University of Oslo

FYS-STK3155

Project 1:
In-depth exploration of Linear Regression methods

Laura Ruffoni
Leonardo Proetto

September 2024

Contents

Abstract

The purpose of this project is to present an in-depth exploration of Linear Regression methods, with a focus on Ordinary Least Square, Ridge Regression and Lasso. By first applying these methods to a Franke's function, we investigate polynomial fittings of varying degrees, highlighting the characteristics of each approach. We employ resampling techniques such as Bootstrap and Cross-validation that allow to enlarge the training data. These ensure the model robustness and will be employed to analyze the bias-variance trade-off and to fine-tune the regularization parameter λ in Ridge and Lasso. Lastly the procedure is expanded to real-world terrain data to determine the most effective model. Synthetic data analysis identified Ridge and Lasso as optimal regularization techniques, significantly enhancing model simplicity. Cross-validation proved effective in training more robust models. In contrast, real-data analysis showed less accurate results. While OLS demonstrated reliability, it carried a risk of overfitting. Overall regularization and cross-validation performed less effectively on real-data. Lasso performed particularly poor results.

1 Introduction

The analysis of big quantities of data represents one of the biggest challenges and opportunities of the modern era. Machine Learning methods allow us to transform big quantities of data into valuable information that can be used to make predictions, support decision-making processes or identify patterns in the data [7].

Linear regression stands out as a powerful statistical tool to address these challenges as it provides a framework for modeling the relationship between input variables and predicted outcomes. This project focuses on exploring different approaches to train a regression model. We will exploit diverse regression techniques to estimate and optimize a model, in particular Ordinary Least Square (OLS) but also Ridge and Lasso regression that we are confident will help us in reducing the model complexity [9].

We will discuss how different Linear Regression techniques perform under varying degrees of polynomial fitting. Furthermore the impact of regularization on model complexity and accuracy is examined. The effect of resampling techniques on enhancing model robustness will be closely analyzed as well as their effectiveness in tuning hyperparameters.

The analysis will begin by applying the techniques to Franke's function and will continue on real topographical data. The goal is understanding the behavior of regression models in real-world scenarios. We will specifically focus on understanding the variance and bias associated with the models. Performance indexes such as MSE and R2 will be useful to compare the efficiency of different models. This work aims to provide a consistent workflow for training Linear Regression models. It emphasizes the importance of understanding underlying principles to select appropriate methods for various data contexts.

2 Methods

2.1 Franke Function and Synthetic Data Processing

To explore the different machine learning algorithms, two sets of 500 data points are created by randomly selecting them from a standard normal distribution with every point $x, y \in [0, 1]$. We will refer to these sets as our inputs, x and y. These will be fed to the Franke Function $f(x, y)$, which is used to generate the outcome variable z.

The number of data points has been chosen to ensure an efficient process while supporting a thorough analysis. This number has been validated through practical tests using different sample

sizes. Tests for each function used in the project are attached as supplemental material and are further discussed in the results section.

The Franke function is a two-dimensional function that consists of four exponential terms. It is particularly useful for testing and comparing interpolation techniques. The output of the Franke function, representing our target, will be addressed as z. To the z variable, we add a normally distributed noise term.

```
# Creation of data
np.random.seed(67)
x = np.random.rand(500,1)
y = np.random.rand(500,1)

x, y = np.meshgrid(x, y)

# Define Franke's function
def FrankeFunction(x, y):
    term1 = 0.75 * np.exp(-(0.25 * (9 * x - 2) ** 2) - 0.25 * ((9 * y - 2) ** 2))
    term2 = 0.75 * np.exp(-((9 * x + 1) ** 2) / 49.0 - 0.1 * (9 * y + 1))
    term3 = 0.5 * np.exp(-(9 * x - 7) ** 2 / 4.0 - 0.25 * ((9 * y - 3) ** 2))
    term4 = -0.2 * np.exp(-(9 * x - 4) ** 2 - (9 * y - 7) ** 2)
    return term1 + term2 + term3 + term4

z = FrankeFunction(x, y) + np.random.normal(0, 1, x.shape)

# Flatten the x, y, z arrays for regression
X = np.column_stack((x.ravel(), y.ravel()))
z = z.ravel()

# Splitting into train and test
X_train, X_test, z_train, z_test = train_test_split(X, z, test_size=0.3,
    random_state=32)

# Splitting into train and validation
X_train, X_val, z_train, z_val = train_test_split(X_train, z_train, test_size=0.1,
     random_state=32)
```

Data is split into training data (70%), used to train the models, and test data (30%). From the training data, a validation set (10%) is also extracted, which is used to compare the different model efficiencies on unseen data. The test data is employed for the final assessment of the chosen model's performance. Throughout the analysis, all machine learning algorithms are tested on polynomial models with degrees ranging from 1 to 15. This allows for observing how the complexity of the model influences its performance.

A standard practice in machine learning algorithms deals with the scaling of the data. This is usually done because sparse features or different scales can lead to slower and biased training. In our case, scaling will help reduce the distance between data values to enhance the correctness of the model. Since the initial data is synthetic, within a small range of values, and all features are on the same scale without differing units, a centering of the data is performed [6].

The preprocessing of data is conducted with the help of `train_test_split()`, `PolynomialFeatures()`, and `StandardScaler()`, which are Scikit-learn built-in functions [2]. Scaling of the data is always performed after the transformation of the design matrix. This is important because using high-degree polynomials on the original data can result in large output values, which can disproportionately impact the model's training compared to smaller values.

2.2 Linear Regression

Regression aims at modeling the relationship between given input variables X and a target y: $y = f(X) + \epsilon$. The task is to build a model $\tilde{y} = \hat{f}(x)$ that approximates as closely as possible the function $f(x)$ with the awareness of the presence of unpredictable and random variability which will prevent it from estimating exactly y. The error term ϵ accounts for unexplained variability in the outcome variable. It can be described as the difference between the real values and their predicted ones $\epsilon(i) = Y(i) - \hat{Y}(i)$. We assume it to be normally distributed having $\mu = 0$ and $\text{Var}(\epsilon) = \sigma^2$, which means that on average, the error does not bias the predictions in either direction. The error is also assumed to be independent of the input variables X, implying that whatever causes this error is unrelated to the features we use in the model [1][5][8].

