UNIVERSITY OF OSLO

Project I

Applied Data Analysis and Machine Learning

Involved students

Johan Blakkisrud
Toralf Husevåg
Frederik Eichenberger

Professor: Morten Hjorth-Jensen

Oslo, October 16, 2023

"Geography is just physics slowed down, with a couple of trees stuck in it"
(Terry Pratchett, 1948-2005)

1 | Abstract

This project evaluated linear regression methods to fit polynomial functions to model the Franke function with added noise, in addition to topographic data of four different Norwegian regions. Ordinary least square, Ridge, and Lasso methods was performed, in combination with resampling techniques such as the bootstrap method or cross-validation. The polynomial functions models the Franke data satisfactory but not the terrain data.

2 | Introduction

Regression methods are applied in various fields, ranging from cancer prediction methods [1] to financial risk assessment [2], even in climate and emission research it has found its applications [3]. In this project, we apply machine learning methods to topological terrain data of four Norwegian cities to find functions representing the earth's surface of these areas. First, the methods of Ordinary least squares, Ridge, and Lasso regression will be tested on the Franke function, which represents a simplified terrain given an elevation based on x and y-coordinates, and later on terrain data. The models will be evaluated based on their complexity using the mean squared error. Furthermore, resampling techniques such as cross-validation and bootstrap methods will be used to optimize and quanitfy the generalized performance of the models.

3 | Theory

3.1 | Franke function

To develop an understanding of different linear regression methods, the Franke function is used as a simplified example

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right). \end{aligned} \tag{3.1}$$

For our purpose, we will reduce the function to the interval $x,\ y \in [0,\ 1]$ and add a small stochastic noise $\varepsilon \sim \mathcal{N}(0,\ 0.1)$ on top. The Franke function is illustrated in figure 3.1 where both an idealized version and the data with noise added (taken as the ground truth in subsequent analyses) are shown.

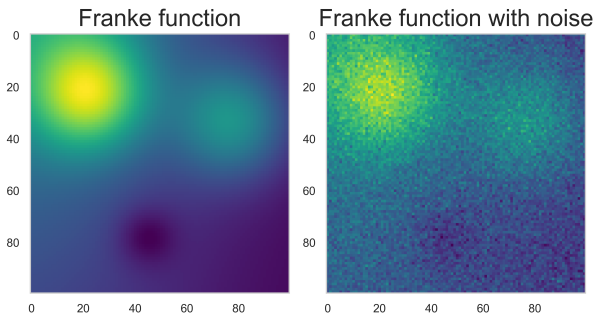

Figure 3.1: The idealized (left) and with added noise (right) representations of the Franke-function

3.2 | Test and training data

To evaluate the final model's ability to generalize a portion of the available data is held-out from training. The model performance on this "unseen" test-data is known as the generalization error.

3.3 | Ordinary least squares

This method assumes a non-stochastic function $f(x)$ disturbed by a stochastic normally distributed noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$, such that the measurement data is y is given by

$$y = f(x) + \epsilon \quad \text{with } \varepsilon \sim \mathcal{N}(0, \sigma^2) \tag{3.2}$$

OLS uses a polynomial approach to fit the data by setting up a design matrix X given by

$$X = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{p-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{p-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{p-1} \end{pmatrix}, \tag{3.3}$$

where n is the amount of datapoints and p the polynomial degree. The fitted data is then obtained by calculating $\tilde{y}$ via

$$\tilde{\mathrm{y}} = X \cdot \beta, \tag{3.4}$$

where β can be though of as weight determining how a feature affects the prediction [4]. These minimise the mean squared error (MSE), which is given by

$$MSE = \frac{1}{n} \sum_{k=0}^{n-1} (y_k - \tilde{\mathrm{y}}_k)^2. \tag{3.5}$$

In case of OLS, the MSE value is our cost function. To minimise this function, β can be determined via

$$\beta = (X^T X)^{-1} X^T y, \tag{3.6}$$

as long as matrix X is a non-singular matrix.

3.4 | Ridge

Ridge regression relies on a different cost function, which is given by

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\} + \lambda \boldsymbol{\beta}^T \boldsymbol{\beta}, \tag{3.7}$$

where the regularization strength (hyperparameter) λ may be used for tuning the model. Thus, the weight factor β is given by

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^T \boldsymbol{y} \tag{3.8}$$

with $\boldsymbol{I}$ being the identity matrix [5]. Ridge, as well as LASSO regression can be very handy in case X is a singular matrix where OLS does not work.

3.5 | LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO) is a useful alternative to OLS and Ridge. LASSO allows the feature coefficients to go to exactly zero, and are as such able to isolate important features (selection).
The cost function to be minimized by the Lasso method is given by

$$C(\boldsymbol{\beta}) = \sum_{i=0}^{p-1} (y_i - \beta_i)^2 + \lambda \sum_{i=0}^{p-1} |\beta_i| = \sum_{i=0}^{p-1} (y_i - \beta_i)^2 + \lambda \sum_{i=0}^{p-1} \sqrt{\beta_i^2}. \tag{3.9}$$

To minimize this function, β is given by

$$\hat{\boldsymbol{\beta}}_i^{\text{Lasso}} = \left\{ \begin{array}{ccc} y_i - \frac{\lambda}{2} & \text{if} & y_i > \frac{\lambda}{2} \\ y_i + \frac{\lambda}{2} & \text{if} & y_i < -\frac{\lambda}{2} \\ 0 & \text{if} & |y_i| \leq \frac{\lambda}{2} \end{array} \right\} \tag{3.10}$$

.

Therefore, for large enough λ values, unique features of the model can be completely excluded since the values are driven to 0, leading to a sparse model [6].

3.6 | Model variance

In our measurement data y, we assume to find a normally distributed noise term $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Thus we can determine the expectation value of our measurement $\mathbb{E}[y_i]$ through

$$\mathbb{E}[y_i] = \mathbb{E}[f(x) + \varepsilon_i] = \underbrace{\mathbb{E}[f(x_i)]}_{\sum_j x_{ij}\beta_j} + \underbrace{\mathbb{E}[\varepsilon_i]}_{=0} = \sum_j x_{ij}\beta_j = \mathbf{X}_{j*}\boldsymbol{\beta}, \tag{3.11}$$

where $\mathbf{X}\boldsymbol{\beta} = \tilde{\boldsymbol{y}}$ is the solution we obtained through OLS. The variance of our measurement is given by

$$\begin{aligned} \mathrm{Var}(y_i) = \mathrm{Var}[f(x_i) + \varepsilon_i] &= \mathrm{Var}[f(x_i)] + \underbrace{\mathrm{Var}[\varepsilon_i]}_{\sigma^2} + 2 \cdot \underbrace{\mathrm{Cov}[f(x_i), \varepsilon_i]}_{=0} \\ &= \mathbb{E}[f(x_i)^2] - \mathbb{E}[f(x_i)]^2 + \sigma^2 \\ &= f(x_i)^2 - f(x_i)^2 + \sigma^2 \\ &= \sigma^2. \end{aligned} \tag{3.12}$$

Since $f(x)$ is assumed to be a non-stochastic value, only the noise contributes to the variance. Furthermore, we can show in case of Ordinary least squares that the expectation value of the optimal parameter $\hat{\boldsymbol{\beta}}$ is given by the β obtained through the matrix inversion,

$$\mathbb{E}[\hat{\boldsymbol{\beta}}] = \mathbb{E}[(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}] = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T \cdot \underbrace{\mathbb{E}[\boldsymbol{y}]}_{\boldsymbol{X}\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{\beta}. \tag{3.13}$$

The variance of the optimal parameter is then given by

$$\begin{aligned} \mathrm{Var}[\hat{\boldsymbol{\beta}}] =& \mathbb{E}\left\{(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])(\hat{\boldsymbol{\beta}} - \mathbb{E}[\hat{\boldsymbol{\beta}}])^T\right\} \\ =& \mathbb{E}\left\{\left((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{\beta}\right)\left((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} - \boldsymbol{\beta}\right)^T\right\} \\ =& \mathbb{E}\left\{\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}\boldsymbol{y}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\right\} + \boldsymbol{\beta}\boldsymbol{\beta}^T \\ & - \boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T \\ =& \mathbb{E}\left\{\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}\boldsymbol{y}^T\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\right\} - \boldsymbol{\beta}\boldsymbol{\beta}^T, \end{aligned} \tag{3.14}$$

where

$$\mathbb{E}[\boldsymbol{y}\boldsymbol{y}^T] = \boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}. \tag{3.15}$$

This yields for the variance

$$\begin{aligned} \mathrm{Var}[\beta] =& (\boldsymbol{X}^T\boldsymbol{X})^{-1}(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{\beta}\boldsymbol{\beta}^T + (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\sigma^2\boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ =& \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}. \end{aligned} \tag{3.16}$$

3.7 | Resampling techniques

Resampling is a family of techniques commonly used in machine learning for evaluating model performance on a variety of adjustable model parameters (hyperparameter-tuning) or the model response to perturbations in the available data (i.e. estimating uncertainties).

To analyze the models we reformulate the MSE given by $\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right]$, such that

$$\begin{aligned} \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(f(x) + \varepsilon - \tilde{\boldsymbol{y}})^2\right] =& \mathbb{E}\left[f(x)^2 + \varepsilon^2 - \tilde{\boldsymbol{y}}^2 + 2f(x)\varepsilon - 2f(x)\tilde{\boldsymbol{y}} - 2\varepsilon\tilde{\boldsymbol{y}}\right] \\ =& \mathbb{E}\left[f(x)^2 + \tilde{\boldsymbol{y}}^2 - 2f(x)\tilde{\boldsymbol{y}}\right] + \sigma^2. \end{aligned} \tag{3.17}$$

Introducing a 'smart 0', given through the expression $\mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}] = 0$, we can separate the terms above yielding

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \underbrace{\mathbb{E}[(f(x) - \mathbb{E}[\tilde{\boldsymbol{y}}]^2]}_{(\mathrm{Bias}(\tilde{\boldsymbol{y}}))^2} + \underbrace{\mathbb{E}[(\tilde{\boldsymbol{y}}}_{\mathrm{Var}(\tilde{\boldsymbol{y}})} - \mathbb{E}[\tilde{\boldsymbol{y}}]^2] + \mathbb{E}[(f(x) - \mathbb{E}[\tilde{\boldsymbol{y}}])(\underbrace{\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})}_{=0}] + \sigma^2. \tag{3.18}$$

Thus, we finally obtain

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \left[\text{Bias}(\tilde{\boldsymbol{y}})\right]^2 + \text{Var}(\tilde{\boldsymbol{y}}) + \sigma^2. \tag{3.19}$$

This now offers us the perspective of analyzing our model quite descriptively. The bias tells us how far our models are away form the actual data, while the variance is an indicator on the width of scattering of our models. Increasing model complexity will decrease the bias but in return will increase the variance. This is shown for a bootstrap sampling in the figure 5.5.

