DATA ANALYSIS AND MACHINE LEARNING
FYS-STK4155

Project 1

Alessia SANFELICI
Carmen DITSCHEID
Lila CASSAN
Marco POZZOLI

October 12, 2023

Abstract

We explored the use of different methods for fitting linear regression to the Franke function and later to topographical terrain data in a region near Stavanger, Norway. We used the OLS, Ridge and Lasso regression methods and combined the OLS method with resampling methods, such as bootstrap resampling, respectively cross-validation. We found that the OLS method gave us the best result, in contrast to Ridge and Lasso performing worse. Furthermore, we studied the underlying theoretical statistics, focusing on the distribution properties of our model as well as the bias-variance trade-off.

1 Introduction

Linear regression, a fundamental statistical technique, denotes a well studied method for a supervised learning approach to predictive modelling. Its simplicity, in combination with the various developed additions, offer a wide range of parameters to tune and techniques to add to generate a high validity model for a data set. Its impact is reaching far beyond the above. Building the foundation for more complex Machine Learning algorithms, e. g. logistic regression for classification, unsupervised methods are derived by linear regression. To benchmark a more sophisticated algorithm, linear regression is used as well to give a first subsumption, whether the developed method is valid. Motivated by the importance and topicality of linear regression, we examine it in this report.

In this project, we explore the impact of the parameters as well as methods on the model output and discover the relation of certain parameters. The former primarily refers to studying the model behaviour with different Lambdas and comparing OLS, Ridge and Lasso, as well as examining the improvement achievable by resampling techniques. The latter consists in extracting the distribution properties of the model and inspecting the bias-variance trade-off. Finally, this project aims to provide a red thread to guide one on how to fit a linear regression to a particular data set.

We start by fitting OLS, Ridge and Lasso regression to a known continuous function, the Franke function. We then examine the cause of our model behaviour by deriving the statistical properties of our model. Subsequently, we regard the bias-variance trade-off and add resampling techniques to improve our results. Finally, we use our gained insights to apply linear regression to terrain data in a region near Stavanger, in Norway. All the computational work has been implemented in Python, through a set of Jupyter notebooks, that can be found in the GitHub folder of the project (see Appendix).

2 Methods

In this section, the theory and the methods which are at the base of the whole work are explained in detail.

2.1 Linear Regression

A linear regression model aims at identifying the pattern of the relationship between a set of input variables X, called independent variables or predictors, and an output variable Y, the so-called dependent variable.
The model assumes that a linear, to be fitted regression function $f(X)$ either describes $E(Y|X)$ or at least depicts a reasonable approximation:

$$f(X) = X\beta.$$

In other words, we assume that our prediction is linearly dependent on the inputs $X = [X_0, X_1, \ldots, X_{p-1}]^T$. $f(X)$ can be written either in matrix form, as above, or as follows:

$$f(X) = (\beta_0 + \sum_{j=1}^{p-1} X_{1j}\beta_j, \ldots, \beta_0 + \sum_{j=1}^{p-1} X_{nj}\beta_j)^T.$$

We assume that our data $X = [X_0, X_1, \ldots, X_{p-1}]^T$ follows an unknown $f(X)$ of the form above, up to a stochastic noise ϵ, with $\epsilon = [\epsilon_0, \epsilon_1, \ldots \epsilon_{n-1}]^T \in \mathbb{R}^n$ and $\epsilon_i \sim_{iid} N(0, \sigma^2)$, with which the the model accounts for $f(X) \neq E(Y|X)$, $f(X) \approx E(Y|X)$. Hence:

$$\begin{aligned} Y &= E(Y|X_0, X_1, \ldots, X_{p-1}) + \epsilon \\ &= (\beta_0 + \sum_{j=1}^{p-1} X_{1j}\beta_j, \ldots, \beta_0 + \sum_{j=1}^{p-1} X_{nj}\beta_j)^T + \epsilon. \end{aligned}$$

A particular fitted regression model is then the result of minimizing a a priori defined cost function. We will examine three different cost functions and thus obtain three different models: *Ordinary Least Squares*, *Ridge regression* and *Lasso regression*. To anticipate already one property: while for OLS and Ridge regression we can obtain the solution of the minimization problem as an analytic formula for $\hat{\beta}$, this is not possible for Lasso regression. Therefore, for the latter, we have to rely on numerical methods.

To compare the different methods, we choose the MSE (explained in detail in 2.5.1) of the test data as the measure of our model's validity. We define the model with the lowest test data MSE as our optimal model.

2.1.1 Notations

We write:

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

where $\mathbf{y}$ denotes the observed output. We use $\tilde{\mathbf{y}}$ to refer to our model's predictions:

$$\tilde{\mathbf{y}} = X\beta,$$

where:

- $\mathbf{y} = [y_0, y_1, \dots y_{n-1}]^T \in \mathbb{R}^n$ is the output vector;
- $X \in \mathbb{R}^{n \times p}$ is the so-called design matrix or feature matrix:

$$X = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{p-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{p-1} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{p-1} \end{bmatrix};$$

- $\beta = [\beta_0, \beta_1, \dots \beta_{p-1}]^T \in \mathbb{R}^p$ is a parameter vector containing the linear regression coefficients. They are the unknown part of the problem;
- $\epsilon = [\epsilon_0, \epsilon_1, \dots \epsilon_{n-1}]^T \in \mathbb{R}^n$ is a vector of random error. We assume that ϵ_i are independent random variables and $\epsilon_i \sim_{iid} N(0, \sigma^2)$.

2.2 Ordinary Least Squares

For OLS we consider the following cost function:

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y_i})^2 = \frac{1}{n}\{(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}})\}.$$

By using the convexity of $C(\beta)$, we obtain its minimum by deriving $C(\beta)$ w.r.t. β and equating the term to zero. The following equation then minimizes the cost function:

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

2.2.1 Statistical Background

The derived hat matrix $H := X(X^TX)^{-1}X^T$ denotes the orthogonal projection to the column space of X. These two properties can be described by the following matrix conditions: $H = H^2$ gives H its projection property. Intuitively speaking, the matrix multiplication by $\mathbf{H}$ projects every $\mathbf{y}$ to the closest $\tilde{\mathbf{y}}$ that is part of the column space: $\mathbf{Hy} = \tilde{\mathbf{y}}$. One would intuitively say that the closest element of the subspace to a element part of the subspace itself, is itself. And this is what the expression $H = H^2$ denotes.

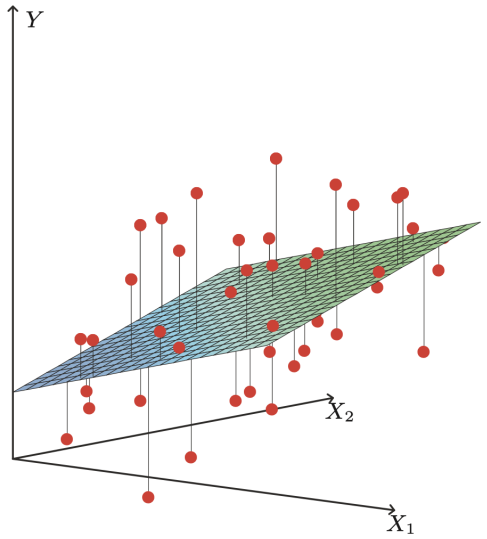

Figure 1: Figure 3.1 [1]: Visualization of orthogonal projection

$H = H^T$ implies the orthogonality of H. Visually this means that our errors $\tilde{\mathbf{y}} - \mathbf{y}$ are vectors parallel to the y axis. In other words, we define our error not as the minimal distance to the model but as the distance between our $\mathbf{y}(X) = f(X) + \epsilon$ and the model evaluated at X: $\tilde{\mathbf{y}}(X) = f(X)$.

The fitted coefficients $\hat{\beta}$ are unique, if the designmatrix X is full of rank. Otherwise X^TX is singular and therefore not invertible. To avoid treating singular matrices and using algorithm such as the singular value decomposition, we e.g. generate a random meshgrid (x, y) to construct the designmatrix used for fitting linear regression to the Franke function. With this, we most

certainly avoid linear dependent columns (x, y) in our designmatrix.

In the case of a singular matrix X, we are not able to invert $X^T X$, but for sufficiently large λ the matrix $(X^T X + \lambda \mathbb{I}_{p\times p})$ is positive definite and thus invertible. This property is one of the reasons we later introduce Ridge regression. Another one, motivating not only Ridge, but also Lasso regression, is numerical stability. To explain this, we first have to dive into shrinkage parameters and regularization resp. penalization terms.
As noted in 2.1 a particular regression model is determined by its cost function. In the following sections 2.3, 2.4 we introduce cost functions for Ridge and Lasso regression. Each of them consist of the OLS cost function extended by a regularization term of the form $\lambda\|\beta\|_2 = \lambda\beta^T\beta$ resp. $\lambda\|\beta\|_2 = \sum_{i=0}^{n-1} |\beta_i|$. We refer to λ as the shrinkage parameter and recall that β denotes the fitted coefficients of our model. In OLS, a slight change in the observed output may cause a significant shift in the fitted parameters β. The penalization term hinders the β from exploding, simultaneously creating a bias towards zero for coefficients in β, the latter giving λ its name. To sum it up, the penalization terms cause a shift of the fitted model towards high bias and low variance, referred to as underfitting and elaborated in detail in 2.8. This manifests itself in a higher tending and thus improved $R2$ score for Ridge and Lasso regression, compared to OLS.

