Applying linear regression methods to terrain data

Bjørnstad, Johannessen, and Merlid

Department of Physics

(Dated: October 7, 2024)

Linear regression models contribute a relatively simple, explainable and computationally efficient way to make data-driven decisions in comparison to more complex methods like neural networks. However, these models do not necessarily perform well on all kinds of data. Furthermore, their polynomial degree and hyper-parameters need to be tuned for each data set. In this paper we implement three linear regression methods, ordinary least squares, Ridge, and Lasso regression, in order to explore their predicative abilities, both compared to each other, and relative to their respective adjustable parameters. We analyze the models using both synthetic data from the Franke function (Franke, 1979, p. 13) as well as real terrain data (Hjorth-Jensen, 2023). We also explore the concept of bias-variance trade-off in the models, using resampling methods such as bootstrapping and k-fold cross-validation. We conclude by reiterating the importance of selecting appropriate model complexities and values for model parameters, and suggest some guidance values for the relevant models and types of data. For the Franke function data, the optimal model is ordinary least squares, with polynomial degree 5 at an MSE of 0.0987, while the terrain data has optimal model ordinary least squares trained by 10-fold cross validation, polynomial degree 10 with 17.686 as MSE. We find that linear regression is not optimal for the complex structure of terrain data.

Contents

I. Introduction

Today, more than ever, we live in a highly data-driven society and ways to handle, interpret and make use of data is at the forefront of many minds. In order to actually utilize and learn from data, one is dependent on reliable methods to perform analyses and extract information.

Statistical learning is a field with a plethora of applications, such as finance, healthcare, engineering and of the more popular ones in the recent years; artificial intelligence.

Among the more common forms, *linear regression* stands out, tracing its origins back to the early nineteenth century with the invention of the *least squares method* (James *et al.*, 2023, p. 5). Linear regression is a collective term, coined only in more recent years, including various linear methods for predicting quantitative values. During mid to late 1900's many more techniques were developed, thereof the methods named 'Ridge' and 'Lasso'. Ridge regression had its debut in the 1970s with the publication of two scientific articles in the journal *Technometrics* (Hoerl, 2020). Lasso regression was first formally presented and named in the article *Regression Shrinkage and Selection Via the Lasso* by Robert Tibshirani in 1996. (Tibshirani, 1996).

In this paper we explore different aspects of linear regression, including ordinary least squares, Ridge, Lasso, data handling, and more. We will first use the Franke function to verify the theoretical grounds. This includes overfitting, the evaluation measures MSE and R^2, the dampening of $\boldsymbol{\beta}$ in penalized regression and bias-variance trade-off. We will study which of the three linear regression methods perform the best on the Franke function. Furthermore, we will implement the models tested on the Franke function on actual terrain data. We will use the resampling techniques bootstrapping and cross-validation. Using MSE as our error measure we will decide on the best overall model for the terrain data, which is the main goal of this paper.

We will first cover the theoretical background in which lay the foundation for our work, before detailing our implementation of methods. The results we found will follow thereafter, and lastly an interpretation and discussion surrounding these results.

II. Theory

A. Linear Regression

Linear regression bases itself on the assumption of a linear relationship between the *predictors* and the *response* (Fahrmeir *et al.*, 2013, p.21-26). Given a data set of p input variables (commonly called predictors), $X = [x_1, x_2, \dots, x_p]$ and a data set of output variables (commonly called the response) one seeks a linear model on the form

$$\tilde{y} = \hat{\beta}_0 + \sum_{j=1}^{p} x_j \hat{\beta}_j. \tag{1}$$

The scalar $\tilde{y}$ is the prediction of the response, $\hat{\beta}_0$ the estimated intercept, and each $\hat{\beta}_j$ the estimated coefficient belonging to its corresponding predictor x_j.

This equation is commonly written in vector form,

$$\tilde{\mathbf{y}} = \boldsymbol{X}^T \hat{\boldsymbol{\beta}}, \tag{2}$$

where given n different sets of input/output-variables (data points), $\tilde{\mathbf{y}}$ is the response-vector and $\boldsymbol{X}$ is a $n \times (p+1)$-matrix called the *design matrix*. Here the $' + 1'$ column in $\boldsymbol{X}$ is a row of ones for inclusion of the intercept in $\hat{\boldsymbol{\beta}}$, and the residual p columns each of the p predictor variables.

The "true" model is assumed the form

$$y = \beta_0 + \sum_{j=1}^{p} x_j \beta_j + \epsilon. \tag{3}$$

Our linear regression models are always estimations of the equation above, and for real-life data there's no way of knowing the true $\boldsymbol{\beta}$ (for generated data there will be exceptions). ϵ is an irreducible *error-term* or *residual-term* representing all variance in the data not explainable by the linear model; any variation due to randomness is included in this term. It is assumed $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and given this assumption one gets the expected value of y_i

$$\mathbb{E}[y_i] = \mathbf{X}_{i,*}\boldsymbol{\beta}, \tag{4}$$

and the variance of y_i

$$\text{var}(y_i) = \sigma^2. \tag{5}$$

Calculations of Eq. 4 and 5 are available in appendix B.

B. Cost- & Loss-Functions

The main objective when solving linear regression problems, is finding the optimal coefficients $\boldsymbol{\beta}$ that minimizes an error measure between y_i and $\tilde{y}_i$. How such an optimal solution evinces is dictated by the definition of the *cost-function*, or *loss-function*, which is simply metrics chosen to measure how much the predictions deviate from the "truth". A cost-function, $\text{Cost}(f, \mathcal{D})$, is used to describe such a metric measuring a group of data-points, while a loss-function, $L(y, \hat{y})$, describes a metric regarding a single data-instance. The cost-function can be expressed in terms of the loss function

$$\text{Cost}(f, \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{y}_i), \tag{6}$$

as the average of the loss-function over the data. Through different choices of cost-function one ends up with different methods for estimation, resulting in different models for the same data-set.

C. Data handling

1. Scaling

In order for all parameters to have an equal starting point when training a model, it's often necessary to scale the data (Hastie *et al.*, 2009, p. 398). Having one parameter with values of order 10^3, and another parameter with values of order 10^{-3}, the model could miss the predictive importance of the latter. Whether or not it is necessary to scale the data, depends on the choice of cost-function; Given two predictors of different magnitudes where the same relative change has equal impact on the response, the predictors of smaller magnitude would have coefficients of inversely proportional magnitude. If a cost-function penalizes bigger coefficients, as some does, the parameter of smaller magnitude would then be penalized more than it should. To alleviate the risk of such problems, it is thus necessary for many methods that all data is scaled before training.

A common choice for scaling is to use the *standard scaler*. This method involves subtracting the mean and dividing by the standard deviation, for each feature (i.e. parameter) respectively. Each of the columns will then have mean equal to zero and a standard deviation of one after the scaling. For the i-th datapoint and the j-th feature one gets the transformation as indicated in Eq. 7 (Hjorth-Jensen, 2024, Linear Regression). Alternatively a variation of the standard scaler is sometimes used, including only subtraction of the mean and omitting the division of standard deviation.

$$x_{ij} \rightarrow \frac{x_{ij} - \overline{x}_j}{\sigma_{x_j}} \tag{7}$$

It's considered good practice to not scale the intercept, as it represents a constant term without variability and scaling it will often worsen it's performance, or defeat it's purpose entirely. The intercept may be taken out during training and later be recalculated as Eq. 8 shows (Hjorth-Jensen, 2024, Resampling methods). Taking out the intercept amounts to removing the column of 1's in the design matrix.

$$\beta_0 = \frac{1}{n}\sum_{i=0}^{n-1} y_i - \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=1}^{p-1} X_{ij}\beta_j \tag{8}$$

2. Splitting data into sets

When training a model the goal is for it to learn the underlying pattern of the data that is (hopefully) representative for the entire population that the data is sampled from. Testing a model on the same data it's trained on could give a false impression of it's accuracy; a model that takes into account variation specific to the data it's trained on (that's not representative for the larger population) would get rewarded in this testing, while a model that does not take it into account would get punished (Hastie *et al.*, 2009, p. 228).

It's therefore necessary to have (at least) two data sets; one to be used for training and one to be used for testing, to give us a more impartial evaluation of the model. The two data sets are named respectively "training data" and "test data". As can be inferred by the names, the training data is used in the training (or *fitting*) of the model, while the test set is used to evaluate the performance of the final chosen model. Additionally one can choose to include a "validation" set in the split; this validation set would then be applied in the model selection phase.

An important concept in train-test splitting is to **never** touch the test-set before the final evaluation. The test-set is supposed to act as never-before seen data for the model, and given a limited data set is the closest one gets to an impartial assessment. If using a scaler, fitting of the scaler should be done after the split and solely on the training data - and this scaler, fit to the training data, then applied to the test data when testing.

Splitting data is an important tool to help avoid *overfitting* and *underfitting* of the model, and ensure a more generalizable fit. Overfitting and underfitting is elaborated further in section II.H; "*Bias-Variance Tradeoff*".

