FYS-STK4155 Project 1

OLS, Ridge and Lasso Regression with Resampling

Ingrid Utseth
Polina Dobrovolskaia

Ingrid Utseth
ingrid.utseth@fys.uio.no
Polina Dobrovolskaia
polinad@student.matnat.uio.no

Dato: October 8, 2018

1 Abstract

This study presents and investigates three polynomial regression methods: Ordinary Least Squares, Ridge regression and Lasso. It presents the analytic characterization by MSE and R^2-score variations for a group of different polynomial models for data sets with different degree of noise. We studied trends and features of each polynomial and investigated which of the methods would fit Franke's function and digital terrain data in the best possible way.

We observed minor differences in performance degree between Ordinary Least Squares and Ridge regression. Of all models, Lasso gave the least favourable modelling outcome. Based on error metrics such as MSE, R^2-score, bias and variance we concluded that polynomial approximation of degree 5 by the Ridge regression was the best fit for our Franke's function. From the studies with the terrain data we got the most fair approximation of our data set with Ridge regression of polynomial degree 8.

Contents

2 Introduction

In the field of Machine Learning there are many algorithmic approaches which may be useful when it comes fitting function to a set of given data. The performance of different algorithms will depend on the size, structure and noise of the data. In this report, we will investigate different fitting techniques and analyze their ability to model medium-sized data sets. The accuracy of each model will be evaluated using Mean Squared Error (MSE), R^2-score, bias and variance.

We will here compare three different polynomial regression algorithms; Ordinary Least Squares, Ridge regression and Lasso. In the first part of this project we will use randomly generated data and Franke's function. We experiment by tuning our algorithms by varying different parameters and thus find strengths and weaknesses of each regression method. We will use a resampling technique, the bootstrap algorithm, to estimate the MSE, R^2-score, bias and variance and extract the confidence interval for our β-coefficients.

In the second part we will dig into a real-life problem by exploring and adapting our previous algorithms to model digital terrain data.

3 Method

3.1 Generated Data

In order to test our methods, we would ideally like an large amount of data; however this is not always possible in real-life experiments. Therefore, we consider the Franke's function with added stochastic noise using the normal distribution, $G(x,y)$,

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

$$z(x,y) = f(x,y) + D \cdot G(x,y), \tag{2}$$

where D is a positive constant indicating the strength of the added noise. Using function represented by equation 1, we can generate as much data as we wish using randomly chosen x and y-values between 0 and 1. Initially, we generated a data set consisting of thousand x, y and z points.

We explore the method of Ordinary Least Squares (OLS) before delving into more advanced methods (Ridge regression and Lasso). This was done by studying how the randomly generated data with different noise levels, D, was fitted to models with different polynomial degree. The analysis of the predicted models was done by investigating the relationship between model complexity (polynomial degree) and error metrics such as MSE.

Before we delve into the analysis of out first polynomial regression method, we need to establish some common grounds that will let us estimate the error of each model and later compare the performance of the different methods.

To evaluate the accuracy of our methods, we calculated the Mean Squared Error, which measures how our model differs from the true data.

$$MSE(\hat{y}, \tilde{\hat{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (\hat{y}_i - \tilde{\hat{y}}_i)^2 \tag{3}$$

$\hat{y}$ is the true value, $\tilde{\hat{y}}$ is the predicted values.

MSE measures the average square in error between real values and the values we obtained from our model. Thus, it is handy to have another parameter that would say something about how close the data is fitted to the regression line. R^2-score is an engine that explains how the predicted model varies in comparison to to the real data variance. It is also worth noting that MSE varies with the scale of the data, while the R^2-score is normalized. This will be useful later, as our digital terrain data is on a much larger scale than our generated data.

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

where $\hat{y}$ is the true value, $\tilde{\hat{y}}$ is the predicted values. $\bar{y}$ is the mean value of the true values $\hat{y}$. In general, we would like the MSE to be as small as possible and the R^2-score to be close to its maximum value of 1.

However, we do not always want our model to follow the data too closely, as this will often result in overfitting. We add noise to the Franke's function, and if our model follows the data too closely, we might miss the overall trend of the function in between all the local minima and maxima. To avoid this, we also looked at variance and bias,

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

$$\text{VARIANCE} = \frac{\sum_{i=0}^{n-1} (\tilde{y}_i - \bar{\tilde{y}}_i)^2}{n}, \tag{6}$$

where y_i is the true values at point i, $\tilde{y}_i$ is the predicted values at a specific point i, and $\bar{\tilde{y}}_i$ is the average predicted value at point i. The sum of the bias and the variance should in general almost equal the MSE. High bias indicates that the model is underfitted; meaning the model does not have parameters to represent our data in a good way. High variance indicates overfitting - the underlying trends of the data is lost. [2]

The *OLS* method is suitable for experimental data with Gaussian measurement errors; which in our case will be the noise. [3] The squared error cost function, $Q(\hat{\beta})$, is defined as the squared difference between the observed (in our case randomly generated) values and the predicted values. In other words, *OLS* is a method where we find the minimal spread of our cost function. Mathematically, this can be represented in the following way,

$$Q(\hat{\beta}) = \sum_{i=0}^{n-1} (z_i - \tilde{z}_i)^2 = (\hat{z} - \hat{X}\hat{\beta})^T (\hat{z} - \hat{X}\hat{\beta}) \tag{7}$$

$$\frac{\delta Q(\hat{\beta})}{\delta \hat{\beta}} = 0 = \hat{X}^T (\hat{z} - \hat{X}\hat{\beta}) \tag{8}$$

$$\hat{\beta}^{\text{OLS}} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\hat{z} \tag{9}$$