2.2.2 Distribution properties of OLS, results of the paper and pencil exercise

Firstly we recall the following two identities: $\mathbf{y} = f(X) + \epsilon$ and $f(X) = X\beta$, as well as two properties of the variance: $Var(Y + a) = Var(Y)$ and $Var(aY) = a^T Var(Y)a$, where Y is a n-valued random variable and $a \in \mathbb{R}^n$ is a non-stochastic vector. The last property follows by applying the linearity of the expectation value to the calculation of the covariance twice.
The linearity of the expectation and the non-stochasticity of $X_{i,*}\beta$ then give:

$$E[y_i] = E[f(X_{i,*}) + \epsilon_i] = E[X_{i,*}\beta] + E[\epsilon_i] = X_{i,*}\beta = \sum_{j=0}^{p-1} X_{ij}\beta_j.$$

Using that $X_{i,*}\beta$ is deterministic and thus the variance is invariant under its translation, we can reduce the variance of y_i to the variance of ϵ_i. Finally we apply a distribution property of ϵ_i:

$$Var(y_i) = Var(X_{i,*}\beta + \epsilon_i) = Var(\epsilon_i) = \sigma^2.$$

We can interpret the deterministic $f(X_{i,*})$ as a normal distributed random variable with variance equal to zero. Then, y_i is, as the sum of two normal distributed random variables, normally distributed as well. With the calculations above, it follows: $y_i \sim N(X_{i,*}\beta, \sigma^2)$.

Particularly referring to OLS now, we derive the distribution that the fitted vector of parameters $\hat{\beta}$ follows:

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T y = (X^T X)^{-1} X^T (f(X) + \epsilon).$$

We first apply linearity and then use $E(\epsilon) = 0$:

$$E[\hat{\beta}_{OLS}] = (X^T X)^{-1} X^T X\beta + (X^T X)^{-1} X^T E[\epsilon] = \mathbb{I}_{p\times p}\beta = \beta.$$

To derive $Var(\hat{\beta}_{OLS})$, we firstly recall that the covariance of a vector of random variables $X = [X_1, X_2, \ldots, X_p]^T$ is defined as the matrix, s.t. $X_{ij} = Cov(X_i, X_j)$ for $i, j \leq p$ and thus by $Cov(X_i, X_i) = Var(X_i)$ the elements on the diagonal equal the variances of the variables X_i. Furthermore, we note that $Var(X)$ and $Cov(X)$ are used interchangeably. Therefore, by $\epsilon_i \sim_{iid} N(0, \sigma^2)$, we obtain that $Cov(\epsilon_i, \epsilon_j) = 0$ for $i \neq j$, by the independence of the ϵ_i and $Var(\epsilon_i) = \sigma^2$ by definition. Then, the vector $\epsilon = [\epsilon_0, \epsilon_1, \ldots \epsilon_{n-1}]^T$ has the variance matrix $Var(\epsilon) = \mathbb{I}_{n\times n}\sigma^2$.
Applying the two above noted properties of the variance, we obtain:

$$\begin{aligned} Var(\hat{\beta}_{OLS}) &= Var((X^T X)^{-1} X^T (X\beta + \epsilon)) \\ &= Var((X^T X)^{-1} X^T \epsilon) \\ &= (X^T X)^{-1} X^T Var(\epsilon) X((X^T X)^{-1})^T \\ &= \sigma^2 (X^T X)^{-1} X^T \mathbb{I}_{n\times n} X (X^T X)^{-1} \\ &= \sigma^2 (X^T X)^{-1}. \end{aligned}$$

Hence, by knowing $E[\hat{\beta}_{OLS}] = \beta$ and $Var(\hat{\beta}_{OLS}) = \sigma^2 (X^T X)^{-1}$, we can define confidence intervals for our parameters β_i, $i \leq p-1$ and therefore e.g. validate our model's results and estimate errors.

2.3 Ridge Regression

For Ridge regression we consider the following cost function:

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y_i})^2 + \lambda \sum_{i=0}^{n-1}\beta_i^2 = \frac{1}{n}\{(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}})\} + \lambda\beta^T\beta.$$

The added $\lambda\beta^T\beta$ is called the regularization term. By using the convexity of $C(\beta)$, we obtain its minimum by deriving $C(\beta)$ w.r.t. β and equating the term to zero. The following equation then minimizes the cost function:

$$\hat{\beta}_{Ridge} = (X^TX + n\lambda I)^{-1}X^Ty.$$

Since we can choose λ freely, we define $\tilde{\lambda} = n\lambda$ and then minimize. The result is the following equation:

$$\hat{\beta}_{Ridge} = (X^TX + \tilde{\lambda} I)^{-1}X^Ty.$$

2.4 Lasso Regression

For Lasso regression we consider the following cost function:

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y_i})^2 + \lambda \sum_{i=0}^{n-1}|\beta_i| = \frac{1}{n}\{(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}})\} + \sum_{i=0}^{n-1}|\beta_i|$$

Due to the regularization term, we cannot, unlike for OLS and Ridge regression, derive an analytic formula for the optimal β. Thus, we have to rely on numerical methods for minimizing $C(\beta)$. In this project we make use of the matching *scikit-algorithm*.

2.5 Accuracy of the model

Evaluating the accuracy of a model is fundamental in the field of machine learning, where it is normal to test and compare different models. In fact, the accuracy of a model tells us how well this model is predicting the values that we want to predict, suggesting us how well it is performing.
The idea behind accuracy is to compare the obtained predictions with the true target values. Then, the final aim is to find a way to maximize the accuracy of the model, by minimizing the dissimilarity between the predictions and the actual target values.
There are several ways to evaluate the accuracy of a model, but for sure the

most famous are the Mean Squared Error (MSE) and the coefficient R^2. As already mentioned in 2.1, in this work, we used the MSE as a way to compare different methods, thus defining the best method as the one that minimizes the test MSE.

2.5.1 Mean Squared Error

The Mean Squared Error (MSE) of a model measures the average of the squared errors, that is, the average squared difference between the predicted values and the actual target values. The MSE of a sample of n data points is computed as:

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

Since this quantity is a sum of squared differences, it is always positive. The lower the MSE is, the better the accuracy of the model is.
The MSE is one of the most widely spread measures for model validity. Its popularity, compared to the relative or absolute error, is driven by a property that follows from squaring the differences: small deviations, referring to deviations smaller than one, are weighed less; while large deviations, deviations larger than one, are punished by the squaring. While this is desired when e. g. fitting a polynomial to a continuous function, it is not necessarily desired when fitting a model to stochastic outputs. For the latter, outliers in the output data can occur, in which case the model is desired to actually not interpolate those, but rather to recognize the underlying trend in the data. Therefore, working with stochastic output, the large penalization of high deviations requires for treating outliers with e.g. scaling.

2.5.2 Coefficient R^2

The coefficient of determination, or R^2, is a measure that provides information about how good the fit of a model is. It provides a way to evaluate how well future samples are likely to be predicted by the model. In particular, in the regression context, it allows to measure how well the regression line approximates the actual data.
If $\tilde{y}_i$ is the predicted i^{th} value of the sample and y_i is the corresponding true value, then the R^2 score is defined as:

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

where we have defined the mean value of the observed data $\mathbf{y}$ as:

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

The coefficient of determination is upper bounded by the value 1, but it can be also negative. This happens because the model can be arbitrarily worse than the worst possible least-squares predictor (equivalent to a horizontal hyperplane at a height equal to the mean of the observed data).

2.6 Train and test data

When applying Machine Learning models, it is common practice to split the available data into two sets: a training set and a test set. Sometimes, even a third set is added, the so-called validation set, but this is not the case. Intuitively, the idea behind this split is to use the training set to train the model, and the test set to test the model (through an evaluation of the model's accuracy).
Even if there is not a general and correct rule about how much data should be included in each set, usually the two sets are not composed by the same number of data. A common choice is to use 80% of the data to form the training set, and the remaining 20% to form the test set.

2.7 Scaling data

Feature scaling in machine learning is one of the most critical steps encountered during the pre-processing of data, before the creation a model. When dealing with real data, it is common to have features with very different units and numerical scales. Moreover, we previously said that the MSE, as many other Machine Learning methods, is largely affected by the scale of data. For this reason, it is very common to scale the features, in order to make them comparable and avoid outlier values.
There exist different techniques to scale data, but the most used are:

- Standardization: it involves subtracting the mean and dividing by the standard deviation of the dataset. For each feature:

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \bar{x}_j}{\sigma(x_j)},$$

where $\bar{x}_j$ and $\sigma(x_j)$ are the mean and standard deviation, respectively, of feature x_j. This way of scaling allows to obtain all features with a mean equal to 0 and a standard deviation of 1. For the datasets where we don't have the standard deviation or we don't wish to calculate it, it's common to simply set it to one (in this case, we just subtract the mean value).

- Min-Max scaling: it is generally used to impose that the features are in a certain interval. To scale each feature to the interval $[a, b]$, we can apply the following formula:

$$x_j^{(i)} \leftarrow (b-a)\frac{x_j^{(i)} - min(x_j)}{max(x_j) - min(x_j)} - a,$$

where $max(x_j)$ and $min(x_j)$ represent the maximum and the minimum value of $x_j^{(i)}$ over the dataset.