D. Resampling methods

When training models, error estimation is crucial both for model selection and model assessment. Without a good metric for the performance of the model it is impossible to gauge how well fit a model actually is. Generally, it's desirable to have as much data as possible for the training of a model. When holding off some of the data for testing, this data can then not be used for training. It's therefore of interest to explore methods that allow for a train-test split of the data, while still keeping as much data as possible for training and giving us a better error estimation.

1. Bootstrapping

Bootstrapping is a general term for one such resampling method where one draws with replacement from the original data set, creating a "new" data set for each bootstrap. Every bootstrap sample should have the same size as the original data set, n. B such samples are generated, and the training repeated for each of them. The b-th bootstrap sample produces model $\hat{f}_b$ (Hastie *et al.*, 2009, p. 249).

Using bootstrap the estimation of the error can be calculated as follows:

$$BS_{error} = \frac{1}{n}\sum_{i=0}^{n-1} L\left(y_i, \frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(x_i)\right) \tag{9}$$

2. K-fold cross-validation

K-fold cross-validation is another method of resampling. The data set is divided into K parts $\mathcal{F}_k$, called folds. The K must be chosen by the developer as they see fit. $K-1$ folds are used as training data, while the remaining fold is used as test data. The model trained when the k-th fold is held out, is denoted f^k. The procedure is repeated K times, holding out a new fold each time. For k-fold cross-validation the estimated error is given as the mean of the K test errors, shown in Eq. 10 (Hastie *et al.*, 2009, p. 241). Her we take into consideration that the folds $\mathcal{F}_k$ may be of different sizes.

$$CV_{error} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|\mathcal{F}_k|}\sum_{i\in\mathcal{F}_k} L\left(y_i, f^k(x_i)\right) \tag{10}$$

There are both advantages and disadvantages of this method. On one hand, it achieves the goal of maintaining a train-test split while keeping a sizeable train set. The estimated error (Eq. 10) will be closer to the true generalization error measure. This is a consequence of the error being averaged over many different models, and thereby to a higher degree taking into account randomness associated with data-selection.

On the other hand, cross-validation will be quite computationally costly for a high K. Another disadvantage is that there is no clear choice of K; here there is a trade-off between bias and variance. A smaller K leads to lower variance, but higher bias. On the other side, a higher K leads to low bias, but high variance. The extreme case of the latter is leave-out-one cross-validation (LOOCV), where every datapoint acts as a fold. It will lead to an unbiased error measure, but the variance will be quite large (Hastie *et al.*, 2009, p. 242).

E. Evaluation measures

The *mean squared error* (MSE) is a popular error measure for linear regression models, and is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2. \tag{11}$$

n denotes the number of data points in the training data, and the prediction for the i-th data point is denoted $\tilde{y}_i$. Given a loss-function of the squared distance between the prediction and the true value, the errors for cross-validation and bootstrapping (respectively Eq. 10 and Eq. 9) will trivially be the equivalent of MSE.

R^2 is a measure of how well the model explains the variance present in the data.

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y}_i)^2}, \tag{12}$$

$R^2 \in (-\infty, 1]$. If $R^2 = 1$ the model perfectly explains all variance, whereas a value of 0 would mean does not explain any of the variance. A prediction $\tilde{\mathbf{y}} = \bar{\mathbf{y}}$, would result in $R^2 = 0$. If $R^2 < 0$, the model is worse than a straight line. The numerator is the sum of the squared residuals, also called RSS. The denominator is the total sum of squares, in short TSS (Martin, 2022, p. 29).

F. Ordinary least squares

Ordinary least squares is a linear regression method that seeks to minimize the following cost function:

$$C(\boldsymbol{\beta}) = \sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{13}$$

Comparing the expression above to Eq. 11, it's clear how the OLS regression method is inherently designed to minimize MSE.

From Eq. 13 the equation for the optimal $\boldsymbol{\beta}$ can be derived. This is done by taking the derivate of the cost function w.r.t. $\boldsymbol{\beta}$ and finding the minimum. The optimal $\boldsymbol{\beta}$ for OLS is shown in Eq. 14.

$$\hat{\boldsymbol{\beta}}_{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \boldsymbol{H}\mathbf{y} \tag{14}$$

The $\boldsymbol{H}$ is popularly called the Hessian matrix. The Hessian matrix for OLS specifically is stated in Eq. 14, but the term ”Hessian matrix” generally describes a square matrix of double derivatives.

Ordinary least squares provides an unbiased estimation - meaning the expected value of the estimated betas is equal to the true betas, as shown in Eq. 15.

$$\mathbb{E}[\hat{\boldsymbol{\beta}}_{OLS}] = \boldsymbol{\beta} \tag{15}$$

$$\text{var}(\hat{\boldsymbol{\beta}}_{OLS}) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} \tag{16}$$

OLS regression is invariant to scaling of the data.

G. Penalized linear regression methods

An extension of the ordinary least squares method is to add a penalization term to the cost-function. There are many reasons why this is often preferred.

Firstly, when OLS is performed it is assumed that the matrix $\boldsymbol{X}^T\boldsymbol{X}$ in Eq. 14 is invertible. This may not always be the case due to correlation between the predictors in the data set, or if $p > n$. In these cases the matrix will not be full rank, i.e. not invertible. A mathematical fix to this is to add a (small) number λ along the diagonal:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} - \lambda\boldsymbol{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{17}$$

These methods also help to reduce overfitting. Intuitively this is a result of penalizing ”too good of a fit” on the training data, meaning it's harder for models to get overfit.

Eq. 17 is the general equation for the coefficients in penalized regression, where the parameter λ controls the regularization. Among the many types of choices for penalization metrics are the L1-norm penalty, also known as Lasso, and the L2-norm penalty, known as Ridge. Different types of penalties yields in different properties and interpretations related to the resulting models.

1. Ridge Regression

In Ridge regression, the L2-norm penalty gives us the following cost-function:

$$C(\boldsymbol{\beta}) = \sum_{i=0}^{n-1}\left(y_i - \sum_{j=1}^{p-1}X_{ij}\beta_j\right)^2 + \lambda\sum_{j=1}^{p-1}\beta_j^2 \tag{18}$$

This can alternatively be expressed as Eq. 19 and 20, where the restraint on $\boldsymbol{\beta}$ is explicitly stated. Here the value of t is directly related to the value of λ (Hastie *et al.*, 2009, p. 63).

$$C(\boldsymbol{\beta}) = \sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{p-1} X_{ij}\beta_j\right)^2 \tag{19}$$

$$\sum_{j=1}^{p} \beta_j^2 \leq t, \tag{20}$$

Ridge regression puts a penalty on all the β-terms except the intercept, which is held out during training. Had it been included, the model would depend on the chosen origin and this dependency undermines the principle of shift invariance (Hastie *et al.*, 2009, p. 63). The value of β_0 is later calculated as Eq. 8 shows.

In the ideal case, when the design matrix $\mathbf{X}$ is orthogonal, we have $\mathbf{X}^T\mathbf{X} = \boldsymbol{I}$. From Eq. 17, we get that:

$$\boldsymbol{\beta}_{Ridge} = (\boldsymbol{I} - \lambda\boldsymbol{I})\mathbf{X}^T\mathbf{y} \tag{21}$$

From Eq. 21, it immediately follows that, in the case of $\mathbf{X}$ orthogonal, we get the relation:

$$\boldsymbol{\beta}_{Ridge} = \frac{1}{1+\lambda}\boldsymbol{\beta}_{OLS} \tag{22}$$

The solutions produced by Ridge regression depend on the scaling of the data. It is therefore especially important to standardize the data as explained in section II.C.1.

2. Lasso Regression

Lasso regression is another penalization method. This method uses an L1-norm penalty, giving us the cost function and constraint on β as:

$$C(\boldsymbol{\beta}) = \sum_{i=0}^{n-1}\left(y_i - \sum_{j=0}^{p-1} X_{ij}\beta_j\right)^2 + \lambda\sum_{j=0}^{p-1}|\beta_j| \tag{23}$$

Similar to Ridge regression, there is an alternative formulation, Eq. 25 and Eq. 25, explicitly stating the constraint on $\boldsymbol{\beta}$;

$$C(\boldsymbol{\beta}) = \sum_{i=1}^{N}\left(y_i - \sum_{j=1}^{p-1} X_{ij}\beta_j\right)^2 \tag{24}$$

$$\sum_{j=1}^{p} |\beta_j| \leq t, \tag{25}$$

The value of t is again directly related to the value of λ (Hastie *et al.*, 2009, p. 68).

The intercept is held out in training, by the same reasoning as for Ridge regression, described in section II.G.1.

3. Ridge vs. Lasso

Both Ridge and Lasso are known as *shrinkage methods*, as they both shrink the coefficients. Lasso additionally goes under the term *selection method*, as the coefficients can be shrunk to zero resulting in only a selection of the features contributing in the model. If the value of λ is sufficiently large, a larger number of coefficients become zero. In Ridge regression, the values of β_j are forced closer to zero, but can never be zero completely.

