Project 1
FYS-STK4155

Mina Tangen & Tham Le (for code: see github.com/minatang1/project1)

October 7, 2019

Abstract

This report looks at the three different regression methods; Ordinary Least Squares (OLS), Ridge and LASSO. First, the three methods are used to fit Franke's Function using polynomials of degree up to $n = 20$ (in different methods). The methods are evaluted using the Mean Squared Error (MSE) and R2 scores. Confidence intervals are produced for the regression estimates. The bootstrap method was implemented as resampling. Then, the same methods are used to fit real terrain data.

We found that the OLS method gave the best results. The polynomial of degree 7 gave the best result for Franke's function using OLS and degree 5 for Ridge after bootstrapping. OLS performed best overall at Franke's function without noise and terrain data. MSE scores were difficult to interpret for the terrain data.

1 Introduction

Linear regression methods is an important part in Machine Learning, fitting the models to the data by modeling a dependent response by predictor values. In modern society it has been more common to see research projects with big and complexed data sets to extract patterns. Making good predicted models with small errors is one of the biggest challenges we face today. And in a world full of different algorithms, it's hard to choose the right one based on the complexity of the data set.

In this project we are using regression to fit a bivariate function, the so-called Franke's function (equation 7 given in the appendix). A plot is shown in figure 1. We then want to use the same methods to fit a surface like the one shown figure 1

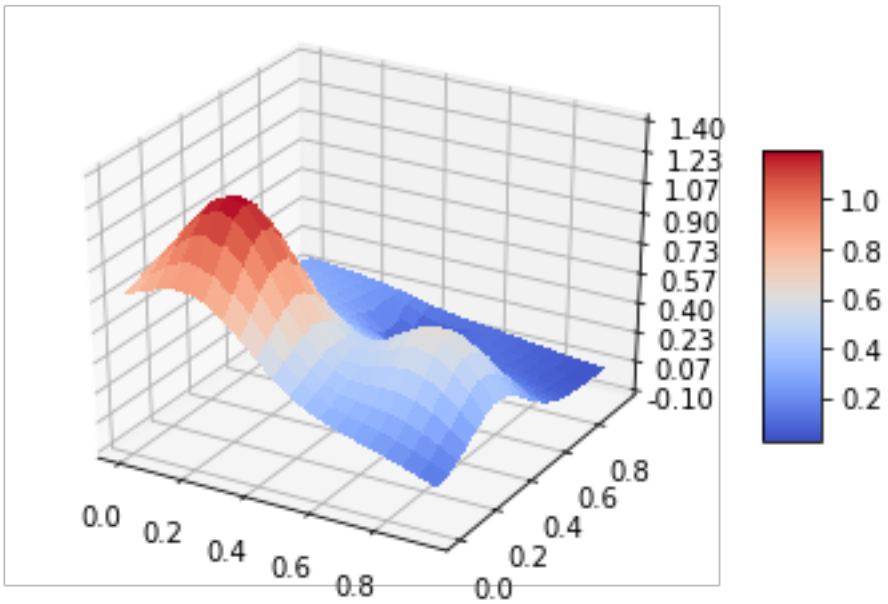


Figure 1: Franke's function

just that this surface is based on real terrain data. We will use bivariate polynomials of degree up to $n = 5$ and use evaluation metrics to evaluate the models. We will also look at how the evaluation metrics changes with differences in the noise we add to our dataset and differences in the hyperparameters that Ridge and LASSO depends on (as described later). The questions we want to answer is thus, which polynomial degree gives the best fit for Franke's function and the terrain data, and what are the optimal choices for the hyperparameters?

The first section describes the methods and theoretical basis for the algorithms used in the project. Here, we also discuss the evaluation metrics and the bias-variance trade-off. The next section presents the results, followed by a discussion of these results. We reach a conclusion in the last section. Additional equations and code examples are presented in the Appendix.

2 Methods

2.1 Regression methods

Regression is comprised of three elements (Mehta et al. (2019)), namely

- ***data***
- ***model***
- ***cost function***

The data consists of input values (x_i, y_i) and corresponding output variables z_i. In our case, (x_i, y_i) represents the coordinates of the plane and z_i represents the heigth. For Franke's function, (x_i, y_i) are simply random values that we feed the function to generate z_i values. For the terrain data, we are given values for (x_i, yi) and the z_i that corresponds to each pair of plane coordinates.

Gievn an input vector X^T ($X^T = (X_1, \ldots, X_p)$ and $X_j = (x_i, yi)$) and output vector Y, we want to make a prediction, $\hat{Y}$ of Y. Our regression ***model*** is defined as (from Hastie et al. (2017))

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j + \epsilon \tag{1}$$

ϵ describes the error in our datapoints (simulated in Franke's function by adding noise). Equation 1 can again be written as

$$\hat{Y} = X^T \hat{\beta} + \epsilon$$

Since we have multiple input vectors X, we use matrix notation to find:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

The matrix $\mathbf{X}$ is called a Vandermonde matrix (or design matrix) (Faul, A. C. (2016)). The setup of this matrix is shown in the Appendix. The Vandermonde matrix will have p rows (as the input and output vectors contains p elements) and n columns, depending of the degree of the polynomial we choose. p is the number of features and represents the compexity of our model. The complexity of our model is in therefore this case determined by the degree of the polynomials. n is the number of observations and represents the sample size.

