UiO : Department of Physics
University of Oslo

FYS-STK4155 Project 1: Regression analysis and resampling methods

Astrid Tesaker, Vemund Thorkildsen & Sofie Tunes

1 Abstract

In the field of data analysis, the importance of providing predictions based on previously learned knowledge has always been vital. In later years, much work has been done on Machine-Learning (ML) algorithms. However, some important concepts in ML is also crucial in classical regression analysis. Some of these concepts include the bias-variance trade-off, in-sample error, out-sample error and resampling.

In this report, we have studied the Franke function, and digital terrain data from southern Norway. Three different regression methods have been studied (Ordinary Least Squares, Ridge and LASSO), along with two different resampling methods (k-fold cross-validation and bootstrap). The widely used error metrics, Mean Squared Error and R^2-score have been employed. The implemented code is available from: `https://github.com/VemundST/Project1`. From the results obtained during this project work, OLS gave the best fit. However, it is believed that through tuning of the penalty parameter λ, Ridge and LASSO can provide better results than presented in this report.

Contents

2 Introduction

Linear regression methods are fundamental and widely used as a tool for prediction. In a modern world with an increased reliance on data analysis, it is vital to be able to draw conclusions about the relationships between variables. This is better understood by using an example that includes a data set. The Boston housing data-set in ScikitLearn's library contains 13 different predictor variables. The goal is to predict the house values using these predictors. However, it would also be useful to study which variables that has the biggest affect on the price, for example size or location. Relationships like these will be possible to analyse using linear regression.

When developing algorithms, it is beneficial to test the algorithms on synthetic data. Here it is possible to control the relationship between noise and actual data. As a testing ground for machine learning, the Franke function is a good place to start. After verifying that the algorithms work as intended, real terrain data are introduced. On both of these data-sets, different regression -and resampling methods will be used. The error of the model when using these methods will be evaluated. Thereafter the results will be presented. A discussion and a short conclusion on how well the methods reproduce the data will conclude this report. Let us start by introducing some basic notation for linear regression.

3 Theory

In this section the different methods, that have been implemented in `Python`, will be described. There will be a short introduction to the basic theory of the regression methods. A thorough description of error metrics, resampling methods and model complexity will also be presented. This will allow us to gain insight into important Machine-Learning concepts, such as in-sample error, out-sample error, overfitting and the bias-variance trade-off.

3.1 Linear regression methods

By employing linear regression, it is possible to find the relationship between observed data ($\mathbf{y}$) and some explanatory variables, often called predictors ($\mathbf{X}$). In order to do so, an assumption that there exists a linear relationship between $\mathbf{y}$ and $\mathbf{X}$ is made (van Wieringen, 2019). This gives the assumptions for linear regression, and the relationship can be written as:

$$\mathbf{y} = f(\mathbf{X}) + \hat{\varepsilon} = \mathbf{X}\hat{\beta} + \hat{\varepsilon}, \tag{1}$$

where it is assumed that the observed data, $\mathbf{y}$, can be described as the sum of a function, $\mathbf{X}\hat{\beta}$, and the noise term $\hat{\varepsilon}$. The predictors, which are organized in the *design matrix* are denoted $\mathbf{X}$. The design matrix can be user defined, and has a size of $[n\ x\ p]$ (van Wieringen, 2019).

In order to fit the data to a polynomial, the unknown $\hat{\beta}$ parameters must be extracted from the data. After $\hat{\beta}$ is extracted, a prediction ($\tilde{\mathbf{y}}$) of the real function can be made:

$$\tilde{\mathbf{y}} = \mathbf{X}\hat{\beta}. \tag{2}$$

The procedure for estimating $\hat{\beta}$ varies for different regression schemes and will be explained further in sections 3.1.1, 3.1.2 and 3.1.3. Common for all these schemes is the reliance on a design matrix, which is shown below:

$$\mathbf{X} = \begin{bmatrix} 1 & x_0^{p_1} & x_0^{p_2} & \dots & \dots & x_0^{p_{n-1}} \\ 1 & x_1^{p_1} & x_1^{p_2} & \dots & \dots & x_1^{p_{n-1}} \\ 1 & x_2^{p_1} & x_2^{p_2} & \dots & \dots & x_2^{p_{n-1}} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1}^{p_1} & x_{n-1}^{p_2} & \dots & \dots & x_{n-1}^{p_{n-1}} \end{bmatrix}. \tag{3}$$

Here $\mathbf{p} = [p_0 = 0, p_1, p_2 \dots p_{n-2}, p_{n-1}]$ describes a generalized model that is fitted to some observed data. Depending on the problem at hand, the design matrix must be adjusted accordingly. This report will explore elevation as a function of x and y, thus the design matrix must also represent a polynomial of x and y. The generalized design matrix of degree $\mathbf{P}$, for fitting a polynomial depending on two variables is shown in equation 4.

$$\mathbf{X} = \begin{bmatrix} 1 & x_0 & y_0 & x_0^2 & x_0y_0 & y_0^2 \dots & x_0^{\mathbf{P}} & x_0^{\mathbf{P-1}}y_0^{\mathbf{1}} & \dots & x_0^1y_0^{\mathbf{P-1}} & y_0^{\mathbf{P}} \\ 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \dots & x_1^{\mathbf{1}} & x_1^{\mathbf{P-1}}y_1^{\mathbf{1}} & \dots & x_1^1y_1^{\mathbf{P-1}} & y_1^{\mathbf{P}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-2} & y_{n-2} & x_{n-2}^2 & x_{n-2}y_{n-2} & y_{n-2}^2 \dots & x_{n-2}^{\mathbf{P}} & x_{n-2}^{\mathbf{P-1}}y_{n-2}^{\mathbf{1}} & \dots & x_{n-2}^1y_{n-2}^{\mathbf{P-1}} & y_{n-2}^{\mathbf{P}} \\ 1 & x_{n-1} & y_{n-1} & x_{n-1}^2 & x_{n-1}y_{n-1} & y_{n-1}^2 \dots & x_{n-1}^{\mathbf{P}} & x_{n-1}^{\mathbf{P-1}}y_{n-1}^{\mathbf{1}} & \dots & x_{n-1}^1y_{n-1}^{\mathbf{P-1}} & y_{n-1}^{\mathbf{P}} \end{bmatrix} \tag{4}$$

3.1.1 Ordinary Least Squares

Ordinary Least Squares (OLS) is one of the simplest methods of regression, and tries to find the parameters $\hat{\beta}$ so that it minimizes the remaining sum shown in equation 5 (Hastie et al., 2005). This leads to a minimal error between $\mathbf{y}$ and $\tilde{\mathbf{y}}$. Instead of solving the above linear algebra problem (equation 2) to find the optimal parameters $\hat{\beta}$, the mean squared error (equation 10) is optimized via the so-called *cost function.* This function gives a measure of the spread between the values $\mathbf{y}$ and the parameterized values $\tilde{\mathbf{y}}$

(Hjorth-Jensen, 2019b).The unknown parameters $\hat{\beta}$, are found by taking the derivative of the *cost function* with respect to $\hat{\beta}$. The cost function is given as:

$$C(\hat{\beta}) = \frac{1}{n}\{(y - \mathbf{X}^T\hat{\beta})^T(y - \mathbf{X}^T\hat{\beta})\}, \tag{5}$$

and by taking the derivative with respect to $\hat{\beta}$ the analytical solution is found as:

$$\hat{\beta}^{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{6}$$

As clearly seen in equation 6, determining $\hat{\beta}^{OLS}$ involves a matrix inversion. In case of a singular, or near singular matrix $(\mathbf{X}^T\mathbf{X})^{-1}$, it then follows that OLS can break down.