The two methods, specifically the difference in how the coefficients approach zero, are illustrated in Fig. 1. Both methods will find a solution where the error function intersects the constraint. For the circular Ridge constraints, this intersection will never coincide with the axis i.e. the coefficients will always be non-zero.

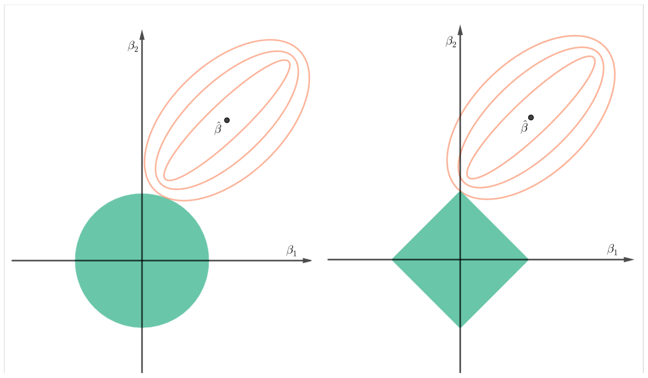

FIG. 1: Illustration of the optimal coefficients in Ridge regression (left) and Lasso regression (right). The green areas illustrate the constraints given in respectively Eq. 20 and Eq. 25, and the red ellipses contours of a least squares error function (Recreated from Hastie *et al.*, 2009, p. 71).

H. Bias-Variance Tradeoff

1. Bias

The bias of a method (or estimator) describes its inability to capture the true relationship being estimated. For a specific model this a measure of the difference between the expected values of the estimator and the true values, as shown in Eq. 26.

$$\text{Bias}(\hat{\boldsymbol{\beta}}) = \mathbb{E}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta} \tag{26}$$

If models were to be trained with the same method on different data sets, those with low bias would all on average predict close to the target values they're trained on. Models with high bias would on the contrary predict values further from the true target. Bias is often described using shots at an archery target, where each shot represents a model trained on different data; high bias is analogous to shots far from the bulls-eye (which is the goal), while lower bias to shots closer to the bulls-eye.

2. Variance

The variance of a model describes the amount of variation between it's predictions and the expected value of the predictions, as shown in Eq. 27. This in turn explains a models sensitivity to fluctuations in the data sets; for a model with high variance the predictions will vary a lot depending on which data set it's trained on, and vice versa.

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \mathbb{E}[\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}]^2] = \mathbb{E}[\hat{\boldsymbol{\beta}}^2] - \mathbb{E}[\hat{\boldsymbol{\beta}}]^2 \tag{27}$$

Relating back to the analogy of a shots at a target, high variance relates to shots that deviate a lot from one another, while lower variance to shots more closely grouped together.

3. The trade-off

When evaluating statistical models one often talks about the expected error of the model, as a measure of how good the model is. This expected error can be decomposed as shown in Eq. 28.

$$\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \mathrm{Bias}(\tilde{\mathbf{y}})^2 + \mathrm{var}(\tilde{\mathbf{y}}) + \sigma^2 \tag{28}$$

Calculations are available in appendix B, here specifically for a model fit with ordinary least squares (OLS) regression. Eq. 28 gives us the expected error composed of a model-bias term, a model-variance term and lastly the variance of ϵ; the irreducible error-term as given in Eq. 3. As the latter of the three is irreducible and wont be affected by changes to the method, one is therefore left with the bias term and variance term to try and reduce for optimizing models. The ideal model would have both low bias and low variance, but as the two terms are inversely related - this is highly improbable. Instead the objective becomes to find the optimal balance between them - this is called the *bias-variance trade-off.*

The edge cases of this trade-off are usually denoted overfitting and underfitting, as mentioned in section II.C.2 and section II.G. Overfitted models have low bias and high variance, while an underfitted models have the opposite; high bias and low variance (Hjorth-Jensen, 2024, Statistical interpretations and Resampling Methods).

An overfitted model is characterized as being too complex; often having too many parameters, degrees of freedom or similar measures of complexity. Overfit models pick up on all small variations in training data, including those caused by noise and randomness. As a consequence these models perform excellent on their training data, but are expected to perform quite terribly on new data sets.

An underfit model is contrastingly characterized as being too simple. Underfit models are not very specialized to the training data, failing to pick up on variations related to the underlying distribution of the data. As a consequence these models perform similarly on training data and test data, but this performance is not very good.

III. Method

A. Data

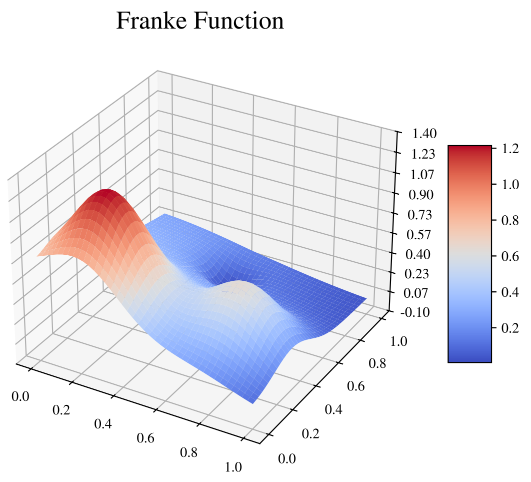

FIG. 2: Franke function for x,y $\in [0, 1]$

1. Franke Function

We first implement our methods on a synthetic data set based on the Franke function (Franke, 1979, p. 13). Originally proposed by Richard Franke in 1979, it is a much used function for testing linear regression and interpolation problems. The Franke function is defined as:

$$
\begin{aligned}
f(x,y) =& \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\right) \\
&+\frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49}-\frac{(9y+1)^2}{10}\right) \\
&+\frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\right) \\
&-\frac{1}{5}\exp\left(-(9x-4)^2-(9y-7)^2\right), \qquad (29)
\end{aligned}
$$

with the domain $[0,1]^2$. It produces a surface in the range $(0,1.25)$ with two Gaussian peaks and a Gaussian dip, imposed on a surface sloping down towards $(x,y)=(1,1)$ (Franke, 1979, p. 13).

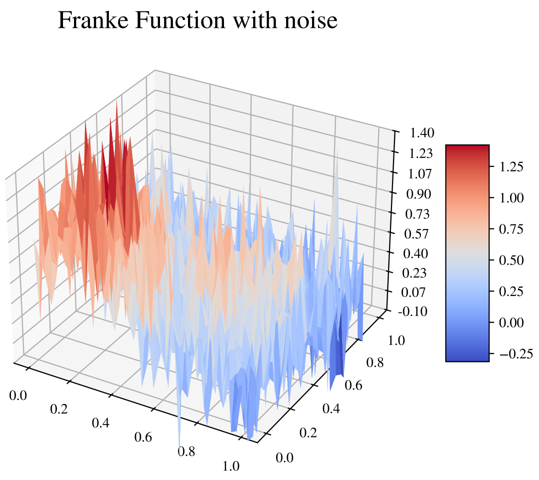

FIG. 3: The Franke function for x,y $\in [0,1]$ with added noise term: $(3/10)\mathcal{N}(0,1)$.

In order to enable potential overfitting in the models, we added a stochastic noise term to the Franke function. The noise term is generated using a normal distribution, and scaled by a factor of 0.3. By choosing this noise term factor, we introduce sufficient amounts of noise to observe overfitting in some models, while maintaining the underlying structure to be recognized by the models (see Fig. 3). Hence, we use the following function in our analysis:

$$
f^*(x,y) = f(x,y) + \frac{3}{10}\mathcal{N}(0,1), \qquad (30)
$$

where $f(x,y)$ is defined in Eq. 29. All mentions of the Franke function henceforth will reference the function defined in Eq. 30.

For the Franke data, we use 41 data points along each axis, for a total of 1681 data point (i.e. 41^2).

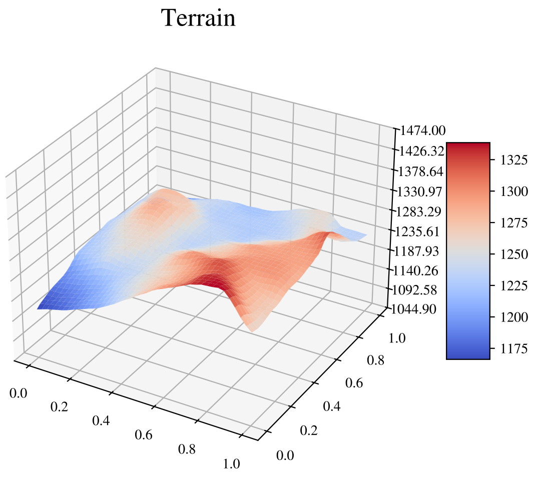

FIG. 4: Terrain data

2. Terrain data

Secondly, we implement our methods on real terrain data (Hjorth-Jensen, 2023). The data is taken from an area close to Stavanger, Norway. We use 41 data points along each axis (i.e. $41^2 = 1681$). The data is plotted in Fig. 4. As with the Franke Function, we have $x, y \in [0,1]$.