2.8 Bias-variance trade-off

The bias-variance trade-off is a important topic in machine learning. Before going into the details, it's important to clarify what *bias* and *variance* are:

- **Bias**
  Bias is the measure of how well the model is able to represent the complexity of the data. For instance, low bias means that the model represents the data very well, meanwhile high bias means that the model is oversimplified, therefore not able to represent the data accurately enough.
  We can also define the bias as:

$$Bias = \mathbb{E}[(y - \mathbb{E}[\tilde{y}])^2].$$

- **Variance** Variance is the measure of how much the model changes when training with different portions of the dataset. A low variance implies a great coherency of the model in every subset that is used for training. A high variance means that the model changes significantly depending on the subset it has been trained with.
  We can also define the variance as:

$$Var = \mathbb{E}[(\tilde{y} - \mathbb{E}[\tilde{y}])^2].$$

After this brief introduction, we can finally take a look at the bias-variance trade-off.
When we choose our model it's very important to take into account the values of bias and variance. As shown in the following image, it's impossible to choose a perfect model, as we can't have low bias and variance at the same time. If we choose an $underfitting$ model (high bias, low variance) it will be very good at being consistent with the different subsets, but it will be oversimplifying the data. If we instead choose an $overfitting$ model (low bias, high variance), the opposite will happen. The reason why $overfitting$ comes in place is that, by increasing model complexity, the model works really well with the training sample, but its performance with the test sample becomes worse, since it is less trained to adapt to new data samples. A good choice is obviously a more balanced model, that is a model with a middle ground in bias and variance values.

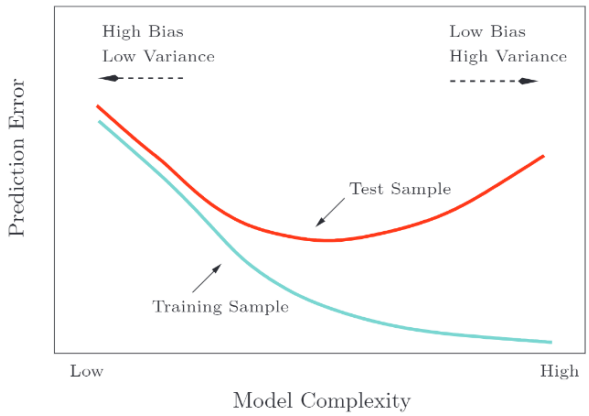

Figure 2: This image [1] shows the relation between the Mean Squared Error and the degree of the model. We notice very different trends for the training and the test dataset. The two lines start off together, until they split and follow different trends: the MSE of the training set decreases, meanwhile the MSE of the test set starts to increase. This is explained by considering bias and variance: on the left side of the graph we have $underfitting$, on the right side we have $overfitting$.

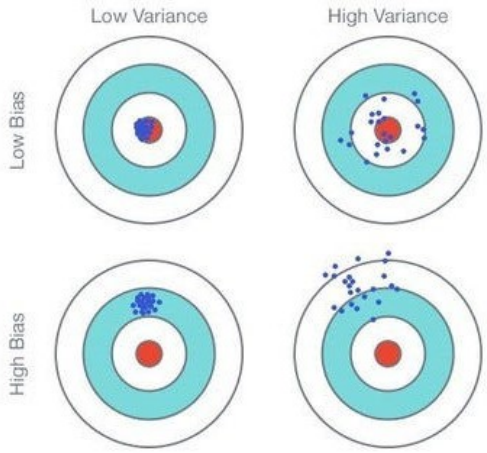

Figure 3: This image shows another way of visualizing the bias-variance trade-off (Fig.1 from [7]) .

2.8.1 MSE decomposition into bias-variance

Consider a dataset consisting of the following data:

$$X = \{(x_i, y_i), i = 0, ..., n-1\},$$

where the data is generated by the noisy model

$$y = f(x) + \epsilon, \quad where \quad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

We find the optimized β parameters for our model by optimizing the cost function of the Mean Squared Error

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

We can show that we can rewrite this in terms of the variance of the model, its bias and the variance of the noise. In other words:

$$\mathbb{E}[(y - \tilde{y})^2] = Bias[\tilde{y}] + Var[\tilde{y}] + \sigma^2.$$

To prove the above formula, we can start by rewriting the cost function as:

$$\mathbb{E}[(y - \tilde{y})^2] = \mathbb{E}[(\tilde{y} + \epsilon - \tilde{y})^2].$$

We then add and subtract the expected value of $\tilde{y}$:

$$\mathbb{E}[(y - \tilde{y})^2] = \mathbb{E}[(\tilde{y} + \epsilon - \tilde{y} + \mathbb{E}[\tilde{y}] - \mathbb{E}[\tilde{y}])^2].$$

Now we can group the argument of the expected value so that we obtain:

$$\mathbb{E}[(y - \tilde{y})^2] = \mathbb{E}[(y - \mathbb{E}[\tilde{y}])^2] + \mathbb{E}[(\tilde{y} - \mathbb{E}[\tilde{y}])^2],$$

and finally:

$$\mathbb{E}[(y - \tilde{y})^2] = Bias[\tilde{y}] + Var[\tilde{y}] + \sigma^2.$$

2.9 Resampling methods

Resampling techniques have central importance in modern statistics. They work by drawing samples from a training set and fitting a model multiple time in order to get more insights about the model itself. In this way, we can learn things about the model that we wouldn't be able to if we just used the original training data once.
The resampling methods that we are going to look at are "$k - fold\ cross - validation$" and "*bootstrapping*".

2.9.1 k-fold cross-validation

The k-fold cross-validation is a popular resampling technique. It involves randomly dividing our data set in a definite number of subsets ($k - folds$) and using each fold as test data while using the remaining subsets as training data. Then, we can calculate individual errors for each prediction (for each fold), and we compute the average.
An extreme case of k-fold cross-validation is the "leave one out" method where only one data point is used for testing. This method can be more robust, but the number of iterations is computationally demanding. Hence, using 5-10 folds strikes a practical balance between computational efficiency and robustness of the error estimation.

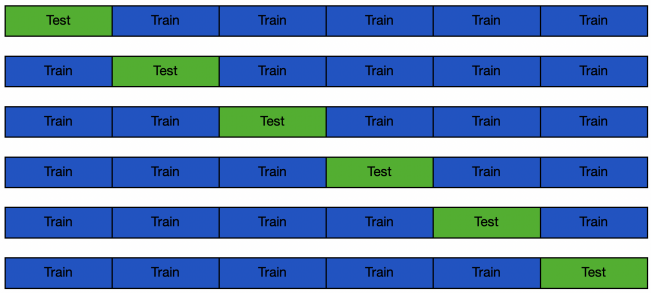

Figure 4: Example of division into subsets for $k - fold$ cross validation : $k = 6$

In this way, we use all the dataset to make our predictions.

2.9.2 Bootstrap

Bootstrap is another popular resampling method. It works by taking a set number of samples and deriving how much our output can change based on the subsets. Instead of relying solely on one sample, bootstrap allows you to simulate many different samples from your data, which helps you better understand the reliability of your conclusions.

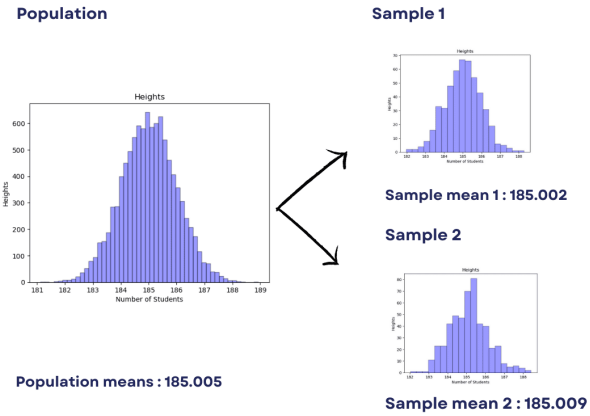

Figure 5: [6] Example of division into subsets, and the different means that we get in the different samples. This gives us an idea of the variance of our model: by taking different samples, we get different outputs.

3 Results

In contrast to the later following application of linear regression methods to Terrain data, we will first fit a model to the Franke function, where the underlying function describing our data is nearly known, with the 'nearly' referring to the added stochastic noise. We want to draw attention to the slightly different approach and usage in fitting linear regression to a known continuous function, compared to fitting it to a data set following an unknown pattern and noise.
When fitting a polynomial to a continuous function with linear regression, we aim at finding a model with uniformly distributed deviations $\mathbf{y} - \tilde{\mathbf{y}}$. We perform an approximation and abstract the pattern our data follows. Thu,s we are willing to accept small errors but try to avoid larger ones, since this means losing information about the underlying trend. By its definition the MSE emphasizes larger deviations by higher punishment.
In contrast, when fitting linear regression to a data set with an unknown pattern, we try to separate the underlying trend from the added noise. We most certainly do not want to model all outliers of the data, even though the MSE pushes the model to this extent. To get rid of that partly unwanted bias to interpolate outliers, we can scale our data. The stressed punishment only applies to deviation larger than one, so we can solve this problem by e.g. *min-max scaling* the output values lie in the interval [0, 1). Therefore, we chose the above scaling for fitting linear regression to the terrain data.

3.1 Franke function

In this part, we examine the fitting of OLS, Ridge and Lasso regression, while varying the polynomial degrees between one and eight, to the Franke function.
The definition of the Franke function is the following:

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

where both x and y are considered between 0 and 1.

