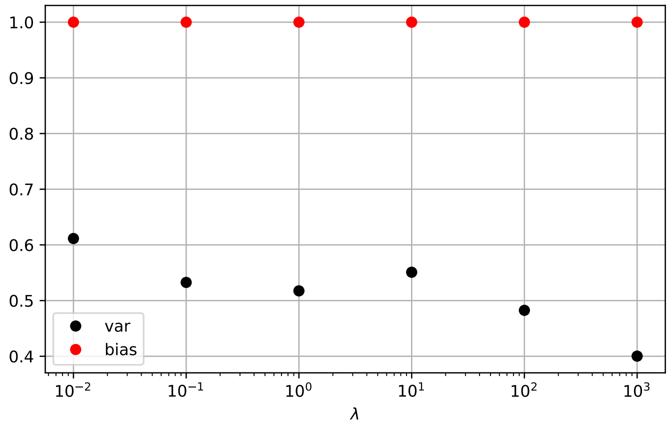

Figure 14: The model performance in terms of bias and variance as function of the hyper parameter is plotted. The bias remains unchanged for all values of λ. The polynomial degree is 16.

Figure 14 shows that the variance decreases as the size of λ increases. However; the bias does not decrease at all. So our model gives a bad fit in terms of the bias for all values of λ. This means that the model are missing the target even for small values of λ. The predictions are in addition to being centered around the wrong value pretty spread. Meaning that this model does not preform well.

6.3 Lasso regression

For this task the lasso regression fail miserably for the considered values of the hyperparameter λ. In Figure 15 and 16, respectively, the Bias and variance, and the MSE is plotted against the hyperparameter λ for a polynomial of degree 14, Which was found to give the best fit. Notice; in figure 15 the bias is quite stable at around 1, while the variance falls to zero. We are therefore in the low variance, high bias regime where the predictions have a low spread, but they miss the target. In Lasso regression the parameter λ kills of the least important β's. It seems that for $\lambda > 1$ it have removed to many and the model just returns zero. The MSE shown in Figure 16 is unbelievable high. For the OLS it was between $0.3-1$, but for Lasso it is over 1000 for scaled data. The number of data points used are around 9000, so again it is hinted that a factor $1/n$ is missing. However, when replacing sklearn's Lasso function with its Ridge method we get results that is comparable to what we found with our own ridge function in section 6.2. Sklerns's Lasso is therefore acting very strange. Note that the error stabilizes for $\lambda = 1$ which is likely due to the model having killed of to many features and is returning zero's.

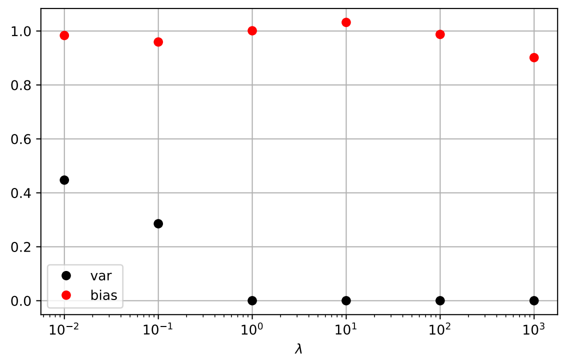

Figure 15: Bias and variance plotted against the hyperparameter λ for the lasso regression for a polynomial of degree 14.

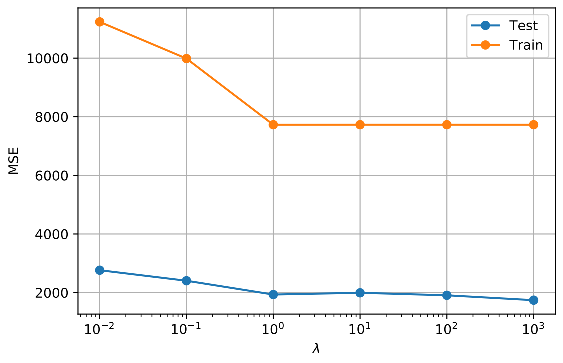

Figure 16: MSE plotted against the hyperparameter λ for the lasso regression. The degree of the polynomial is 14.

6.4 Our best fit

In the above discussion we found that the best fit was with the OLS method for a polynomial of degree 14. In Figure 17 the best fitted polynomial is plotted against the real data. We can see that the fit is not perfect, but we can recognise the most important features. The model does miss the deepest and highest points, in addition to making the valleys to broad.

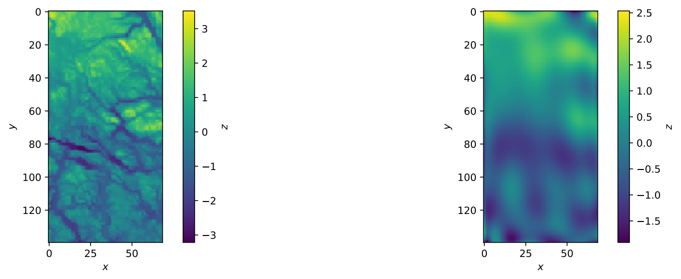

Figure 17: Plots of the mountains and our best fitted function.

7 Summary and Conclusion

We have tested different methods for linear regression for fitting a polynomial to a two dimensional function and a region outside of Stavanger. We estimated the MSE, bias and variance of our models using the bootstrap re-sampling technique and cross-validation. We found that the lowest MSE was obtained when using OLS method, which we also saw when varying the hyperparameter for the other two methods. We therefore conclude that the OLS method is best for this type of problems.

8 Feedback

In general we think this exercise has been good. It has been easy to get help on the computer lab and the online learning material(jupyter notebooks etc.) have been really helpful.

References

[1] J. Friedman T. Hastie, R. Tibshirani. *The elements of statistical learning : Data mining, inference, and prediction*. Springer Science Business Media, 2009.

[2] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre GR Day, Clint Richardson, Charles K Fisher, and David J Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics reports*, 810:1–124, 2019.

[3] Morten Hjorth-Jensen. Week 35: From ordinary linear regression to ridge and lasso regression. https://compphysics.github.io/MachineLearning/doc/pub/week35/html/week35.html.

[4] Morten Hjorth-Jensen. Week 36: Statistical interpretation of linear regression and resampling techniques. https://compphysics.github.io/MachineLearning/doc/pub/week36/html/week36.htmll.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, 2006. https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf.

[6] Morten Hjorth-Jensen. Week 37: Summary of ridge and lasso regression and resampling methods. https://compphysics.github.io/MachineLearning/doc/pub/week37/html/week37.htmll.