3.7.1 | K-Fold cross-validation

K-fold cross-validation divides data into k subsets (folds) of (as far as possible) equal sizes, which are used to evaluate the model. Given q hyperparameters for the model, each having N possible states, an ensemble of N^q models (for each combination of hyperparameters) are trained on $k-1$ folds before being evaluated on the final fold - known as the *validation set*. This process is repeated k times, such as all folds have been used once for validation. To find the optimal subset of hyperparameters across the q-dimensional grid, one may simply take the lowest average MSE or a similar cost function across the k validation folds. This process is known as *hyperparameter tuning*.

3.7.2 | The bootstrap

The bootstrap method, as defined by Efron B. [7], draws a random sample of values with a given sample size from the data set and uses it to calculate a singular model. Each pick in the bootstrap sample are drawn with replacement, allowing a single observation from the original data to occur multiple times in the bootstrapped sample. Repeating the drawing of a random sample and estimating the model's quantity allows us to obtain a distribution of model predictions, which can be analyzed in terms of variance and the bias of our predictions, as depicted in figure 5.5.

4 | Materials and Methods

4.1 | Datasets

This work contains two parts with their respective data set. The first data set is a synthetic one generated from the Franke function by calculating the function on a regular xy-grid. The function was sampled from 0 to 1 with regular intervals of 0.01.

```
1
2 STEP_SIZE = 0.01
3
4 x = np.arange(0, 1, STEP_SIZE)
5 y = np.arange(0, 1, STEP_SIZE)
6
7 x, y = np.meshgrid(x, y)
8
9 franke_z = project_utils.FrankeFunction(x, y, add_noise=True)
```

Here, the function `FrankeFunction` computes the Franke function according to the definition given in equation (3.1). A boolean parameter `add_noise` will add a Gaussian distributed noise, with a default value of 0.1, but can be set with the parameter `sigma`.
The second dataset is geographical data from the NASA's Shuttle Radar Topography Mission, an 11 day mission in february 2000, which mapped the elevation of 80% of the earth's surface using a dual antenna setup [8]. Each image consists of 3601×1801 single-channel gray-level values as 16-bit integer representations. Four images from norway are considered in this analysis (figure 4.1), but the subsequent subchapters uses the Møsvatn image data to exemplify the pipeline.

4.2 | Overall implementation

The analysis of the Franke-function was performed under the constriction of little use of libraries. Thus each sub-task was written as a separate function in a python script named `analysis_franke.py`. The analysis of the second data set, using less constrictions, was implemented using library functions and an analysis class. Shared functions and classes were put in the module file `utils.py`. Both of the files can be found in the accompanying github repository that can be found on this link.

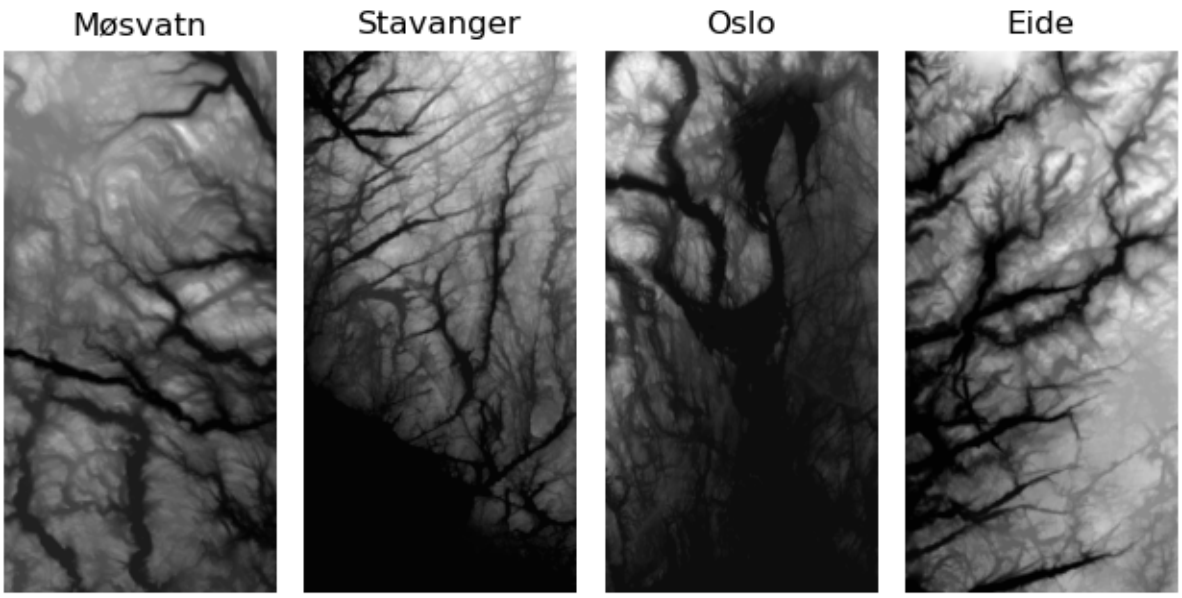

Figure 4.1: The four terrains considered in this analysis, all from the Shuttle Radar Topography Mission.

4.3 | Splitting the data into train and tests partitions

Two ways to reserve a portion of the data for testing was considered: either randomly selecting pixels to belong in the test set or selecting a coherent chunk of the image for the test set (see figure 4.2).
A fixed random seed of 42 was used throughout. For the randomly selected pixels, the following snippet, illustrated with data set one, was used:

```
1
2  franke_z_flat = franke_z.ravel()
3
4  x = np.ravel(x)
5  y = np.ravel(y)
6
7  index_vals = np.array(range(np.prod(franke_z.shape)))
8
9  idx_train, idx_test = train_test_split(
10     index_vals, test_size=0.3, random_state=42)
```

And for the splitting of training/test in chunks:

```
1  index_vals = np.array(range(np.prod(franke_z.shape)))
2  idx_train = index_vals[:int(len(index_vals) * (1 - self.test_set_ratio))]
3  idx_test = index_vals[max(idx_train) + 1:]
```

4.4 | Normalization

Looking at the histograms in figure 4.3, both the Franke and terrain data seem to be normally distributed. Therefore, the data sets were scaled and centered to zero mean and set to a standard deviation of 1 using z-score standardization, having the mean and standard deviation estimated from each training set.

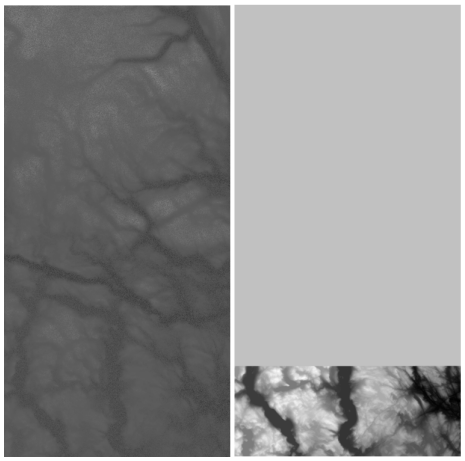

Figure 4.2: Test sets for the two methods of dividing the images into training and test partitions, either by randomly selecting pixels (left) or by a coherent chunk (right). The blank region / intermediate zero-valued pixels are added to the test sets for visualization, and are not included in the actual test sets.

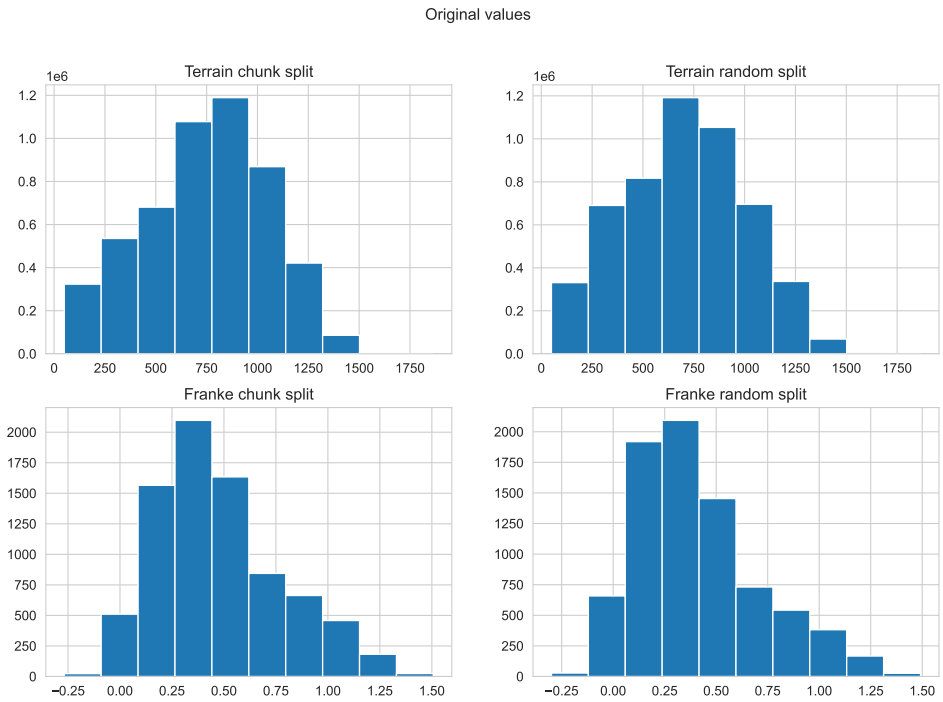

Figure 4.3: Histograms showing the training data for the Franke function and the Møsvatn terrain SRTM data, for both methods of splitting into train and test sets.

As the input features are coordinates, each value appears only once and are therefore shifted to a zero mean using min-max feature-scaling.

4.5 | Part A

The ordinary least square solution of the optimization problem was found by the function `OLS` found in `utils.py`. The function returns the mean squared error (MSE) and R^2 of the training data, the fitted training data and the β-values. These β-values were calculated by

```
beta = np.linalg.pinv(X.T.dot(X)).dot(X.T).dot(z)
```

The design matrix X was made by the function `generate_design_matrix`. This function generates design matrices for a sum of polynomials for a set number of polynomials.
The MSE and R^2 were calculated for increasing number of polynomials. The full implementation can be found in the function `part_a` in `analysis_franke.py`.

4.6 | Part B

Here, ridge-regression was performed by implementing a simple function `ridge` that takes as input a design matrix, test data and a λ-value. It uses matrix inversion for obtaining the beta values:

```
beta = np.linalg.inv(X.T.dot(X) + lmb*np.eye(X.shape[1])).dot(X.T).dot(z)
```

Values of λ between 10^{-3} to 10^3 were explored and the MSE and R^2 for train and test-data was calculated. The full implementation can be found in the function `part_b` in `analysis_franke.py`.