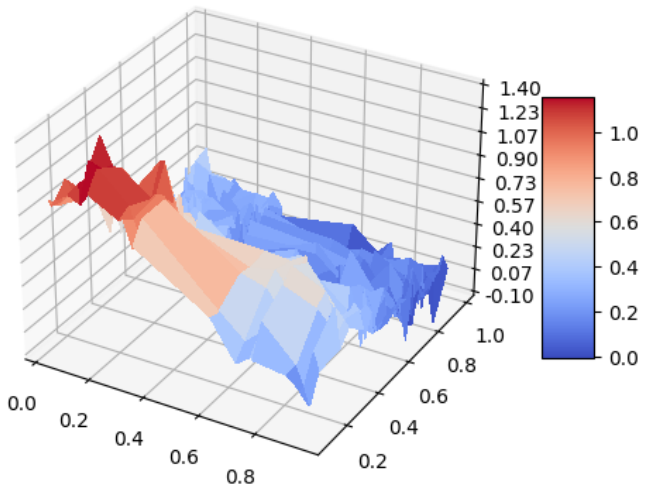

Figure 6: Plot of the Franke function with added stochastic noise.

In this exercise we only applied limited scaling to the data by centering the input and output data X resp. z. Firstly, we subtracted from the design matrix X its column-wise mean. We then calculated our model's intercept by using the expression from the lecture notes [4], which results from deriving the cost function, sometimes also referred to as loss function, by β_0 and setting it to zero. Due to the convexness of our cost function, this led us to find its minimum with respect to β_0. It follows the equation: $z_{intercept} = E_{emp}(z_{train}) - E_{emp}(X_{trainscaled}) * \beta$. Note that $X_{trainscaled}$ denotes a matrix and thus in this case we define the expectation as the vector consisting of the column-wise expectation values. Since we above centered our design matrix X, the mean vector of $X_{trainscaled}$ equals the zero vector. Our equation reduces to: $z_{intercept} = E_{emp}(z_{train})$. Lastly, we scaled the output by subtracting the calculated intercept $z_{intercept}$. Thus the intercept of any OLS, Ridge and Lasso model fitted to the scaled data equals 0.
Removing the intercept mainly impacts Ridge and Lasso Regression, where the fitted coefficients β appear in the regularization term and thus impact the cost function. The performed scaling allows us to fit the model without β_0, and therefore eliminating its punishment of the cost function.
Since our produced data is evenly distributed in the interval [0,1), there is no need to identify and treat outliers, thus we do not scale by dividing by the standard deviation. The analogue scaling of the output data follows the

same reasoning.
We perform splitting the data in test and train data once, and then apply the OLS, Ridge and Lasso fitting with polynomials for each degree. The uniformly splitting of the data allows us a better comparison of the MSEs later on.
As reasoned in the method section, we generate random input vectors x and y to aim at avoiding perfect correlation of the columns of our design matrix X.

3.1.1 Fitting OLS

After fitting OLS regression, we crated two plots to study the behavior of β and the relationship between polynomial degree, MSE and coefficient R^2.

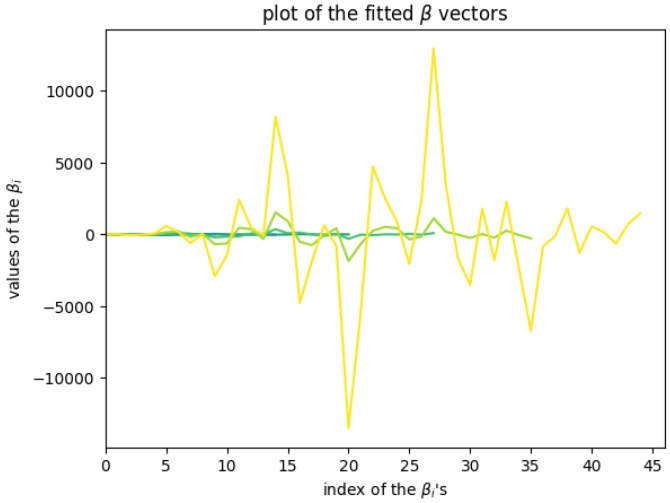

Figure 7: An intuitive approach to the Bias-Variance trade-off: Regarding the beta values as a function of the degree of the fitted polynomial, we recognize a high variance in beta for the degrees 7 and 8. This allows us to assume that modelling our data with a polynomial degree of 7 or higher is likely to be an overfitted model.

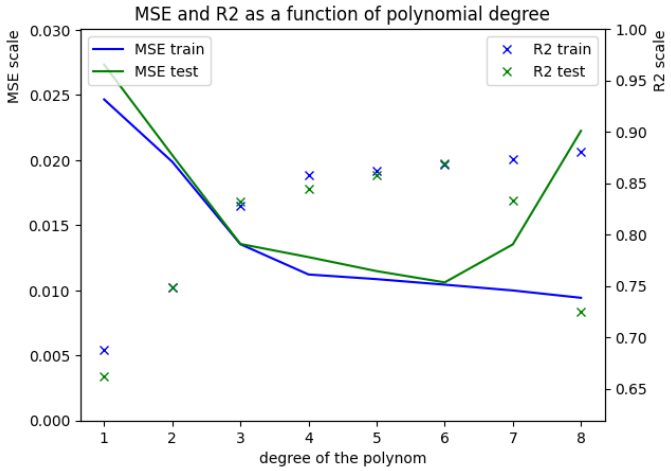

Figure 8: The polynomial OLS fit of degree six generated the lowest MSE on the test data and thus is our optimal model.

Train data and test data MSE decline at first, as an increased model complexity captures the underlying trend of the data more precisely. This allows us to conclude that increasing the degree improves our prediction, implying that those fits illustrate the High Bias, Low Variance area. Aligning with the intuitive approach above, we observe an increasing of the test data MSE for fitted polynomials with degree seven and higher, while the train data MSE declines, as expected and in the methods section, further with higher degrees. The polynomial fit of degree six and higher depicts the typical Low Bias, High Variance area.
Fitting a polynomial of degree six therefore generates the optimal results with a test data MSE of 0.010611. The train data MSE is with 0.009426 at its lowest at the polynomial degree eight.

3.1.2 Fitting Ridge regression

We fitted the Ridge regression for 20 λ values in the interval $[0.0001, 206.9138]$. By using the logspace operator the majority of values of λ lie closely to the value 0.0001.

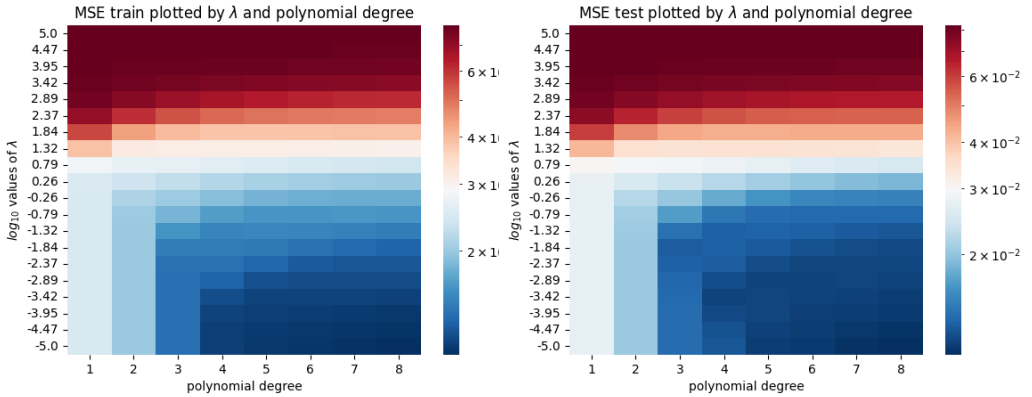

Figure 9: A heatmap to visualize the MSE of the train data (left) as well as the test data (right) as functions of the polynomial degree and lambda: Both the train data and the test data MSE reach its optimum for the parameters [lambda, polynomial degree] = [0.00001, 8], where the train data MSE is at 0.010594 and the test data MSE at 0.010737.

As Figure 9 states, our optimal model is given by performing the Ridge regression with the parameters [λ, polynomial degree] = $[0.0001, 8]$, which gives us a test MSE of 0.010737. We note that the OLS fitting performed slightly better, recalling that the optimal model, OLS fitting with polynomial degree six, generated a test data MSE of 0.010611.

3.1.3 Fitting Lasso regression

The same analysis can be done for Lasso regression. In this case, our model's test data MSE for optimal parameters [lambda, polynomial degree] = [0.0000335982, 6] denotes at 0.011642. The optimal train data MSE is given by the model denoted by [lambda, polynomial degree] = [0.0001, 5]. Due to the regularization term, the train data MSE does not, in contrast to OLS, decrease in the degree of polynomials.

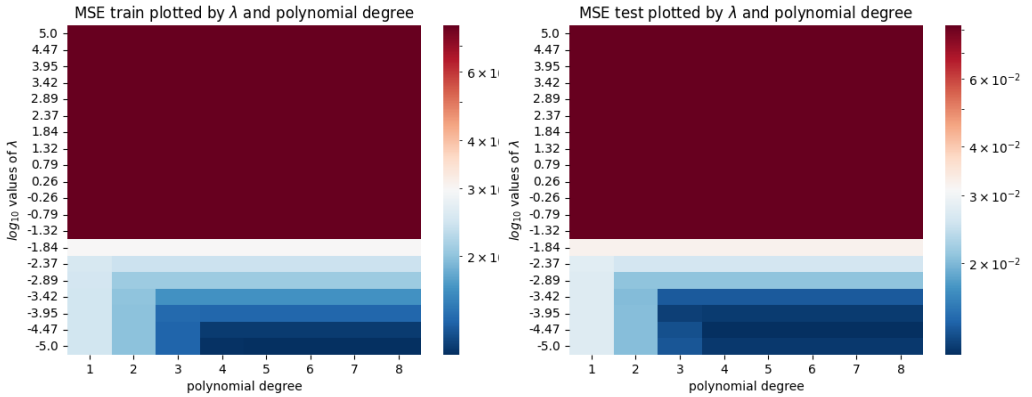

Figure 10: A heatmap to visualize the MSE of the train data (left) as well as the test data (right) as functions of the polynomial degree and lambda: The MSE of the train data is the lowest for the parameters [lambda, polynomial degree] = [0.0000335982, 6].