Rearranging the equation above, we can identify $\hat{\beta}^{\text{OLS}}$. The process of modeling from "observed" data to predicted data can be stated in the following way,

$$f(x,y) + D\cdot G(x,y) \longrightarrow z(x_i,y_i) = \tilde{z}_i + \epsilon_i = \sum_{\substack{j=0\\k=n-1}}^{\substack{j=n-1\\k=0}} \beta_j x_i^j \cdot y_i^k \tag{10}$$

Here $\tilde{z_i}$ is our observed data with $\epsilon_i = D\cdot G(x)$ - stochastic noise; an usual phenomenon in experimental data. $\hat{X}$ is a matrix containing our polynomial parameters; in this case columns of powers of $\hat{x}$ and $\hat{y}$. The size of the $\hat{X}$ is dependent on the polynomial degree that we wish to fit the observed data to. The predictive performance of our model is thus controlled by degree of $\hat{X}$. When the model gets more complex, fitting with higher order polynomials, the model can become more influenced by noise rather than the overall trends in the data. [3] This often leads to overfitting.

In order to choose the best complexity for our model, we studied OLS models with polynomial degree between 2 and 20. We also experimented with different degrees of noise to evaluate the impact it had on the performance of the model. We would later reuse the optimal complexity from these tests with our two other regression methods, as they build on OLS.

When presented with real-life data, one does not always have opportunity to generated more data in order to test the models. One way to solve this problem is to perform a re-sampling of the data. The total data set is split two parts, training and test data. We shuffled our data set and set aside about 20% of the data to be used for testing of the trained model. On the remaining data we used the bootstrap algorithm with 100 re-sampling steps to train the model with OLS.

Next, we estimated MSE, R^2-score, bias and variance against the test data (see appendix 6.2.4 for Python code example). We also estimated the confidence interval for the β-coefficients.

If the data contains a lot of noise, the β-coefficients from OLS may have a high variance and become unstable; small changes in the data results in large changes in the model. [4] One way to mitigate this problem is to use shrinkage methods such as Ridge regression.

$$\hat{\beta}^{\text{RIDGE}} = (\hat{X}^T\hat{X} + \lambda\hat{I})^{-1}\hat{X}^T\hat{z} \tag{11}$$

$\hat{X}$ and $\hat{z}$ is defined as before, while λ is a non-negative number controlling the amount of shrinkage. Essentially, a few of the β^{RIDGE} parameters are reduced to zero, thus the model gets a better performance degree and stability.

We continued using the most appropriate level of complexity from our OLS analysis. In order to determine an optimal λ-value, we compared the MSE and R^2-scores of eight Ridge regression models with varying shrinkage powers. We used new generated test data to estimate the MSE and R^2-score.

The amount of noise in the data can also influence the fit. Therefore we also investigated some of the most promising λ-values and their mean square errors with varying levels of noise. Again, we used new generated test data to estimate the MSE.

We calculated the confidence interval of the β-values for the model with the most promising λ-value and the degree of noise $D = 0.1$, again using new generated test data. Finally, we estimated the Mean Squared Error, R^2-score, bias and variance of the final model using only our original, 1000-point data set and the bootstrap algorithm.

The final regression method tested was the Lasso, another shrinkage method,

$$\hat{\beta}^{\text{LASSO}} = \arg\min_{\beta}\{\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{P} x_{ij}\beta_j)^2 + \alpha\sum_{j=i}^{P}|\beta_j|\} \tag{12}$$

In this case, we implemented the Lasso method using Python's machine learning library sci-kit learn (sklearn). As with Ridge regression, we assume that our investigation of optimal polynomial degree with the OLS method still holds. As with Ridge regression, the Lasso includes a constant α that controls the amount of shrinkage. We cannot assume that the optimal value λ from the Ridge regression analysis will give us a good result with the Lasso. Therefore, we investigated how various α-values affected the MSE and R^2-score for the Lasso. We also investigated how a Lasso model for a function with some degree of noise was influenced by some of the α-values.

After finding a suitable α-value, we adjusted some of the other variables of the sklearn lasso function. `max_iter`, the maximum number of iterations it will run until convergence is reached, was set to 5000. `fit_intercept` was set to `False`. Our function call going forwards looked like this,

```
Lasso(alpha=a, max_iter=5000, fit_intercept=False)
```

Using the bootstrap algorithm, we calculated the MSE, R^2-score, variance and bias of our final Lasso model. Like previous, the degree of stochastic noise $D = 0.1$ and we performed 100 rounds of resampling.

3.2 Terrain Data

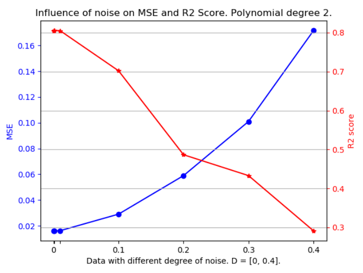
After analysing the three models with generated data, we performed similar analysis on digital terrain data. We selected a small area in the vicinity of Rjukan, Norway. The size of the matrix containing the terrain data is [250, 250].