We now want to fit a linear model to our data. In this project, we have used three different regression methods: ordinary least squares, ridge regression and LASSO regression. We define a ***cost function***. The cost function looks at the difference between the data and what the model predicts. We want to find the parameters $\hat{\beta}$ that minimize this function. The three regression methods differ in that they have different estimates for $\hat{\beta}$.

2.1.1 Ordinary Least Squares (OLS)

The first regression method we used is the Ordinary Least Squares (OLS) method.

In the OLS method we want to minimize the residual sum of squares (Hastie et. al (2017)):

$$\text{Residual Sum of Squares}(\beta) = \frac{1}{n} \sum_{i=1}^{n-1} (z_i - \bar{z}_i)^2 \tag{2}$$

where z_i is the real height given and $\bar{z}_i = f(x_i, y_i)$ is the estimation of that height. This is our cost function.

By minimizing the residual sum of squares, we mean that we want to find values for β that gives the smallest possible value for equation 2. Applying equation 2 to our model given above, we find that we want to minimize the following:

$$\text{Residual Sum of Squares}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

We can rewrite this to find

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0 \tag{3}$$

This equation can be solved for β (and denote $\hat{\beta}^{OLS}$):

$$\hat{\beta}^{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{4}$$

Equation 3 will have a uniquie solution (given in equation 4) if $\mathbf{X}^T\mathbf{X}$ is non-singular (i.e. invertible). The solution vector $\hat{\beta}^{OLS}$ has dimension $\mathbb{R}^p$.

As just explained, we have to assume that $\mathbf{X}^T\mathbf{X}$ is non-singular. This is often true when $n >> p$, i.e., when the number of observations is much larger the the number of features. However, in other cases, $\mathbf{X}^T\mathbf{X}$ might not be invertible, and there is no unique solution to equation 3. There are many decomposition algorithms that can help in computing an inverse to reach a solution, like the Singular Value Decomposition (SVD) and LU factorization (Faul (2017)). Another approach to tackling this issue is to add a regularization parameter, which can make $\mathbf{X}^T\mathbf{X}$ singular. The following two methods introduces such regularization parameters.

2.1.2 Ridge regression

In ridge regression, we add a regularization penalty (λ) to the size of $\hat{\beta}$. This is a way of "shrinking" the coefficients (Hastie et al. (2017)).

We find an expression similair to that of the OLS, but with the penalty λ added:

$$\hat{\beta^{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

The greater the penalty, the greater we shrink $\hat{\beta}$. We note that if $\lambda = 0$ then $\hat{\beta}^{ridge}$ equals the regression estimate from the OLS method (equation 4).

As explained above, adding a regularization parameter can solve inverse problems we might encounter using the OLS method. Introducing a regularization parameter also helps with cases of poor generalization, which means cases in which the model cannot generalize to other situations than it has been trained for in the training set. This happens in cases where $p >> n$, i.e. the number of features, p (model complexity) exceeds the number of observations, n (sample size) (Mehta et al. 2019). How model complexity and sample size affects the performance of our model is discussed later in the section *Bias-variance trade-off*.

2.1.3 LASSO regression

LASSO stands for "least absolute shrinkage and selection operator" (Mehta et al. (2019)). This method, like Ridge regression, uses a regularization parameter to shrink coefficients.

LASSO tends to make the coefficients zero ("sparse" solutions (Mahte et al. (2019)))

As for the other two methods, we find an expression for the estimator $\hat{\beta}^{LASSO}$:

$$\hat{\beta}^{LASSO} = \underset{\beta}{argmin} \left\{ \frac{1}{2} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{i,j}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{5}$$

Equation 5 has no analytical solution, we therefore use Scikit-learn to implement this part (see *Implementation*)

2.2 Resampling methods

2.2.1 Bootstrap

Resampling methods is an important statistical technique and a key tool in machine learning. One of the most common methods are the Bootstrap method. Bootstrap is an expression used on statistical methods, new datasets are simulated by the original data set. Bootstrap occurs by repeatedly drawing random sample set from the sample space with replacement to achieve robust estimates (Hastie et al, (2017)). The method is quite general and don't require a distribution assumptions.

The procedure of Bootstrap method follows:
Split the data into training and test sets
Set a number of samples n

1. Draw a sample set from the training set with replacement.
2. Compute and fit a model to the sample set.
3. Test the model on the test set.
4. Repeat this process n times

After the process, calculate the mean of the evaluation scores (chapter 2.3).

2.3 Model evaluation

We used two different evaluation metrics to evalute the models; R2 and MSE.

Mean Squared Error (MSE) taking the average of the square of the values, where the values is the differences between the observed values y_i and the predicted ones $\hat{y}_i$. The bigger the values the larger the error and 0 means that the regression model is perfect.