4.7 | Part C

Carried out as in part b but with the use of LASSO instead of ridge-regression. To perform the calculation, the function `linear_model.LASSO` from the library `sklearn` was used and the MSE and r^2 of both training and test-data was calculated:

```
clf = linear_model.Lasso(alpha=lmb,
    fit_intercept=False,
    max_iter=10000)

clf.fit(X_train, z_train)

z_pred_train = clf.predict(X_train)
z_pred_test = clf.predict(X_test)

MSE_train = mean_squared_error(z_train, z_pred_train)
MSE_test = mean_squared_error(z_test, z_pred_test)

R2_train = r2_score(z_train, z_pred_train)
R2_test = r2_score(z_test, z_pred_test)
```

A range of λs from 10^{-3} to 10^3 was explored. The full implementation can be found in the file `analysis_franke.py` in the function `part_c`

4.8 | Part E

The bias-variance trade-off was explored by the bootstrap re-sampling method. The number of bootstraps was set to 100, and the number in each set was set to 200, which is roughly 3% of the training data.
The boostrap-method for a single polynomial is illustrated in the following snippet, where the function `OLS` previously described was used:

```

p = 4 # The order of polynomial

X_test = project_utils.generate_design_matrix(
    x[idx_test], y[idx_test], p)

z_test = franke_z_flat[idx_test]

x_train = x[idx_train]
y_train = y[idx_train]
z_train = franke_z_flat[idx_train]

k_bootstraps = 100
n_samples = 200

print("The relativel size of the bootstrap samples: ",
    n_samples/len(x_train))

bootstrap_MSE = np.zeros(k_bootstraps)
bootstrap_bias = np.zeros(k_bootstraps)
bootstrap_variance = np.zeros(k_bootstraps)

for i in range(k_bootstraps):

```

```
    _x, _y, _z = resample(x_train,
    y_train,
    z_train,
    n_samples=n_samples)

    _X = project_utils.generate_design_matrix(_x, _y, p)

    _MSE, _R2, _Z, _beta = project_utils.OLS(_X, _z)

    z_pred = X_test.dot(_beta)

    MSE = mean_squared_error(z_test, z_pred)
    bias = np.mean((z_test - np.mean(z_pred))**2)
    variance = np.var(z_pred)

    bootstrap_MSE[i] = MSE
    bootstrap_bias[i] = bias
    bootstrap_variance[i] = variance

result = {"Polynomial": p,
    "Error": np.mean(bootstrap_MSE),
    "Bias": np.mean(bootstrap_bias),
    "Variance": np.mean(bootstrap_variance)}
```

This was done for polynomial up to the order of 9, and the error, bias and variance calculated. The full implementation is found in `part_e`.

4.9 | Part F

A simplified K-mean cross validation function was implemented and can be found as `k_fold_cross_validation` that accepts parameter x, y and z for a 2D-function together with a randomized list of indices, type of regression, λ-value of relevant, number of folds and the order of polynomial. The functions returns a pandas data frame of the results:

```
result = {
    "MSE_train",
    "MSE_test",
    "R2_train",
    "R2_test",
    "Polynomial",
    "Lambda",
    "Type",
    "Folds"}
```

The k-fold cross validation was implemented thusly:

```
index_groups = np.array_split(idx, k)

MSE_test_kfold = np.zeros(k)
MSE_train_kfold = np.zeros(k)
R2_test_kfold = np.zeros(k)
R2_train_kfold = np.zeros(k)

for g, i in zip(index_groups, range(k)):

    x_fold = x[g]
    y_fold = y[g]
    z_fold = z[g]

    x_train = x[~g]
    y_train = y[~g]
    z_train = z[~g]

    X_train = generate_design_matrix(x_train, y_train, p)
    X_test = generate_design_matrix(x_fold, y_fold, p)

    if type == "OLS":
        MSE_train, R2_train, z_tilde_train, beta_train = OLS(X_train, z_train)
        z_tilde_test = X_test.dot(beta_train)

MSE_test = MSE(z_fold, z_tilde_test)
R2_test = R2(z_fold, z_tilde_test)

```

```
MSE_test_kfold[i] = MSE_test
MSE_train_kfold[i] = MSE_train
R2_test_kfold[i] = R2_test
R2_train_kfold[i] = R2_train

MSE_train = np.mean(MSE_train_kfold),
MSE_test = np.mean(MSE_test_kfold),
R2_train = np.mean(R2_train_kfold),
R2_test = np.mean(R2_test_kfold),
```

This was calculated for OLS, ridge and LASSO-regression with λ-values from 10^{-3} to 10^{3}.

4.10 | Second implementation - Geographical data

The pipeline for analyzing the four topograpic terrain SRTM datasets, based on parts A-F, are shown in figure 4.4. The analysis was performed using methods and models from the scikit-learn library, implemented into the `TerrainAnalyser` class in `utils.py`. The terrain analysis utilized 3-fold cross-validation to estimate the optimal combination of ridge penalization α and polynomial degree p by minimizing the average MSE across the validation sets. The two methods of splitting the data into train / test / validation sets were treated as separate data sets for each SRTM dataset.

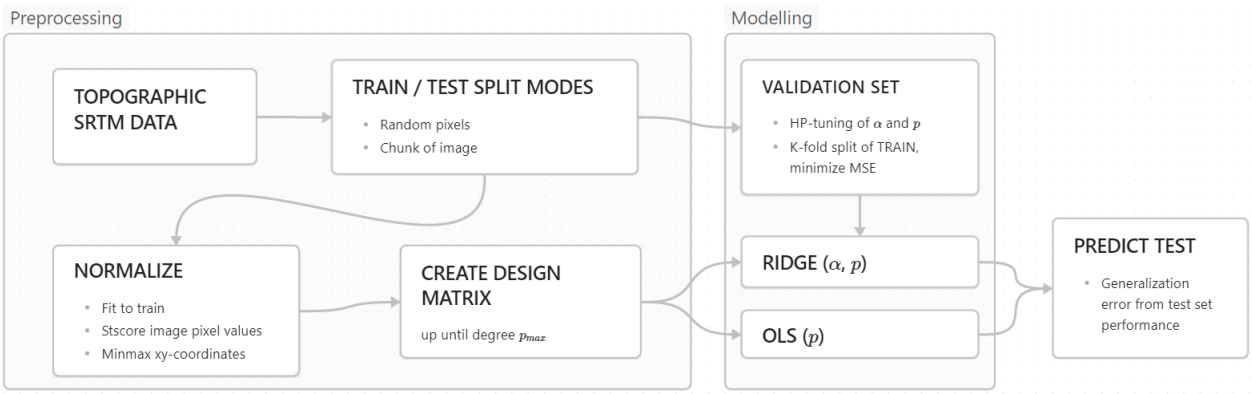

Figure 4.4: The pipeline for analyzing each of the SRTM datasets.

4.11 | Time-complexity

The OLS-approach was used on a terrain data-set that was down-sampled with a factor of 12, leaving a matrix size of 300 by 150. This dataset was fitted with polynomials ranging from 1 to 71 in intervals of four and the computation time recorded. This was performed on a Thinkpad P52 with 32 GB of RAM and an Intel i7 processor. The script to perform the timed calculations is `test_function_terrain.py` found in the github repo.

5 | Results

5.1 | OLS

Figure 5.1 shows the MSE and R^2 of the Franke function with increasing number of polynomials.

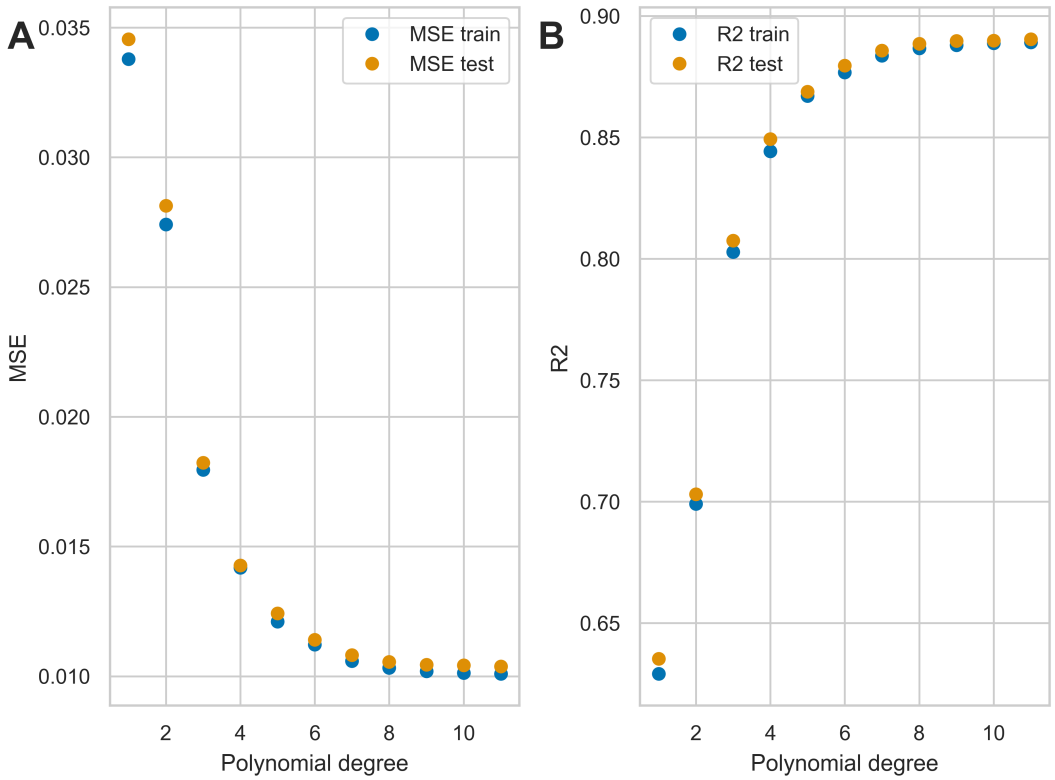

Figure 5.1: MSE and R^2-value of ordinary least square fitting the Franke-function for training and test data. With increasing model complexity, the error seems to decline. A plateau is reached after the 7^{th} degree.

Figure 5.2 shows the noisy Franke function (the ground truth), the idealized Franke function, the predicted results for an increasing number of polynomial as well as a difference map.