Not all square matrices are diagonalizable and is so-called defective matrices. The condition $\mathbf{X}^T\mathbf{X} = \mathbf{X}\mathbf{X}^T$ is not fulfilled. The Singular Value Decomposition (SVD) algorithm can decompose any general matrix $\mathbf{X}$ into terms of a diagonal matrix and two orthogonal/unitary matrices.

It states that a general $m \times n$ matrix $\mathbf{X}$ can be written in terms of a diagonal matrix of dimensionality $n \times n$ and two orthognal matrices $\mathbf{U}$ and $\mathbf{V}$. Where $\mathbf{U}$ has dimensionality $m \times m$ and $\mathbf{V}$ dimensionality $n \times n$. This leads to the expression $\mathbf{X} = \mathbf{U}\sum\mathbf{V}^{\mathbf{T}}$. From this, it follows that SVD can be employed for determining the pseudo-inverse of a singular matrix, and thereby avoiding the singularity problem (Hastie et al., 2005).

3.1.2 Ridge regression

Ridge regression is a technique for analyzing multiple regression data that suffer from collinearity. The term collinearity refers to the event where two (or multiple) covariates (the columns of $\mathbf{X}$) are highly linearly related. When the design matrix is high-dimensional, it is more likely that the columns are collinear, thus leaving the design matrix singular (van Wieringen, 2019). A crude but effective way of circumventing the singularity issue is to add a small value (λ) to the diagonal of $(\mathbf{X}^T\mathbf{X})$, which assures that the matrix is always invertible. The equivalent cost function for Ridge can then be expressed as:

$$C(\mathbf{X}, \hat{\beta}) = \frac{1}{n}\{(\mathbf{y} - \mathbf{X}\hat{\beta})^T(\mathbf{y} - \mathbf{X}\hat{\beta})\} + \lambda\hat{\beta}^T\hat{\beta}. \tag{7}$$

By once again taking the derivatives with respect to $\hat{\beta}$, a modified matrix inversion problem is obtained. The matrix inversion problem with limited values of λ will not suffer from singularity problems.

$$\hat{\beta}^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}. \tag{8}$$

Where I is a $[p\ x\ p]$ identity matrix. The hyperparameter λ is a tuning parameter, which controls the shrinkage of $\hat{\beta}$. Larger values of λ leads to greater amounts of shrinkage.

3.1.3 LASSO regression

LASSO stands for Least Absolute Shrinkage and Selection Operator. LASSO is a shrinkage method like Ridge, with subtle but important differences (Hastie et al., 2005). Like Ridge, LASSO fits the linear regression model by minimizing the sum of squares augmented with a hyperparameter (penalty parameter). The difference with regards to Ridge regression lies in the penalty function (van Wieringen, 2019). The penalty function in LASSO adds the absolute value of the magnitude of the $\hat{\beta}$-parameters as a penalty to the cost function. The addition of this term means some features are entirely ignored by the model. Having some coefficients be exactly zero often makes a model easier to interpret, and can reveal the most important features of a model (Guido, 2016). With the penalty term, the cost function for LASSO takes the form:

$$C(\mathbf{X}, \hat{\beta}; \lambda) = \frac{1}{n}\{(\mathbf{y} - \mathbf{X}\hat{\beta})^T(\mathbf{y} - \mathbf{X}\hat{\beta})\} + \lambda\sqrt{\hat{\beta}^T\hat{\beta}}. \tag{9}$$

Contrary to OLS and Ridge regression, there is no analytical expression for finding $\hat{\beta}$. So the extraction of $\hat{\beta}$ is done iteratively. The iterative search for $\hat{\beta}$ is done by employing gradient decent methods, where another hyperparameter called learning rate (γ) is important (Hjorth-Jensen, 2019b). The "Lasso" function from `Scikit-Learn` takes measures to determine the learning rate, so this will not be the focus of this report.

3.2 Error metrics

In order to estimate the performance of the model (how well the prediction fits the data) it is important to make use of error estimation metrics. In this work, the Mean Squared Error and R^2-score has been used.

3.2.1 Mean Squared Error (MSE)

The MSE is an average of the squared difference between the predicted data points $\tilde{\mathbf{y}}$ and their actual value $\mathbf{y}$. Mathematically this can be expressed as:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y_i - \tilde{y_i})^2, \tag{10}$$

This means that a MSE close to zero gives a good estimate for the model. Note also that the cost function given by equation 5 is equal to equation 10. In other words, OLS attempts to minimize the Mean Squared Error.

3.2.2 R^2- score

The calculations of MSE can be used to find the R^2-score.

$$R^2 = 1 - \frac{MSE}{MSE_0} = 1 - \frac{\sum(y_i - \tilde{y}_i)^2}{\sum(y_i - \bar{\mathbf{y}})^2}. \tag{11}$$

The R^2-score measures the relationship between the variance of the predicted model and the total variance of the input data. When the total variance for the data is much larger than the variance of the prediction, the R^2-score becomes close to the optimal value of one.

3.3 Training & test data

To assess whether a model prediction is reliable, splitting of the input data in training and test datasets have been done. In this way, the estimation of the $\hat{\beta}$ parameters and the prediction is done on separate subsets of the data (Guido, 2016). The splitting of the data should be randomized, i.e shuffling the data before splitting. Figure 1 shows an idealized representation of the train and test error as a function of model complexity. Given an infinite amount of model complexity, the training error will move towards zero. However, this might leave little room for flexibility, so it is important to also evaluate the test error to avoid *overfitting*, where the model might fit noise.

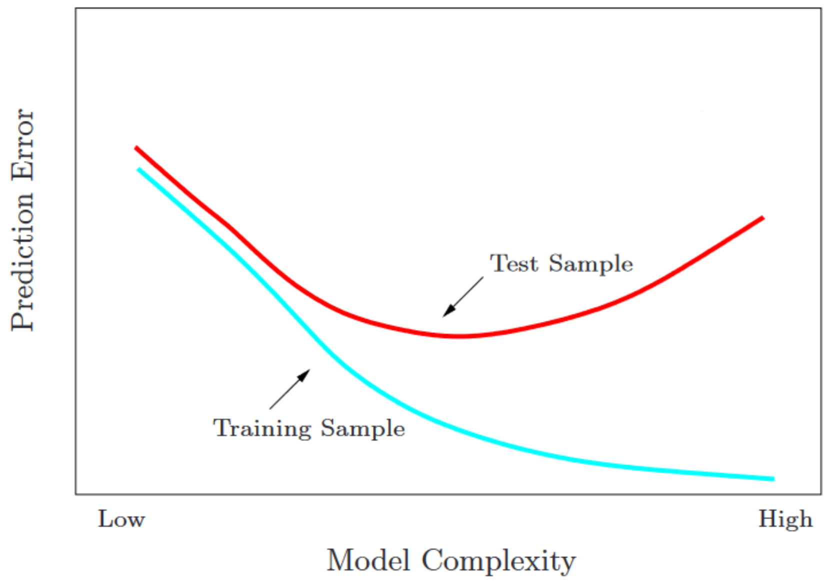


Figure 1: Training and test error as a function of model complexity (modified from Hastie et al. (2005)

There is no explicit rule for splitting data, but a rule-of-thumb is to use between 65 % and 80 % for training, and use the rest for testing. Sometimes there is not enough data to train on, which might give an uncertain estimate of the future predictions. In this case it is common to employ resampling methods (Murphy, 2012).

3.4 Resampling methods

As mentioned, resampling methods can be used for model assessment when the amount of input data is limited. Resampling involves repeatedly drawing samples from a training set and refitting a model of interest on each subset (Hjorth-Jensen, 2019b). This allows us to determine if the model fits the general function governing the input data, or just a subset of the data.