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

The R^2-score function measures the variance explained by the model divided by the total variance, and tell us how well the regression model fits the data by percentage from a scale 0-100 %.

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

where $\bar{y}$ is the mean of the values.

2.4 Bias-variance trade-off

As mentioned above, generalization denotes the models abilitiy to learn and generalize to other situations than the one it has been trained on. We can for instance have a model that overfits the data, which means that a complex model, at small sample sizes, is tricked to believe that noise represent real patterns in the data, rather than filtering it as noise (Mahte et al. (2019)).

Bias describes the part of the generalization error which exist due to wrong assumptions, such as assuming that the data is linear when it actually quadratic. A high-bias model i most likely to unfit the training data.
Variance is this part of the generalization error due to the model's excessive sensitivity to small variations is in the training data. A model with many degrees of freedom (such as a high-degree polynomial model) is likely to have high variance, and thus to overfit the training data.

Increasing a model's complexity will typically increases its variance and reduce its bias. Conversely, reducing a model's complexity increases its bias and reduces its variance. This is why it is called a **bias-variance tradeoff**.

The predictive performance of the model therefore depends on the number of columns in the design matrix $\mathbf{X}$ and the amount of data we have. We must therefore either choose a less complex model or increase the sample size by collecting more data. Figure 8 in the *Results* section shows how the predicition error of our model depends on the complexity of the model.

Mathematically, we can look at it this way:
We know that the true data is generated from a noisy model

$$y^1 = f(x) + \boldsymbol{\varepsilon}$$

Our $\boldsymbol{\varepsilon}$ is normally distributed with mean zero and standard deviation σ^2. In our project the derivation of the ordinary least squares method (OLS) are defined as an approximation for the function f with the parameters $\boldsymbol{\beta}$ and the design matrix $\mathbf{X}$ as

$$\hat{y} = \mathbf{X}\boldsymbol{\beta}$$

We found our parameters $\boldsymbol{\beta}$ by optimizing the means squared error by the cost function

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

We can rewrite this (see Appendix) to find:

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

where

$$Bias[\hat{y}]^2 = \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{y}\right])^2$$

and

$$Var[\hat{y}] = \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{y}\right])^2,$$

i. e.,

$$= Bias[\hat{y}]^2 + \sigma^2 + Var[\hat{y}]$$

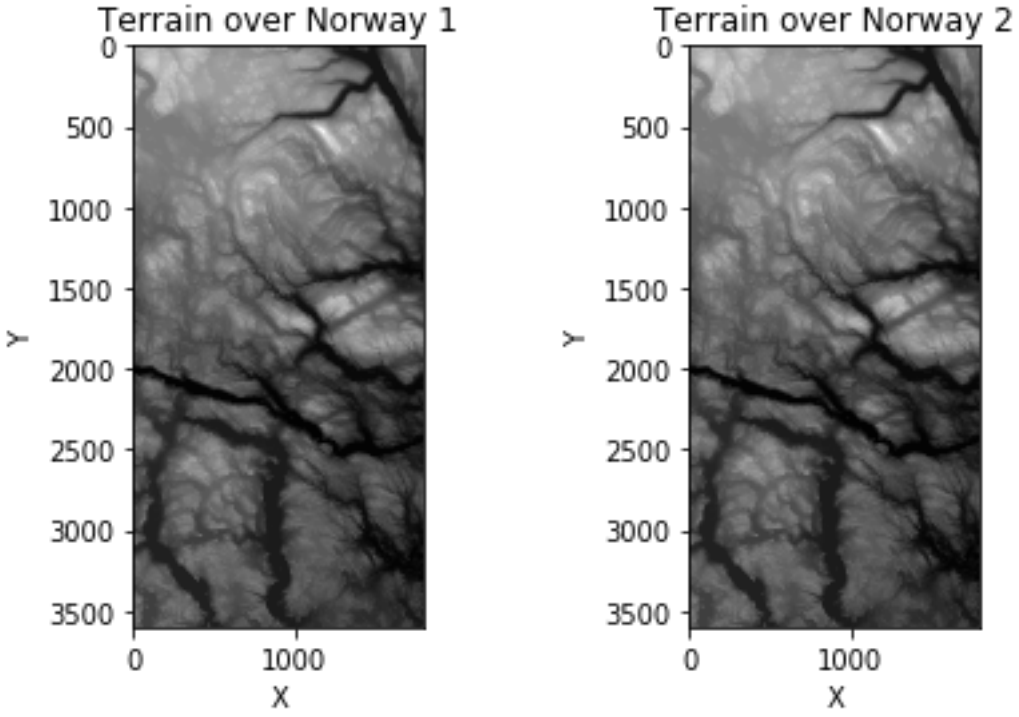
[1] Not to be confused with the input values y_i described above

2.5 Implementation and data

The dataset was generated by using Python's `arange` to simply produce a list of numbers (x, y) to use as input to Franke's function. After feeding these numbers to Franke's function, we add noise. The "amount" of noise added can simply be tuned by a scalar ν multiplied with the noise:

$$z(x, y) = f(x, y) + \nu \cdot \epsilon \tag{6}$$

The terrain data was extraced from the website https://earthexplorer.usgs.gov/[2]. We used randoms patches of the dataset and then took the mean of the statistics we found from those patches.