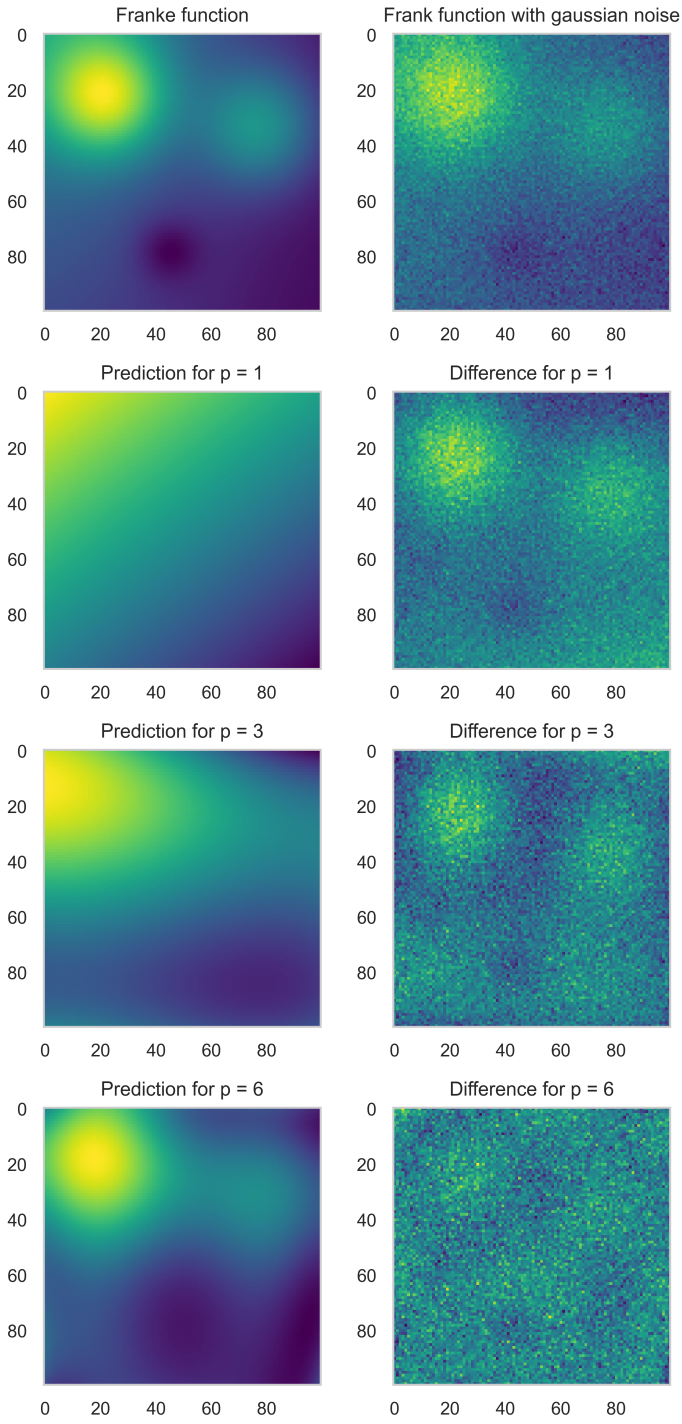

Figure 5.2: The noisy and idealized Franke function, the predicted Franke function, and the signed difference between the true and predicted results for an increasing number of polynomials. Notice that the difference-map initially has a structure but it disappears for higher polynomials.

5.2 | Ridge

The MSE and R^2 for both training and test for different values of λ is shown in figure 5.3.

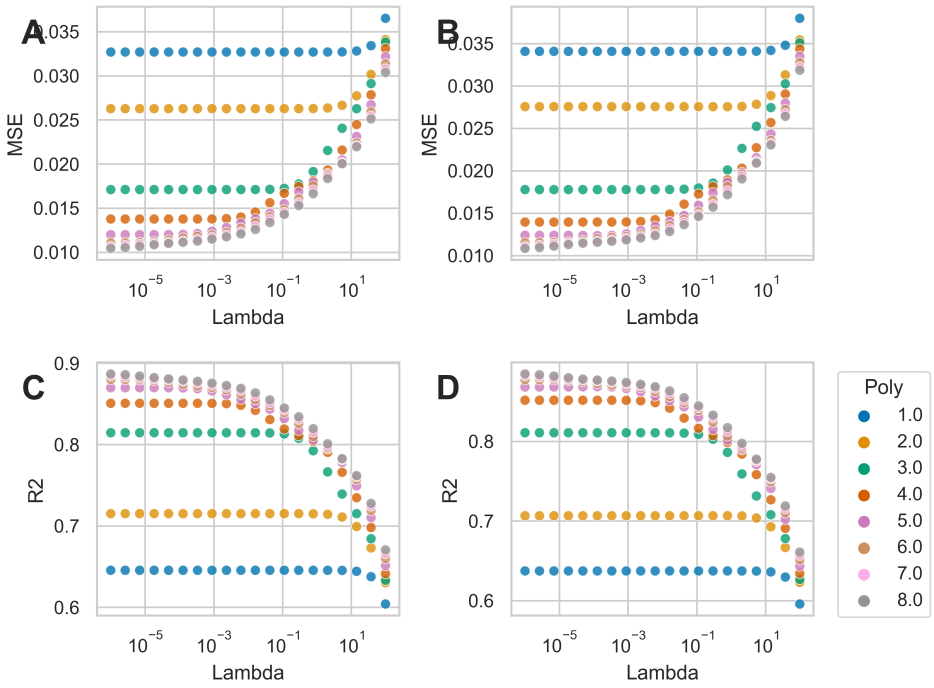

Figure 5.3: MSE (A and B) and R^2-value (C and D) of Ridge regression depending on the regularisation parameter λ. On the left side, the training errors are displayed. On the right side, the testing ones. The results are enhanced for smaller λ-values. Thus, it approximates the ordinary least square method. As seen in OLS, the test and training error is minimized for higher model complexity.

5.3 | LASSO

For the LASSO-analysis, the resulting MSE and R^2 for various λ-values are shown in figure 5.4

5.4 | Bootstrap

The bias, variance and error as a function of model complexity, i.e. number of polynomials are shown in figure 5.5.

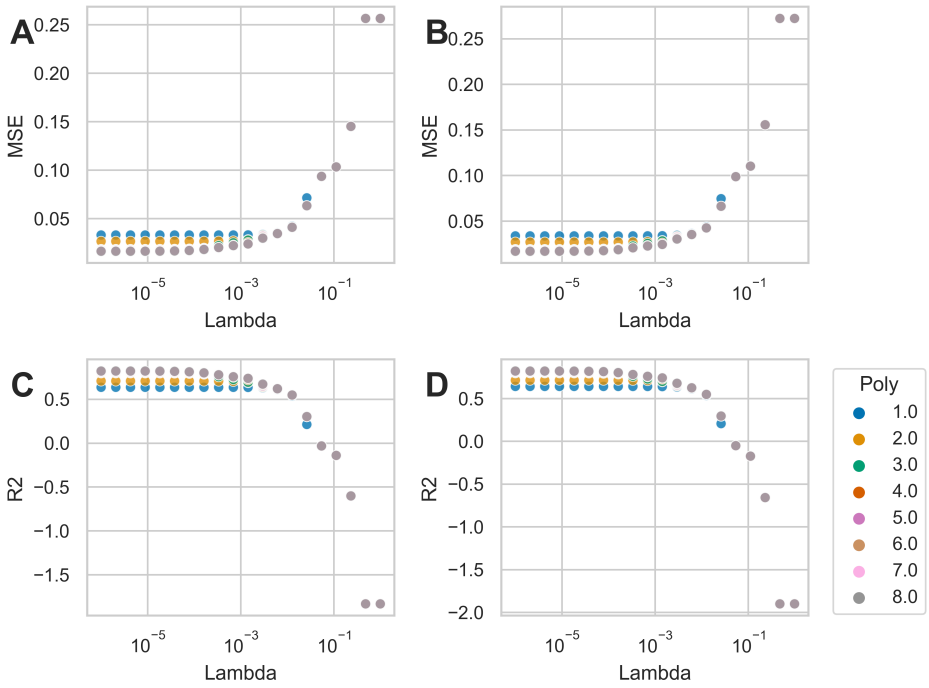

Figure 5.4: MSE (A and B) and R^2-values (C and D) of LASSO regression depending on the regularisation parameter λ. On the left side, the training errors are displayed. On the right side, the test. The results are enhanced for smaller λ-values. Thus, it approximates the ordinary least square method. As seen in OLS, the test and training error is minimized for higher model complexity.

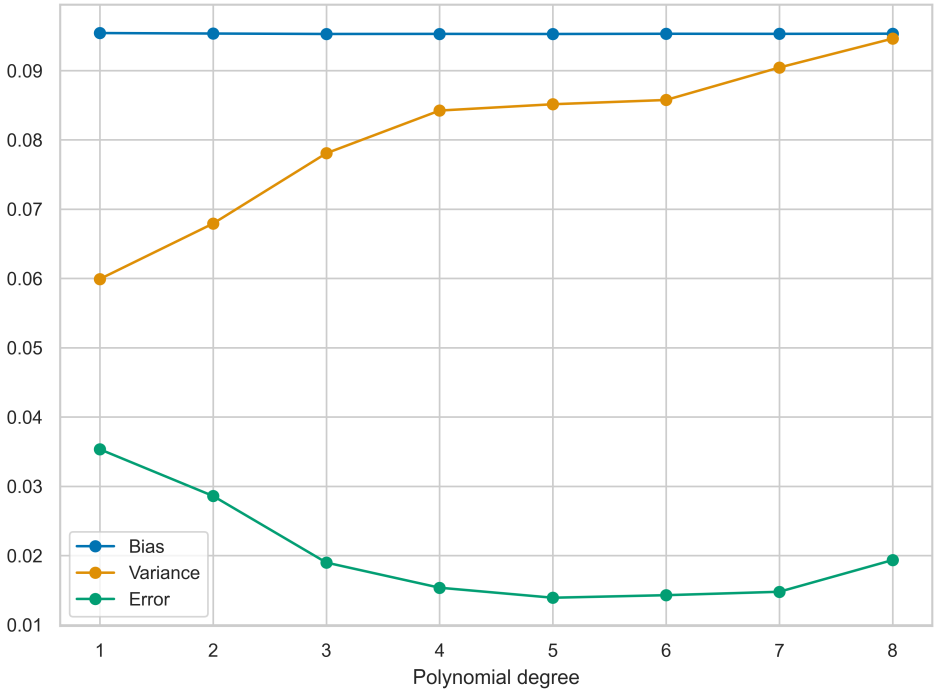

Figure 5.5: Bias-variance-plot for the Franke-function. The error is expected to have a minimum between under- and over-fitting, the bias to monotonically decrease and the variance to increase.

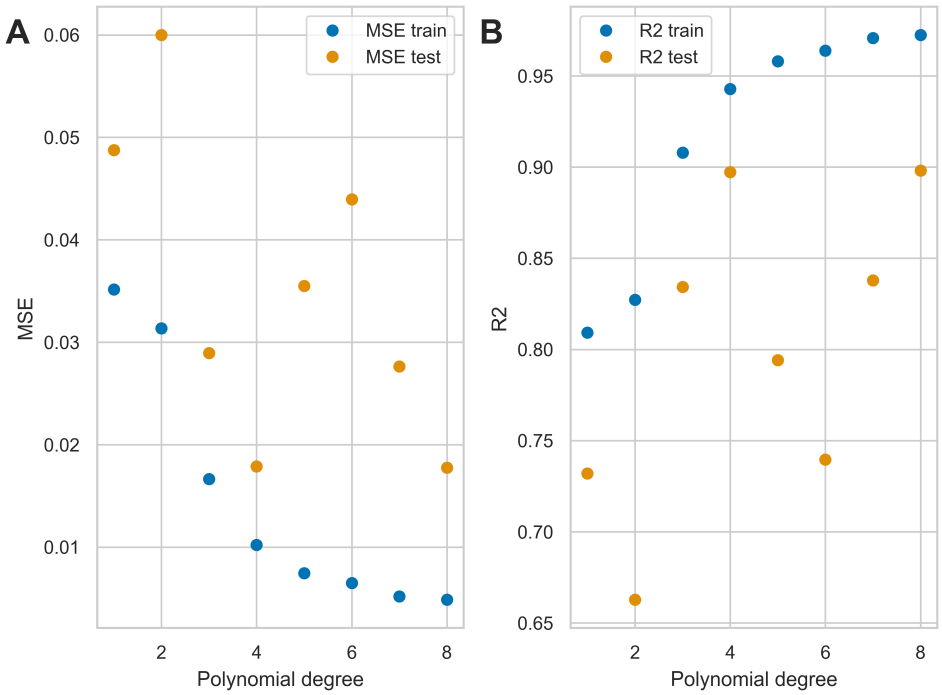

Figure 5.6: Train and test-MSE and R^2 for five fold cross validation using OLS for the Franke-function

5.5 | K-fold cross validation

The MSE and r^2-values for the three different regression analyses are shown in figures 5.6, 5.7 and 5.8 for five-fold cross validation. These are repeated in figures A.1, A.2 and A.3 for ten-fold cross-validation in the appendix.