The heatmap hints at a constant MSE, on the train as well as test data, for large values of λ. In fact this is true and corresponds to $\beta = 0_{p\times 1}$ for all $\lambda \geq 0.04833$ and all fitted polynomial degrees. This means that the punishment by the regularization term for the above noted λ is so high that the choice $\beta = 0_{p\times 1}$ in fact minimizes the cost function, which then equals:

$$C(0_{p\times 1}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 + \lambda \sum_{i=0}^{n-1}|\beta_i| = \frac{1}{n}\sum_{i=0}^{n-1} y_i^2 \ .$$

3.1.4 Comparison of models

In our performed fittings, the OLS Regression with the parameter [polynomial degree] = [0.0001, 6] generates the best result with a test MSE of 0.0106112. Below, the plot of all models fitted with polynomial degree six shows the behaviour of the test data MSE as a function of λ.

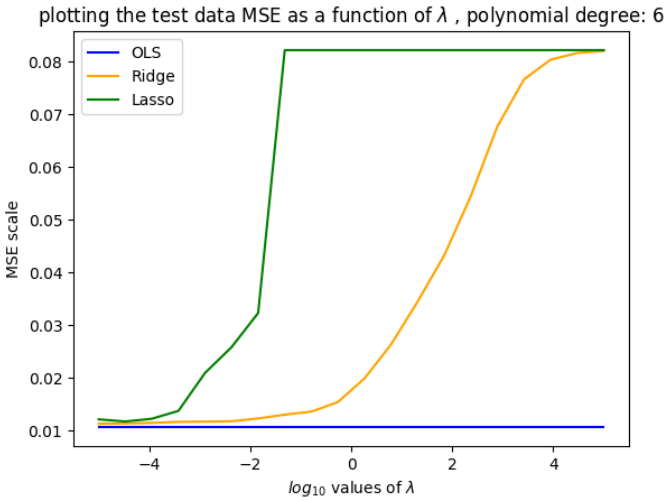

Figure 11: The graph captures the fast increasing of the test data MSE in augmenting λ. It also shows the faster increasing of Lasso regression compared to Ridge, as mentioned above.

3.1.5 Bias-variance trade-off

We now examine the bias-variance trade-off for OLS regression, by studying the relationship between the Mean Squared Error and the complexity of our model (the degree of the polynomial). Our goal is to recreate Figure 2 by using actual data.
We proceed by calculating the MSE for every degree of the polynomial. We plot the data, putting MSE on the y-axis and the degree of the polynomial on the x-axis. We use the logarithm with base 10 to make the reading of the graph easier.

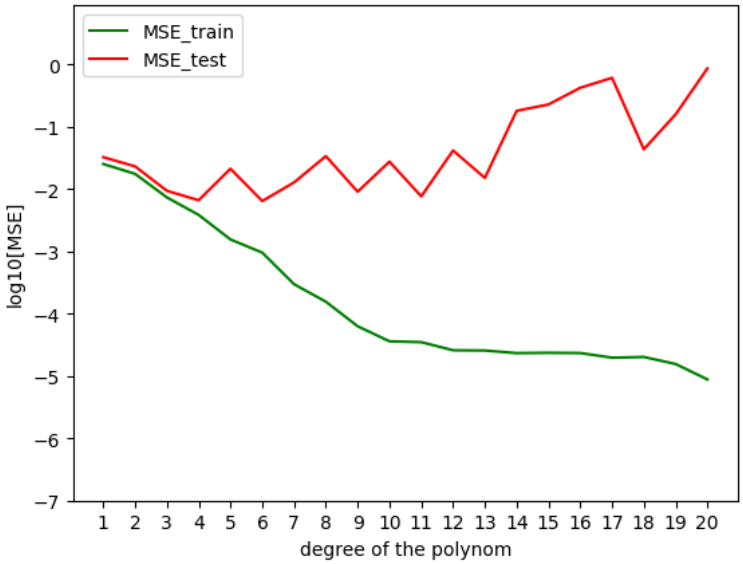

Figure 12: We notice the same behaviour described in the paragraph 2.8, obviously the curve is less smooth.

For the first four degrees, the two curves are basically the same. This is caused by high bias and low variance (underfitting, low complexity of the model). From the fifth degree onward the curves are splitted, taking completely different paths: the test MSE increases, while the train MSE decreases and then becomes more or less stable. This behavior is a consequence of low bias and high variance (overfitting, since the model is too complex).

3.1.6 Bootstrap

Still focusing on OLS regression, the next step consists in analyzing the values of bias, variance and error based on the complexity of our model. We decided to operate with 100 bootstrap samples.
We proceed by generating data and setting the number of 100 samples. We then calculate the value of error, bias and variance for every degree of the polynomial. The results are shown in the following plot:

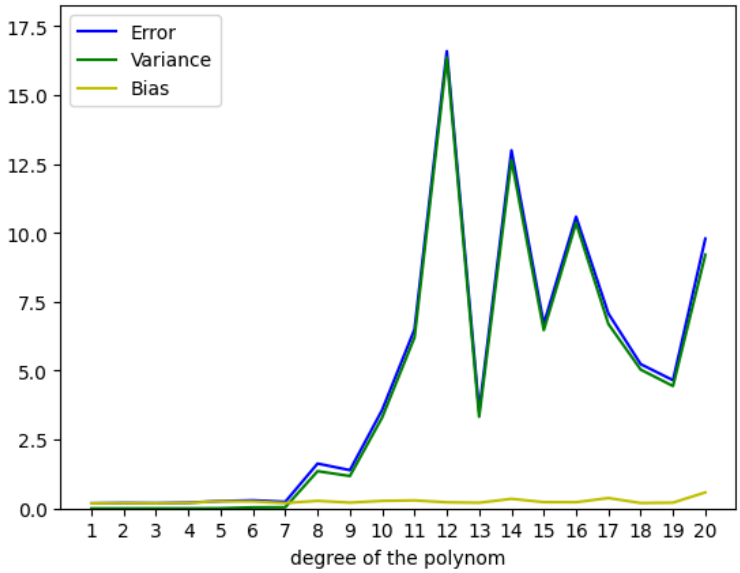

Figure 13: We notice that bias remains constant, with a very low value. Meanwhile, up to the seventh degree, the error is close in value with the bias, since the variance is close to zero. From the eighth degree onward, the error and the variance increase and go in pair.

This figure depicts perfectly the relationship between variance and bias, in particular from degree 8 onward, where the variance increases sharply (overfitting). As a consequence, the error also reaches high values, as its behavior directly depends on the variance.

3.1.7 K-fold cross-validation

By adding k-fold cross-validation, we tried to make our model more robust i.e. reducing the impact of outliers due to the division into subsets on the MSE. We used numbers of folds between 2 and 10. When we increase the number of folds, the mean MSE is less affected by potential outliers in the individual MSEs, because they can compensate, and they can't have an important weight.

By computing the absolute difference between the MSE for $k = 10$, which should be the most stable one, we can distinguish larger deviations for small values of k. However, this difference is lower than $0.15 \times MSE$ for $k = 5$. Increasing the number of fold will therefore always make the results more

stable, but we can reasonably choose a number of folds between 5 and 10 in order to avoid expensive calculations. In the following figures, we can observe that, for OLS, Lasso and Ridge, when we increase the number of folds, the difference to MSE for $k = 10$ decreases, so the MSE becomes more stable. We can still have greater differences for $k_i > k_{i-1}$ due to the randomness of the division into subsets, but the tendency stays the same.

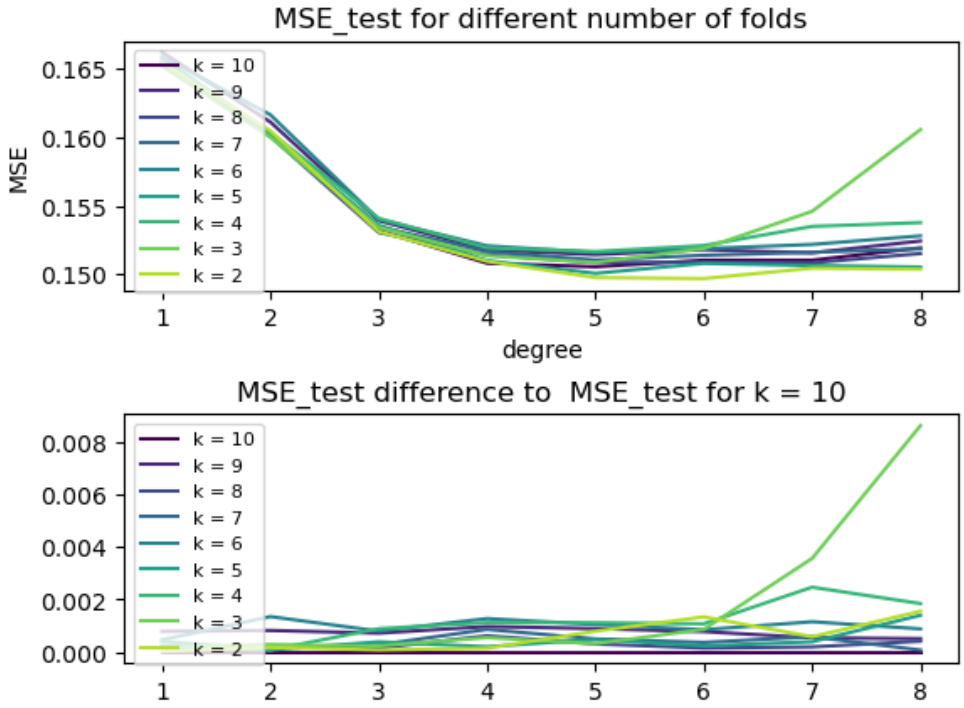

Figure 14: While the first graph shows the MSE with respect to polynomial degree, for different number of folds in $[2, 10]$, the second plot allows to understand better the influence of the choice of k on the MSE. It displays the difference between the MSE for k in $[2, 10]$ and the MSE for $k = 10$ for OLS regression. This result decreases as we increase the number of folds i.e. the MSE is getting more stable, but the improvement that can be made computing the regression with one more fold shrinks.