(a) Terrain 1

(b) Terrain 2

Figure 2: Terrain plots

The OLS and Ridge methods were implemented "manually", i.e. based on equation 4 and 2.1.2 respectively. The functions are shown in the Appendix. For LASSO, as noted, there is no analytical solution, so we used *scikit-learn's* functionality for this. The first two methods were also compared to the results generated from *scikit-learn*.

We implemented the bootstrap method. We used *scikit-learn's* to split our data into training and test data. Then we used the different methods on the training data, and used the set of β's we got on the test data, to se how well it performed.

[2]We used the first example file provided

3 Results and discussion

3.1 Generated data

(a) OLS fit 1

(b) OLS fit 2

Figure 3: OLS fits

Figure 3 shows the fit produced using the OLS method both on top (3a) and alone (3b).

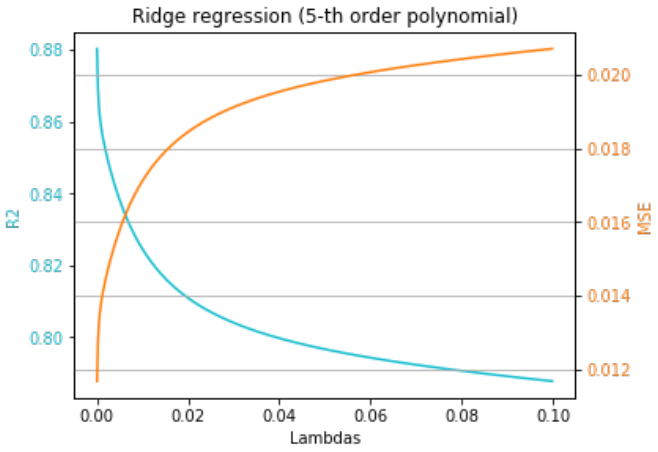


Figure 4: R^2 and MSE scores as a function of lambda (Ridge), $n = 5$

Tuning of lambda Figure 4 and **??** show how R2 and MSE scores change with the hyperparameter λ. We find an ideal value to be quite low, around 0.001 for both, and so used these values throughout, when looking at other aspects of the analysis. However, small values of λ for LASSO makes it behave strange, probably due to how *scikit-learn* uses this to calculate the learning rate and so the method might converge.

Confidence intervals for β Tables 1, 2 and 3 show the confidence intervals (made as described in Appendix) for β for the three different methods.

Note that the variances in tables 1, 2 and 3 are the same for the different β's. This is because the variance is found along the diagonal of $((\mathbf{X}^T\mathbf{X})^{-1}$. This matrix is called a correlation matrix. A correlation matrix is a m×m-table showing the correlation coefficients between to variables in each cell and the variances appears along the diagonal (Bock, T.). The covariance appears in

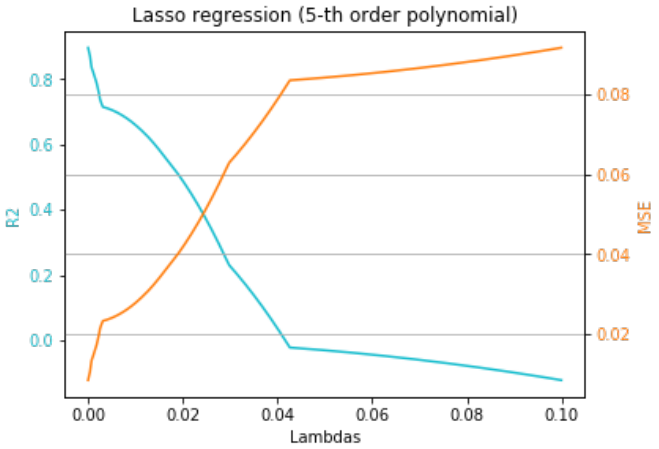


Figure 5: R^2 and MSE scores as a function of lambda (Ridge), $n = 5$

Table 1: Confidence interval (OLS)

β	Lower limit	Upper limit	Var
β_0	0.3278	0.7198	Var: 0.59371
β_1	4.83956	5.23156	Var: 7.55623
β_2	3.48951	3.88151	Var: 7.55623
β_3	-17.78303	-17.39103	Var: 40.34353
β_4	-8.43359	-8.04159	Var: 40.34353
β_5	-12.37218	-11.98018	Var: 30.64174
β_6	9.92037	10.31237	Var: 97.75542
β_7	-9.96499	-9.57299	Var: 97.75542
β_8	27.0471	27.4391	Var: 70.09944
β_9	22.73779	23.12979	Var: 70.09944
β_{10}	14.25747	14.64947	Var: 108.53728
β_{11}	30.45056	30.84256	Var: 108.53728
β_{12}	-29.64032	-29.24832	Var: 80.29257
β_{13}	-3.84756	-3.45556	Var: 73.83169
β_{14}	-33.00839	-32.61639	Var: 80.29257
β_{15}	-12.72548	-12.33348	Var: 44.83459
β_{16}	-16.76062	-16.36862	Var: 44.83459
β_{17}	8.86653	9.25853	Var: 37.83166
β_{18}	5.20763	5.59963	Var: 36.35623
β_{19}	-3.35257	-2.96057	Var: 36.35623
β_{20}	16.36135	16.75335	Var: 37.83166

the non-diagonal cells. The covariance tell us the measure of how linear independent there are between to values. The goal of the correlation matrix is to see the patterns in a large amount of data. So in our case, the diagonal elements represents the variance.