When no prior assumption on $f(x)$ is present we can assume it is linear. Linear regression offers significant advantages due to its simplicity and interpretability. The model's linear equation provides a clear understanding of how each independent variable influences the dependent variable, making it easy to make inference:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \tag{1}$$

If X represents the design matrix and β is the vector of parameters, the previous equation can be expressed as:

$$y = X\beta + \epsilon \tag{2}$$

For a given data point i, the predicted outcome $\hat{y}_i$ can be written as:

$$\hat{y}_i = X_i\beta$$

and the true outcome as:

$$y_i = X_i\beta + \epsilon_i$$

To compute the expected value of the true outcome y_i, we start with:

$$\mathbb{E}[y_i] = \mathbb{E}[X_i\beta + \epsilon_i] = \mathbb{E}[X_i\beta] + \mathbb{E}[\epsilon_i]$$

Since $X_i\beta$ is deterministic, we have:

$$\mathbb{E}[X_i\beta] = X_i\beta$$

Also, for the error term ϵ_i, we know that:

$$\mathbb{E}[\epsilon_i] = 0$$

Thus, the expected value of the true outcome simplifies to:

$$\mathbb{E}[y_i] = X_i\beta$$

Next, we calculate the variance of the true outcome y_i. By definition:

$$\text{Var}(y_i) = \mathbb{E}\left[(y_i - \mathbb{E}[y_i])^2\right]$$

Substituting the expression for y_i:

$$y_i = X_i\beta + \epsilon_i$$

yields:

$$\text{Var}(y_i) = \mathbb{E}\left[(X_i\beta + \epsilon_i - X_i\beta)^2\right]$$

Simplifying this expression:

$$\text{Var}(y_i) = \mathbb{E}\left[\epsilon_i^2\right] = \text{Var}(\epsilon_i) = \sigma^2$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. Therefore, the distribution of y_i is:

$$y_i \sim \mathcal{N}(X_i\beta, \sigma^2)$$

For the entire vector y, this result generalizes to:

$$y \sim \mathcal{N}(X\beta, \sigma^2 I) \tag{3}$$

This formulation captures both the mean and variance structure of the model, where the mean is determined by the linear relationship between the predictors and the outcome and the variance is constant across all observations.

Once a model family is chosen, it is essential to identify the optimal parameters which represent the impact of each feature on the outcome. To achieve this, we employ a cost function, which quantifies the error between the predicted outcomes and the actual data. The goal is to find the values of the coefficients that minimize this cost function. Linear regression offers various approaches to employ this estimation: including Ordinary Least Squares (OLS), Ridge regression, and Lasso regression. Throughout this project, each of these methods is explored, highlighting their advantages and disadvantages

To understand the performance of the model various indexes like Mean Squared Error and R Squared are used.

The Mean Squared Error (MSE) measures the average squared difference between true and predicted values.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4}$$

Ideally an MSE of zero is desired [11].

R^2 gives a measure of how much of the total variance of the target variable is explained by the model. If it is equal to 1.0 the model explains all the variance and it is the best case possible [11].

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5}$$

The MSE on training data is not informative for model selection, it will always favor the more complex model. Throughout the work validation MSE is used for decision-making to understand the best performing model. The final accuracy of the chosen model is then assessed by the test MSE.

2.3 Variance-Bias Trade-off

Managing the relationship between validation error and training error is crucial to prevent overfitting or underfitting. Overfitting occurs when a model adapts too well to the training data. This will reduce its performance on unseen data. In contrast, underfitting arises when the model is too simple, yielding high errors both in the test and training set.

In this context the balance between the bias and variance of the model is critical. Bias and variance measure two different sources of error in an estimator [3].

The bias reflects the difference between the true and the estimated function. The variance quantifies the sensitivity of the model to changes in the training data. The flexibility of the model can lead to a low bias but can increase the model variance, potentially resulting in overfitting.

Simpler model instead can keep the variance low but exhibit higher bias, with a greater risk of underfitting. Balancing these quantities is crucial to minimize overall error, as it directly influences the ability of the model to generalize well on unseen data [4].

The MSE provides a comprehensive estimate of all potential sources of error between the true function and the estimated function.

$$\mathbb{E}[(y - \tilde{y})^2] = \text{Bias}[\tilde{y}]^2 + \text{Var}[\tilde{y}] + \sigma^2 \tag{6}$$

This equation illustrates that the total expected error can be attributed to the bias, variance, and irreducible error:

$$C(X, \hat{\beta}) = \mathbb{E}[(y - \hat{y})^2] = \text{Bias}[\hat{y}]^2 + \text{Var}[\hat{y}] + \sigma^2$$

Starting from:

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

where $y = f(x) + \epsilon$, ϵ is the irreducible noise (with $\epsilon \sim \mathcal{N}(0, \sigma^2)$), and $\hat{y} = \hat{f}(x)$. Expanding the square:

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

Since $\mathbb{E}[\epsilon] = 0$ and ϵ is independent of $\hat{f}(x)$, the cross-term $\mathbb{E}[(f(x) - \hat{f}(x)) \cdot \epsilon]$ is zero. Thus:

$$\begin{aligned}\mathbb{E}[(y - \hat{y})^2] &= \mathbb{E}[(f(x) - \hat{f}(x))^2] + \mathbb{E}[\epsilon^2] \\ &= \mathbb{E}[(f(x) - \hat{f}(x))^2] + \sigma^2\end{aligned}$$

Breaking down the first term:

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

Expanding this expression:

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

Since $f(x)$ is deterministic, $\mathbb{E}[f(x)] = f(x)$, and the term $2\mathbb{E}[(f(x) - \mathbb{E}[\hat{f}(x)])(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))]$ becomes zero. Thus:

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

Recognizing that:

$$\begin{aligned}\text{Bias}(\hat{f}(x))^2 &= (f(x) - \mathbb{E}[\hat{f}(x)])^2 \\ \text{Var}(\hat{f}(x)) &= \mathbb{E}[(\mathbb{E}[\hat{f}(x)] - \hat{f}(x))^2],\end{aligned}$$

we get:

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

Finally, combining the terms:

$$C(X, \hat{\beta}) = \text{Bias}^2[\hat{y}] + \text{Var}[\hat{y}] + \sigma^2 \tag{7}$$

Achieving the lowest validation error typically indicates the optimal balance between bias and variance, where the model effectively captures the underlying data patterns.