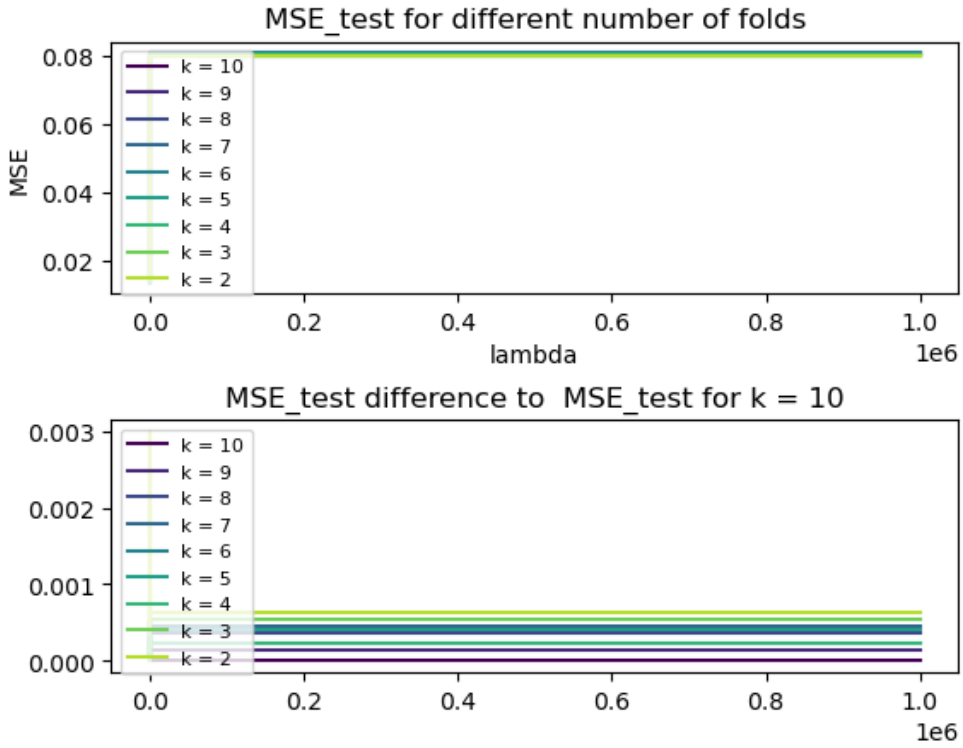

Figure 15: The first of the two graphs displays the MSE with respect to λ for Ridge regression for different values of k. The second plot shows the difference between the MSE for k in $[2, 10]$ and the MSE for $k = 10$ for Ridge regression. Decreasing the number of folders increases the difference of MSE, but the values are still very small (we are talking about differences of the order 10^{-3} or even less).

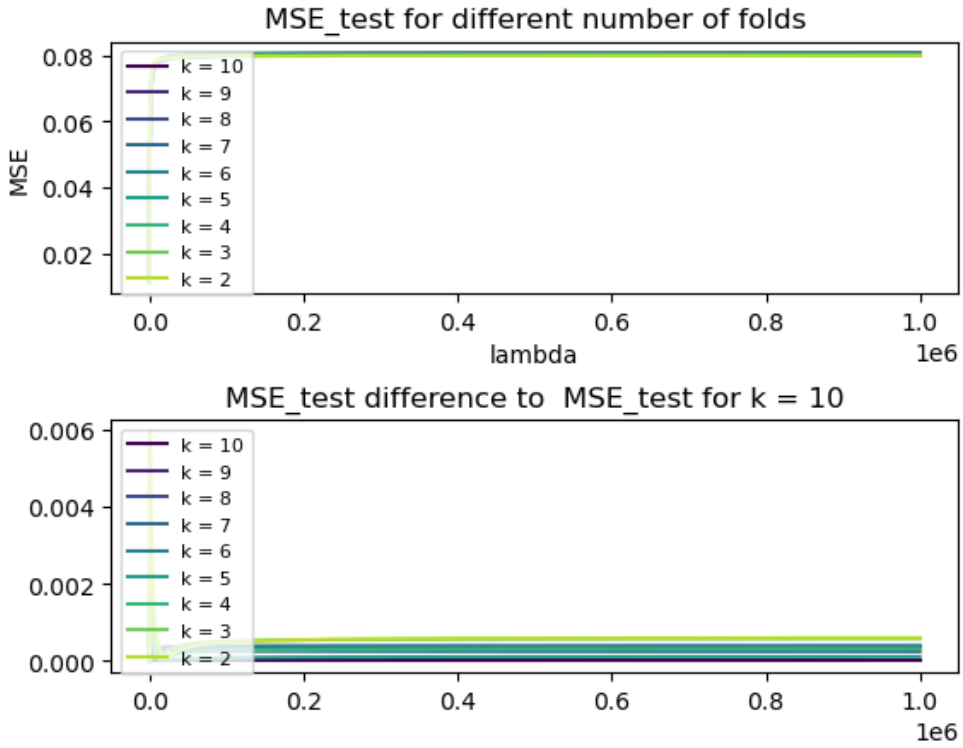

Figure 16: The first of the two graphs displays the MSE with respect to λ for Lasso regression for different values of k. The second plot shows the difference between the MSE for k in $[2, 10]$ and the MSE for $k = 10$ for Lasso regression. Decreasing the number of folders increases the difference of MSE, but the values are still very small (we are talking about differences of the order 10^{-3} or even less).

As mentioned before, increasing the number of folds, in this case, doesn't give a significant improvement to the MSE values (the difference between the curves is minimal). This is the reason why, in order to avoid expensive computations, we chose $k = 5$ for the analysis of the terrain data.

3.2 Terrain data

After the application of our models to an artificial and simple context - the Franke function - the next step is to study a real case. The data used in this section are digital terrain data about the altitude and the graphical coordinates of a region located near Stavanger, in Norway. The original dataset is composed of a grid of dimension 3601x1801.
The data can be represented by the following figures:

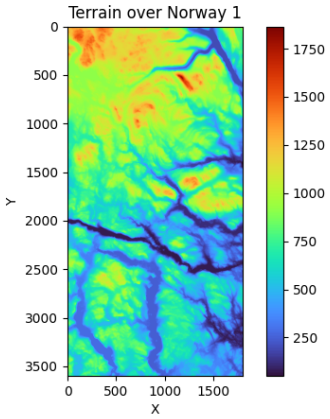

Figure 17: Terrain Data, 2D

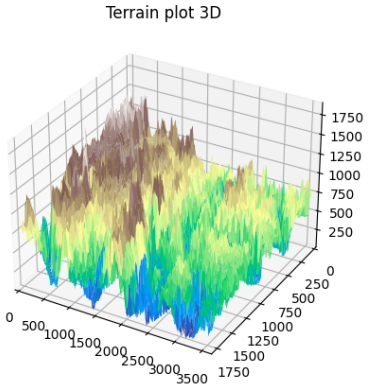

Figure 18: Terrain data, 3D

In order to avoid the complications related to the problem of dealing with a large number of data (due to the high dimension of the matrix to invert), we considered it useful to reduce the number of points to take into consideration in the analysis. For this purpose, we randomly selected 1500 points from the available data, with the aim of studying them in detail. Afterwards, we prepared the data by scaling both the inputs and the output, by implementing our own code for the min-max scaling and applying it to all the points. We decided to use min-max scaling so that all of our data take values in the interval $[0, 1]$. Here, it is necessary to apply this method because we have dimensions varying in different ranges: in this way, all the data can be compared without problems of any kind. The obtained points are represented in the following plot:

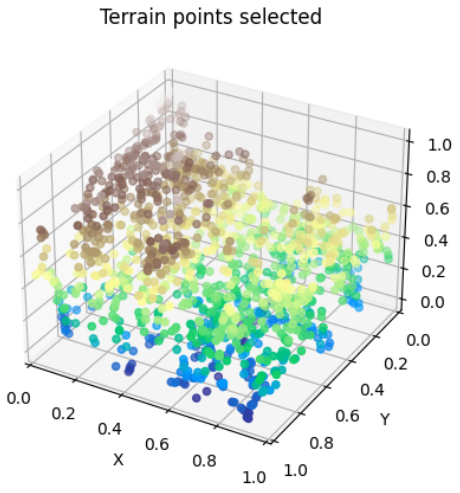

Figure 19: Representation of the randomly selected 1500 points from the data set

Before applying any other method, in order to complete the pre-processing phase, it is necessary to split our data into a training set and a test set. We used the already existent *sklearn* function (*train_test_split*), to create a test sample using 20% of the selected data.

The analysis has been developed both with and without the implementation of cross-validation as resampling technique. If the two versions of the code and the relative results can be found in the GitHub folder, here just the former is presented. Since the complete analysis and the comparison of the two versions has already been presented with the Franke function, here we just show the results obtained with a 5-fold cross-validation (the reasons for this decision can be understood from 3.1.7).