3.4.1 *K*-fold cross-validation

K-fold cross-validation involves splitting the input data into k different folds. The steps for doing k-fold cross validation can be summarized as:

1. Split data in k folds
2. Train the model on k - 1 folds, leave one out
3. Predict on the dataset that was left out
4. Compute error metrics
5. Repeat steps 2-4 for k folds
6. Average error metrics

It is common to set k to a value between 5 and 10 (Murphy, 2012), which has been done in this project work.

3.4.2 Bootstrap

Bootstrap is another resampling technique that has been employed in this project. Bootstrapping can be summarized as:

1. Create subset of data by drawing M random datapoints with replacement
2. Train the model on the subset
3. Predict on the data-set that was left out
4. Compute error metrics
5. Repeat steps 2-4 for N bootstraps
6. Average error metrics

Drawing with replacement involves that one datapoint can be picked twice, and some datapoints will not be picked at all (Hastie et al., 2005). In this project, the Bootstrap technique was used to evaluate the Bias-variance tradeoff.

3.4.3 Bias-variance trade-off

The Mean Squared Error can be decomposed in terms bias, variance and an irreducible error:

$$
\begin{aligned}
C(\mathbf{X},\hat{\beta}) &= \frac{1}{n}\sum^{n-1}(y_i-\tilde{y_i})^2 = E[(\mathbf{y}-\tilde{\mathbf{y}})^{\mathbf{2}}] \\
&= E[(f+\varepsilon-\tilde{\mathbf{y}})^{\mathbf{2}}] \\
&= E[(f+\varepsilon-\tilde{\mathbf{y}}+E[\tilde{\mathbf{y}}]-E[\tilde{\mathbf{y}}])^2] \\
&= E[(\mathbf{y}-E[\tilde{\mathbf{y}}])^2]+Var[\tilde{\mathbf{y}}]+\varepsilon^2 \\
&= \frac{1}{n}\sum_i(f_i-E[\tilde{\mathbf{y}}])^2+\frac{1}{n}\sum_i(\tilde{\mathbf{y}}-E[\tilde{\mathbf{y}}])^2+\varepsilon^2
\end{aligned} \tag{12}
$$

The first term in equation 12 describes the bias. Bias is the difference between the model prediction and the correct value, often denoted:

$$
Bias^2=\frac{1}{n}\sum_i(f_i-E[\tilde{\mathbf{y}}])^2 \tag{13}
$$

The variance is the variability of the model prediction for a given data point. Or in other words, the variance is a value which describes the spread of the prediction.

$$
Var=\frac{1}{n}\sum_i(\tilde{\mathbf{y}}-E[\tilde{\mathbf{y}}])^2, \tag{14}
$$

The last term in equation 12 is the irreducible error, which means the equation can be rewritten as:

$$
E[(\mathbf{y}-\tilde{\mathbf{y}})^2]=Bias^2+Variance+Irreducible\ Error. \tag{15}
$$

Bias can be interpreted as an error caused by a models inability to sense smaller fluctuations in the data-set (Hjorth-Jensen, 2019b). For instance, data governed by a fifth degree polynomial can not be properly fitted to a second degree model, the error this induces is due to the model being *biased.* Bias typically decreases with model complexity, as higher order models can sense smaller fluctuations (Hjorth-Jensen, 2019b). The variance can be interpreted as the opposite of bias. For higher degree polynomials, the model can begin to *overfit*, i.e fitting noise into the prediction. This is denoted as the variance of the prediction.

3.4.4 Confidence interval

A confidence interval gives an estimate on how well $\hat{\beta}$ is predicted. For 95% confidence interval employing OLS, the confidence interval can be defined as:

$$c_{\beta_i} = \beta_i \pm 1.96 SE(\beta_i). \tag{16}$$

Where c_{β_i} is the confidence interval for a given β_i, and the $SE(\beta_i)$ can be found by taking the square root of $Var(\beta_i)$, which can be found by extracting the diagonal of $(\mathbf{X^T X})^{-1}\sigma^2$. The same procedure can be employed for Ridge, but the $Var(\beta_i)$ is now calculated as the diagonal of $(\mathbf{X^T X} + \lambda \mathbf{I})^{-1}\mathbf{X^T X}(\mathbf{X^T X} + \lambda \mathbf{I})^{-1}\sigma^2$. When employing LASSO regression, there is no analytical expression for finding the confidence interval. It is possible to find the confidence interval, but it has not been done in this project work.

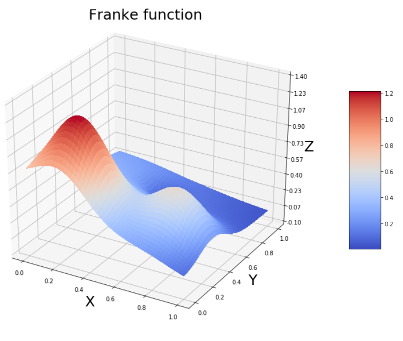
4 Data sets

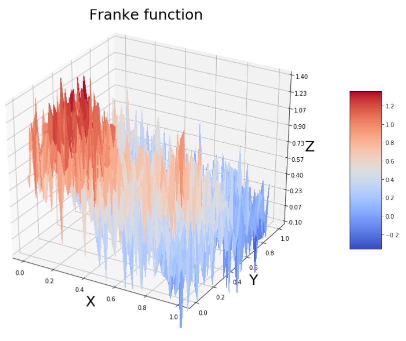
4.1 The Franke function

To study the various regression methods, the Franke function was used. The Franke function is a weighted sum of four exponentials and is broadly used for testing various surface interpolation techniques and fitting algorithms (Hjorth-Jensen, 2019a). The function is defined for $x, y \in [0, 1]$, and reads as follows:

$$\begin{aligned} f(x,y) = &\frac{3}{4} exp(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}) \\ &+ \frac{3}{4} exp(-\frac{(9x+1)^2}{49} - \frac{9y+1}{10}) \\ &+ \frac{1}{2} exp(-\frac{(9x-7)^2}{4} - \frac{(9y-3))^2}{4}) \\ &- \frac{1}{5} exp(-(9x-4)^2 - (-9y-7)^2)). \end{aligned} \tag{17}$$

Figure 2 shows the Franke function with and without noise.





(a)

(b)

Figure 2: The Franke function without noise (a) and with noise = 0.25 (b)

4.2 Terrain data

After testing the algorithms on the Franke function, it was time to introduce real data. Digital terrain data was downloaded from the Github site https://github.com/CompPhysics/MachineLearning/tree/master/doc/Projects/2019/Project1/DataFiles, and the file SRTM_data_Norway_1.tif was chosen. This file contains terrain data from a region in Norway called Møsvatn Austfjell (shown in figure 3), which is located in Telemark county. A small part of the data **(75x75)** was extracted (shown in figure 4), and the data-set was normalized by using the following relation:

$$data_{norm} = \frac{data - d\bar{a}ta}{std(data)} \tag{18}$$

Where $d\bar{a}ta$ denotes the mean value, and $std(data)$ denotes the standard deviation.