First, we analyzed fit of the terrain data with the OLS method. We experimented with models of various polynomial complexity and visually compared the result to the real image of our data set. Using the bootstrap algorithm, again with 20% of the data used in testing and 100 rounds of resampling, we calculated MSE, R^2-score, bias and variance to gauge the performance of obtained models.

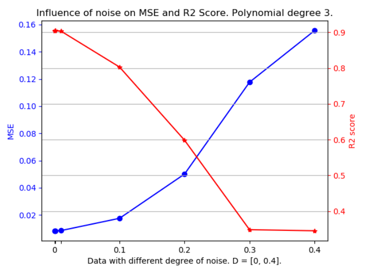
Continuing with the most suitable polynomial degree from the OLS analysis, the next step was to find the right λ-value for Ridge regression. Again, the different λ-values were evaluated using the bootstrap algorithm with 20% of the data used in testing and 100 resamples. Similarly, we evaluated various α-values for the Lasso. In this case, however, we were forced to adjust the number of bootstrap resampling rounds down to 10 and the `max_iter` parameter of sklearn's lasso function down to 1000 for reasons of computational speed. This will have a negative effect on the final result.

4 Results and discussion

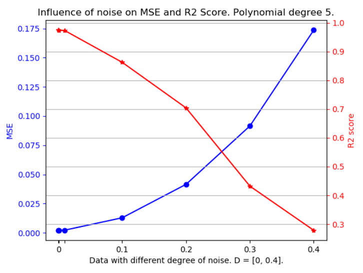
4.1 Generated data



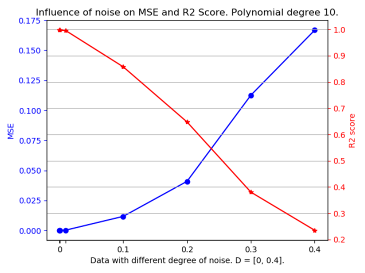
(a) MSE and R2 for degree 2.



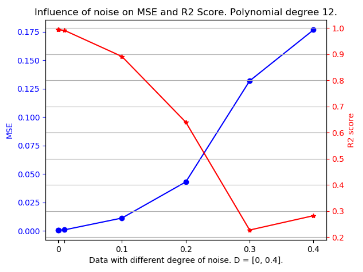
(b) MSE and R2 for degree 3.



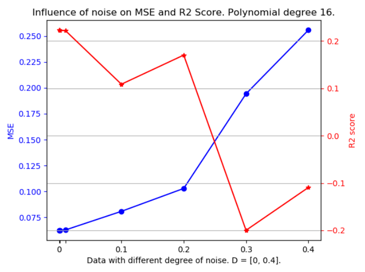
(c) MSE and R2 for degree 5.



(d) MSE and R2 for degree 10.



(e) MSE and R2 for degree 12.



(f) MSE and R2 for degree 16.

Figure 1: MSE and R^2-score for data with stochastic noise of degree, $D = [0, 0.4]$ fitted by polynomial models with degree 2, 3, 5, 10, 12 and 16.

The first method investigated was OLS. We needed to find a suitable complexity for the model, thus we studied the fit of models with varying polynomial degrees between 2 and 20 with different stochastic noise, $D \in [0, 0.4]$. Figure 1 illustrates the effect of increased degree of stochastic noise plotted against the MSE and R^2-score for some selected models with varying polynomial degrees.

The MSE curves for models with polynomial degree from 2 up to 10 are somewhat similar; they grow almost exponentially as the noise increases. For higher polynomial degrees, from 12 to 16, the curve starts off rising exponentially for smaller degrees of noise, and then displays linear disturbances for larger noise. As expected, noise disturbs the learning process of the model and thus affects performance. Comparing the R^2-score curve of models with polynomial degrees 2 to 10 we see a linear drop as more noise is introduced. The disturbance in the curve was first seen in models with polynomial degree 11 and higher. As for the model with degree 16 we observe that the R^2-score score curve erupts for D = 0.2 and then again D = 0.4. This is likely a result of overfitting - in these models, noise is considered a feature of the data.

Studying different models we see that the MSE decreases drastically with increased polynomial degree. This is noticeable even in models with second and third polynomial degrees. As seen in figure 2, the values for MSE and R^2-score start to flatten out around polynomial degree 5. From this figure, we were able to get a good visual confirmation that there was no need to use a more higher polynomial degree than 5 to obtain a model that provides a good explanation for our data set.