2.4 Ordinary Least Squares

The Ordinary Least Squares (OLS) method is commonly used in machine learning to estimate the parameters. To find the parameters that minimize the sum of squared residuals the derivative of the cost function is taken with respect to beta and set to zero. The formula for the optimal parameters is:

$$\hat{\beta} = (X^T X)^{-1} X^T y \tag{8}$$

This formula involves the inversion of the matrix $(X^T X)^{-1}$ which can lead to problems if the $X^T X$ matrix is singular or near singular. In this case, the Singular Value Decomposition (SVD) is employed. It is one of the most powerful linear algebra algorithms and provides a numerically stable matrix decomposition [9].

$$X = U \Sigma V^T \tag{9}$$

Stated that the errors ϵ are independently and identically distributed with zero mean and constant variance σ^2, using OLS the expected value and variance of the estimated coefficients β can be computed. From the OLS formula:

$$\hat{\beta} = (X^T X)^{-1} X^T y \quad \text{where} \quad y = X\beta + \epsilon$$

Substituting y:

$$\hat{\beta} = (X^T X)^{-1} X^T (X\beta + \epsilon)$$

$$\hat{\beta} = (X^T X)^{-1} X^T X\beta + (X^T X)^{-1} X^T \epsilon$$

Simplifying the first term:

$$\hat{\beta} = \beta + (X^T X)^{-1} X^T \epsilon$$

Now, take the expectation:

$$\mathbb{E}[\hat{\beta}] = \mathbb{E}\left[\beta + (X^T X)^{-1} X^T \epsilon\right]$$

Since $\mathbb{E}[\epsilon] = 0$, we get:

$$\mathbb{E}[\hat{\beta}] = \beta + (X^T X)^{-1} X^T \mathbb{E}[\epsilon] = \beta$$

Under these assumptions, the OLS estimator $\hat{\beta}$ is an unbiased estimator of the true coefficients β . This means that the expected value of $\hat{\beta}$ is equal to the true β. Furthermore the variance is computed knowing that:

$$\hat{\beta} = \beta + (X^T X)^{-1} X^T \epsilon$$

$$\text{Var}(\hat{\beta}) = \text{Var}\left(\beta + (X^T X)^{-1} X^T \epsilon\right)$$

The variance of β is zero because it is a constant, so we are left with:

$$\text{Var}(\hat{\beta}) = \text{Var}\left((X^T X)^{-1} X^T \epsilon\right)$$

Using the variance property:

$$\text{Var}(A\epsilon) = A\text{Var}(\epsilon)A^T$$

where $A = (X^T X)^{-1} X^T$

$$\text{Var}(\hat{\beta}) = A\text{Var}(\epsilon)A^T$$

Since $\text{Var}(\epsilon) = \sigma^2 I$:

$$\text{Var}(\hat{\beta}) = (X^T X)^{-1} X^T \sigma^2 I (X^T X)^{-1} X^T$$

Simplifying:

$$\text{Var}(\hat{\beta}) = \sigma^2 (X^T X)^{-1}$$

The variance of the OLS estimator depends on both the error variance σ^2 and the design matrix. If the columns of X are highly correlated, $X^T X$ becomes nearly singular and the variance of $\hat{\beta}$ increases, leading to less reliable estimates. Assuming that β follows a normal distribution, we can construct the confidence interval as:

$$\mu_\beta \pm z \cdot \frac{\sigma_\beta}{\sqrt{n}} \tag{10}$$

where z for a 95% confidence level. In explicit form, the confidence interval is:

$$\mu_\beta \pm 1.96 \cdot \frac{\sigma_\beta}{\sqrt{n}}$$

To perform the OLS method over different polynomial degrees and discuss the results, two functions have been implemented in this work. In the `ols()` function, for each degree, the design matrix is shaped with the addition of all polynomial combinations. All the data, except for the intercept column, is then centered.

```
scaler = StandardScaler(with_std=False)
X_train_poly[:, 1:] = scaler.fit_transform(X_train_poly[:, 1:])
X_val_poly[:, 1:] = scaler.transform(X_val_poly[:, 1:])
```

The intercept column is created by the `PolynomialFeatures()` function and consists of all ones. Centering it would generate a zero-only column, leading to a non-invertible matrix. Afterwards, the SVD function is employed to obtain the parameters of each model. The decision to use SVD for the inversion of the design matrix derives from its reliability in the presence of non-invertible matrices. For each polynomial, the validation MSE and train MSE are calculated to show their behavior with respect to different degrees. The same considerations are made for the metric R^2. We expect the train MSE, as discussed before, to always decrease as the complexity of the model increases. This reasoning holds for training R^2, which should constantly increase.

2.5 Regularization techniques

So far, the only approach used to explore the dependency between X and y, has been trying different polynomial fits. This helped understanding how more complex models can better capture the relationship between input and output. Ridge and Lasso introduce a different perspective on estimating the true function by regularization. These processes try to make the model simpler, not by explicitly selecting features, but working directly on the loss function [7]. The regularization process introduces a penalty term into the loss function. This term accounts for the complexity of the model. It imposes a constraint on the magnitude of the model coefficients. The cost function for Ridge regression becomes:

$$L(\beta) = \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{11}$$

Differently from Ridge, Lasso penalizes the sum of the absolute values of the coefficients. As a result it excludes less important variables by shrinking their coefficient to 0.

$$L(\beta) = \sum_{i=1}^{n} \left(y_i - \mathbf{x}_i^T \beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{12}$$

The hyperparameter λ controls the strength of shrinkage. λ imposes a balance between the model performance and its complexity. Increasing values of λ apply a stronger shrinkage of the model parameters. When set to 0, no regularization is applied, and the model becomes equivalent to OLS regression. Throughout the analysis, a consistent list of lambdas has been employed in regularization algorithms: 0.0001, 0.001, 0.01, 0.1, 1, 10.