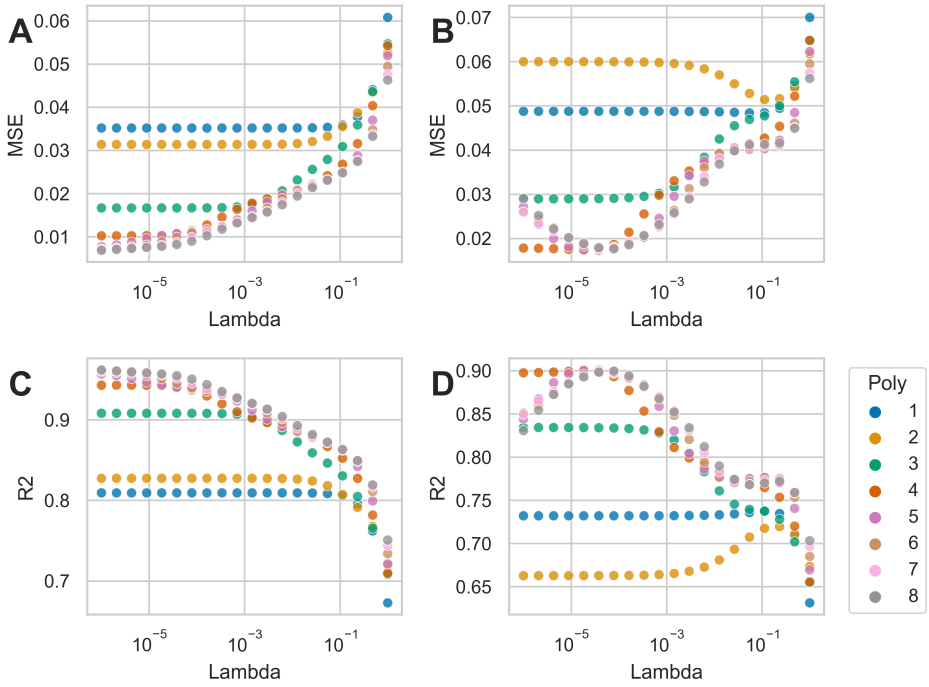

Figure 5.7: Five fold cross validation using ridge regression on the Franke function

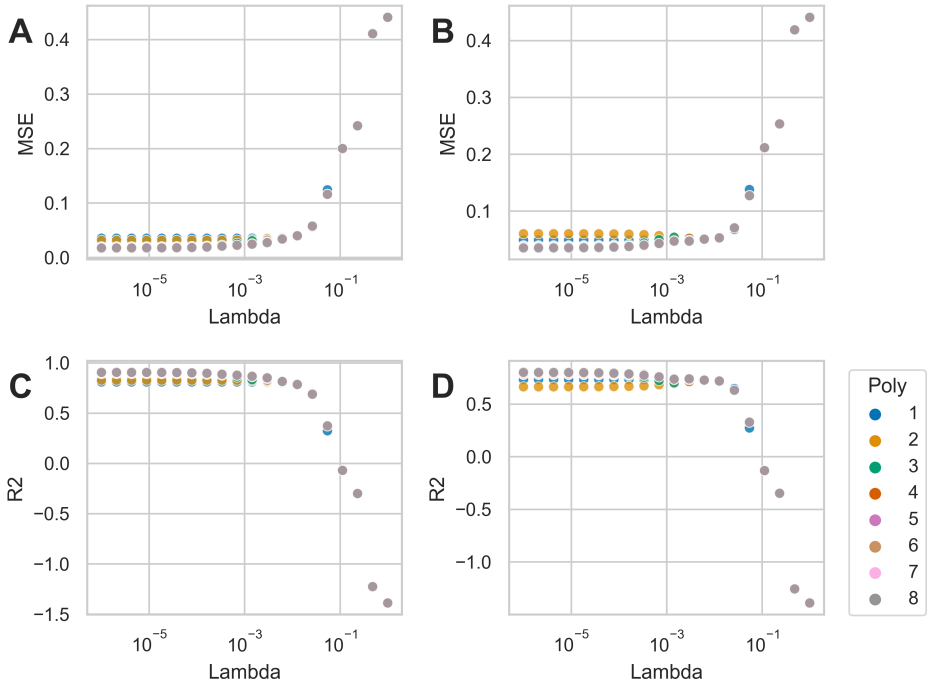

Figure 5.8: LASSO-regression with five fold cross validation

5.6 | Geographical data

The OLS models are seen to increasingly fit the training partitions of the terrain data with increasing p, as seen in figure 5.9. While regional differences are seen to affect the training score more than method of splitting the data, the modelling performing the worst on test seem to be split into test by chunks. For the random pixel splits, a calculation with order of polynomial up to $p = 71$ was performed, showing the MSE and p^2 for training increase while the test-values flattens out (figure 5.13).

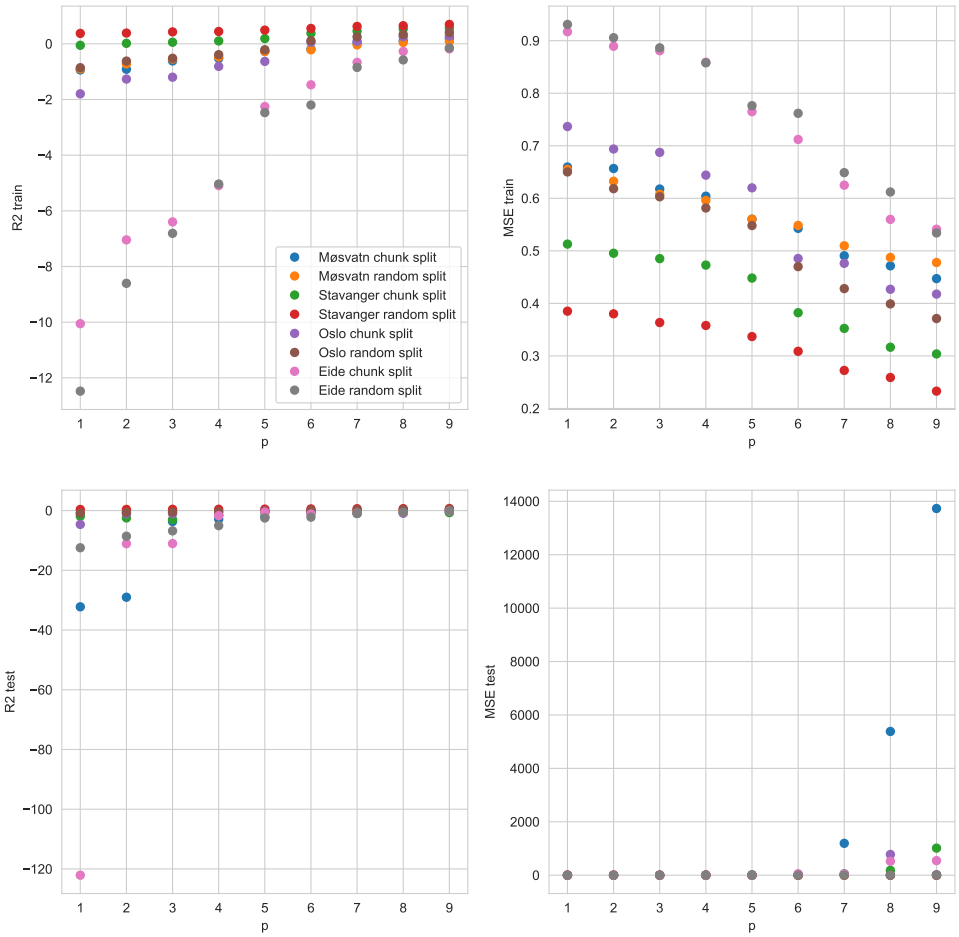

Figure 5.9: OLS results for the four topographic terrain data sets, for the two methods of dividing into train and test.

A bias-variance-plot of the terrain data for Møsvatten is shown in figure 5.11
The hyper-parameter grid-search for the ridge-penalized model is shown for the Oslo terrain data in figure 5.12, with the remaining regions in Appendix A (Figure A.5, A.4, A.6).
The optimal combination of α and p is largely dependent on the method of splitting data into training and test, more than choice of terrain data, as seen in table 5.1. There is no visible difference between training and average validation MSE for the randomly split data, and the model performances are seen to increase with higher p and lower regularization strength α. However, when splitting the validation sets into chunks the average validation MSE is the highest for lower values of p. The optimal α for all regions are in the order of 10^4, except for the Møsvatn region (having an optimal polynomial degree of one).
The time per polynomial for an OLS-calculation quickly increases, and the time-dependence can be found in figure A.7 in the appendix. The predicted image for five, 50 and 100 polynomials can be found in figure

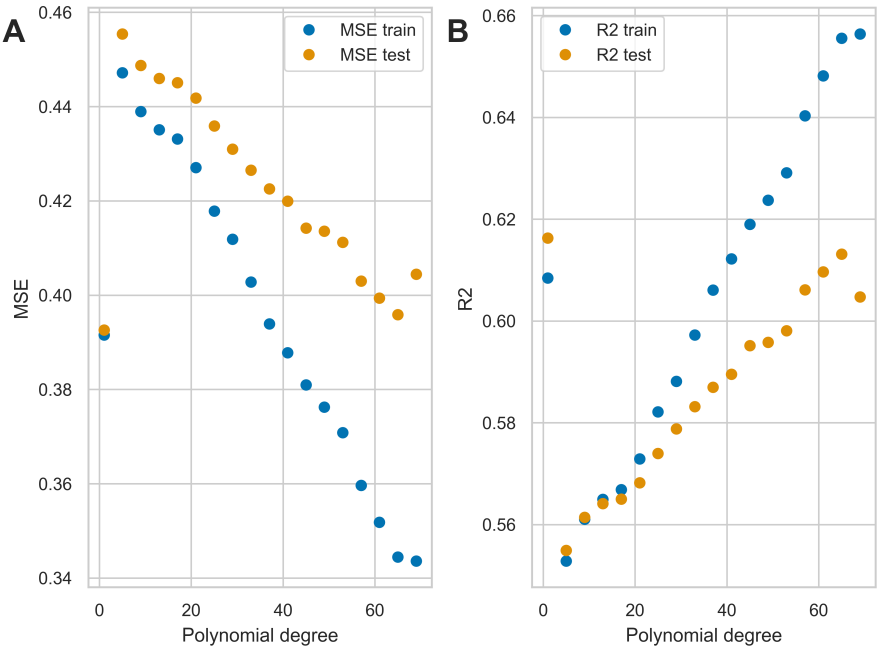

Figure 5.10: Training and test-values for the Møsvatten data set using the OLS-regression method. The data set was downsampled by a factor of 12 to speed up computation.