3. Scaling and train/test-split

To ensure our different models are trained and tested on the exact same data, making them comparable, we perform the train/test split of the data initially, in a separate data handling class. After setting up our design matrix (II.A), we scale it using the `StandardScaler` class from *Scikit learn* (SKLearn, 2024). We use the standard scaler as discussed in section II.C.1. From this discussion, it is clear that we want some scaling of the data. We prefer standard scaler over i.e. min-max scaling, as it transforms the data to have mean 0, which is good for Ridge and Lasso. It also puts more weight on outliers (Raschka *et al.*, 2022, p. 120), which is important for discussing concepts such as overfitting and variance. The data handling class also implements bootstrap sampling of the data, as well as splitting data in equally sized sets to use for cross validation.

B. Implementation of models

As our data contains two numerical input variables for each data points (x and y), our design matrix needs to include all combined polynomials of these variables up to the max degree. For example, for a model of polynomial degree 2, a row in our matrix looks like Eq. 31, where (x,y) are the input coordinates of that data point. The number of rows equals the number of data points, and will be scaled as described above after being split into train and test parts.

$$[x,\ y,\ x^2,\ xy,\ y^2] \tag{31}$$

We implement a general `RegModel` class, with subclasses for each model type. In these classes we implement functions for fitting the models on data (both simple data, and bootstrap and cross validation sets), performing predictions based on these fits, and calculating MSE and R^2 for these models.

For both the ordinary least squares and Ridge models we have implemented the fit and prediction function manually, through equations Eq. 14 and Eq. 17 respectively. As Lasso regression has no explicit formula to perform the fit, i.e. calculate the optimal coefficients, we utilize the `Lasso` class from *Scikit learn* (SKLearn, 2024).

C. Hyperparameter selection

All the linear methods in this project (OLS, Ridge and Lasso) have hyperparameters which results in different models. These hyperparameters are the polynomial degree and the weight parameter for the regularization for Ridge and Lasso regression (λ). Optimizing the ordinary least squares model is simple, as we only need to test it on different polynomial degrees. We run a set of bootstrap samples for each degree complexity, and choose the model with the lowest MSE.

For Ridge and Lasso however, we require selecting two parameters simultaneously. As different polynomial degrees might obtain the best MSE at different values for λ, it does not suffice to lock one and test the other only for that value. We use grid search, a standard technique for tuning multiple hyper parameters (Yang and Shami, 2020, p. 302). Grid search is an exhaustive search over a selection of values for each parameter. We perform two layers of grid search on each type of model and set of data. In the first layer we test model-complexities ranging from 1 to 11, and lambdas from 1×10^{-5} up to 9 (with increasing step size according to order of magnitude). We use 200 bootstrap samples, and pick the values for polynomial *degree* and λ resulting in the lowest MSE. In the second layer, we use only the complexities $d-1,\ d,\ d+1$, and 40 different λ-values ranging from $(4/5)\lambda$ to $(6/5)\lambda$. We do this in order to fine tune our selections of hyper-parameters.

IV. Results and discussion

A. Hyperparamters

Through grid search the optimal combination of λ- and degree-values were found to be:

		Franke Function	**Terrain Data**
Ridge	λ	2.4×10^{-2}	3.0×10^{-5}
	degree	7	4
Lasso	λ	1.7×10^{-5}	2.2×10^{-4}
	degree	7	6

TABLE I: Best pairs of λ- and degree values found through grid search

In the following sections, the Lasso and Ridge regression models are trained with their respective λ values from Tab. I.

B. Franke function

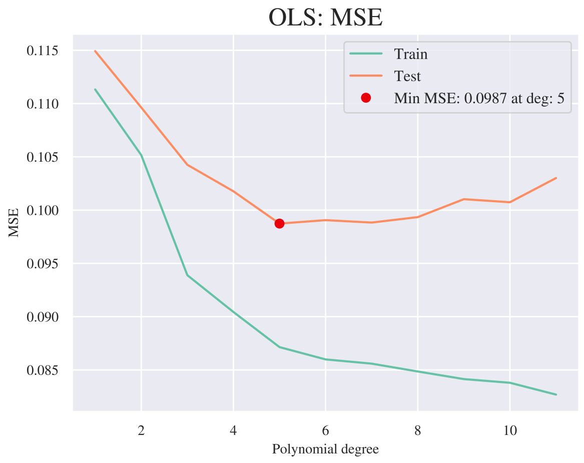

FIG. 5: Mean squared error (MSE) for the ordinary least squares model both on training- and test data. The minimum MSE is marked with a red dot. This displays the same concept as figure 2.11 in Hastie et.al. (Hastie *et al.*, 2009, p. 38).

Fig. 5 shows how the mean square error (MSE) initially decreases as the polynomial degree increases. This seems reasonable as the Franke function makes up a complicated surface and higher degree polynomials are necessary to replicate it.

Furthermore, we notice that the test error reaches its minimum at polynomial degree equal to 5. This minimum is chosen as the optimal polynomial degree. Beyond this point, the test error increases and the training error decreases. This is due to overfitting, and the fact that OLS is designed to minimize the MSE of the training data.

When the model is underfitted, i.e. a too low polynomial degree, we have high bias and low variance. At the other end however, when the polynomial degree is too large, the bias is low and the variance is high.

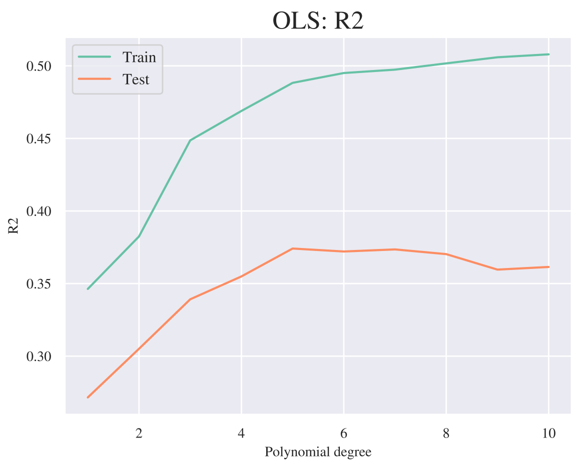

FIG. 6: R^2 for the ordinary least squares model on training- and test data.

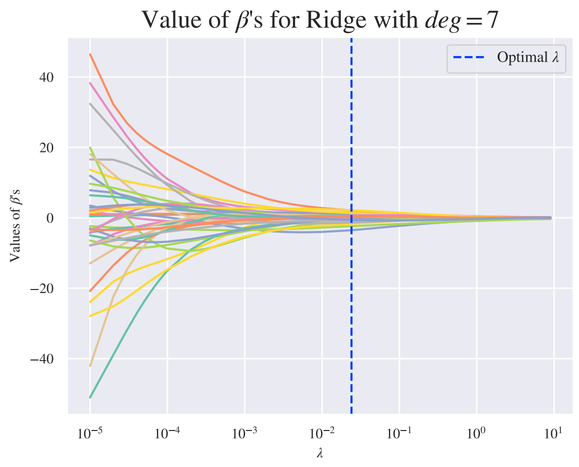

FIG. 7: The values of the parameters $\boldsymbol{\beta}$ for the Ridge regression model with polynomials up to a degree of seven for increasing values of λ. Each colored line represents one β_j.

As shown in Fig. 6, the R^2 value increases as higher degree polynomials are introduced to the ordinary least squares model. This is reasonable as we likely need higher degree polynomials to explain more variance in the data. The increase in R^2 halters at around degree equal to 10. At this point, introducing higher degree polynomials does not help explain further variance in the data. The value of R^2 for the test data is substantially lower then the one for the training data. This might be a sign that the model does not describe the underlying structure of the data precisely, in other words doesn't generalize well. This claim is further supported by an R^2 value of 0.50 on the training data, meaning that our model only explains 50 % of the variance in the training data. However, this is expected due to the large amount of noise we introduced in Eq. 30.

The $\boldsymbol{\beta}$-values for our Ridge and Lasso regression models are presented in Fig. 7 and Fig. 8 respectively. For both models, we observe that the coefficients are forced towards zero as the penalty increases. For the Ridge regression coefficients, all have non-zero values regardless of the size of the penalty. For Lasso regression however, we observe that some are brought to zero at a sufficiently large penalty; closer to $\lambda = 10^0 = 1$ all values of the β's are brought to zero. A λ value of this size seems highly unreasonable, as it corresponds to a constant prediction, i.e. a flat plane normal to the z-axis. The vertical line in each plot shows the optimal λ in combination with the polynomial degree found through the grid search (numbers are available in Tab. I).

We find it interesting that the Lasso regression model has a far lower optimal λ value than the Ridge regression model. However, we also note how the y-axis values in Fig. 7 and in Fig. 8 are different. Taking the latter into consideration, the beta values chosen for the final models are comparable.