Table 2: Confidence interval (Ridge)

β	Lower limit	Upper limit	Var
β_0	0.52095	0.91295	Var: 0.59371
β_1	2.48263	2.87463	Var: 7.55623
β_2	2.09851	2.49051	Var: 7.55623
β_3	-10.76059	-10.36859	Var: 40.34353
β_4	-7.60571	-7.21371	Var: 40.34353
β_5	-1.75528	-1.36328	Var: 30.64174
β_6	6.0122	6.4042	Var: 97.75542
β_7	1.6983	2.0903	Var: 97.75542
β_8	3.461	3.853	Var: 70.09944
β_9	0.67427	1.06627	Var: 70.09944
β_{10}	6.90676	7.29876	Var: 108.53728
β_{11}	5.42815	5.82015	Var: 108.53728
β_{12}	0.22934	0.62134	Var: 80.29257
β_{13}	3.82003	4.21203	Var: 73.83169
β_{14}	-2.80012	-2.40812	Var: 80.29257
β_{15}	-6.28271	-5.89071	Var: 44.83459
β_{16}	-2.79464	-2.40264	Var: 44.83459
β_{17}	-3.52449	-3.13249	Var: 37.83166
β_{18}	0.05861	0.45061	Var: 36.35623
β_{19}	-3.298	-2.906	Var: 36.35623
β_{20}	0.95116	1.34316	Var: 37.83166

Note also in table **??** that a lot of the coefficients have confidence interval $[-1.96 - 1.96]$, which means that the coefficient is 0, as discussed above.

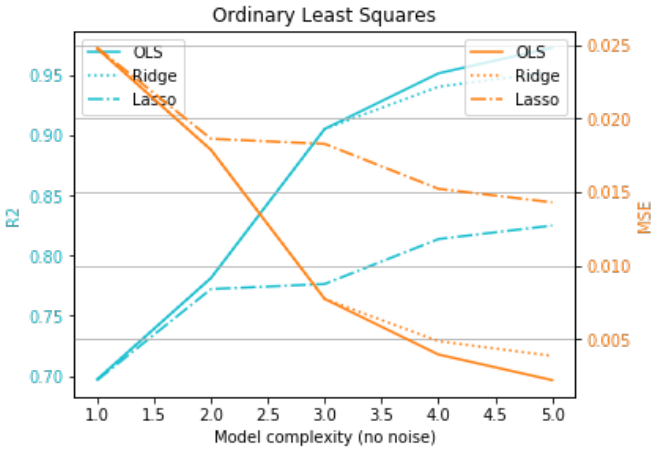


Figure 6: R^2 and MSE scores as a function of model complexity, $\nu = 0.1$

Table 3: Confidence interval (LASSO)

β	Lower limit	Upper limit	Var
β_0	0.79379	1.18579	Var: 0.59371
β_1	-0.70764	-0.31564	Var: 7.55623
β_2	-0.196	0.196	Var: 7.55623
β_3	-0.4639	-0.0719	Var: 40.34353
β_4	-1.50749	-1.11549	Var: 40.34353
β_5	-0.13789	0.25411	Var: 30.64174
β_6	-0.196	0.196	Var: 97.75542
β_7	-0.196	0.196	Var: 97.75542
β_8	0.28827	0.68027	Var: 70.09944
β_9	-0.196	0.196	Var: 70.09944
β_{10}	-0.196	0.196	Var: 108.53728
β_{11}	-0.196	0.196	Var: 108.53728
β_{12}	-0.14733	0.24467	Var: 80.29257
β_{13}	-0.196	0.196	Var: 73.83169
β_{14}	-0.196	0.196	Var: 80.29257
β_{15}	-0.196	0.196	Var: 44.83459
β_{16}	0.41754	0.80954	Var: 44.83459
β_{17}	-0.196	0.196	Var: 37.83166
β_{18}	-0.196	0.196	Var: 36.35623
β_{19}	-0.196	0.196	Var: 36.35623
β_{20}	-0.196	0.196	Var: 37.83166