During the implementation of Ridge and Lasso regression, the optimization set adopted does not include the intercept. This is preferable in order to focus the regularization on the feature coefficients [9]. Trials that included the intercept are included in the code material of this project. The intercept column added by `PolynomialFeature()` is removed from the design matrix, and the training of the model is performed with the parameter `fit_intercept` set to `False`. The intercept is computed and later added to the predicted values.

2.5.1 Ridge

In the implementation of the Ridge algorithm, two functions are used. The `Ridge_noint()` function generates and centers the polynomial combinations within the design matrix for each degree. To find the optimal parameters, the `lambda_ridge_noint()` function is employed. This function tests different values of λ in manually implemented loss functions. By comparing the MSE, the best lambda is then chosen for each degree.

Finally, `Ridge_noint()` plots the training and validation MSE as a function of model complexity to help us decide which one fits best unseen data. The same analysis is also performed using R Squared as performance metric.

2.5.2 Lasso

Lasso does not have an analytical solution for the optimization of its cost function. As a consequence it relies on iterative optimization techniques for the minimization process. These methods are not described in this project. To implement Lasso regression the same workflow used for Ridge regression is engaged. Built-in Scikit-learn functions have been leveraged to calculate the parameters for each polynomial degree [2].

2.6 Resampling Techniques

Resampling techniques are statistical methods that help enhance the training procedure, especially when data is limited. The scope is to allow multiple evaluations on different subsets of data, which helps in understanding the model's robustness. Two of the most used methods are Bootstrap and Cross validation, whose performance has been tested in this work.

2.6.1 Bootstrap

In this project, the bootstrap approach is employed to evaluate OLS model's performance by repeatedly calculating its error across various fits and predictions. Specifically, for different degrees, multiple bootstrap samples (500) are generated from the training data. The model is then trained on each sample and the assessment of its predictions on the validation set is performed using MSE. Additionally, the variance and bias are calculated for each polynomial model. To better understand how error, variance, and bias behave with increasing complexity, a plot is constructed. This iterative process provides information on the model's accuracy and helps quantify the bias and variance associated with different polynomial degrees. The code employed has been adapted from Morten Hjorth-Jensen's lesson of week 37 [10].

2.6.2 Cross-Validation

Cross-validation is used for model evaluation and hyperparameter selection. This technique involves dividing the dataset into k folds, where the model is trained on a subset and tested on the remaining data. The model is trained k times, each time using a different fold as the test set, while the remaining $k-1$ folds serve as the training set. In this paper, both 5 and 10 folds Cross-Validation are applied on OLS, Ridge, and Lasso algorithms. In the current analysis, a validation set is not created manually but directly from the `cross\_val\_score()` function from Scikit-learn [2].

Since data are already randomly shuffled during the splitting into train and validation data, no further shuffling before applying cross-validation is computed. Shuffling also before the cross-validation has been tested, but no significant difference in the results has been noticed.

Cross-validation enhances model robustness by allowing for repeated training and evaluations of each model. In Ridge and Lasso, this iterative process also aids in identifying the optimal lambda value by comparing model performance across different lambda values. This helps in understanding how different regularizations affect the model performance.

For each method, the mean MSE is plotted as a function of the model's complexity to provide a visual understanding of its behavior. During the implementation of all regression methods with cross-validation, Scikit functions have been employed [2].

2.7 Real Topographical Data

Thus far, the study focused on analyzing different regression algorithms and training approaches using synthetic data, with the goal of identifying the most effective model. After gaining a deeper understanding of the different machine learning techniques employed, the study is extended to real-world data. The goal is to comprehend how different algorithms perform if applied to more complex and noisy data.

The analysis is carried out following the same steps conducted with synthetic data. Ordinary Least Squares, Ridge, and Lasso are applied to different order polynomials. For Ridge and Lasso, the following values of lambda are employed: 0.0001, 0.001, 0.01, 0.1, 1.

Both 5-fold and 10-fold Cross-Validation are applied to evaluate which model fits the data best. Terrain data from a region close to Stavanger in Norway is used for the analysis, specifically the selected data file 'SRTM_data_Norway_1.tif', from the website `https://earthexplorer.usgs.gov/`.

```
# Load the terrain
terrain = imageio.imread('SRTM_data_Norway_1.tif')

# Design Matrix creation and target variable
terrain_sample=terrain[:50,:50]
height, width = terrain_sample.shape

x = np.arange(0, width)
y = np.arange(0, height)

x, y = np.meshgrid(x, y)
X = np.column_stack((x.ravel(), y.ravel()))
z = terrain_sample.ravel()

# Splitting into train and test
X_train, X_test, z_train, z_test = train_test_split(X, z, test_size=0.3,
    random_state=32)
# Splitting into train and validation
X_train, X_val, z_train, z_val = train_test_split(X_train, z_train, test_size=0.1,
     random_state=32)
```

The input of the analysis consists of the x and y coordinates of each terrain point, while the outcome variable is the elevation of the specific point. Only the first 50 rows and columns from the matrix file are employed to reduce the quantity of the data, in order to obtain faster-running algorithms and more interpretable results. The input data is randomly divided into training, validation, and test sets. Then, with respect to the analyzed degree, the corresponding polynomial combinations are added to the design matrix, which is finally scaled. Polynomials up to 25 degrees are tested to gain a wide understanding of the behaviors of the algorithm.

Real-world data is expected to be less regular than synthetic data and also more noisy. For this reason, the scaling process assumes higher importance to ensure reliable results. Therefore, data are not only centered as for the synthetic data analysis but also normalized. The preprocessing employs the same Scikit-learn built-in functions that were previously used [2]. The choices of the different models are based on validation MSE.