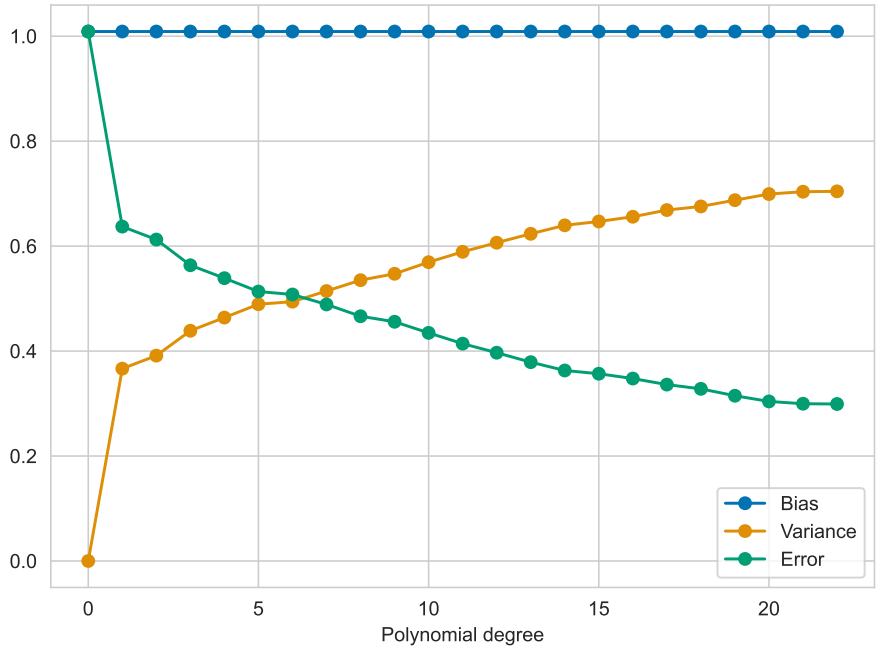

Figure 5.11: Bias-variance plot of the Møsvatten data set. Image data was downsampled by a factor two to speed up computation.

	Best model	Min valid MSE	Test R2	Best model	Min valid MSE	Test R2
	Chunk split			Random split		
Møsvatn	$p = 1$, α=1.0	0.698	-0.684	$p = 9$, $\alpha = 1.0$	0.476	0.523
Stavanger	$p = 4$, $\alpha = 3.511 \times 10^4$	0.512	-1.547	$p = 9$, $\alpha = 1.0$	0.277	0.767
Eide	$p = 2$, $\alpha = 1.233 \times 10^4$	0.952	-0.225	$p = 9$, $\alpha = 1.0$	0.577	0.465
Oslo	$p = 4$, $\alpha = 3.511 \times 10^4$	0.739	-3.72	$p = 9$, $\alpha = 1.0$	0.452	0.630

Table 5.1: Optimal hyperparameters for each terrain and split mode, found by grid-based minimization of average MSE across a 3-fold cross-validation for ridge-regularized linear models. The model with optimal hyperparameters are evaluated on the test data.

5.13.

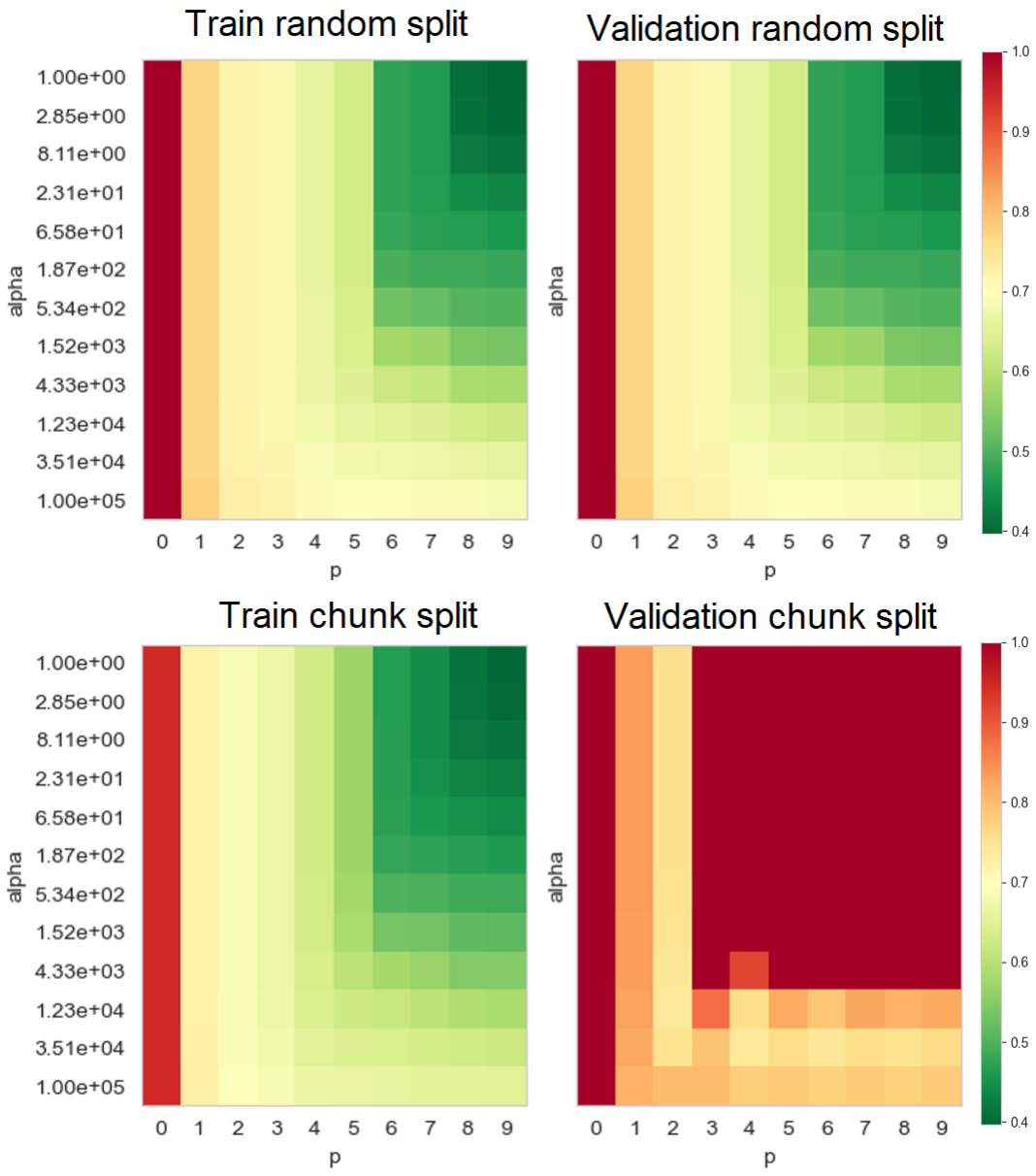

Figure 5.12: MSE for each considered combination of α and p in a ridge-regularized linear model for both training and averaged 3-fold cross-validation, for the Oslo region SRTM data.

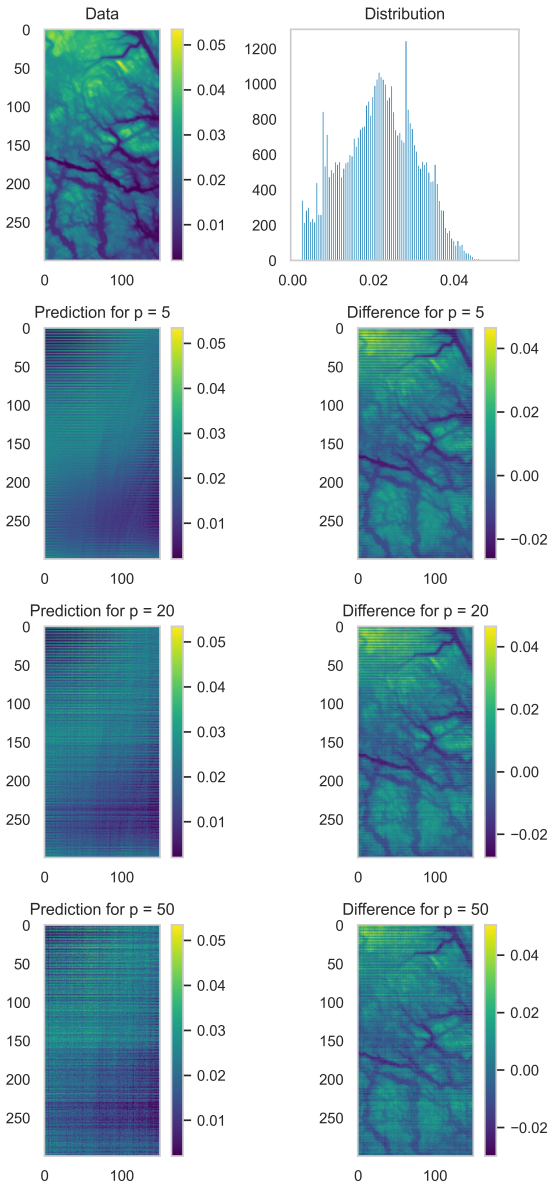

Figure 5.13: The OLS model prediction of terrain data for different order of polynomials. The whole image domain is used to for prediction where the weights are from the training data picked with random pixel sampling. Image data was downsampled to 300 by 150 from 3601 by 1801 matrix size prior to analysis.

6 | Discussion

6.1 | Preprocessing

Procedures in preprocessing, such as normalization, should be performed using transformations determined and decided using training data only. Thus, any influence on the final model by the testing data is avoided, leaving it completely ”unseen” and avoiding *data leakage*.
The scale of the data to analyse, both input features and outcome, may affect how the estimated regression coefficients are estimated in relation to each other. To avoid this issue the data is usually normalized. This process harmonizes the models for comparison between inputs of varying scales, such as the four topographic maps, increasing interpretability. Additionally, the normalization decreases the risk of under- or overflow issues when estimating model parameters, e.g. when having input on a very small scale. Normalization may also increase model performance.
For the Franke function, we opted to not scale the data, given the similar sizes of the input data (being coordinates in a well defined range of 0 to 1). The analysis was performed with and without a normalization (using the standard normalizer in the `sklean` library) of the function-values, and did it did have a large impact on the results.

6.2 | Boostraping and bias-variance

In the case of the Franke function, we observed an initial decrease in error followed by an increase. This trend was consistent for both the Franke function data and the terrain data. The variance demonstrated a steady increase, while the bias, although diminishing with increased model complexity, did so at a relatively slow rate. Several factors may contribute to these observations.
One potential explanation is that the dataset size is relatively large, and bootstrapping is typically more effective with sparse data, making it difficult to discern a significant difference between bootstrap samples. We conducted experiments where we varied both the number of bootstraps and the sample size, ranging from a few percent of the training set to as much as 70 percent, and did not observe a substantial drop in bias.