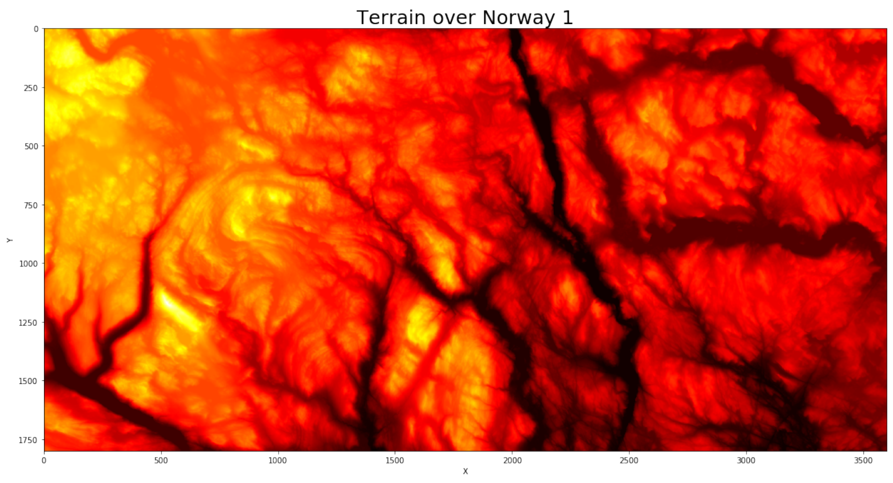
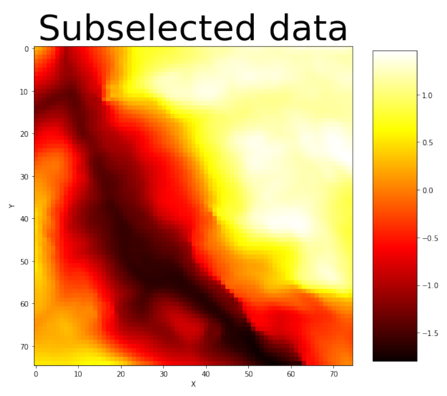
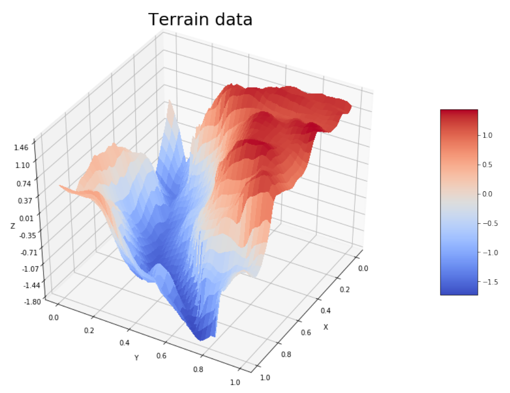


Figure 3: Terrain data from Telemark, Norway



(a)



(b)

Figure 4: Subselected terrain data heatmap (a) and surface plot (b)

5 Implementation

Based on the theory presented in section 3, the algorithms have been implemented in `Python`. `functions.py` includes all the algorithms that have been developed during this project work. The code can be accessed from `https://github.com/VemundST/Project1`, where `main.ipynb` has been used to generate the results presented in this report.

The code can roughly be divided in three: Regression, error metrics and resampling. Error metrics and resampling methods have been implemented from scratch, together with OLS and Ridge. The functionalities of `Scikit-Learn` was employed for Lasso regression and for splitting the input data in training and test sets. SVD was taken from the `numpy` library. It is included in the code, but the results presented in this report was obtained without SVD. Included in the `GitHub` repository, there is a test-script, used for benchmarking the codes against `scikit-learn`.

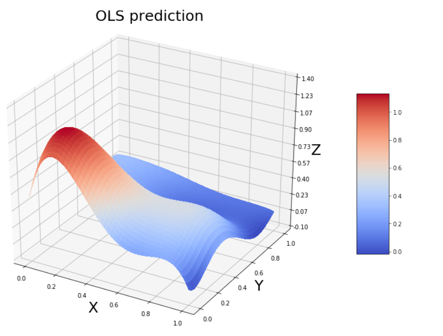
The code shown below was used for creating the design matrix. As seen from the code, there is a short comment describing the inputs and outputs of the functions, along with a description of its intended use. Such a description is avaliable for all the codes included in `functions.py`.

```
def DesignDesign(x, y, power,ravel=False):
    '''
    This function employs the underlying pattern governing a design matrix
    on the form [1,x,y,x**2,x*y,y**2,x**3,(x**2)*y,x*(y**2),y**3 ....]
    x_power=[0,1,0,2,1,0,3,2,1,0,4,3,2,1,0,...,n,n-1,...,1,0]
    y_power=[0,0,1,0,1,2,0,1,2,3,0,1,2,3,4,...,0,1,...,n-1,n]
    input:  x,y = 1D spatial vectors, or raveled mesh if ravel=True
            power = polynomial degree
            ravel = False (input vectors will not be meshed and raveled)
                  = True (input vecotors wil be kept as is)
    output: DesignMatrix = The design matrix
    '''
    concat_x    = np.array([0,0])
    concat_y    = np.array([0,0])
    for i in range(power):
        toconcat_x = np.arange(i+1,-1,-1)
        toconcat_y = np.arange(0,i+2,1)
        concat_x   = np.concatenate((concat_x,toconcat_x))
        concat_y   = np.concatenate((concat_y,toconcat_y))
    concat_x      = concat_x[1:len(concat_x)]
    concat_y      = concat_y[1:len(concat_y)]
    if ravel:
        X = x
        Y = y
    else:
        X,Y            = np.meshgrid(x,y)
        X              = np.ravel(X)
        Y              = np.ravel(Y)
    DesignMatrix = np.empty((len(X),len(concat_x)))
    for i in range(len(concat_x)):
        DesignMatrix[:,i]     = (X**concat_x[i])*(Y**concat_y[i])
    return DesignMatrix
```

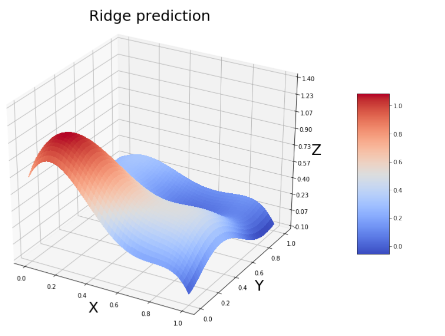
6 Results

In this section the results from the implementation in `Python` will be presented. A stochastic noise with a normal distribution $N(0, \sigma^2)$, where $\sigma = 0.25$, was added to the Franke function. The results presented will mostly be discussed as a function of model complexity and the hyperparameter λ.

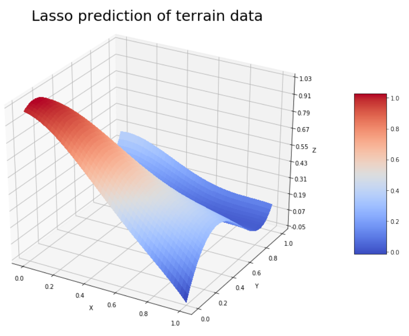
6.1 Franke Function



(a)



(b)



(c)

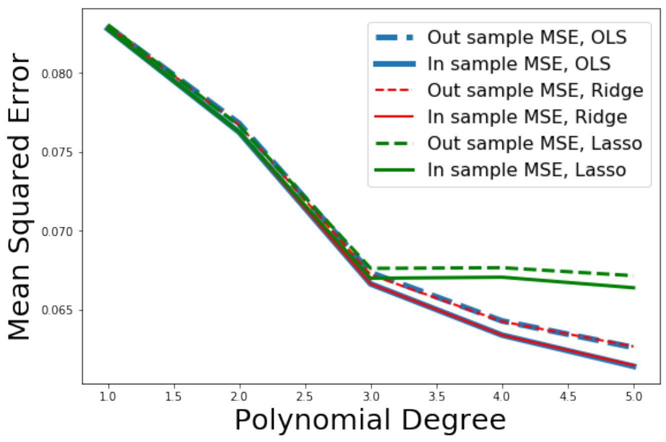
Figure 5: Predicted surface for a fifth degree design matrix by employing OLS (a) Ridge (b) and Lasso (c) as regression scheme. λ was set to 10^{-4} for these specific plots.