3 Results

Here below are presented the results of each analysis. These results are validated through some inspections. In particular, all the functions employed have been tested with 500 random data points with values between 0 and 1, applied to a simple one-degree function without added noise. The validation consists in obtaining as best performing a first degree polynomial having MSE equal to 0. This material can be found attached to this project.

3.1 Synthetic Data

3.1.1 Ordinary Least Squares

The best model fitted by the OLS algorithm is determined based on the minimum validation MSE.

The Figure 1 shows the behavior of the validation and training error as the polynomial degree increases from 1 to 15. A significant MSE decrease is visible for both curves for lower-degree polynomials, showing an improvement in fit. The curve starts to flatten around degree 6, with the validation error reaching a minimum at degree 11, corresponding to an MSE of 1.006. For higher degrees, the curve starts to rise. This aligns with the expectation that increasing the degree improves model fit, up to a certain point, beyond which overfitting starts to occur.

The Figure 2 illustrates the variation in the R^2 metric for both training and validation sets. Similar to MSE, the value increases rapidly for the first polynomial degrees before flattening. A peak in performance is reached at 11 degrees, corresponding to an R^2 value of 0.0805. In both plots a) and b), the training error keeps improving, demonstrating how complex models continue to enhance their fit to the training data. In the plot c), the same analysis is shown up to 50 degrees, with the intention of better showing the expected behavior of the training MSE, which keeps decreasing while the validation MSE rises after a certain level of complexity.

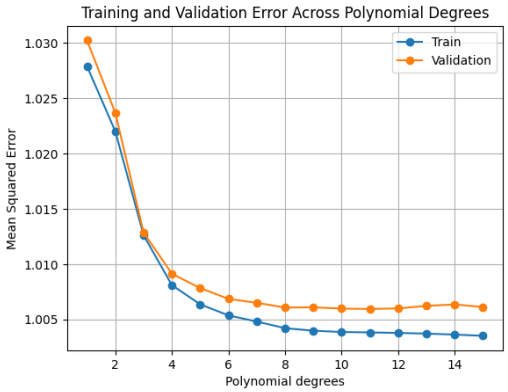

Figure 1: Validation and training MSE as complexity increases

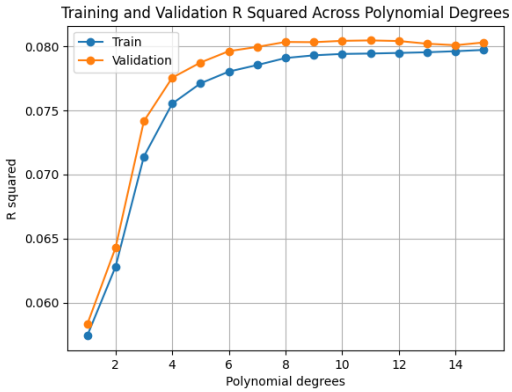

Figure 2: Validation and training R^2 as complexity increases

In the graph 3 it is shown the behavior of the first five parameters with respect to complexity. The intercept, B1 and B2 are relatively constant along complexity while the next generated beta values fluctuates.

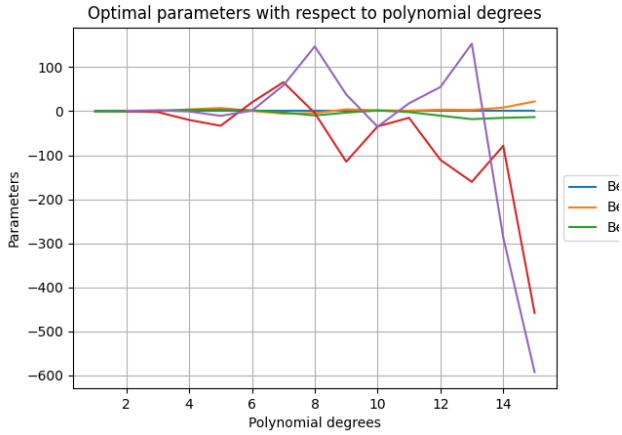

Figure 3: First five parameters along model's complexity

3.1.2 Ridge

For Ridge regression, the best model based on MSE is obtained at degree 2, with a regularization parameter λ of 0.0001. Its validation MSE is equal to 1.32, while the R^2 value is 0.134. From Figures 4 and 5, it is clearly visible that as the degree of the polynomial increases beyond 2, the validation MSE and R^2 gradually worsen. The decline in performance for higher-degree models can be attributed to overfitting. Observing the values of the optimal lambda for each degree, a trend is noticeable. In fact, for degrees 1 and 2, a value of 0.0001 is picked; for degree 3, a value of 0.1; and for all other degrees, lambda is set to 1. This shows how a stronger regularization is applied as the model complexity rises. The best performance is achieved with low regularization applied to a relatively simple model.

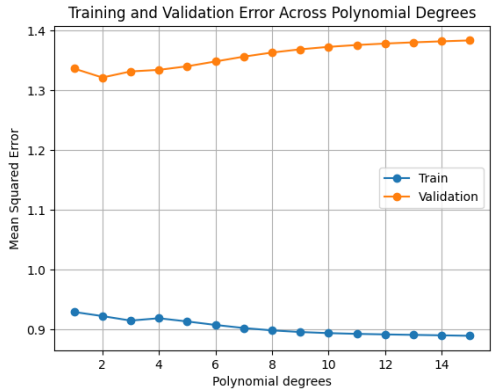

Figure 4: Validation and training MSE as complexity increases

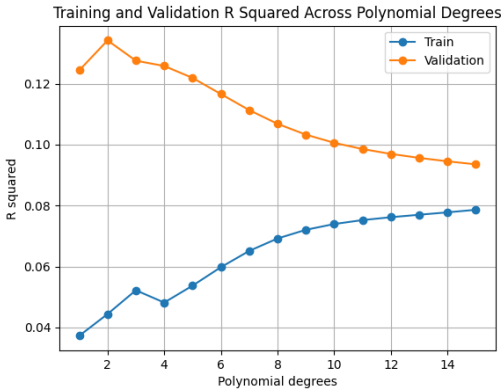

Figure 5: Validation and training R^2 as complexity increases

3.1.3 Lasso

The best performing model, based on both MSE and R^2, is obtained with a polynomial degree of 2 and a regularization parameter $\lambda = 0.0001$. This model achieves a validation MSE of 1.3219 and an R^2 value of 0.1342. Comparatively, lower degrees also show relatively good performance. However, models with higher degrees exhibit less accurate performance in terms of validation MSE and R^2, indicating overfitting to the training data.