Model complexity and noise Figure 6 shows

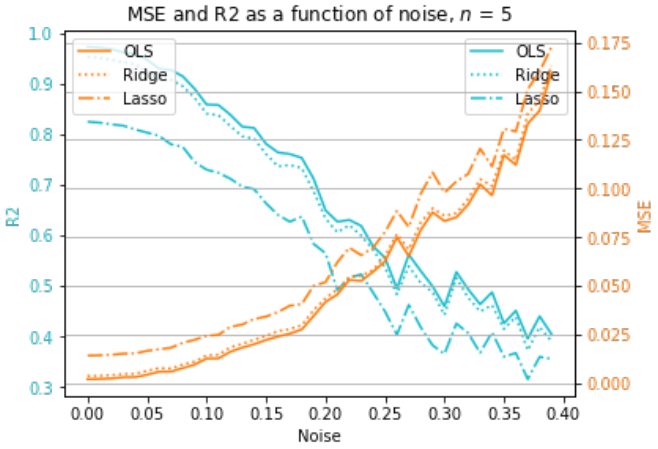


Figure 7: R^2 and MSE scores as a function of noise, $n = 5$

Figure 7

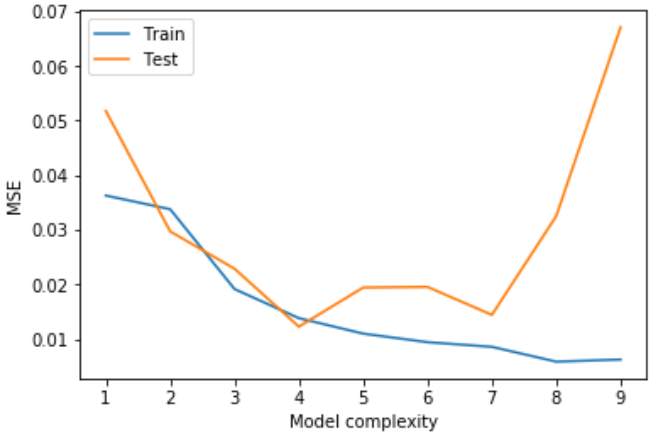


Figure 8: Bias-variance trade-off

Bias-variance trade-off Figure 8 shows the bias-variance trade-off situation described above. We see that as the model complexity increases, we get better estimates both for the training and test case. However, as the model gets too complex, the test error increases and the model is overfitting.

Table 4: OLS

Polynomial degree	R2	MSE
n = 1	0.6779	0.0266
n = 2	0.7491	0.0207
n = 3	0.8839	0.0096
n = 4	0.9248	0.0062
n = 5	0.9699	0.0025
n = 6	0.9797	0.0017
n = 7	0.9904	0.0008
n = 8	0.9890	0.0009
n = 9	0.9854	0.0012

Bootstrap Tables 4, 5 and 6 shows the R2 and MSE scores after 5000 bootstraps for different polynomial degrees. The same information is shown in figure 9

As we can see, polynomial of degree 7 performs the best for OLS, whereas degree 5 performs best for Ridge. For LASSO, it seems the scores converges somehow, which we found strange. We do not believe that degrees of polynomial 2 up to 9 performs equally. We conclude from this that OLS performs best (R2 = 0.9904 and MSE = 0.0008 vs. R2 = 0.9180 and MSE = 0.0068). OLS and Ridge has the same performance up until degree 3.

Bootstrap gave overall better results.

Table 5: Ridge regression

Polynomial degree	R2	MSE
n = 1	0.6778	0.0266
n = 2	0.7490	0.0208
n = 3	0.8783	0.0101
n = 4	0.8807	0.0099
n = 5	0.9699	0.0090
n = 6	0.9040	0.0079
n = 7	0.9130	0.0307
n = 8	0.9180	0.0068
n = 9	0.9213	0.0065

Table 6: LASSO regression

Polynomial degree	R2	MSE
n = 1	0.5974	0.0333
n = 2	0.6286	0.0307
n = 3	0.6286	0.0307
n = 4	0.6286	0.0307
n = 5	0.6286	0.0307
n = 6	0.6286	0.0307
n = 7	0.6286	0.0307
n = 8	0.6286	0.0307
n = 9	0.6286	0.0307

3.2 Terrain data

Tables 7, 8 and 9 show the mean of the R2 and MSE scores for 30 randomly chosen patches of the terrain 1 data. Lambda values for Ridge and LASSO were $\lambda = 0.001$ and $\lambda = 0.01$ respectively. As we can see, the MSE values are enormous. We suspect that this has more to do with our implementation than it is an actual measure of the method. We therefore look at the R2 scores for the following analysis: OLS performs best with polynomial degree 10. Ridge and Lasso perform best with degree 20, but still worse than OLS.

Problems We encountered numerous problems in this project, which of course affected the results. We were unable to fix all problems, as discussed in various parts above. We found implementation of MSE scores a bit difficult, as shown above. It is therefore difficult to say something meaningful for MSE scores for the terrain data.