The R^2-scores approach a value of 1 for a higher degree polynomial model. As noise is present in the data, this is likely due to overfitting. With a limited amount of training data, it is usually a better choice to go with a simpler model; simply because it requires less data to train a simple model than a complex one. [3]

β	Value	Interval
β_0	0.580335	[0.577083, 0.583586]
β_1	5.130898	[5.095107, 5.166688]
β_2	2.985933	[2.948834, 3.023031]
β_3	-21.552724	[-21.723636, -21.381811]
β_4	-7.425274	[-7.558579, -7.291969]
β_5	-8.579148	[-8.755066, -8.403230]
β_6	22.037899	[21.659317, 22.416481]
β_7	28.672446	[28.393282, 28.951611]
β_8	7.654489	[7.366452, 7.942527]
β_9	-2.532593	[-2.922809, -2.142378]
β_{10}	0.125837	[-0.266709, 0.518383]
β_{11}	-36.871729	[-37.176799, -36.566660]
β_{12}	2.726144	[2.439877, 3.012410]
β_{13}	-18.007491	[-18.310734, -17.704248]
β_{14}	19.271342	[18.866642, 19.676043]
β_{15}	-6.190463	[-6.345071, -6.035856]
β_{16}	11.637605	[11.495853, 11.779356]
β_{17}	8.788225	[8.653327, 8.923123]
β_{18}	-10.051728	[-10.188907, -9.914549]
β_{19}	13.138633	[13.004816, 13.272450]
β_{20}	-11.558098	[-11.716012, -11.400185]

Table 1: β-values and their confidence intervals calculated using OLS for Franke's function with noise degree $D = 0.1$.

The model of polynomial degree 5 seem to have a reasonably good MSE and

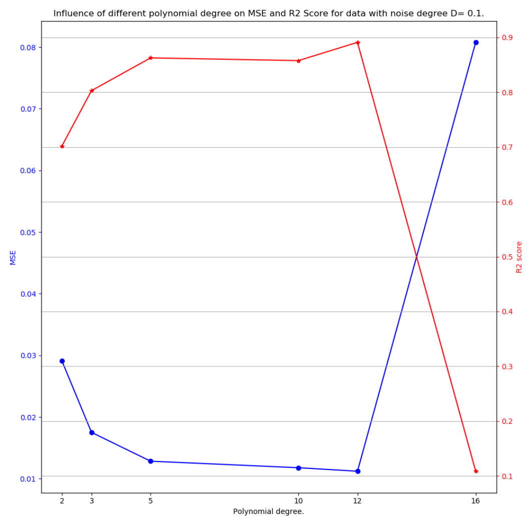


Figure 2: MSE and R^2-score for data with noise degree D $= 0.1$ for models with polynomial fit of degree 2, 3, 4, 5, 10, 12 and 16.

MSE	0.01294
R^2-score	0.84086
Bias	0.01257
Variance	0.00036

Table 2: Average MSE, R^2, bias and variance for the OLS model of polynomial degree 5 and noise with degree, D $= 0.1$. The bootstrap algorithm implemented with $n = 100$ re-samples.

R^2-score, while still being more stable than some of the higher order models. We will continue using this polynomial degree when analyzing the two other methods. Using OLS with polynomial degree 5 and the bootstrap algorithm with 20% of the data set aside for testing and 100 rounds of re-sampling iterations for training data, we calculated the bias and the variance of this model (see table 2, which seems to be decently low. We also calculated the confidence intervals for all the β-coefficients (see table 1).

For Ridge regression, on the other hand, it was necessary to analyze the accuracy of several models calculated using various logarithmic λ-values. The results are presented in figure 3.

The best fitting λ-value seems to be in a range of 10^{-4}. Lambda below 10^{-4} have no noticeable impact on the accuracy of the model, while larger values dramatically decreases the accuracy. We also investigated how the degree of stochastic noise influenced the model. Plotting the MSE against the strength of the stochastic noise for the three most promising λ-values, we can see that the MSE steadily increases with increasing noise. We observe some minor deviations between the three λ-s. As the strength of the noise increases, the MSE becomes more unstable - a smaller λ-value will no longer necessarily produce an satisfactory small MSE. Examining figure 3 and figure 4, $\lambda = 10^{-4}$ seems to be

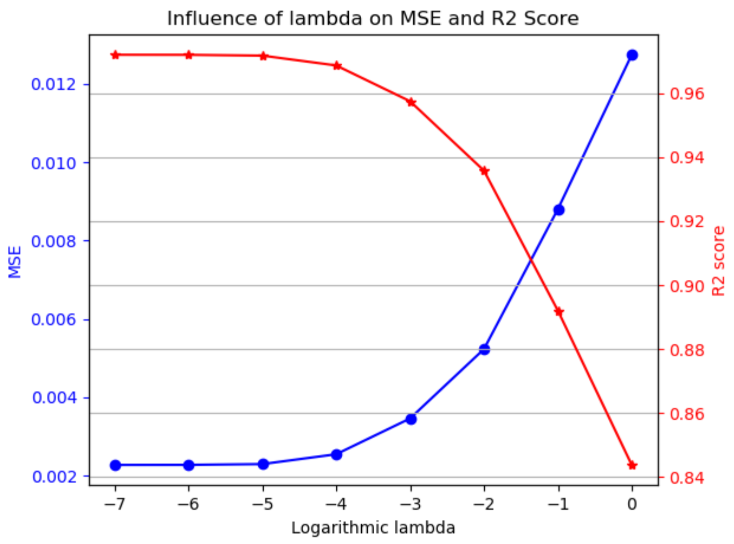


Figure 3: Mean Square Errors and R^2-score plotted vs the logarithmic λ.