Across all degrees, the optimal λ values were consistently low, with most degrees favoring $\lambda = 0.001$, except for degrees 1 and 2, which performed better with $\lambda = 0.0001$. This suggests that the models required only slight regularization to achieve their best performance.

3.1.4 Bootstrap

The bias-variance trade-off analysis is conducted using the bootstrap method on polynomial regression models with degrees ranging from 1 to 15. The resulting plot, Figure 6, shows the behavior of MSE, bias, and variance as a function of the polynomial degree.

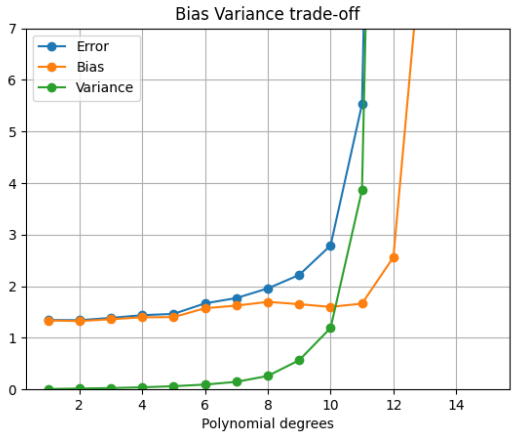

Figure 6: Bias-Variance trade-off for OLS regression with bootstrap resampling across polynomial degrees 1 to 15.

At lower polynomial degrees, bias remains relatively high while variance is low, indicating underfitting. As the polynomial degree increases beyond this point variance begins to increase. This sharp rise in variance, particularly after degree 9, corresponds to overfitting as the model becomes too complex and follows more noise from the training data. The validation error follows the same behavior. However, the bias should decrease in higher-degree models. Unexpectedly, this is not the case as beyond the 10th degree bias rapidly increases. This behavior could be associated with the random processes employed in our code, for example in the extraction or the split of the synthetic data. Changing random seeds could lead to different results.

The optimal degree is found to be at degree 2, where the MSE reaches its lowest value of 1.34. At this degree, the model achieves the best balance between bias and variance, leading to the most accurate generalization on unseen validation data. Beyond degree 2, the MSE increases slightly due to rising variance, while the bias continues to decrease.

3.1.5 Cross-Validation for OLS

The best-performing model is identified both in the 5-fold and 10-fold case as the polynomial regression with degree 1. Specifically, the MSE for the 5-fold cross-validation is 0.9786, while the MSE for the 10-fold cross-validation is 0.9788. The minimal differences between the MSE values from the 5-fold and 10-fold cross-validation indicate that the choice of folds does not significantly affect the generalization performance of the model in this case.

The graph 7 shows that the MSE remains relatively low and stable for polynomial degrees ranging from 1 to 13. This minimal change in the error across these degrees suggests that increasing the complexity of the polynomial model does not significantly improve the prediction accuracy.

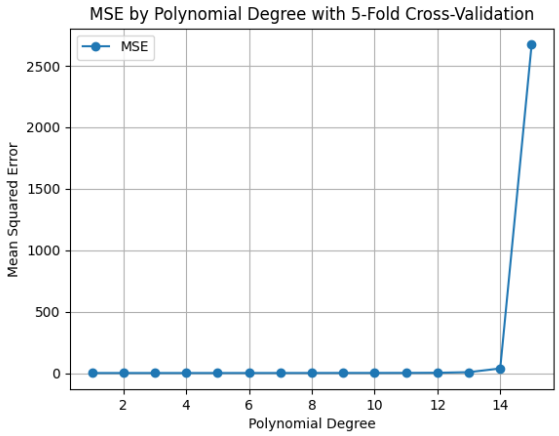

Figure 7: MSE along model's complexity trained by 5-fold Cross Validation

3.1.6 Cross-Validation for Ridge

In the 5-fold cross-validation case, the lowest MSE (0.9782) is achieved with a polynomial of degree 1 and a regularization parameter (λ) of 1. The error increases as the complexity of the model increases. Similarly, in the 10-fold cross-validation scenario, the lowest MSE (0.9786) is also observed at degree 1 with the same λ value. As the degree of the polynomial increases, the MSE remains fairly stable up to around degree 8 but begins to increase as higher-degree polynomials are considered (fig. 8). The

MSE reaches 0.9944 at degree 15, again reflecting the overfitting problem with higher complexity of the model.

Overall, the results highlight that regularization effectively controls model complexity, particularly in lower-degree polynomials. However, as the degree of the polynomial increases, the regularization becomes insufficient to counterbalance the increased complexity, resulting in overfitting. It is noted that there is a consistency in the choice of $\lambda = 1$ across different degrees suggesting that the models favor a relatively strong regularization.