The hyperparameter in the Ridge and Lasso regression analysis are initially set to 10^{-4} and are using polynomials in x and y up to fifth order. From table 1, which shows the MSE and R^2-score for a fifth degree polynomial, it is evident that OLS shows the best MSE and R^2-score, while Lasso performs worst. Ridge is slightly worse than OLS, but almost comparable, which is visualized in figure 5.

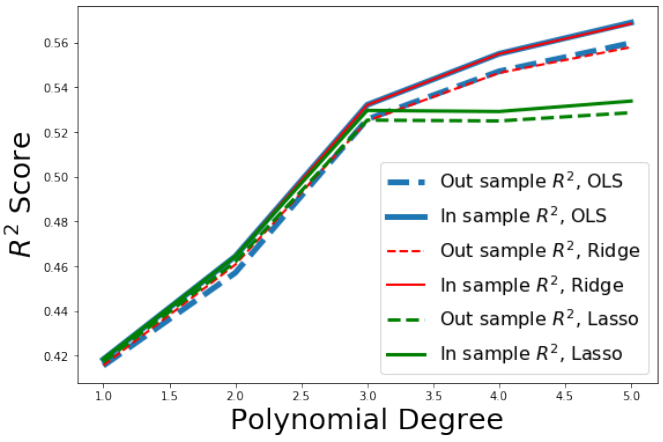
Table 1: MSE and R^2-score for OLS, Ridge and Lasso with polynomial 5 and $\lambda = 10^{-4}$.

	OLS	Ridge	Lasso
MSE	≈ 0.06360	≈ 0.06607	≈ 0.07513
R^2	≈ 0.55190	≈ 0.54169	≈ 0.47070

The in-sample (training) error and out-sample (test) error is shown in figure 6. These results were obtained by doing k-fold cross-validation, to ensure that the results are representative for the full dataset. As also seen in table 1, OLS and Ridge exhibit comparable results. Lasso performs similarly to ridge and OLS up to a power of three, but is significantly worse for fourth and fifth degree polynomials. For all regression methods, the out-sample error is worse than the in-sample error.



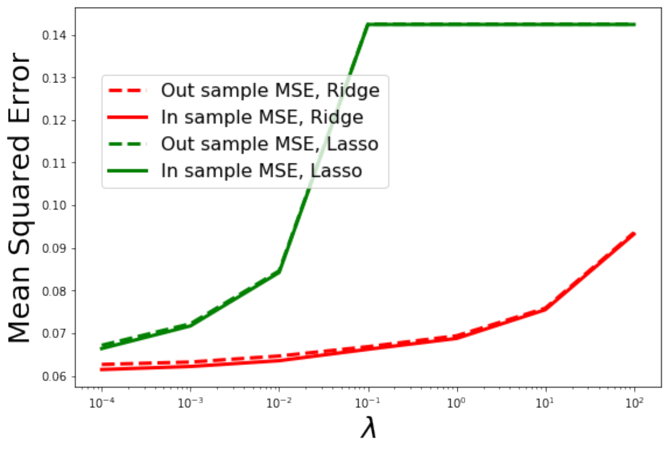
(a)



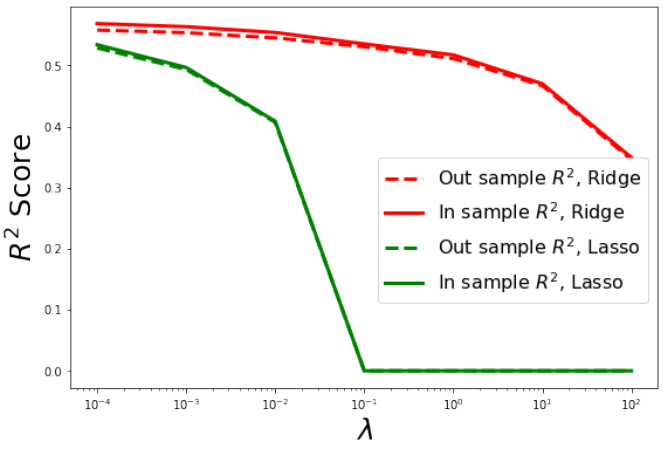
(b)

Figure 6: MSE (a) and R^2- score (b) for franke function with OLS, Ridge and Lasso as function of polynomial degree

Figure 7 shows the MSE and R^2-score as function of λ for a fifth degree polynomial. For larger values of λ, the error gets larger for both Lasso and Ridge. It is also worth noting that the difference between in-sample error and out-sample error becomes smaller for larger values of λ. OLS is not included in these plots as it does not depend on any hyperparameter.



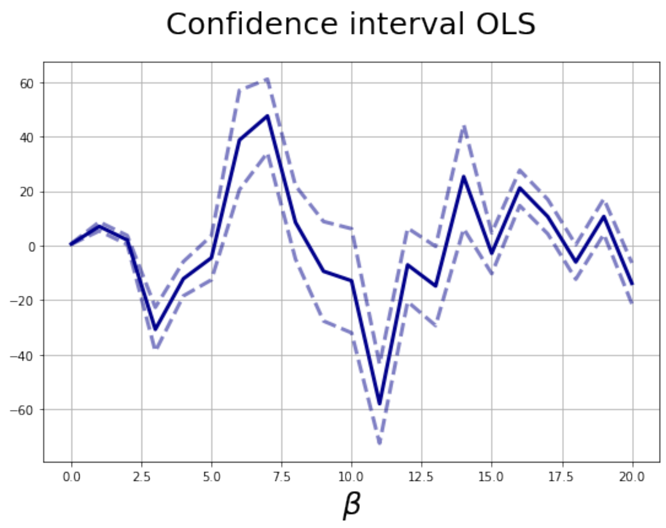
(a)



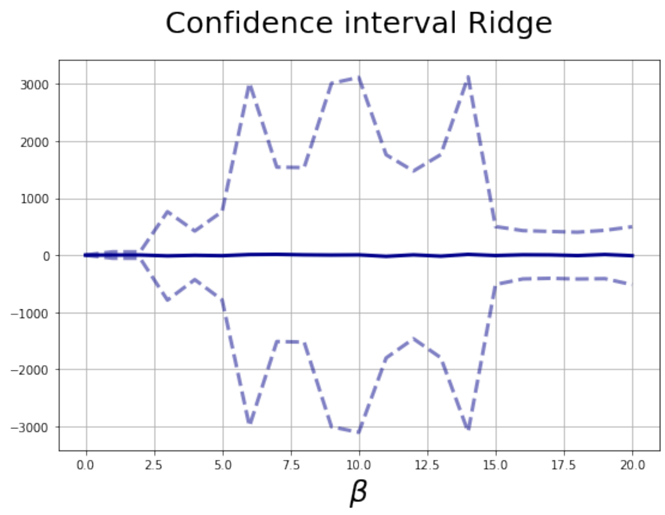
(b)

Figure 7: MSE (a) and R^2- score (b) for franke function with Ridge and Lasso as function of λ. Results obtained by k-fold cross validation with k=5

The confidence interval for respectively OLS (figure 8a) and Ridge (figure 8b) show extreme variability. The confidence interval for Ridge is much larger than for OLS.



(a)



(b)

Figure 8: Confidence interval of the parameters $\hat{\beta}$ for OLS (a) and Ridge (b). Results obtained by k-fold cross validation with k=5

From figure 9, it is possible to evaluate the Bias-variance trade-off. The bias is high for lower polynomial degrees before stabilizing at lower value for higher degree polynomials. The variance is low for smaller values, before getting larger for higher order polynomials. It can also be seen that the bias does not move towards zero, but rather stabilizes at a higher value. This will be discussed further in section 7.