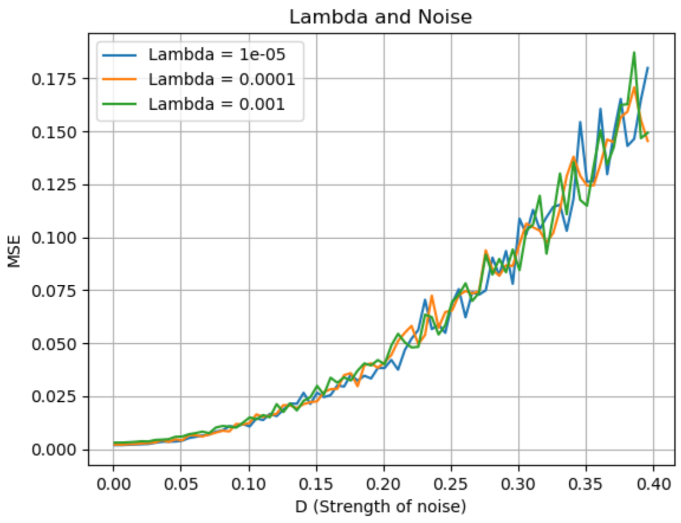


Figure 4: Mean Square Errors plotted vs D, a positive constant determining the level of noise in Franke function.

an appropriate choice for our further studies. This model was used to calculate β-s together with confidence interval, which can be seen in table 3.

Finally, we used the bootstrap algorithm to estimate MSE, R^2-score, variance and bias. The result is displayed in table 4.

From the R^2-score for Ridge regression with noise degree $D = 0.1$, polynomial degree 5 and $\lambda = 10^{-4}$, we can see that this model is somewhat better

β	Value	Interval
β_0	0.571799	[0.569002, 0.574597]
β_1	4.856842	[4.830227, 4.883458]
β_2	3.144542	[3.117835, 3.171248]
β_3	-20.737251	[-20.858250, -20.616251]
β_4	-6.844436	[-6.945798, -6.743075]
β_5	-9.815735	[-9.935063, -9.696408]
β_6	21.557576	[21.288146, 21.827006]
β_7	23.316729	[23.098794, 23.534664]
β_8	10.370145	[10.152637, 10.587652]
β_9	0.752966	[0.486935, 1.018997]
β_{10}	0.215224	[-0.071634, 0.502082]
β_{11}	-28.137927	[-28.382726, -27.893128]
β_{12}	-0.244428	[-0.471688, -0.017167]
β_{13}	-19.828444	[-20.072827, -19.584061]
β_{14}	15.277838	[14.991775, 15.563900]
β_{15}	-6.585491	[-6.702140, -6.468841]
β_{16}	9.109862	[8.991593, 9.228131]
β_{17}	5.773429	[5.652329, 5.894529]
β_{18}	-5.153574	[-5.273127, -5.034022]
β_{19}	12.021091	[11.903473, 12.138708]
β_{20}	-9.672372	[-9.789678, -9.555066]

Table 3: β-values calculated using Ridge regression with $\lambda = 10^{-4}$ for Franke's function with noise degree $D = 0.1$.

MSE	0.01193
R^2-score	0.86922
Bias	0.01158
Variance	0.00035

Table 4: MSE, R^2-score, bias and various calculated using Ridge Regression with bootstrap algorithm, resampling the data 100 times.

than the OLS model for the same polynomial degree. At this point, we would prefer to fit our data with Ridge regression rather than OLS.

The final model we investigated was Lasso. First, we needed to find a suitable α-value.

MSE	0.01550
R^2-score	0.83434
Bias	0.01521
Variance	0.00029

Table 5: MSE, bias and various calculated using the Lasso and the bootstrap algorithm with $n = 100$ resamples.

Figure 5 clearly shows that there is a clear reduction in the accuracy of the model as the α increases from 10^{-5} to 10^{-4}. Similar to our analysis with

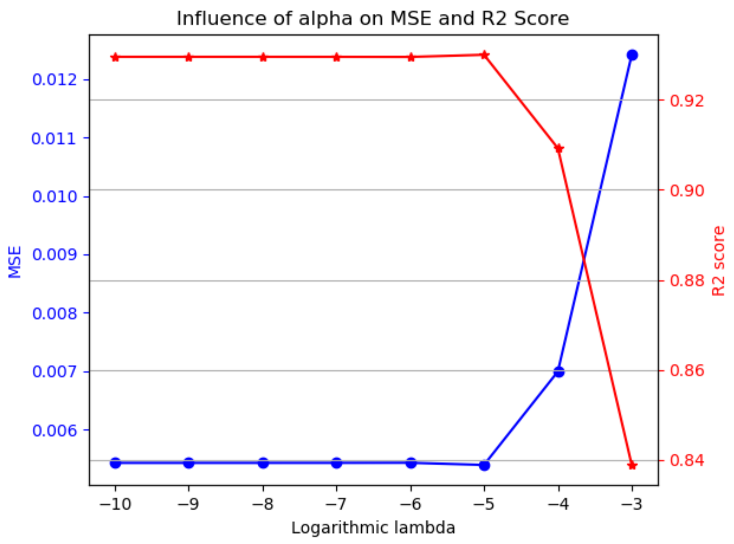


Figure 5: Mean Square Errors and R^2-score plotted vs the logarithmic α.