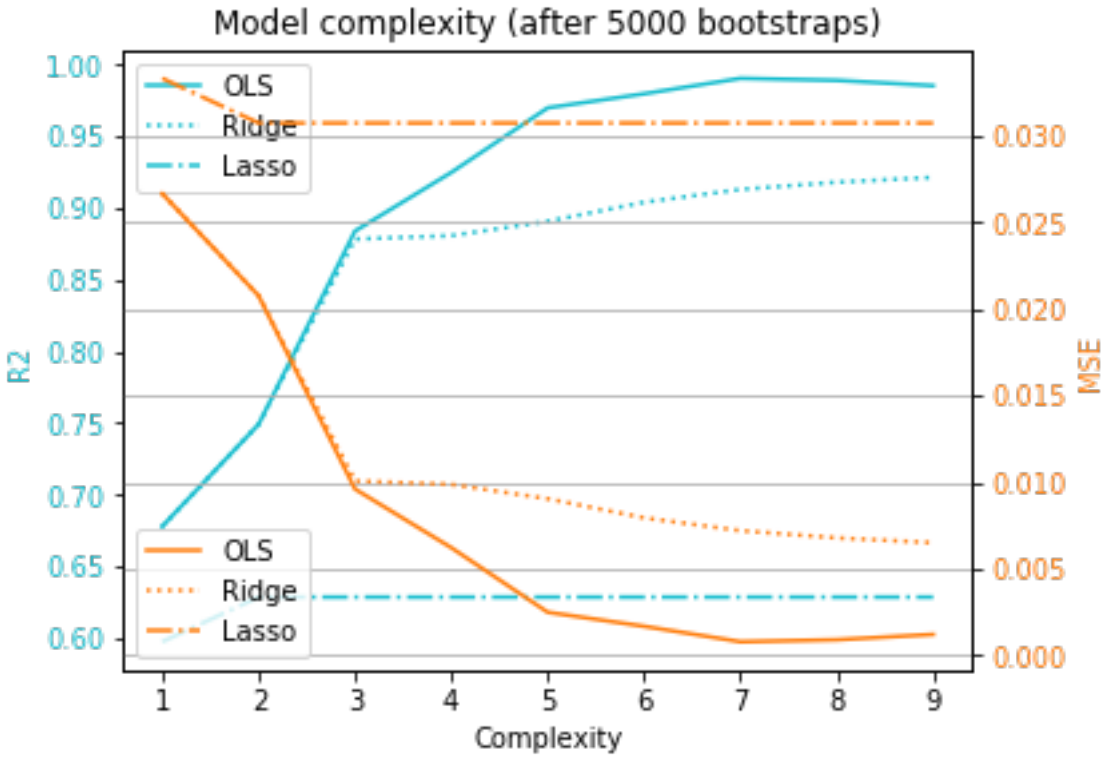


Figure 9: R2 and MSE as a function of model complexity after 5000 bootstraps

4 Conclusion

We conclude that OLS overall performed best for us. LASSO and it coefficients made it difficult for us to get proper results, due to various reasons. Resampling (through implementation of the bootstrap algortihm) gave better results, so we see that this is important. We could clearly see the expected bias-variance trade-off tendency as we expected, and so we will keep this fine balance between model complexity and dataset in mind in future projects involving machine learning algorithms.

5 Appendix

5.1 Mathematical functions

5.1.1 Franke's function

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

Table 7: Terrain data (OLS)

Polynomial degree	R2	MSE
n = 2	0.5877	3163.55
n = 5	0.9049	453.74
n = 10	0.9789	116.01
n = 15	-0.0684	2028831.16
n = 20	-0.018	4475979.74

Table 8: Terrain data (Ridge)

Polynomial degree	R2	MSE
n = 2	0.5877	3163.55
n = 5	0.8902	533.39
n = 10	0.9301	330.02
n = 15	0.9415	276.06
n = 20	0.9463	251.25

5.1.2 Bivariate polynomial

$$f(x,y) = \sum_{i,j} a_{i,j} x^i y^j \tag{8}$$

5.1.3 Vandermonde matrix

Each element of the first column of the Vandermonde matrix has the value 1. The other columns represent the different terms in a bivariate polynomial. That means that one column will correspond to the x^2y term, another will correspond to the xy^3 term and so on.

The Vandermonde matrix (or design matrix) of degree 3 is shown as an example:

5.1.4 Bias-variance trade-off

We found our parameters $\boldsymbol{\beta}$ by optimizing the means squared error by the cost function and want to show:

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

By deifinition, the variance is defined for any random variable X as

$$VAR[\mathbf{X}] = \mathbb{E}[\mathbf{X}^2] - (\mathbb{E}[\mathbf{X}])^2$$

We rearranging it and get

$$\mathbb{E}[\mathbf{X}^2] = VAR[\mathbf{X}] + (\mathbb{E}[\mathbf{X}])^2$$

Table 9: Terrain data (LASSO)

Polynomial degree	R2	MSE
n = 2	0.7595	167451322.22
n = 5	0.8644	176352105.81
n = 10	0.8900	177839653.61
n = 15	0.8950	178071263.96
n = 20	0.8968	178125605.48

Since f is determinstic, also entirely determined by its initial state and inputs and we know $y = f + \varepsilon$ and $\mathbb{E}[\varepsilon] = 0$. We can rewrite

$$\mathbb{E}[y] = \mathbb{E}[f + \varepsilon] = \mathbb{E}[f] = f$$

And since $Var[\varepsilon] = \sigma^2$,

$$Var[y] = \mathbb{E}[(y - \mathbb{E}[y])^2] = \mathbb{E}[(y - f)^2] = \mathbb{E}[(f + \varepsilon - f)^2]$$
$$= \mathbb{E}[\varepsilon^2] = Var[\varepsilon] + (E[\varepsilon])^2 = \sigma^2$$