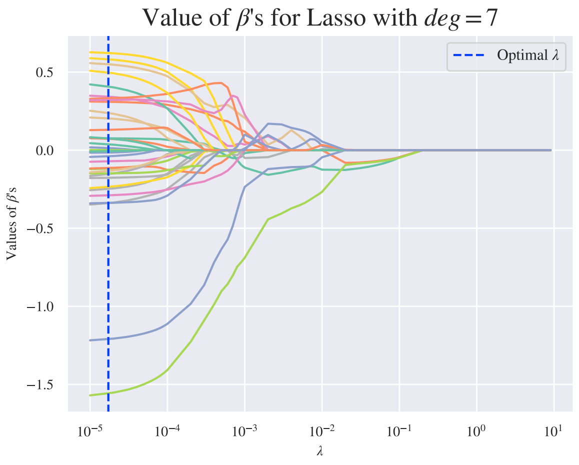

FIG. 8: The values of the parameters $\boldsymbol{\beta}$ for the Lasso regression model with polynomials up to a degree of seven for increasing values of λ. Each colored line represents one β_j.

We want to find the best model among OLS, Ridge and Lasso regression. In Fig. 9 we see the MSE on the test data for all three. The best model on the Franke function data is OLS with a MSE of 0.0987. This is found at a polynomial degree of five.

The Lasso regression model does not perform identically with OLS when λ is set to zero. As fitting the Lasso model requires iterative tuning, we can not expect to reach the exact optimal β's. For continuity, we decided on having a constant value for max iterations while fitting the Lasso models, at `max_iter=1000`. Increasing this would result

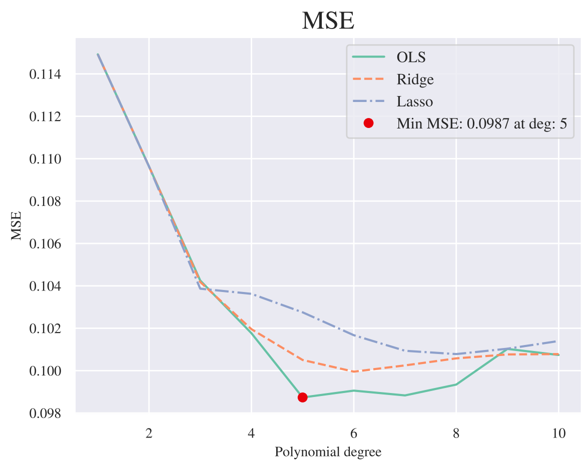

FIG. 9: The MSE on the test data for OLS, Ridge and Lasso regression models. The minimum MSE is marked with a red dot.

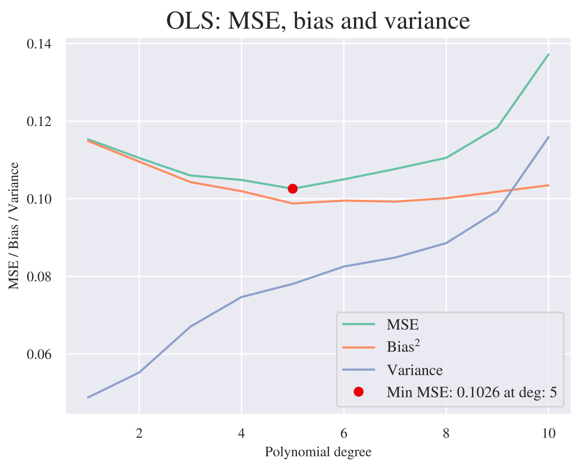

FIG. 10: MSE decomposition in terms of bias and variance term for an ordinary least squares model trained with bootstrapping. The minimum MSE is marked with a red dot.

in a substantial increase in computation time for the grid search. The Lasso regression model performs worse out of the three partly because it has not yet reached the optimal solution.

Furthermore, the Ridge regression model performs only slightly worse than OLS. We suspect that the λ value found through the grid search is not the optimal one, as a λ value of 10^{-5} yield a lower minimal MSE for the test data. The cause of this is not apparent to us. Possible explanations might be that the grid search uses another train test split than the plotting files. This is to be able to run them separately, i.e. we don't want to have to run the relatively slow grid search every time we make a new plot. To avoid this being a large source of errors, we use bootstrapping in the grid search. In this specific plot however, we only train on a simple train test split, which, if unlucky, can cause results that are not theoretically optimal. At a closer inspection of different λ values greater than zero for Fig. 9, the Ridge regression model never outperforms OLS. The conclusion of OLS being the best model holds regardless.

With the use of bootstrap samples, we have calculated the MSE decomposed into a bias and a variance term for an OLS model. In Fig. 10 the bias is initially high, but the variance is low. This is typical for an underfitted model. As the complexity increases, the bias goes down, whereas the variance increases. The lowest MSE is reached at some trade-off between the two. As the model complexity is further increased, the variance substantially increases and ensures an even higher MSE than at the beginning. This is a sign of an overfitted model.

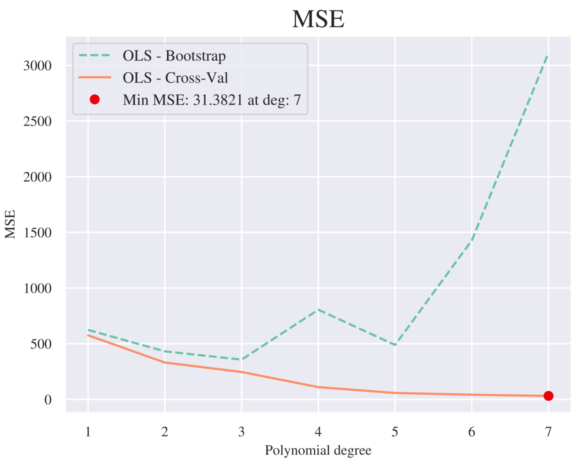

FIG. 11: MSE for OLS plotted against the polynomial degree, two different ways: 10-fold cross-validation and bootstrapping with 100 bootstrap samples. The minimum MSE is marked with a red dot.

C. Terrain data

In Fig. 11 we compare bootstrapping to cross-validation. Both resampling techniques are used on the ordinary least squares model. The minimal MSE value for the cross-validated OLS model is lower than for the bootstrapped model.

There might be several reasons why the MSE is lower for cross-validation compared to bootstrapping. For the model trained with cross-validation, more data is utilized than for the one trained with bootstrap. In the latter, we split into a training and a test data set, whereas cross-validation trains on the entire data set. Indeed, as we use an 80/20 train-test split, and bootstrap on average uses 63.2% of the train data each sample, the bootstrap ends up using only 50.6% ($0.8 \cdot 63.2\%$) of the training data on average (Hastie *et al.*, 2009, p. 251). Compared to 10 folds cross validation, which uses 90% of the data for training each fold, it is expected that the bootstrap model will overfit at a much lower complexity.

We could have used the entire data set to draw bootstrap samples from, which would have allowed us to use more data. However we would then have to keep track of which samples were not included in the bootstrap sample and use these as the test set for the specific bootstrap model. This was not done, as it would have required us to implement bootstrapping twice for the different purposes in this report.

For the cross-validated model we tried different number of folds in the range 5 to 10. The results were very similar, with $k = 10$ giving the lowest MSE. Seeing as computing power was not an issue in this part, we chose k = 10 as our final value. This is used for all bootstrapped models for the terrain data.

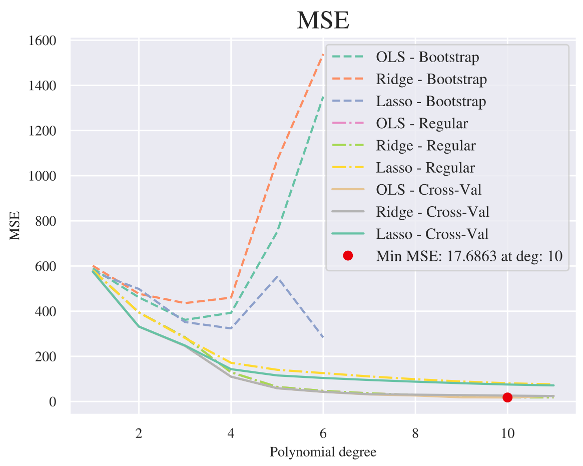

FIG. 12: MSE for OLS, Ridge and Lasso regression against polynomial degree. Each model is trained three times; with no resampling, using 10-fold cross-validation and lastly, using bootstrapping with 100 bootstrap samples. The minimum MSE is marked with a red dot.

Our main goal is to find the model providing the best fit, with respect to the MSE, for the terrain data. We have implemented three different model types; OLS, Ridge, and Lasso. Furthermore, we can use no resampling, bootstrapping or cross-validation. Combining this yields nine different models. In Fig. 12 we compare the MSE on the test set for all. We find OLS with 10-fold cross-validation to be the best model, with a MSE value of 17.6863 at a polynomial degree of 10. This plot is only created for degrees of complexity up to 11 seeing as there are problems with numerical precision beyond this point (numbers smaller than $\approx 10^{-15}$). The bootstrapped models are excluded from the plot beyond a polynomial degree of seven, as including them would dominate the visualization.