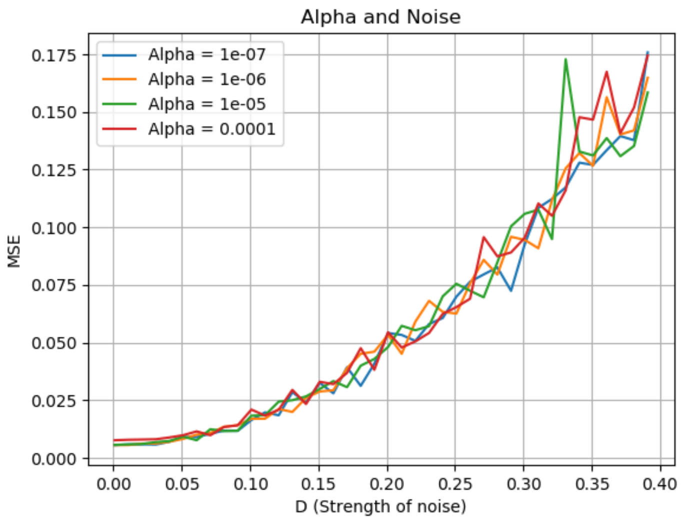


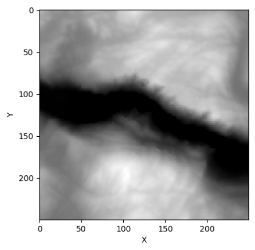
Figure 6: Mean Square Errors for various α-values vs the degree of stochastic noise D.

Ridge regression, there are large variations in the MSEs for all α-s as the degree of noise increases. Based on this, we chose $\alpha = 10^{-5}$ in our final model. In the same manner, the MSE, R^2-score, bias and variance of the final model was derived by the use of the already introduced bootstrap algorithm.

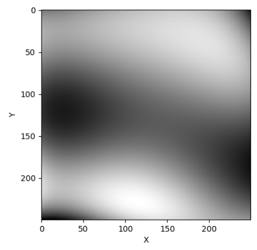
In table 5, we can see that the performance of Lasso model was slightly worse than the models produced by OLS and the Ridge regression methods.

We argue that this might be due to the `max_iter`-parameter. Increasing the number of iterations might have produced a better result. However, this will also be computationally expensive, especially when using the bootstrap algorithm with many re-sampling rounds.

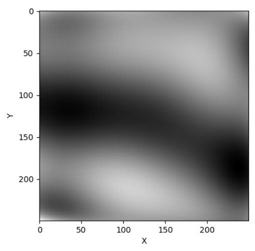
4.2 Terrain data



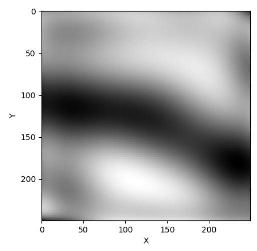
(a) Original terrain data



(b) OLS model, degree=5



(c) OLS model, degree=7



(d) OLS model, degree=9

Figure 7: Original terrain data from Rjukan and the predicted models of the same data fit by OLS method.

Figure 7 illustrates the original data set and compares it to some of the OLS models with polynomial degrees 5, 7 and 9. As expected, small-scale details are lost and smoothed out in our models. However, the shape and dimensions of the deep valley of Rjukan, the main feature of this data set, is preserved in our models.

Degree	5	6	7	8	9
MSE	18420.6274	11577.34248	9885.85454	6400.56027	5690.38056
R^2-score	0.6990	0.81451	0.84025	0.89591	0.90939
Bias	18413.6247	11570.34652	9878.29669	6394.14613	5312.43266
Variance	7.0027	6.99596	7.55786	6.41415	377.94790

Table 6: MSE,R^2 -score, bias and variance calculated using OLS and the bootstrap algorithm with $n = 100$ re-samples.

The MSE, R^2-score, variance and bias for increasingly complex OLS models for the terrain data is presented in table 6. It seems that a polynomial $degree =$ 5 is no longer sufficient as there is a sharp increase in R^2-score from $degree = 5$

to *degree* $= 6$. As the complexity increases, the MSE and R^2-score seems to stabilize. The difference between the R^2-score for a model with *degree* $= 8$ and the model with *degree* $= 9$ is only about 0.01. As the variance increases drastically between *degree* $= 8$ and *degree* $= 9$, it seems that a polynomial *degree* $= 8$ is a good compromise between underfitting and overfitting.

In the following analysis using the Ridge regression and the Lasso, we continued using this level of complexity.

As the scale of the data is larger now than when we used Franke's function with input variables between 0 and 1, it is reasonable to look for a larger λ-value to perform Ridge regression analysis on the terrain data. We calculate the MSE, R^2-score, bias and variance for λ-values between 10^{-2} and 10^6.

log(λ)	MSE	R2	Bias	Variance
-2	6293.421900	0.897197	6286.519504	6.902396
-1	6325.990890	0.897937	6319.453227	6.537663
1	6311.103615	0.897007	6304.786463	6.317152
2	6576.030716	0.895841	6569.631100	6.399616
3	6885.034802	0.888714	6878.113715	6.921086
4	6995.749995	0.887709	6988.856209	6.893786
5	8246.653905	0.864878	8240.009614	6.644291
6	8813.738344	0.856626	8806.501578	7.236766

Table 7: MSE, R^2 -score, bias and variance calculated using OLS and the bootstrap algorithm with $n = 100$ resamples.