Our ε and $\hat{y}$ are independent, we can the rewrite the equation as

$$\mathbb{E}[(y - \hat{y})^2]$$

$$= \mathbb{E}[(f + \varepsilon - \hat{y})^2]$$

$$= \mathbb{E}[(f + \varepsilon - \hat{y} + \mathbb{E}[\hat{y}] - \mathbb{E}[\hat{y}])^2]$$

$$= \mathbb{E}[(f - \mathbb{E}[\hat{y}])^2] + \mathbb{E}[\varepsilon^2] + \mathbb{E}[(\mathbb{E}[\hat{y}] - \hat{y})^2] + 2\mathbb{E}[(f - \mathbb{E}[\hat{y}])\varepsilon] + 2\mathbb{E}[\varepsilon(E[\hat{y}] - \hat{y})] + 2\mathbb{E}[(\mathbb{E}[\hat{y}] - \hat{y})(f - \mathbb{E}[\hat{y}])]$$

$$= (f - \mathbb{E}[\hat{y}])^2 + \mathbb{E}[\varepsilon^2] + \mathbb{E}[(\mathbb{E}[\hat{y}] - \hat{y})^2] + 2(f - \mathbb{E}[\hat{y}])\mathbb{E}[\varepsilon] + 2\mathbb{E}[\varepsilon]\mathbb{E}[\mathbb{E}[\hat{y}] - \hat{y}] + 2\mathbb{E}[\mathbb{E}[[\hat{y}] - \hat{y}](f - \mathbb{E}[\hat{y}])$$

$$= (f - \mathbb{E}[\hat{y}])^2 + \mathbb{E}[\varepsilon^2] + \mathbb{E}[(\mathbb{E}[\hat{y}] - \hat{y})^2]$$

$$= (f - \mathbb{E}[\hat{y}])^2 + Var[y] + Var[\hat{y}]$$

$$= Bias[\hat{y}]^2 + Var[y] + Var[\hat{y}]$$

$$= Bias[\hat{y}]^2 + \sigma^2 + Var[\hat{y}]$$

We find that the bias is given as

$$Bias[\hat{y}]^2 = \frac{1}{n}\sum_i (f_i - \mathbb{E}\,[\tilde{y}])^2$$

and the variance is given as

$$Var[\hat{y}] = \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\,[\tilde{y}])^2$$

5.1.5 Confidence intervals

So, a 97.5 % confidence interval tells us that we can be 97.5 % confident that our estimated value for, in this case, β, lies within this interval.

To produce the confidence intervals, we have used the following equation from Hastie et al. (2017):

$$[\beta_j - z^{(1-\alpha)} v_j^{1/2}\hat{\sigma} \quad , \quad \beta_j + z^{(1-\alpha)} v_j^{1/2}\hat{\sigma}] \tag{9}$$

where v_j is the j-th diagonal element of $(\mathbf{X}^T\mathbf{X})^{-1}$ and multiplied with $\hat{\sigma}$ is the standard deviation of β (STD_β). $z^{(1-\alpha)}$ denotes the $(1-\alpha)$ percentile (of the normal distribution), i.e. $z^{(1-\alpha)} = 1.96$ for the 97.5 % confidence interval. So, equation 9 can (in the case of a 97.5 % confidence interval for β), be expressed as:

$$\beta \pm 1.96 \cdot STD_\beta \tag{10}$$

5.2 Code snippets

Ordinary least squares

```
def linreg_ols(X, z):
   # Solving for beta beta = np.linalg.inv(np.transpose(X).dot(X)).dot(np.transpose(X)).dot(z)
   y_predict_ols = X  beta
return beta, y_predict_ols
```

Ridge regression

```
def ridgereg(X, z, lambda_ridge):
   # Solving for beta beta_ridge = np.linalg.inv(np.transpose(X).dot(X) + lambda_ridge*np.identity
   y_predict_ridge = X  beta_ridge
   return beta_ridge, y_predict_ridge
```

6 Bibliography

Bock, T. (20XX) What is a correlation matrix? Downloaded from: https://www.displayr.com/what-is-a-correlation-matrix/ (05.10.2019)

Devore, J. L. & Berk, K. N. (2012) *Modern Mathematical Statistics with Applications* 2nd. ed. Springer.

Faul, A. C. (2016) *A Concise Introduction to Numerical Analysis* CRC Press.

Hastie, T., Tibshirani, R. & Friedman, J. (2017) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* 2nd. ed. Springer.

Mehta, P. Bukov, M., Wang, C-H., Day, A. G. R., Richardson, C. Fisher, C. K. & Schwab, D. J. (2019) *A high-bias, low-variance introduction to Machine Learning for physicists* Physics Reports (Vol 810) https://doi.org/10.1016/j.physrep.2019.03.001.