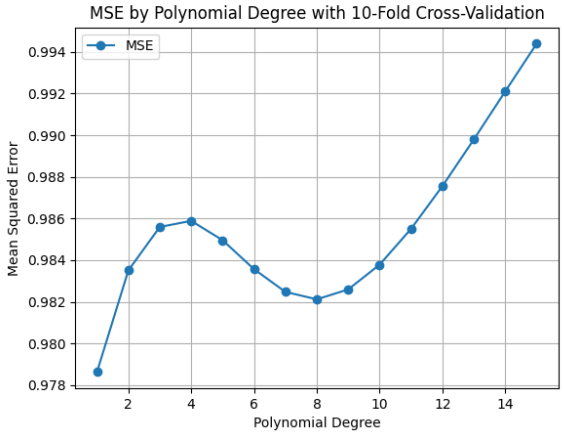

Figure 8: MSE along model's complexity trained by 10-fold Cross Validation

3.1.7 Cross-Validation for Lasso

In the 5-fold cross-validation for Lasso, the optimal model is found with a polynomial of degree 1, the optimal regularization parameter is small ($\lambda = 0.0001$). As the degree of the polynomial increases, the optimal λ increases to 0.01 for degrees up to 15, resulting in stable MSE values around 0.9904. This suggests that for higher-degree polynomials, the Lasso model effectively shrinks the coefficients to prevent overfitting. The results from the 10-fold cross-validation have a slightly different pattern. The lowest MSE (0.9788) is also obtained at degree 1, but this time with a lambda value of 0.0001. For higher degrees, the optimal lambda varies between 0.001 and 0.01, resulting in MSEs values ranging from 0.9817 to 0.9862. This analysis suggests that Lasso's regularization is more effective when cross-validated on a larger number of folds. The MSE consistency for higher degrees suggests that increasing complexity beyond a certain point does not improve model performance, and Lasso is able to prevent overfitting by shrinking coefficients to zero where appropriate.

3.1.8 Final Model Evaluation

Following the identification of Ridge Regression as the best-performing method, the model is then trained with the entire training dataset to obtain the most accurate estimates of the model parameters. The optimal parameters obtained from this training phase are $[0.4101, -0.4871, -0.7322]$. The trained Ridge Regression model is applied to the test dataset. The MSE on this unseen data is approximately 1.0118, indicating a satisfactory performance.

	Mean Squared Error (MSE)	Best Model Degree
Ordinary Least Squares (OLS)	1.006	11
Ridge Regression	1.32178	2
Lasso Regression	1.32185	2
Bootstrap OLS	1.3416	2
OLS with 5-Fold Cross-Validation	0.9786	1
OLS with 10-Fold Cross-Validation	0.9788	1
Ridge with 5-Fold Cross-Validation	0.97823	1
Ridge with 10-Fold Cross-Validation	0.97863	1
Lasso with 5-Fold Cross-Validation	0.97861	1
Lasso with 10-Fold Cross-Validation	0.97879	1

Table 1: Mean Squared Error (MSE) and Best Model Degree across various regression techniques. The minimum MSE value is highlighted in red.

3.2 Real Topographical Data

3.2.1 Ordinary Least Squares

In the OLS analysis on real data the most efficient model is identified at degree 21, which achieves an MSE of 4.06. Beyond the optimal degree the validation MSE starts rising again as expected. Once again, this suggests that while higher polynomial degrees generally improve model fit, they may also introduce complexity that does not always translate to better performance on unseen data.

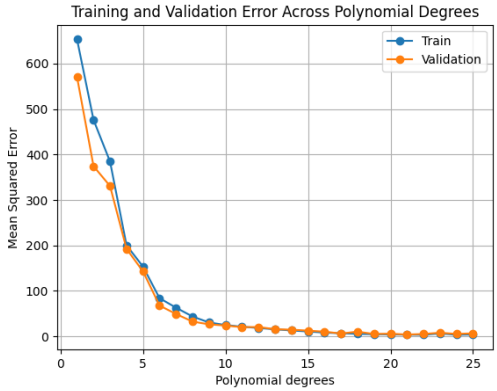

Figure 9: Validation and training MSE as complexity increases

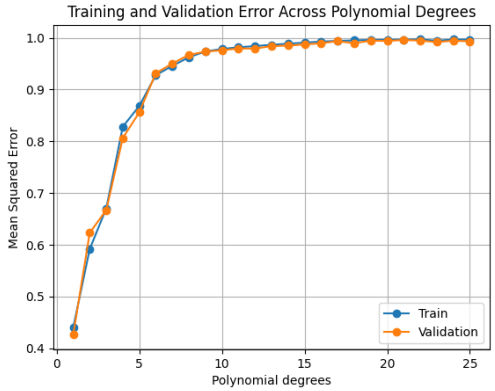

Figure 10: Validation and training R^2 as complexity increases

3.2.2 Ridge

The best model is identified as the one having degree 24, with a λ of 0.0001, achieving an MSE of 25.14. As the polynomial degree increases, a notable trend of decreasing MSE is present. From the results of the analysis shown in the plot 11 it's clear how after a sharp drop in the MSE values for the first degrees the decrease starts to be less important in every step. To have a better compromise

between performance and complexity, a lower degree could be seen as optimal. It seems reasonable to pick the degree in which the curve starts to flatten, gaining in model simplicity with a negligible performance loss. In this case a degree of 9 seems coherent.

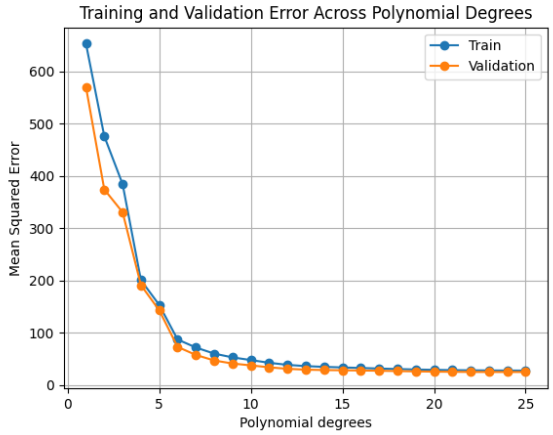

Figure 11: Validation and training MSE as complexity increases

3.2.3 Lasso

The optimal model was identified at degree 25, with a regularization parameter λ of 0.0001, resulting in a MSE of 76.04. Also for Lasso holds the same reasoning mentioned in the Ridge analysis results, choosing a simpler model having slightly worse performance to higher degree polynomials seems a reasonable approach. In this case a degree of 10 is identified as appropriate by evaluating the MSE values and the figure 12.