The varying λ-values seem to have little impact on the performance of the model (see table 8). We have to increase λ to 10^3 in order to see small change in the R^2-score. The change finally becomes significant around $\lambda = 10^5$. Taking our previous analysis of Franke's function (figure 4) into consideration, it seems that the choice of λ is less significant when large amounts of noise is present. As the landscape around Rjukan is hardly flat (i.e. there is a high degree of "noise"), it is not unreasonable that the behaviour of the models are somewhat unpredictable, similar to figure 4 at noise degree $D = 0.20$ and above.

However, Ridge regression does produce a better result than OLS for λ-values between 10^{-2} and 10^1. This was expected from our analysis of Franke's function.

For Lasso, we calculate MSE, R^2-score, bias and variance of α-values between 10^{-6} and 10^1.

Over all, Lasso was not only significantly slower than the two previous methods (as mentioned above, we had to decrease the number of re-sampling rounds in order to run the bootstrap algorithm), it also produced worse results. This can most likely results of adjustment of the `max_iter` parameter in sklearn's lasso function. The function's objective is to minimize equation 12 and with a lower number of iterations the convergence will be poor.

log(α)	MSE	R2	Bias	Variance
-6	20381.291118	0.674222	20376.461456	4.829662
-5	19880.195211	0.673421	19875.258833	4.936377
-4	20375.741351	0.674470	20370.137752	5.603599
-3	20515.513774	0.674209	20510.341929	5.171845
-2	20524.769098	0.666755	20519.964360	4.804738
-1	20384.090086	0.677750	20379.163353	4.926733
0	20090.388209	0.675932	20086.629132	3.759076
1	21907.431639	0.652828	21900.789804	6.641835

Table 8: MSE, R^2, bias and variance calculated using OLS and the bootstrap algorithm with $n = 10$ resamples.

5 Conclusions and perspectives

The objective of this project was to study and determine performance value of three different polynomial regression methods - Ordinary Least Square, Ridge regression and Lasso. The fit of each model from these methods were evaluated using MSE, R^2-score, bias and variance. The bootstrap algorithm allowed us to estimate these metrics even for limited data sets.

Plots of MSEs and R^2-scores revealed that models with larger polynomial degrees fit better; however, they were more unstable. The model that best explained our Franke's function data had polynomial degree 5, while for our larger scale real terrain data set we preferred to fit our data to a model with polynomial degree 8.

Ridge regression produced the most interpretive model for both Franke's function and digital terrain data. Supported by the theoretical and experimental reasoning, this was an expected outcome, as Ridge regression is a more superior in separating noise from the real data trends. There are some details that could be considered in further studies. Firstly, we used random numbers to add noise to Franke's function and to generate some data sets for testing, so there are some fluctuations in the obtained MSE and R^2-score. A better option would probably have been to use a specific seed to generate the same pseudo-random numbers each time the function was called. Another field of investigation could be to apply other approximations than polynomial; for instance trigonometric function approximations. We could also have tried a different re-sampling technique, for example k-fold cross-validation, in order to estimate the error metrics for each model.

6 Appendix

6.1 Mathematical Formulas

6.1.1 Mean Squared Error

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

$\hat{y}$ is the true value, $\tilde{\hat{y}}$ is the predicted values.

6.1.2 R^2-score

$$R^2(\hat{y}, \tilde{\hat{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{14}$$

$\hat{y}$ is the true value, $\tilde{\hat{y}}$ is the predicted values. $\bar{y}$ is the mean value of the true values $\hat{y}$.

6.1.3 β-coefficients confidence interval

$$\beta_i \pm 1.645\sqrt{\frac{\hat{\sigma}_{ii}^2}{N}} \tag{15}$$

where

$$\hat{\sigma}^2 = \frac{\sum_{i=0}^{N-1}(\hat{y}_i - \tilde{\hat{y}}_i)^2}{N}(\hat{X}^T\hat{X})^{-1} \tag{16}$$

or

$$\hat{\sigma}^2 = \frac{\sum_{i=0}^{N-1}(\hat{y}_i - \tilde{\hat{y}}_i)^2}{N}(\hat{X}^T\hat{X} + \lambda I)^{-1} \tag{17}$$

The square root of $\hat{\sigma}^2$ in equation 16 is the estimated standard error for the β-coefficients calculated sing OLS. It is used in equation 15 to construct a 90% confidence interval. Similarly, one can estimate the standard error for Ridge regression using equation 17.

6.1.4 Bias

$$bias = \frac{\sum_{i=0}^{N-1}(y_i - \bar{\tilde{y}}_i)^2}{N} \tag{18}$$

where $\bar{\tilde{y}}_i$ will be the average predicted value of y at point i.

6.1.5 Variance

$$bias = \frac{\sum_{i=0}^{N-1}(\tilde{y}_i - \bar{\tilde{y}}_i)^2}{N} \tag{19}$$

where $\bar{\tilde{y}}_i$ will be the average predicted value of y at point i.