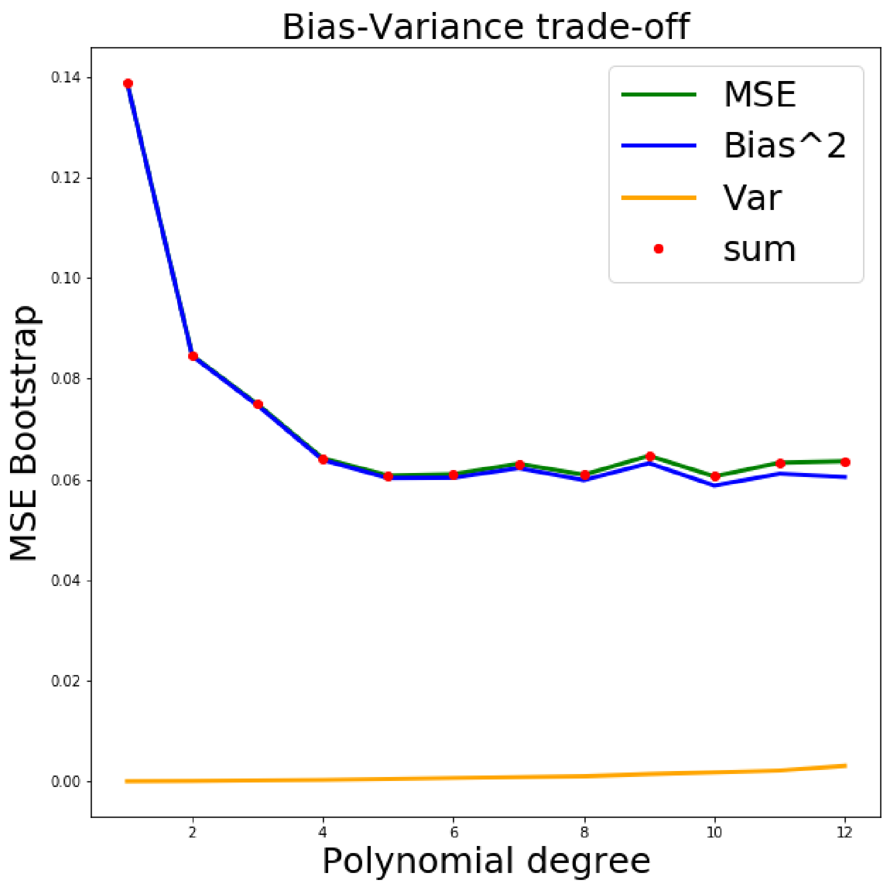
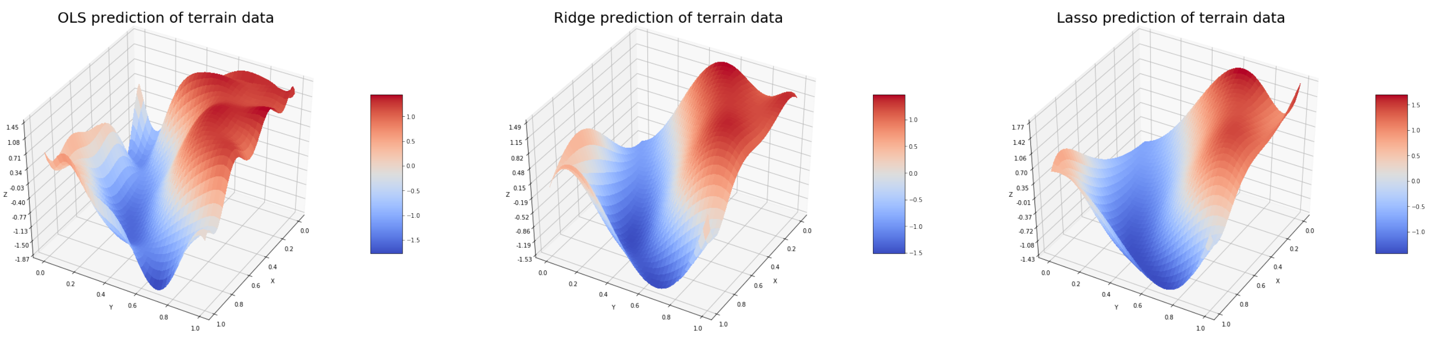


Figure 9: MSE, variance and bias of OLS as a function of model complexity. Obtained by doing 100 bootstraps

6.2 Terrain data

As mentioned, a small subset of the terrain data were chosen for the calculations. Figure 10 shows the predicted surface of the terrain data. From this figure one can see that OLS makes the best prediction, which table 2 agrees with. Looking at table 2 one can see that the regression methods gives lower errors for the terrain data compared to the franke function. After OLS, Ridge gives a better prediction than LASSO.



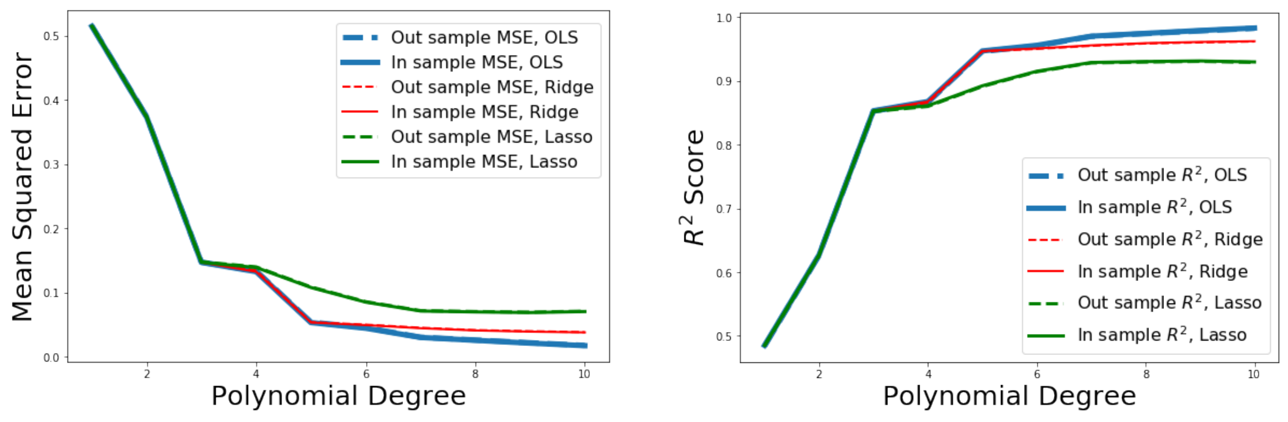
(a) (b) (c)

Figure 10: Prediction for the terrain data for a tenth degree design matrix by employing OLS (a), Ridge (b) and Lasso (c) with $\lambda = 10^{-4}$

Table 2: MSE and R^2-score for OLS, Ridge and Lasso with polynomial 10 and $\lambda = 10^{-4}$ using the terrain data.