6.3 | Penalization in the Franke Function

In the case of both Ridge and LASSO regression, it became evident that the utilization of regularization did not offer a clear advantage. This observation was consistent across different polynomial orders, as the mean squared error (MSE) continued to converge even for smaller values of λ. Furthermore, as the λ-values increased in magnitude, the MSE exhibited a rapid deterioration.

6.4 | Choice of number of polynomial

The Franke-function can be properly modelled with a polynomial of degree six, after that, the model is overfitted. This is indicated both by the minimum in the bias-variance and how the MSE and R^2 plateaus at around $p = 6$.
Modeling terrain data, especially when dealing with a high degree of complexity, using a fixed and limited set of polynomials has proven to be a daunting challenge, as indicated by our analysis. Even with a relatively modest number of polynomials, the computational demands increase significantly.
With a moderately powerful laptop, it takes approximately three seconds to calculate a fit with 50 polynomials, while a 100-polynomial function requires around 22 seconds. It’s worth noting that this performance is achieved using non-optimized computer code and hardware. More critically, our analysis reveals that the test mean squared error (MSE) converges as the number of polynomials increases, suggesting that achieving a satisfactory model using polynomials alone may be unattainable, as the returns diminish with additional complexity.
An alternative approach could involve exploring a different set of functions as a basis for the modeling. However, the challenge lies in identifying the most suitable candidates for this task. Additionally, another option is to more aggressively down-sample and smooth the terrain data, retaining only the prominent variations, such as larger valleys and plateaus. This approach, while reducing computational demands, comes at the cost of losing fine-grained details, necessitating careful consideration of its applicability.

6.5 | Polynomial generalization to topographic data

Whether the real-world topographic terrains were successfully modelled using a two-dimensional polynomial, were more dependent on method for splitting into train and test (chunk versus random pixels) than choice of terrain region - as seen by the systematic differences in validation MSE and test R^2 in table 5.1. If the models were truly finding some underlying pattern to generalize using only coordinates as input, the method of splitting into train and test data should be invariant. Qualitatively, evaluating the overfit predictions seen in figure 5.13, the resulting images does not look very much like real-world terrain data. The terrain region also seem to have some impact on the general performance across the models, varying somewhat in choice for hyperparameters and test scores. The polynomial model seem to better fit the Stavanger region, using the random test split. This may be due to the large homogenous region in the lower part of the image, which is hard to "expect" as part when largely being a part of the test set when creating a chunk-split test set.
The large discrepancy between test scores for the two split methods, illustrates how choices early in the pipeline may affect the final evaluation of model performance. Further work to investigate generalization of polynomial models to highly complex multi-scale terrain data, might include some continuum of methods for splitting data into train and test - e.g. by considering neighbouring pixels into multiple chunks as intermediate steps.

7 | Conclusion

From this work we can make the following conclusions:

1. Using polynomials to model terrain data, even of a high degree (i.e. $p = 100$), is unable to provide a satisfying model using OLS with or without penalization techniques such as Ridge. Despite the MSE valu
2. The Franke-function can be adequately modelled with a polynomial of degree six, and there seem to be little indication that a ridge- or LASSO-approach with a penalization have a clear benefit.

References

[1] F.J. Shaikh and D.S. Rao. "Prediction of Cancer Disease using Machine learning Approach". In: *Materials Today: Proceedings* 50 (2022). International Virtual Conference on Advanced Nanomaterials and Applications (VCAN), pp. 40–47. ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2021.03.625. URL: https://www.sciencedirect.com/science/article/pii/S2214785321027206.

[2] Daniel Broby. "The use of predictive analytics in finance". In: *The Journal of Finance and Data Science* 8 (2022), pp. 145–161. ISSN: 2405-9188. DOI: https://doi.org/10.1016/j.jfds.2022.05.003. URL: https://www.sciencedirect.com/science/article/pii/S2405918822000071.

[3] Yi Zhou. "Regression analysis and driving force model building of CO2 emissions in China". In: *Scientific Reports* (2021). DOI: 10.1038/s41598-021-86183-5. URL: https://doi.org/10.1038/s41598-021-86183-5.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[5] Morten Hjorth-Jensen. "Lecture notes". In: (). URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer New York, NY, 2006.

[7] Bradley Efron. "Bootstrap Methods: Another Look at the Jackknife". In: *Breakthroughs in Statistics: Methodology and Distribution*. Ed. by Samuel Kotz and Norman L. Johnson. New York, NY: Springer New York, 1992, pp. 569–593. ISBN: 978-1-4612-4380-9. DOI: 10.1007/978-1-4612-4380-9_41. URL: https://doi.org/10.1007/978-1-4612-4380-9_41.

[8] Farr et al. "The Shuttle Radar Topography Mission". In: *Reviews of Geophysics* (2007). DOI: 10.1029/2005RG000183.

A | Appendix

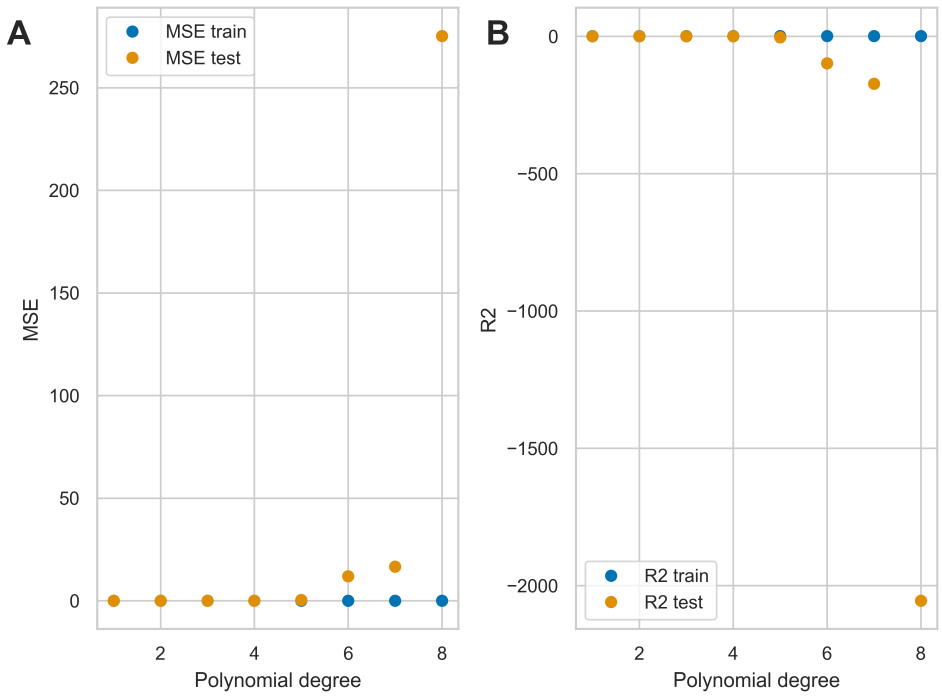

Figure A.1: Train and test-MSE and R^2 for ten fold cross validation using OLS

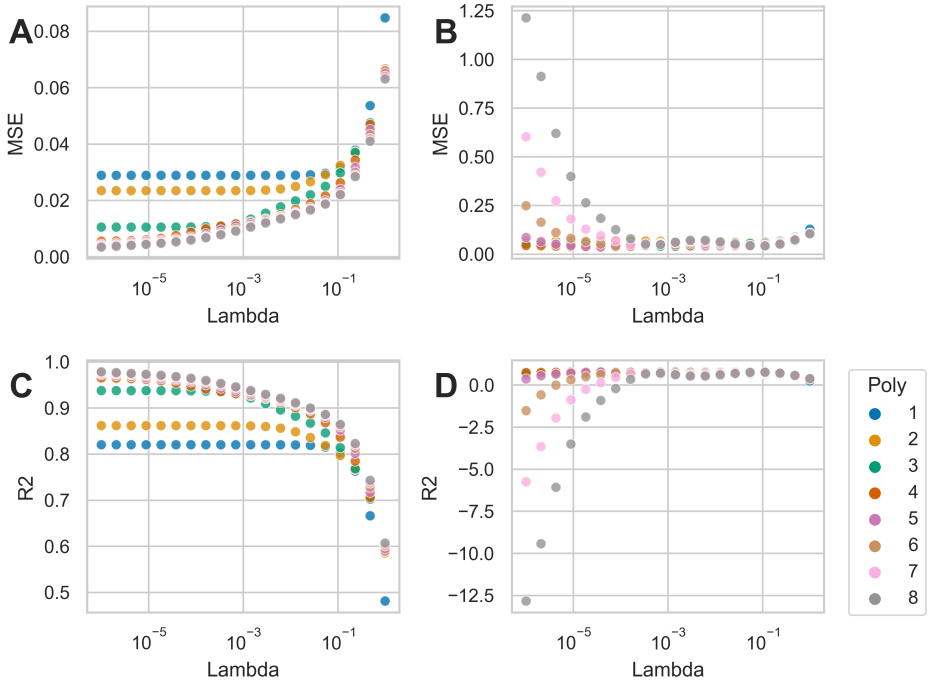

Figure A.2: Ridge regression and ten-fold cross-validation on the Franke-function

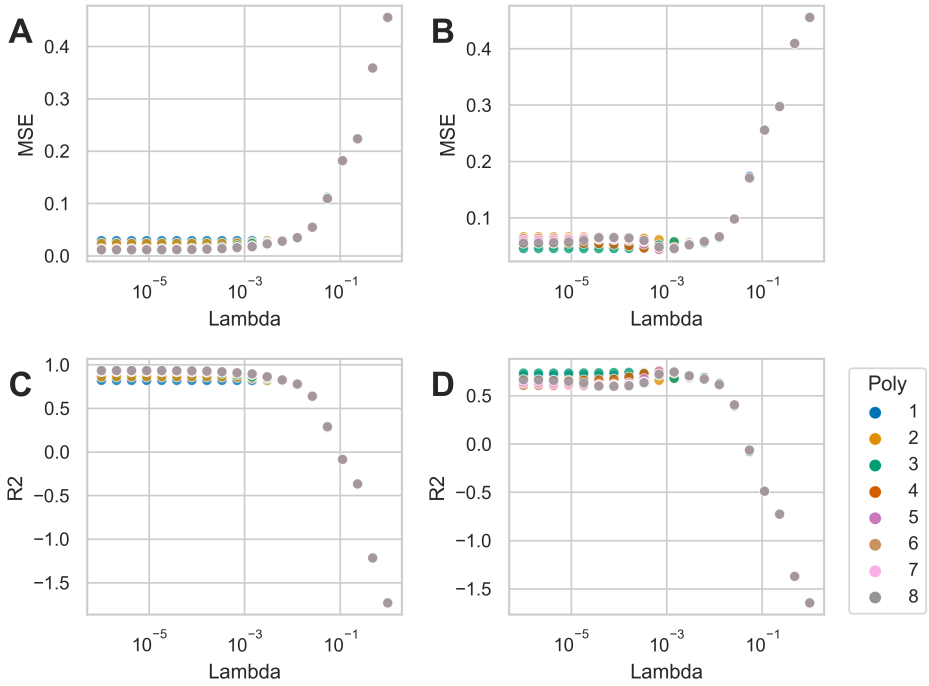

Figure A.3: Ten fold cross validation using LASSO-regression on the Frank-function.