6.2 Code examples

6.2.1 OLS

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

def ols(x, y, z, degree = 5):
    # Estimate function z using OLS
    xyb_ = np.c_[x, y]
    poly = PolynomialFeatures(degree)
    xyb = poly.fit_transform(xyb_)
    beta = np.linalg.inv(xyb.T.dot(xyb)).dot(xyb.T).dot(z)

    return beta
```

6.2.2 Ridge Regression

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures

def RidgeRegression(x, y, z, degree=5, l=0.0001):
    # Calculate matrix with x, y - polynomials
    M_ = np.c_[x, y]
    poly = PolynomialFeatures(degree)
    M = poly.fit_transform(M_)

    # Calculate beta
    A = np.arange(1, degree + 2)
    rows = np.sum(A)
    beta = (np.linalg.inv(M.T.dot(M) +
        l * np.identity(rows))).dot(M.T).dot(z)

    return beta
```

6.2.3 β-coefficients 90% confidence interval

Confidence interval for OLS.

```
def betaConfidenceInterval_OLS(z_real, beta, X):
    # Calculate variance squared in the error
    z_hat = X.dot(beta)
    N, P = np.shape(X)
    sigma_2 = (np.sum(np.power((z_real-z_hat), 2)))/N

    # Calculate the variance squared of the beta coefficients
    var_beta = np.diag(sigma_2*np.linalg.inv((X.T.dot(X))))

    # The square root of var_beta is the standard error. Use it to
                                  calculate confidence
                                  intervals
    i_minus = beta - 1.645*np.sqrt(var_beta/N)
    i_plus = beta + 1.645*np.sqrt(var_beta/N)

    return i_minus, i_plus
```

Confidence interval for Ridge regression. Note that the variable `l` represents the shrinking parameter λ.

```
def betaConfidenceInterval_Ridge(z_real, beta, X, l):
    # Calculate variance squared in the error
    z_hat = X.dot(beta)
    N, P = np.shape(X)
    sigma_2 = (np.sum(np.power((z_real-z_hat), 2)))/N

    # Calculate the variance squared of the beta coefficients
    XTX= X.T.dot(X)
    R, R = np.shape(XTX)
    var_beta = np.diag(sigma_2*np.linalg.inv((XTX + l*np.identity(R
                                  ))))

    # The square root of var_beta is the standard error. Use it to
                                  calculate confidence
                                  intervals
    i_minus = beta - 1.645*np.sqrt(var_beta/N)
    i_plus = beta + 1.645*np.sqrt(var_beta/N)
```

```python
    return i_minus, i_plus
```

6.2.4 The Bootstrap algorithm

```python
import numpy as np
from sklearn.utils import resample
from sklearn.preprocessing import PolynomialFeatures
from RidgeRegression import RidgeRegression
from Lasso import Lasso
from OrdinaryLeastSquares import ols
from Analysis import R2

def bootstrap(x, y, z, p_degree, method, n_bootstrap=100):
    # Randomly shuffle data
    data_set = np.c_[x, y, z]
    np.random.shuffle(data_set)
    set_size = round(len(x)/5)

    # Extract 20\% of data as test-set, never used in training.
    x_test = data_set[0:set_size, 0]
    y_test = data_set[0:set_size, 1]
    z_test = data_set[0:set_size, 2]
    test_indices = np.linspace(0, set_size-1, set_size)

    # And define the training set as the rest of the data
    x_train = np.delete(data_set[:, 0], test_indices)
    y_train = np.delete(data_set[:, 1], test_indices)
    z_train = np.delete(data_set[:, 2], test_indices)

    Z_predict = []

    MSE = []
    R2s = []
    for i in range(n_bootstrap):
        x_, y_, z_ = resample(x_train, y_train, z_train)

        if method == 'Ridge':
            # Ridge regression, save beta values
            beta = RidgeRegression(x_, y_, z_, degree=p_degree)
        elif method == 'Lasso':
            beta = Lasso(x_, y_, z_, degree=p_degree)
        elif method == 'OLS':
            beta = ols(x_, y_, z_, degree=p_degree)
        else:
            print('ERROR: Cannot recognize method')
            return 0

        M_ = np.c_[x_test, y_test]
        poly = PolynomialFeatures(p_degree)
        M = poly.fit_transform(M_)
        z_hat = M.dot(beta)

        Z_predict.append(z_hat)

        # Calculate MSE
        MSE.append(np.mean((z_test - z_hat)**2))
        R2s.append(R2(z_test, z_hat))

    # Calculate MSE, Bias and Variance
    MSE_M = np.mean(MSE)
    R2_M = np.mean(R2s)
    bias = np.mean((z_test - np.mean(Z_predict, axis=0, keepdims=
                                        True))**2)
    variance = np.mean (np.var(Z_predict, axis=0, keepdims=True))
    return MSE_M, R2_M, bias, variance
```

Bibliography

[1] Hastie, T., Tibshirani, R., Friedman, J. (2009) p. 242-243 *The Elements of Statistical Learning.* Springer Media

[2] Geng, D., Shih, S. (2017) *Machine Learning Crash Course: Part 4 - The Bias-Variance Dilemma.*

`https://ml.berkeley.edu/blog/2017/07/13/tutorial-4/`

[3] Mehta, P., Wang, C., Day, A. G. R., and Richardson, C. *A high-bias, low-variance introduction to Machine Learning for physicists*

`https://arxiv.org/pdf/1803.08823.pdf`

[4] Murphy, Kevin P. (2012)., *Machine Learning: A Probabilistic Perspective.* The MIT Press