	OLS	Ridge	Lasso
MSE	≈ 0.01809	≈ 0.03902	≈ 0.07066
R^2	≈ 0.98187	≈ 0.96091	≈ 0.92934

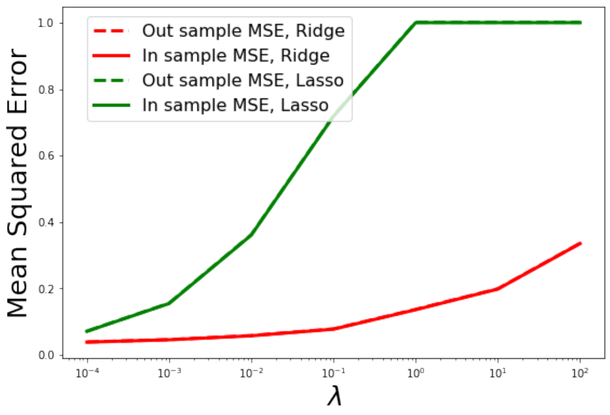
Figure 11 shows the results of k-fold cross validation with in- and out-sample for terrain data. The three regression methods, performs in a similar way up to a fourth degree. The spread between the in-sample and out-sample error is not as wide as for the Franke function. This will be discussed further in section 7.



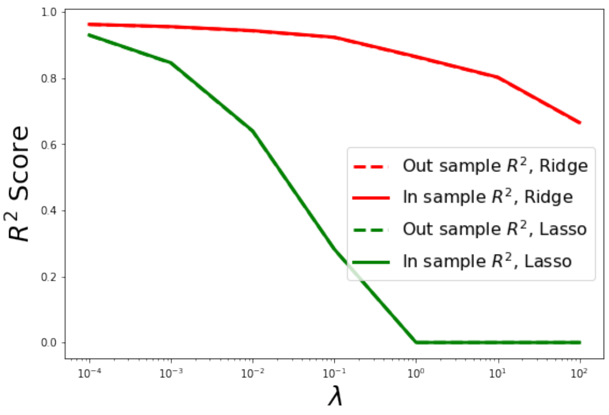
(a) (b)

Figure 11: MSE (a) and R^2- score (b) for the terrain data with OLS, Ridge and Lasso as function of polynomial degree

Figure 12 shows the MSE and R^2-score as function of λ for a tenth degree polynomial using the terrain data. As for the Franke function, larger values of λ gives a bigger error for both Lasso and Ridge. The difference between in-sample error and out-sample error is smaller than for the Franke function.



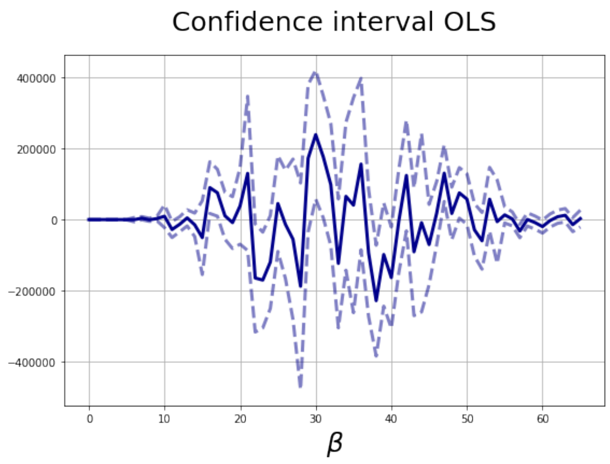
(a)



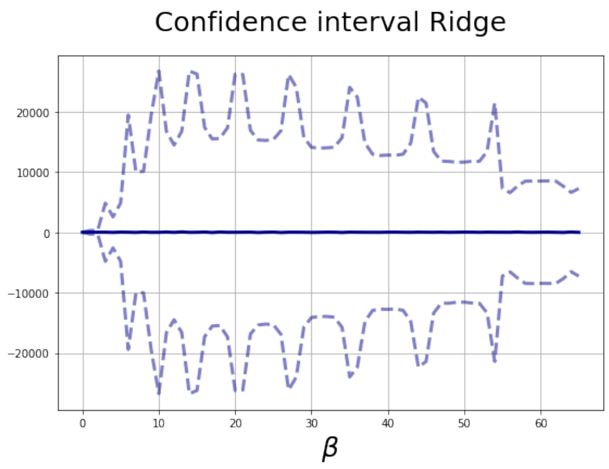
(b)

Figure 12: MSE (a) and R^2- score (b) for the terrain data with OLS, Ridge and Lasso as function of λ. Results obtained by k-fold cross validation with k=5

The confidence interval for the terrain data using OLS and Ridge is shown respectivly in figures 13a and 13b. As for the Franke function, the confidence interval for Ridge is much bigger than for OLS.



(a)



(b)

Figure 13: Confidence interval of the parameters $\hat{\beta}$ with noise level set to 1, for OLS (a) and Ridge (b) on the terrain data.

Figure 14 shows the Bias-variance tradeoff for the terrain data. One can see that the variance is more steady than for the Franke function, and the bias gets even smaller. There is also less difference between the MSE and the bias.

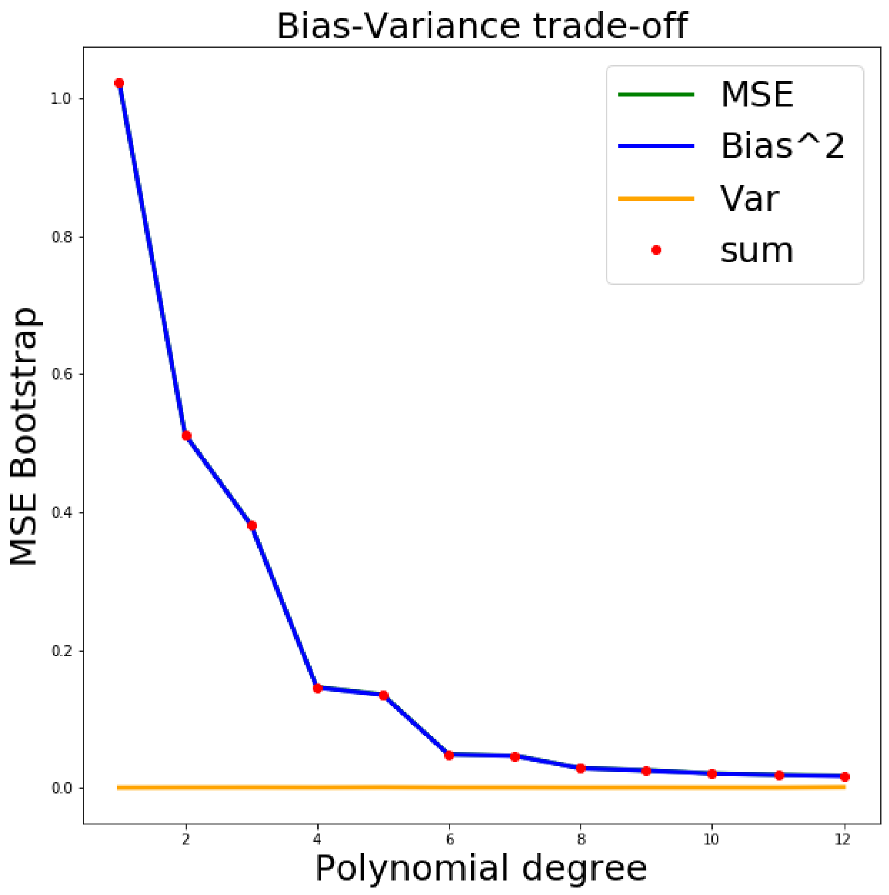


Figure 14: MSE, variance and bias of OLS as a function of model complexity. Illustrates the bias variance tradeoff for the terrain data. Results obtained by doing 100 bootstraps.

7 Discussion

From the results obtained for regression of the Franke function and terrain data, it might seem tempting to conclude that OLS is best. However, it has the distinct drawback of breaking down due to collinearity within the columns of the design matrix. Lasso and Ridge regression does not suffer from the same limitations.