We believe that 10-fold cross-validation is performing best, as it combines being trained on the most data (90% of the data each fold), while also being trained multiple times. This results in a more stable model, that handles new test sets better than any other. It is possible that cross-validation with more than 10 folds would have performed even better, however we avoided this due to it requiring more computational power. Even though it would be feasible to train it on data sets of this size, it would not be well generalizable for larger data sets.

Choosing the number of data points for the terrain data was challenging. Including more data points meant having a data set with more variance. This leads to models with a far higher MSE. Another aspect we considered was the computational time. Especially the grid search is already time consuming with 1681 data points.

Furthermore, it is interesting to discuss whether linear models are sufficient for these kinds of problems or not. Intuitively one might think that more complex is better, and that the only downside of more complex models are computational efficiency. While being more computationally efficient is definitely an advantage that should not be neglected for large quantities of data, it's far from the only upside of these simpler models.

Firstly, linear models generally propose more explicit relationships, i.e. they're easier to interpret. To what extent different feature-variables affect the response-variable is simply stated by the slope's relation to the corresponding parameters, and can thus be deferred straight from the coefficients. The explainability of a model is crucial for justifying the use of analytical and predictive models in close to all real-world problems.

Secondly, and maybe most obviously, if the true relationship we're trying to model is actually linear (or approximately so) - a linear model will be ideal for trying to mimic this relationship. While some more complex models might be able to imitate this performance, there is no reason to use overly complicated methods that increase the risk of overfitting, reduce the interpretability and are more computationally heavy.

V. Conclusion

We have clearly observed that within the field of linear regression, there are major implications from selection of model type, polynomial degree and hyper-parameters. Furthermore the number of data points, the type of data, and selection of methods for training and testing of the models, result in divergent results. Hence, selecting good models, and tuning their complexities and hyper-parameters is a fine yet crucial art.

In spite of some instabilities in test results on the different models, we have found clear indications of which models perform well under different circumstances tested in this paper. In particular, we have seen that a lack of training data or too high polynomial degree leads to overfitting, while higher values for the hyper-parameter λ and larger quantities of training data, leads to more stable models.

For both the synthetic data set based on the Franke function and the real terrain data, we observed that versions of the ordinary least square models resulted in the lowest MSE. Even though we suspect that our grid search implementation for finding optimal λ-values for the Ridge and Lasso models could have a weakness due to how we implemented bootstrap, manual testing of some other λ-values suggests that OLS would always perform best anyways.

Our best performing models for the two data sets are the following: For the Franke data, the best model is ordinary least squares with polynomial degree 5, resulting in an MSE of 0.0987. For the terrain data, we have an ordinary least squares model trained through 10-fold cross validation. This model has polynomial degree 10, and an MSE of 17.686.

Due to our low R^2-scores, we conclude that linear regression is not optimal for modelling this kind of data. That being said, the models are still far better than random guessing, and at a relatively low computational cost they could still be a viable option.

Future improvement of our results could be made by making sure the train and test data is exactly the same between the grid search and plotting of the models. Increasing number of data points and adding more samples to bootstrapping could also improve the results, to a cost of requiring more computational power.

Linear regression as a field is already heavily researched, and most results in a paper like this is just confirming theory, as well as tuning hyper-parameters. Due to this, a similar paper like this could be useful for any given data set, if one wants to figure out whether linear regression will perform well modelling it.

References

L. Fahrmeir, T. Kneib, S. Lang, and B. Marx (2013), *Regression: Models, Methods and Applications* (Springer Berlin Heidelberg).

R. Franke (1979), *A critical comparison of some methods for interpolation of scattered data.* (No. NPS53-79-003. Naval Postgraduate School Monterey CA.).

T. Hastie, R. Tibshirani, and J. H. Friedman (2009), *The elements of statistical learning: Data mining, Inference, and Prediction*, 2nd ed., Springer series in statistics (Springer, New York, NY).

M. Hjorth-Jensen (2023), "Srtm data norway 1," `https://github.com/CompPhysics/MachineLearning/commits/master/doc/Projects/2023/Project1/DataFiles/SRTM_data_Norway_1.tif`, data file.

M. Hjorth-Jensen (2024), "Applied data analysis and machine learning," .

R. W. Hoerl (2020), Technometrics **62** (4), 420–425, https://doi.org/10.1080/00401706.2020.1742207.

G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor (2023), *An introduction to statistical learning: with Applications in Python*, 1st ed. (Springer International Publishing).

P. Martin (2022), *Linear Regression: An Introduction to Statistical Models* (SAGE Publications Ltd).

S. Raschka, Y. Liu, and M. V. (2022), *Machine Learning with PyTorch and Scikit-Learn* (Packt).

SKLearn (2024), "Scikit learn." `https://scikit-learn.org/stable/index.html`, python package.

R. Tibshirani (1996), Journal of the Royal Statistical Society: Series B (Methodological) **58** (1), 267–288, https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1996.tb02080.x.

L. Yang, and A. Shami (2020), Neurocomputing **415**, 295–316.

A. Appendices

1. Appendix A: Source code

The code used for this project is available at https://github.com/GauteJ1/FYS-STK-projects. Instructions for running the code are located in `project_1/README.md`. All plots and results in this paper are easily reproducible in the Jupyter notebook files in this repository, by following the instructions mentioned above.

2. Appendix B: Pen and paper calculations

In this section, the calculations for some of our theretical expressions are presented.

$$\begin{aligned}\mathbb{E}[y_i] &= \mathbb{E}[\sum_j x_{ij}\beta_j + \epsilon_i] \\ &= \mathbb{E}[\sum_j x_{ij}\beta_j] + \mathbb{E}[\epsilon_i] \\ &= \sum_j x_{ij}\beta_j + 0 \\ &= \mathbf{X}_{i,*}\boldsymbol{\beta} \end{aligned} \tag{A1}$$

Eq. A1: Calculation of the expectation value of the true y_i, using the assumption that $\mathbb{E}[\epsilon_i] = 0$ as $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

$$\begin{aligned}\mathrm{var}(y_i) &= \mathbb{E}[(y_i - \mathbb{E}[y_i])^2] \\ &= \mathbb{E}[y_i^2] - \mathbb{E}[y_i]^2 \\ &= \mathbb{E}[(\sum_j x_{ij}\beta_j + \epsilon_i)^2] - \mathbb{E}[y_i]^2 \\ &= \mathbb{E}[(\sum_j x_{ij}\beta_j)^2] + \mathbb{E}[\epsilon_i^2] + 2\mathbb{E}[\sum_j x_{ij}\beta_j\epsilon_i] - \mathbb{E}[y_i]^2 \\ &= \mathbb{E}[y_i]^2 + \mathbb{E}[\epsilon_i^2] + 2\mathbb{E}[y_i]^2 \cdot 0 - \mathbb{E}[y_i]^2 \\ &= \sigma^2 \end{aligned} \tag{A2}$$

Eq. A2: Calculation of the variance of the true y_i. Here $\mathbb{E}[\epsilon_i] = 0$ and $\mathbb{E}[\epsilon_i^2] = \mathrm{var}(\boldsymbol{\epsilon}) = \sigma^2$, again as $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

$$\begin{aligned}\mathbb{E}[\hat{\boldsymbol{\beta}}] &= \mathbb{E}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}] \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{y}] \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\ &= \mathbb{1}\boldsymbol{\beta} \\ &= \boldsymbol{\beta} \end{aligned} \tag{A3}$$

Eq. A3: Calculation of the bias for the OLS estimator $\hat{\boldsymbol{\beta}}$. The result from Eq. A1 is used going from the second to the third line.

$$\begin{aligned}\mathrm{var}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}[(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])^2] \\ &= \mathbb{E}[(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])^T] \\ &= \mathbb{E}[(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})^T] \\ &= \mathbb{E}[(\hat{\boldsymbol{\beta}} - \boldsymbol{\beta})(\hat{\boldsymbol{\beta}}^T - \boldsymbol{\beta}^T)] \\ &= \mathbb{E}[\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T] - \mathbb{E}[\hat{\boldsymbol{\beta}}\boldsymbol{\beta}^T] - \mathbb{E}[\boldsymbol{\beta}\hat{\boldsymbol{\beta}}^T] + \mathbb{E}[\boldsymbol{\beta}\boldsymbol{\beta}^T] \\ &= \mathbb{E}[\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T] - \boldsymbol{\beta}\boldsymbol{\beta}^T - \boldsymbol{\beta}\boldsymbol{\beta}^T + \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= \mathbb{E}[\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T] - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= \mathbb{E}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y})^T] - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= \mathbb{E}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\mathbf{y}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-T}] - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{y}\mathbf{y}^T]\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-T} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\mathbf{X}^T + \mathbb{E}[\boldsymbol{\epsilon}^2])\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-T} \\ &\quad - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= \boldsymbol{\beta}\boldsymbol{\beta}^T + \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} \end{aligned} \tag{A4}$$