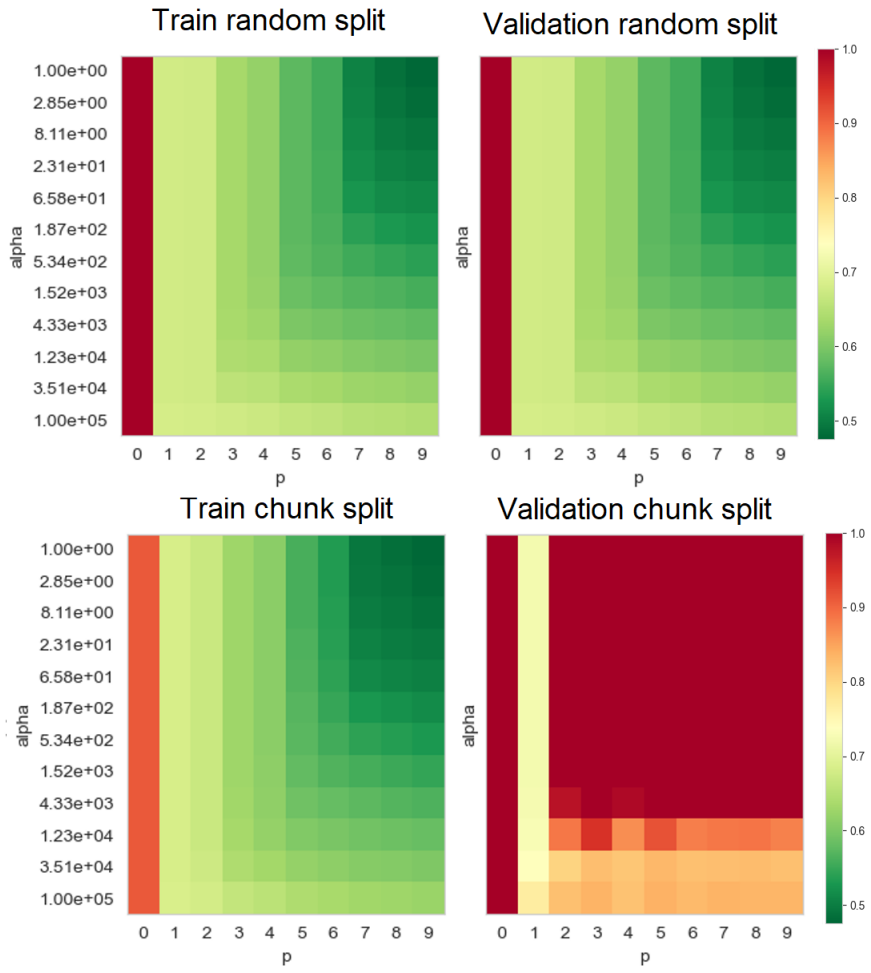

Figure A.4: MSE for each considered combination of α and p in a ridge-regularized linear model for both training and averaged 3-fold cross-validation, for the Møsvatn region SRTM data.

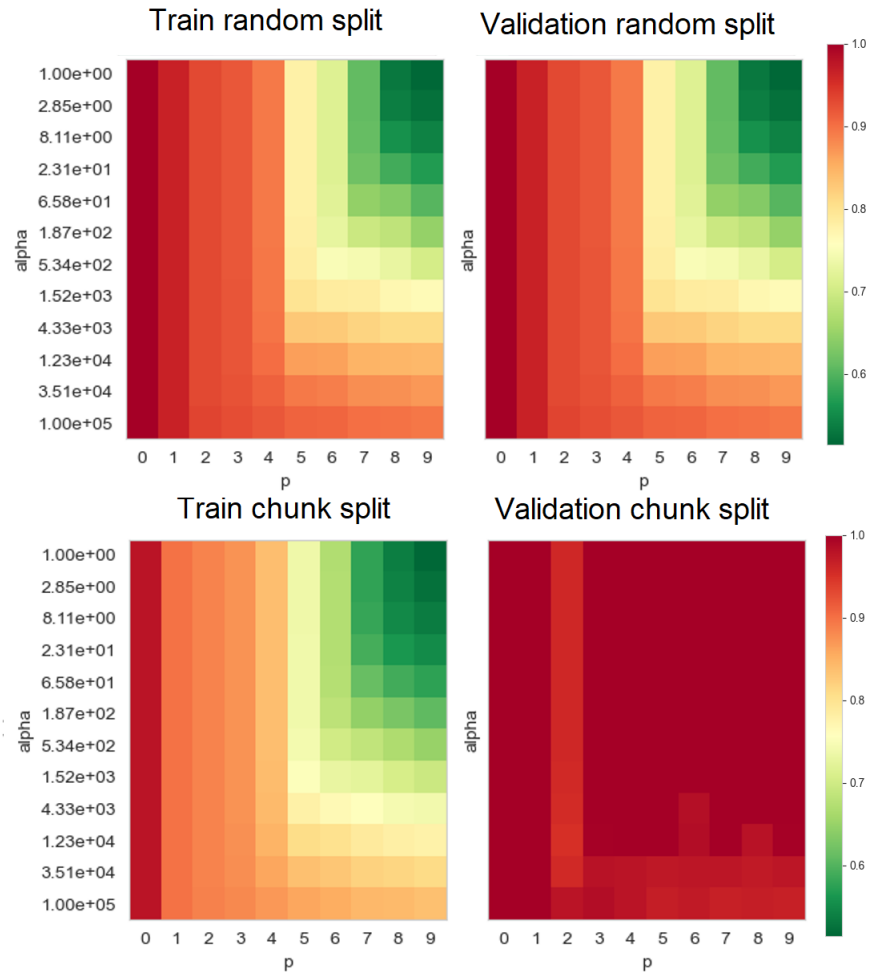

Figure A.5: MSE for each considered combination of α and p in a ridge-regularized linear model for both training and averaged 3-fold cross-validation, for the Eide region SRTM data.

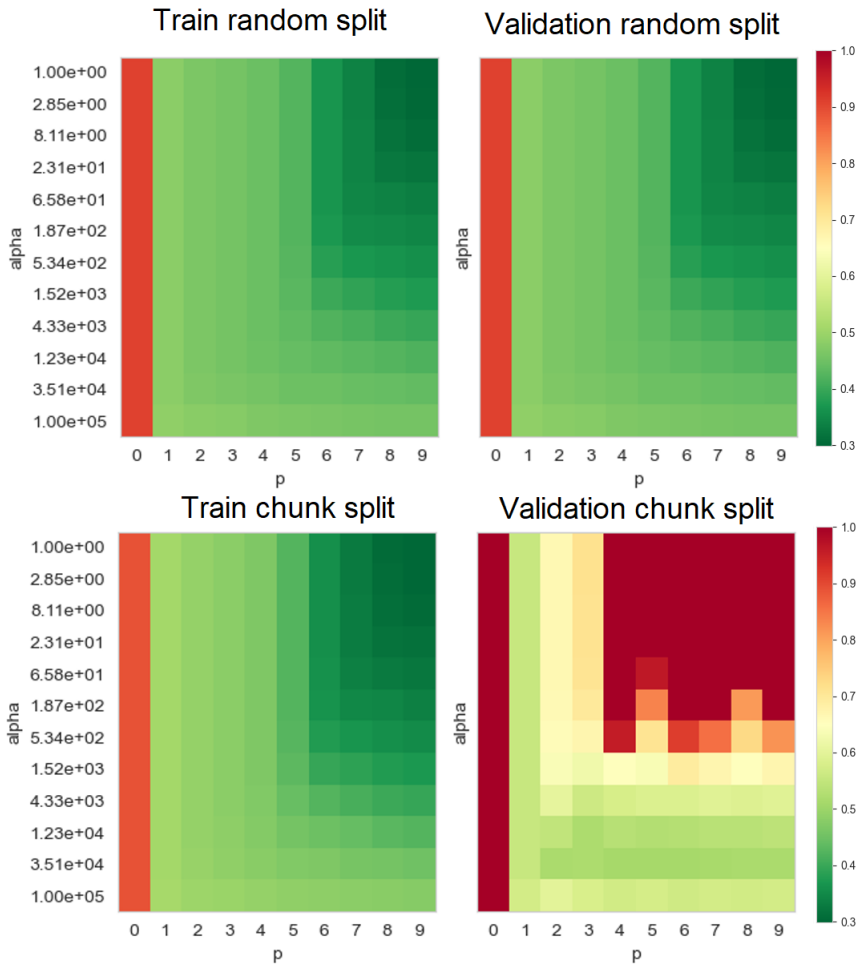

Figure A.6: MSE for each considered combination of α and p in a ridge-regularized linear model for both training and averaged 3-fold cross-validation, for the Stavanger region SRTM data.

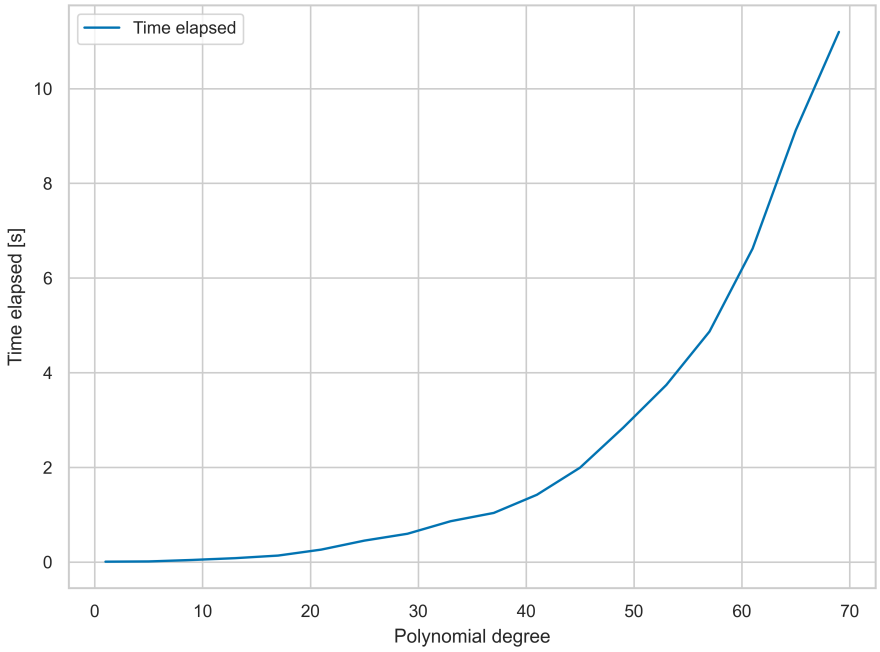

Figure A.7: The time elapsed for as a function of polynomial for a down-sampled terrain-image of size 300 by 150. Performed on a Lenovo Thinkpad P52 with 32 GBq of RAM and an i7-processor