3.2.1 Fitting OLS

Firstly, we applied Ordinary Least Squares regression. The idea was to study the relationship between the MSE, the coefficient R^2 and the polynomial degree. The values have been calculated for polynomial degrees belonging to the interval $[1, 20]$.

Figure 20 shows the obtained results. Focusing on the plot on the left, we can observe that, in general, the test error is higher than the train error,

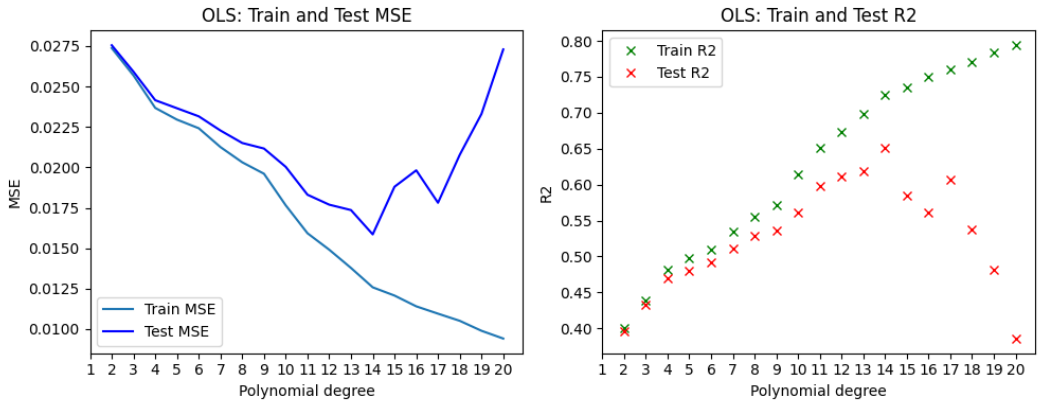

Figure 20: Representation of train and test MSE and R^2 for OLS regression. The train MSE, decreasing as complexity increases, results to be lower than the test MSE, whose values tend to become even worse from degree 15 onward (overfitting). The coefficient R^2 has a specular behavior: it increases for both train and test sets until degree 14. Then, again from degree 15, the test values start to get worse.

especially for higher complexities. The positive aspect is that, up to degree 14, both train and test errors have a decreasing trend, thus confirming that complexity reduces the MSEs. From degree 15 onward, while the train error keeps getting better and better, the test error starts to become worse than before. This behavior is due to the phenomenon of overfitting: since the model with high complexity is constructed to perform very well with the test data, when it deals with a new sample of points (the test set), its performance accuracy is reduced. The minimum value of the MSE for the test sample is obtained with a polynomial degree of 14. With this degree, also the train MSE is low.

Focusing on the R^2 plot, the values corresponding to the train set are generally better than the ones related to the test set. With both sets, the coefficient increases, going from degree 1 to degree 14. Again, starting from degree 15 there is a change in the behavior of the R^2 value related to the test set: it starts decreasing. This behavior reflects perfectly what we have just seen for the test MSE: the overfitting problem comes in place again.

3.2.2 Bias-variance trade-off

The phenomenon of overfitting can be further analyzed by studying the decomposition of the MSE into Bias and Variance.

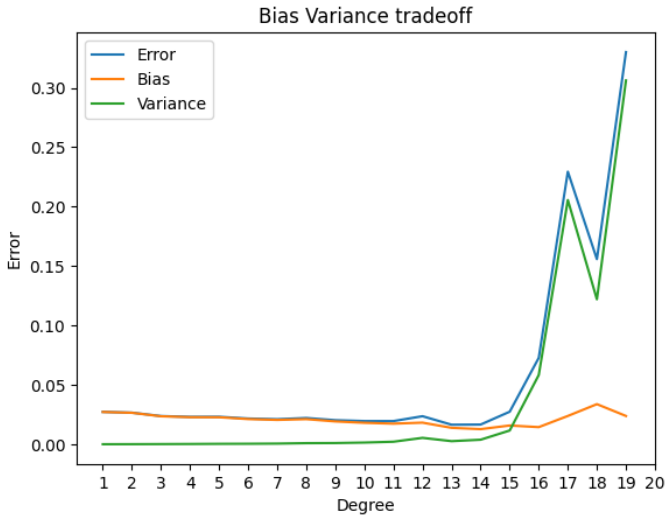

Figure 21: Bias-Variance trade-off for OLS: for low complexity the variance is 0 and the only contribution to the MSE is given by the bias. For large polynomial degrees, the variance reaches very high values, while the bias remains more or less stable. From degree 15 onward, we can see the phenomenon of overfitting.

The above figure shows exactly what we want to depict: the so-called Bias-Variance trade-off. Focusing on the left part of the plot, with small polynomial degrees, we can notice that the variance is equal to 0 (until degree = 11), while the bias is a bit higher. Here, only the bias gives a contribution to the error, meaning that we are "underfitting" our model. As the complexity of the model increases (i.e as the polynomial degree increases), the variance of the model, starting from very low values, increases sharply. This worsening in variance, particularly visible for the highest degrees, is due again to the overfitting problem. While the Bias is more or less stable, the variance is largely affected by a too close adaption of the model to the training set. As a consequence, even small changes in the data bring significant changes to the model for large polynomial degrees. The turning point where the overfitting phenomenon arises is around degree 14/15. This confirms what has been

seen in the previous plots: the test MSE and the test R^2 started to get worse around the same degree.

3.2.3 Fitting Ridge regression

The second step consisted in applying Ridge Regression: the model has been computed with 100 different values of $\lambda \in [10^{-5}, 10^5]$, but also with different polynomial degrees. The idea was to study how the MSE is influenced by variations in both λ and polynomial degree. For this purpose, the following table has been created:

Ridge: MSE and optimal Lambda per polynomial degree

Degree	MSE	log10(lambda)
1.0	0.026173	-0.4545
2.0	0.0247	-1.6667
3.0	0.023186	-1.4646
4.0	0.022787	-2.9798
5.0	0.02243	-5.0
6.0	0.021881	-5.0
7.0	0.02167	-5.0
8.0	0.021477	-5.0
9.0	0.021353	-5.0
10.0	0.021327	-5.0
11.0	0.021356	-5.0
12.0	0.02138	-5.0
13.0	0.02137	-5.0
14.0	0.02133	-5.0
15.0	0.021281	-5.0
16.0	0.021239	-5.0
17.0	0.021212	-5.0
18.0	0.021199	-5.0
19.0	0.021195	-5.0
20.0	0.021195	-5.0

Figure 22: Relationship between polynomial degree, test MSE and $log_{10}(\lambda)$ for Ridge regression: the error tends to decrease as the polynomial degree increases (even if the values are more or less the same from degree 9 onward). The majority of the optimal MSE for each degree is given by the same value of λ (10^{-5}).

The table shows the values of $log_{10}(\lambda)$ that minimize the test MSE for each polynomial degree, together with the correspondent MSE value. Even in this case, when the model complexity increases, the MSE tends to decrease. The optimal values of MSE are obtained with different values of λ, but almost

all of them correspond to $\lambda = 10^{-5}$. That is why we now show (Figure 23) the plot of MSE against polynomial degree for this value of λ, compared with the same graph plotted with a different value of λ. The goal is to see the difference between the two cases, by analyzing the behavior of the error with respect to the polynomial degree.

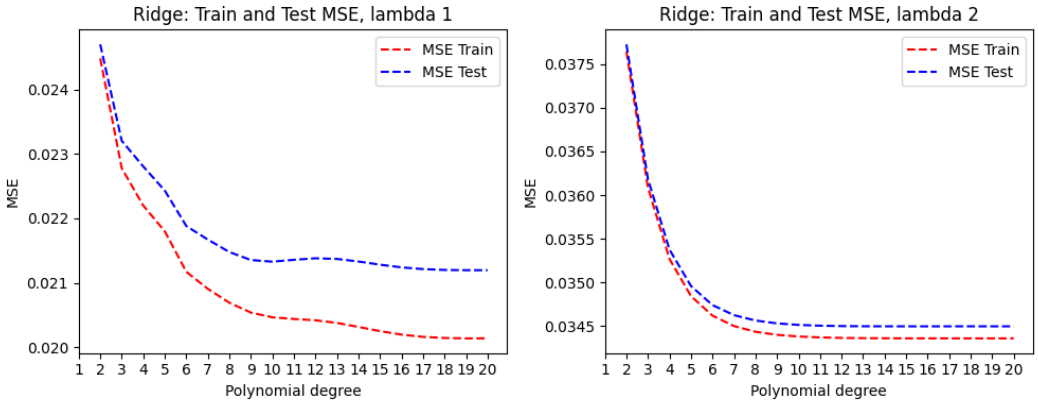

Figure 23: Representation of MSE for Ridge regression with different values of λ, according to the polynomial degree. The two plots are very similar: both train and test errors tend to decrease as complexity increases. The main difference is the gap between the train and the test MSEs: in the first case it is larger, while in the second case the two errors are very close to each other.

The figure on the left is computed with the first value of λ, $\lambda = 10^{-5}$, while the second one is obtained with an intermediate value: $\lambda \approx 954.5$, which correspond to the 80^{th} value of the tested lambdas. Looking at the two graphs together, it can be immediately noticed that their shape is similar: they both start with high levels of MSE for low polynomial degrees, but as long as the complexity increases, the errors decrease. With the first value of λ, the MSE is always lower than for the second case, which means that the first model is better than the second one. In both cases, the test error is worse than the train error. With $\lambda = 10^{-5}$, the difference between the train and the test MSEs is larger: with the second value, the errors, although higher, are very similar to each other, meaning that the model has a similar performance with both the train and the test sets in this case.