Eq. A4: Calculation of the variance of the ordinary least squares estimator.

$$\begin{aligned}\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] &= \mathbb{E}[(\mathbf{y} - \mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2] \\ &= \mathbb{E}[(\mathbf{y} - \mathbb{E}[\tilde{\mathbf{y}}])^2] + \mathbb{E}[(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})^2] \\ &\quad + 2\mathbb{E}[(\mathbf{y} - \mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}] - \tilde{\mathbf{y}})] \\ &= \mathbb{E}[(\mathbf{f} + \boldsymbol{\epsilon} - \mathbb{E}[\tilde{\mathbf{y}}])^2] + \mathbb{E}[(\tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}])^2] \\ &= \mathbb{E}[(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] \\ &\quad + 2\mathbb{E}[(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])\boldsymbol{\epsilon}] + \mathbb{E}[(\tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}])^2] \\ &= \mathbb{E}[(\mathbf{f} - \mathbb{E}[\tilde{\mathbf{y}}])^2] + \mathbb{E}[(\tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] \\ &= \mathrm{Bias}(\tilde{\mathbf{y}})^2 + \mathrm{var}(\tilde{\mathbf{y}}) + \sigma^2 \end{aligned} \tag{A5}$$

Eq. A5: Decomposition of the expected loss in terms of bias and variance. As $\mathbf{f}$ is unknown, $\mathbf{y}$ is used in calculations.

3. Appendix C: Plots not included in the report

The following are all plots requested (or hinted at) in the assignment which we do not include in the report itself.

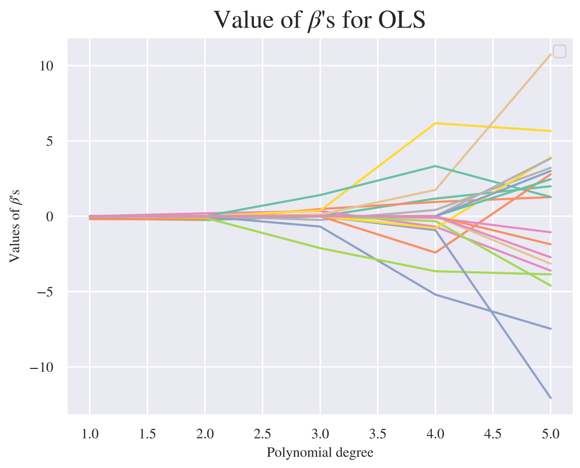

FIG. 13: The values of the parameters $\boldsymbol{\beta}$ for the OLS regression model trained on Franke function data against the polynomial degree. Each colored line represents one β_j.

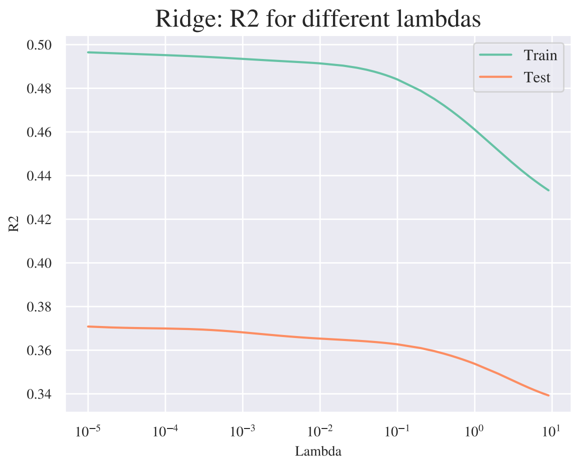

FIG. 15: R^2 for the Ridge regression model trained on Franke function data against the penalty parameter λ

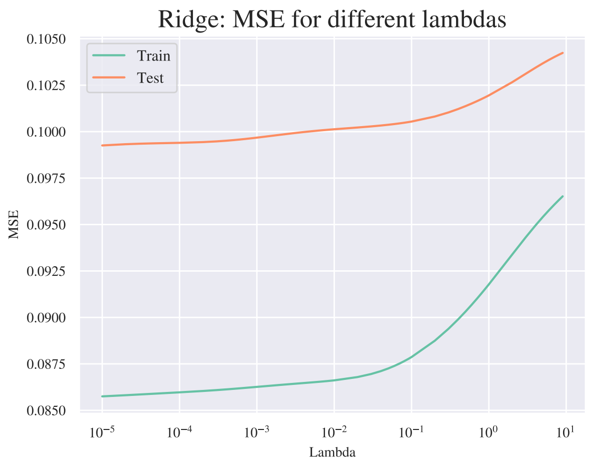

FIG. 14: MSE for the Ridge regression model trained on Franke function data against the penalty parameter λ

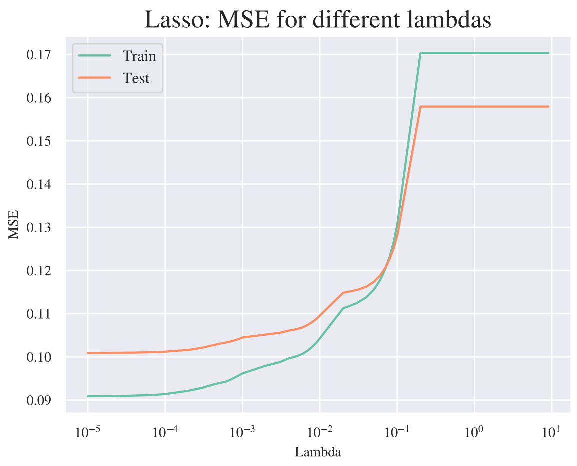

FIG. 16: MSE for the Lasso regression model trained on Franke function data against the penalty parameter λ

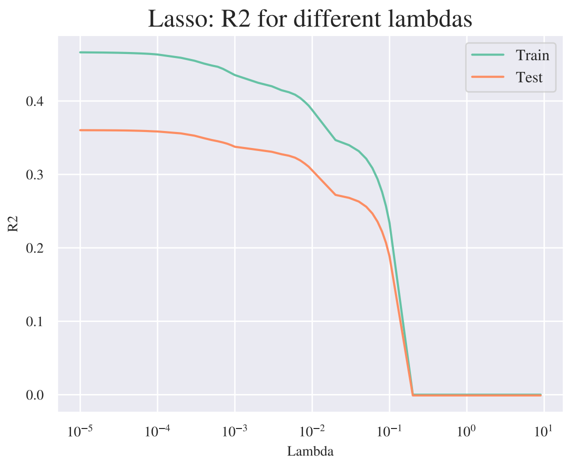

FIG. 17: R^2 for the Lasso regression model trained on Franke function data against the penalty parameter λ

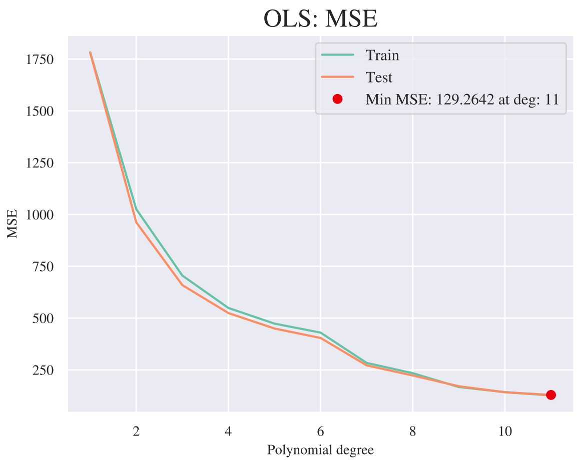

FIG. 19: MSE for the OLS regression model trained on terrain data against the polynomial degree.

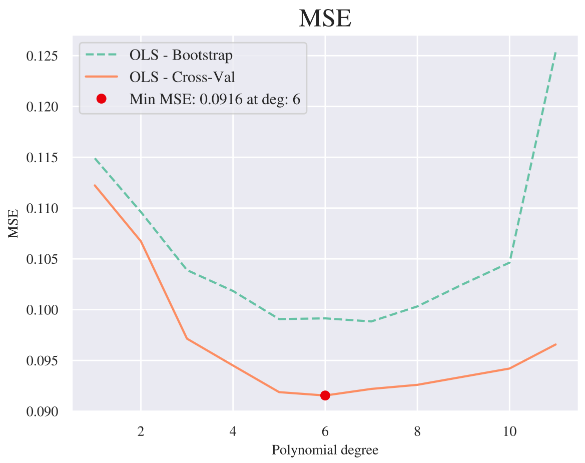

FIG. 18: Comparison of an OLS regression model trained on Franke function data with 10-fold cross-validation and bootstrapped with 100 bootstrap samples.