Ridge takes measures to avoid singular matrices by adding the hyperparameter λ to the diagonal of $\mathbf{X}^\mathbf{T}\mathbf{X}$ as seen in equation 8. λ is also known as a *penalty parameter*, as it shrinks the $\hat{\beta}$-parameters. By letting $\lambda \implies \infty$ in equation 8, it naturally follows that $\hat{\beta}$ will tend towards zero. However, the parameters will never be exactly zero. Lasso regression also adds a regularization term to the cost function. In contrast to Ridge, Lasso regression penalizes $\hat{\beta}$ harder, and in some instances put the $\hat{\beta}$-parameters exactly to zero.

In figure 6, it is evident that LASSO has larger error for polynomial degrees four and five. This is also mirrored for higher degree polynomials for the terrain data (figure 11). Evaluating this together with figures 7 and 12, a possible explanation of the higher LASSO error, is that the chosen penalty parameter ($\lambda = 10^{-4}$) is too harsh. Thus leaving many estimators inoperative. Figure 15 illustrates that the number of non-zero (active) parameters stabilizes for higher degree polynomials. Therefore, it is not expected that higher degree polynomials will yield lower error if the penalty parameter is held constant.

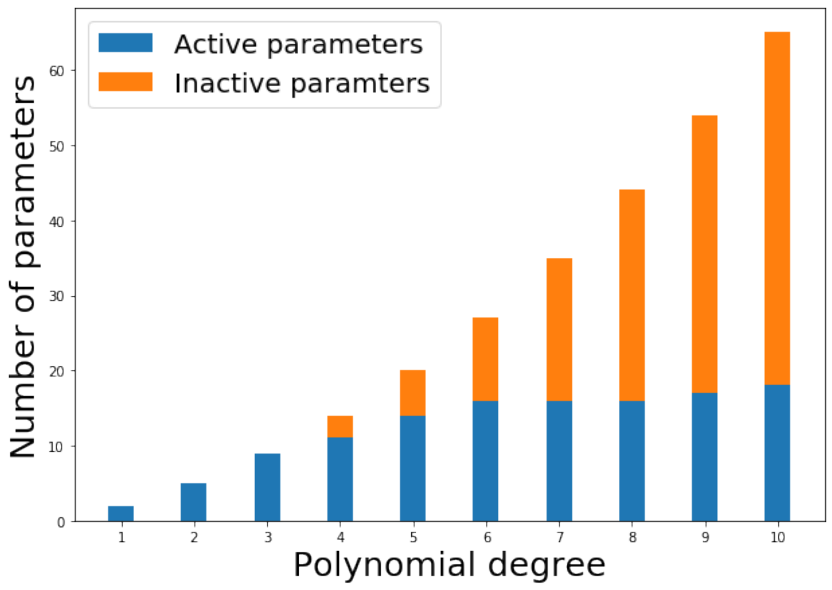


Figure 15: Active parameters compared to inactive parameters for LASSO regression as a function of polynomial degree with $\lambda = 10^{-4}$

Figure 6 illustrates the concept of in-sample and out-sample error. The in-sample error is constantly lower than the out-sample error. For the controlled data, it is known a-priori that the MSE should move towards $\sigma^2 = 0.25^2 = 0.0625$ for the Franke function. However, in case of OLS, it is in practice no limit to how low the in-sample error will get. The idealized error plot shown in figure 1 does not seem to be representative for either the Franke function or the terrain data. The idealized figure illustrates the principle of overfitting, where the out-sample error increases as the model is increasingly fitting noise. This phenomenon is not well represented in the plots included in this report, which might be due to too low model complexities. For the terrain data, the is little difference between in- and out-sample error (figure 11). A possible explanation is that the model complexity must be increased drastically in order to start overfitting.

Bootstrap was used to evaluate the bias-variance tradeoff. Resampling is necessary to get an estimate of the expectation value of the prediction (denoted $E[\tilde{y}]$ in equation 12). For high-complexity models, the variance tends to become larger, while the bias decreases. This effect can be seen from the bias-variance plot for the Franke function (figure 9), but is not present in the plot for the terrain data (figure 14). This discrepancy might be due to the higher noise levels in the Franke function. From both these plots however, the relationship described by equation 12 does not hold. As expected, the $bias^2$ gets lower for higher model complexities. However, it stabilizes before it becomes zero. This effect is caused by an inability to separate the noise term included in equation 12. In equation 13, f_i denotes the real function $f(x)$ in equation 1, i.e without the noise included. As this function is hard to obtain in the real world, the bias has been calculated with the noise term included in this report.

The problem of overfitting can be limited by employing the Ridge or LASSO regression schemes. If the penalty parameter λ is tuned correctly, both of the penalizing regression schemes has the ability to provide a better fit for higher order polynomials. This is due to the variance being limited by the penalty parameter. Tuning of λ has not been done in this project work. It is also worth noting that LASSO can be used for parameter selection, as LASSO emphasizes the most important parameters.

Looking at figures 7 and12, it is evident that the error becomes larger with increasing λ. This is true for both of the penalizing regression schemes, but LASSO seems to be affected to a greater extent. The difference between in- and out-sample error decreases for higher λ-values. This might be attributed to an increased generality of the model. In other words, by increasing λ, bias is added to the model. This limits overfitting and therefore emphasizes the most important features of the data.

In this report, the noise level was set to 0.25 for the Franke function. The chosen noise level will greatly impact several of the calculations shown in section 6. With a greater noise level, the discrepancy between in-sample and out-sample error will increase. Greater noise levels will also lead to a larger confidence interval for all regression schemes. For the Franke function, the noise level is user defined, thus legitimizing the confidence interval. For the terrain data, the noise level is not a-priori knowledge, and was set to 1 in the calculations. Thus, the confidence interval is believed to be much smaller than shown in figure 13.

8 Conclusion

The regression schemes presented in this report are widely used, and have been studied extensively. Ridge and LASSO regression has been studied as a function of both polynomial degree and their reliance on the hyperparameter λ, while OLS has been studied as a function of polynomial degree. OLS gave the best results based on the error estimations presented in this report. It must however be noted, that neither LASSO or Ridge have been tuned as function of the penalty parameter or polynomial degree, and it is believed that they show greater promise for improving the results.

The inclusion of the penalty parameter when employing Ridge and LASSO can be used as a measure to limit overfitting. However, it should be tuned, in order to obtain the optimal results. It can be acknowledged that employing resampling techniques, both as a validation of a models general fit, and as a tool for analyzing the bias-varaince tradeoff, is important.

Apart from tuning ridge and LASSO as a function of λ and polynomial degree, lasso could have been used for determining the most important features. These features could then be extracted, and used as an input for OLS, which might improve the results.

9 Documentation

The material used in this project can be found in the group member's repository in `GitHub`:
`https://github.com/VemundST/Project1`
The Python 3.7 main script can be found at:
`https://github.com/VemundST/Project1/blob/master/main.ipynb`.
While the functions can be found at:
`https://github.com/VemundST/Project1/blob/master/functions.py`.
The figures can be found in the folder in `GitHub`:
`https://github.com/VemundST/Project1/tree/master/figures`.

References

Andreas C. Mueller & Sarah Guido. Introduction to machine learning with python, 2016.

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2), 2005.

Morten Hjorth-Jensen. Project 1 on machine learning,, 2019a.

Morten Hjorth-Jensen. Data analysis and machine learning: Linear regression and more advanced regression analysis, 2019b. URL https://compphysics.github.io/MachineLearning/doc/pub/Regression/pdf/Regression-minted.pdf.

Kevin P. Murphy. Machine learning- a probabilistic perspective,, 2012.

Wessel N. van Wieringen. Lecture notes on ridge regression, 2019. URL https://arxiv.org/pdf/1509.09169.pdf.