3.2.4 Fitting Lasso regression

The same analysis can be repeated with Lasso regression: again, the model has been studied with 100 different values of $\lambda \in [10^{-5}, 10^{5}]$, but also with different polynomial degrees.

As for Ridge Regression, we created a table (Figure 24) to compare the values of lambda minimizing the MSE for each polynomial degree. As we can see from the figure, even in this case, the majority of the best values of MSE are given by the same value of λ: $\lambda = 10^{-5}$. In general, we can notice that the MSE is larger than the values in Ridge's table, meaning that Lasso is working worse with our data.

Lasso: MSE and optimal Lambda per polynomial degree

Degree	MSE	log10(lambda)
1.0	0.026173	-5.0
2.0	0.0247	-5.0
3.0	0.023317	-4.2929
4.0	0.023284	-5.0
5.0	0.02324	-5.0
6.0	0.023176	-5.0
7.0	0.023102	-5.0
8.0	0.023025	-4.798
9.0	0.022968	-4.798
10.0	0.022928	-5.0
11.0	0.022903	-5.0
12.0	0.02287	-5.0
13.0	0.022835	-5.0
14.0	0.022807	-5.0
15.0	0.022786	-5.0
16.0	0.022771	-5.0
17.0	0.022761	-5.0
18.0	0.022756	-5.0
19.0	0.022754	-5.0
20.0	0.022754	-5.0

Figure 24: Relationship between polynomial degree, test MSE and $log_{10}(\lambda)$ for Lasso regression: again, the error decreases as the degree increases. Also here, the majority of lambda values are equal to 10^{-5}.

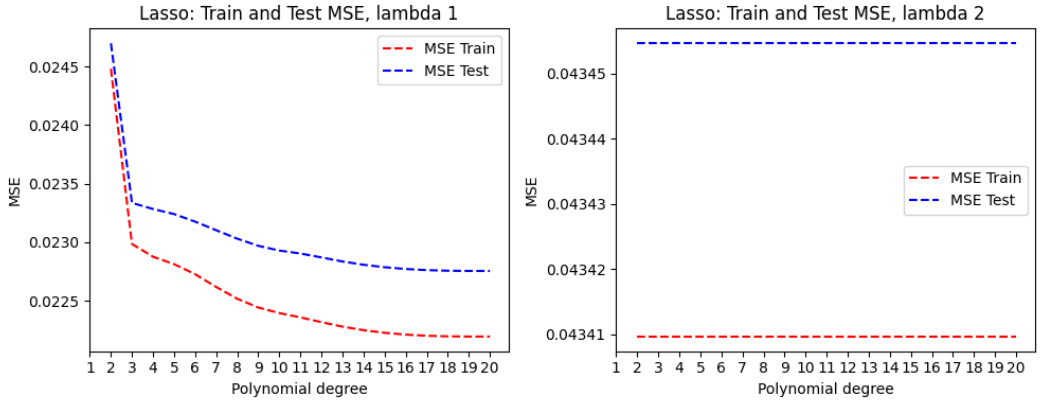

Figure 25: Representation of MSE for Lasso regression with different values of λ, according to the polynomial degree. The left graph is similar to the left one of Ridge: for larger degrees the test error is worse than the train error. On the contrary, the second plot is different: here the MSEs are stable and the test error is worse than the train error.

Following what has been previously presented with Ridge regression, Figure 25 shows the same graphs as before, obtained with the same values of λ. The idea is to compare the results and see what changes between the two methods. For the first plot, the result is more or less the same as before: both train and test errors decrease with the increase of the polynomial degree. If the two errors are very similar at the beginning, from degree equal to 3 onward, the test error is always worse than the train error, thus confirming what has been seen before with both OLS and Ridge (overfitting phenomenon). On the contrary, the second figure appears completely different: here the errors are stable for all the degrees. Moreover, the test MSE is larger, even if just a bit, than the train MSE. The shape of the right plot is due to the same reason explained in subsection 3.1.3: for large values of λ, all the βs are equal to 0, thus making the cost function constant.

3.2.5 Comparison of models

Now that we have analyzed the results for each single method, we can finally make a comparison between the three techniques. We decided to evaluate their performance with a polynomial degree equal to 14 (which was the polynomial degree with the smaller test MSE for Ordinary Least Squares

regression).

Figure 26 shows the obtained results: the plot allows to compare the test Mean Squared Errors obtained with the three methods, also using different values of λ for Ridge and Lasso. The OLS regression, producing a MSE just above 0.015, results to be the best model for fitting our data. The other two methods are worse for each value of λ: even if they do no produce bad results with low values of λ, their performance is not comparable to OLS, which clearly generates a smaller error. In addition, for large values of λ the MSE of both Ridge and Lasso increases a lot, until reaching a plateau. This behavior was expected, since, for large values of λ, both Ridge and Lasso generate predicted values approaching 0, thus producing a stable error. The difference between the two methods is that Lasso shrinks the values of the parameters much earlier: this is why the MSE for Lasso becomes stable before Ridge's MSE (as already seen with the Franke function).

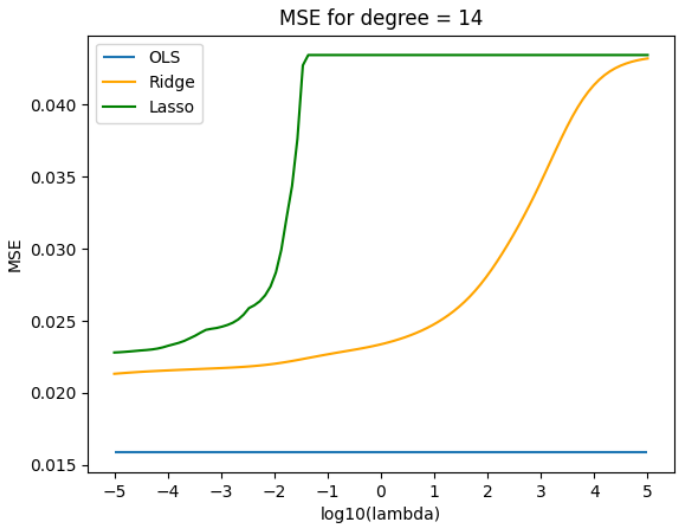

Figure 26: Comparison between OLS, Ridge and Lasso regression for a polynomial degree equal to 14: in this case, OLS results to be the best model, since it provides a smaller MSE with respect to the other two methods.

4 Conclusion

In the present work, the goal was to estimate the most suitable model comparing the performance of different linear regression methods (OLS, Ridge and Lasso) on two types of data. In order to get a better reliability of our results, we also studied and implemented resampling methods, in particular bootstrapping and k-fold cross validation.
In our analysis of the Franke function, the results indicate that OLS is the best method among these three, as it minimizes the mean squared error, demonstrating a better ability to fit the underlying data. For OLS, focusing on the phenomenon of overfitting for large polynomial degrees, we examined the bias-variance trade-off, exploring how the polynomial degree impacts the model's generalization ability from training data to test data. Thanks to a graph illustrating this trade-off, we identified the polynomial degree at which the bias and variance curves diverge, revealing an optimal compromise for our model. The best solution is to choose a balance between bias and variance, thus minimizing the Mean Squared Error.
All our findings were then used to analyze the terrain of Stavanger, Norway. Again, the Ordinary Least Squares fit performed the best, giving in general lower MSE values with respect to Ridge and Lasso.
This study is important for understanding the right process for selecting the right method for analyzing our data, achieving the most accurate predictive models. We believe that the reason why OLS is better for both our dataset is due to the following fact: we are putting ourselves in a region dominated by the bias, where the variance hardly matters for the overall error (since it has very low values). As a consequence, the regularization schemes aimed at reducing the variance (using Ridge and Lasso) don't give a benefit to the MSE. Maybe, by studying higher polynomial degrees, where the OLS is overfitting the data (i.e. the variance is high), shrinkage methods, like Ridge and Lasso, with specific values of λ, could work better. This could be a topic to study in detail, therefore we propose it as a possible expansion of this work.

5 Appendix

The implemented code is provided in the below linked GitHub Repository. We supply the code for all tasks in the folder "code". The report is also available.
In addition, in the folder "Additional material and testing" you will find the used graphs and the testing we implemented to verify our results.

https://github.com/alessiasanfelici/FYS-STK4155-Project-1/tree/main

References

[1] Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Verlag, Berlin, 2009 (Chapter 2-3)

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016 (Chapters 2-14)

[3] Davide Chicco, Matthijs J. Warrens, Giuseppe Jurman, *The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation*, PeerJ Computer Science, 7: e623, 2021 July 5

[4] M.Hjorth-Jensen, 2023, GitHub, *https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html* (Chapters 3-5)

[5] Vladimir Mikulik, 2017, *Why Linear Regression is a Projection*, https://medium.com/@vladimirmikulik/why-linear-regression-is-a-projection-407d89fd9e3a

[6] *https://www.kdnuggets.com/2023/03/bootstrapping.html*

[7] David Watson, *The Rhetoric and Reality of Anthropomorphism in Artificial Intelligence*, Minds and Machines (2019) 29:417–440, 21 September 2019

[8] *Why does ridge estimate become better than OLS by adding a constant to the diagonal?*, https://stats.stackexchange.com/questions/118712/why-does-ridge-estimate-become-better-than-ols-by-adding-a-constant-to-the-diago