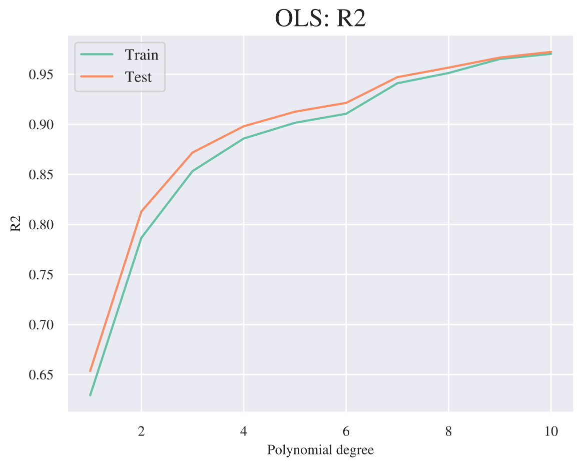

FIG. 20: R^2 for the OLS regression model trained on terrain data against the polynomial degree.

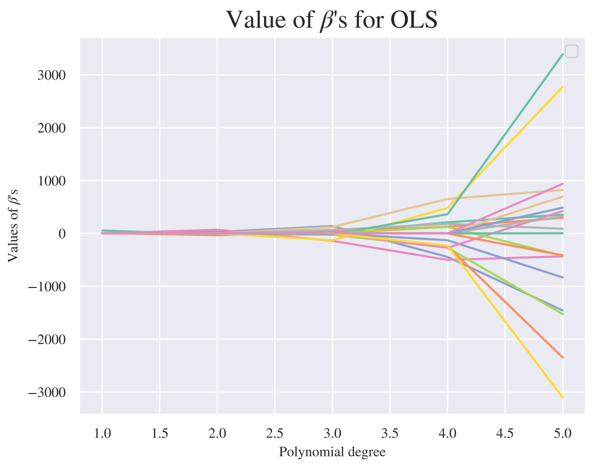

FIG. 21: The values of the parameters $\boldsymbol{\beta}$ for the OLS regression model trained on terrain data against polynomial degree. Each colored line represents one β_j.

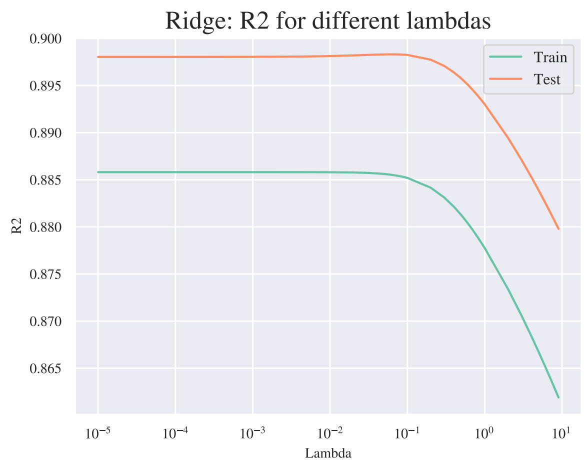

FIG. 23: R^2 for the Ridge regression model trained on terrain data against the polynomial degree.

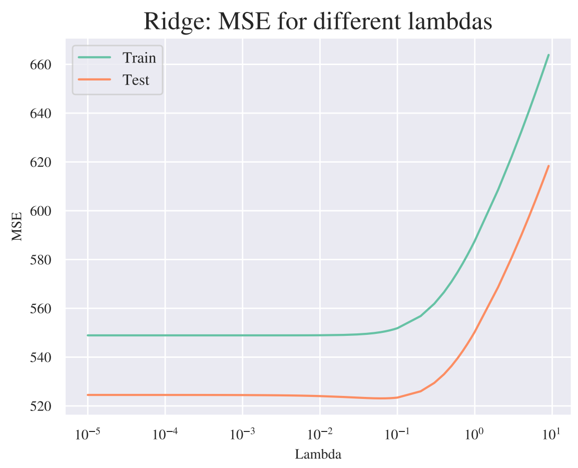

FIG. 22: MSE for the Ridge regression model trained on terrain data against the polynomial degree.

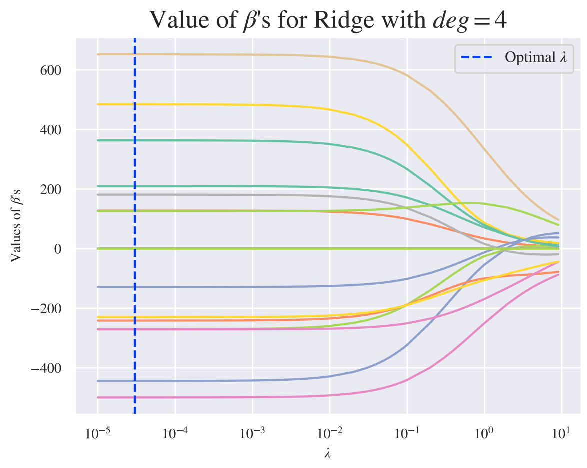

FIG. 24: The values of the parameters $\boldsymbol{\beta}$ for the Ridge regression model trained on terrain data with polynomials up to a degree of seven for increasing values of λ. Each colored line represents one β_j.

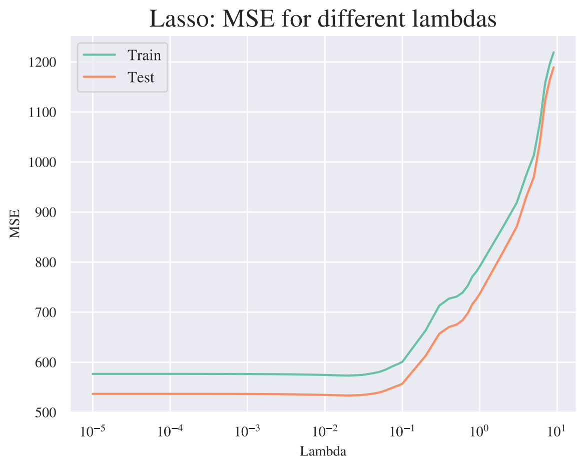

FIG. 25: MSE for the Lasso regression model trained on terrain data against the polynomial degree.

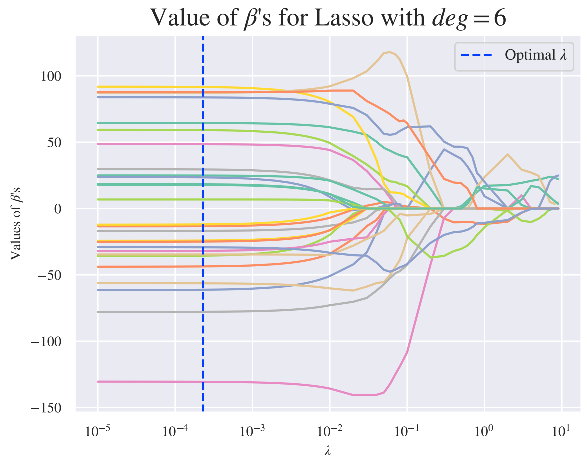

FIG. 27: The values of the parameters $\boldsymbol{\beta}$ for the Lasso regression model trained on terrain data with polynomials up to a degree of seven for increasing values of λ. Each colored line represents one β_j.

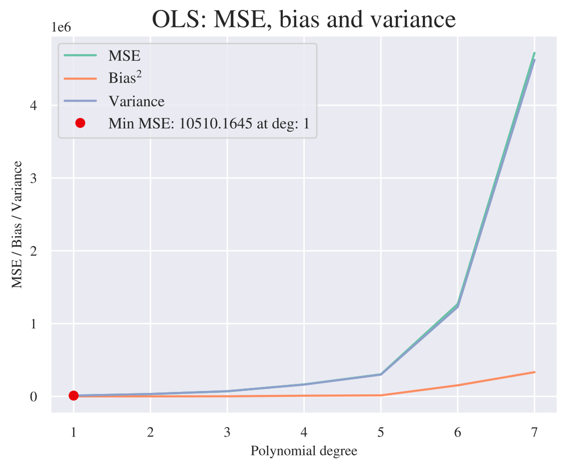

FIG. 28: MSE decomposition into terms of bias and variance term for an ordinary least squares model trained on terrain datawith bootstrapping with 100 samples. The minimum MSE is marked with a red dot.

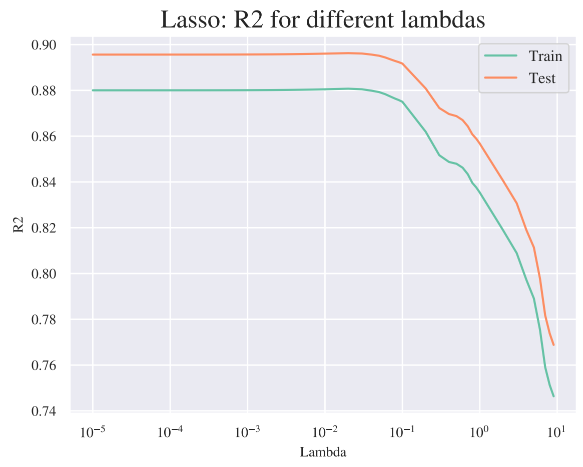

FIG. 26: R^2 for the Lasso regression model trained on terrain data against the polynomial degree.