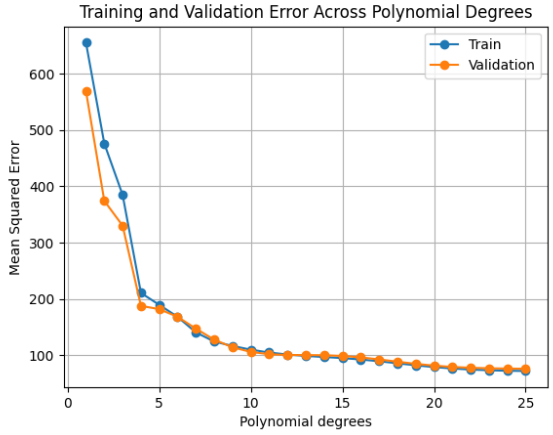

Figure 12: Validation and training MSE as complexity increases

3.2.4 Cross-Validation for OLS

With 5-fold cross-validation, the best-performing model is found at degree 16, having a MSE of 13.7139. As shown in the plot 13, the MSE demonstrates a consistent decline as the degree rose,

beginning from 647.16 at degree 1 and reaching its lowest value at degree 16. A similar trend is observed with 10-fold cross-validation (figure 14), where the optimal model also emerged at degree 20, yielding an MSE of 12.122. For both 5-fold and 10-fold cross-validation a model with degree 9 seems to be a correct balance between performance and simplicity of the model.

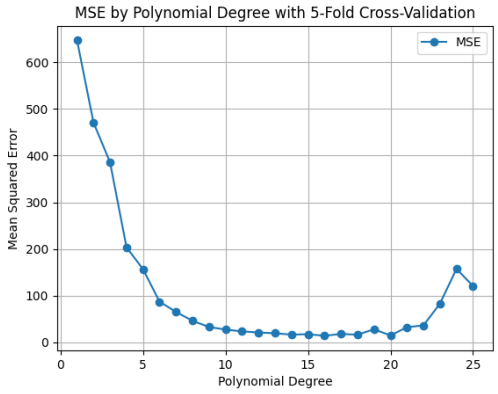

Figure 13: MSE along model's complexity trained by 5-fold Cross Validation

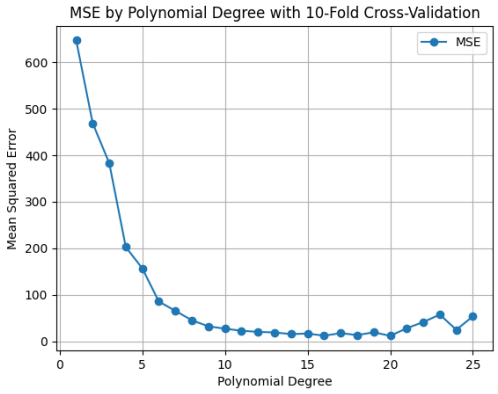

Figure 14: MSE along model's complexity trained by 10-fold Cross Validation

3.2.5 Cross-Validation for Ridge

Employing cross-validation for Ridge regression, the MSE shows a steady decrease as the polynomial degree increases. The best performance is achieved at higher degrees. For the 5-fold cross-validation, the lowest MSE is obtained at degree 21, with an MSE of 30.661 and a regularization parameter of 0.0001. Similarly, for 10-fold cross-validation the lowest MSE is also achieved at degree 21, with a nearly identical MSE of 30.484 and the same λ of 0.0001. The optimal λ is small, implying minimal regularization. Both cross-validation methods identify the 21-degree polynomial as the best model. This conclusion is supported by OLS regression, confirming that the 21-degree polynomial provides the best fit for the data. For both 5-fold and 10-fold cross-validation, a model with degree 11 seems to strike a correct balance between performance and simplicity of the model.

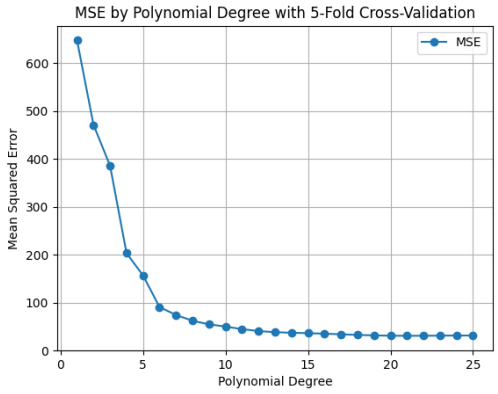

Figure 15: MSE along model's complexity trained by 5-fold Cross Validation

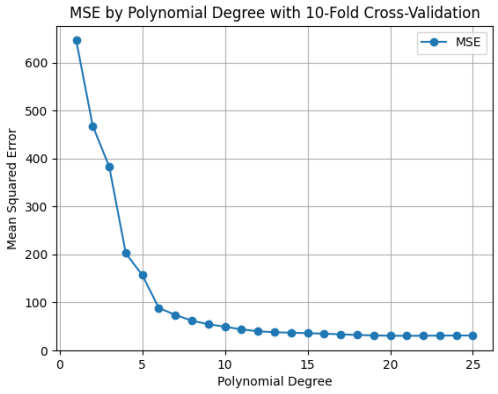

Figure 16: MSE along model's complexity trained by 10-fold Cross Validation

3.2.6 Cross-Validation for Lasso

With 5-fold cross-validation, the optimal model appears at degree 15, achieving a MSE of 138.531, with a lambda of 0.0001. Similar behavior is observed with 10-fold cross-validation, where the best model was also found at degree 15, with a slightly lower MSE of 137.647 and the same optimal lambda of 0.0001.

3.2.7 Final Model Evaluation

	Mean Squared Error (MSE)	Best Model Degree
Ordinary Least Squares (OLS)	4.057	21
Ridge Regression *	25.148	24
Lasso Regression *	76.036	25
OLS with 5-Fold Cross-Validation *	13.713	16
OLS with 10-Fold Cross-Validation *	12.122	20
Ridge with 5-Fold Cross-Validation *	30.6617	21
Ridge with 10-Fold Cross-Validation *	30.4845	21
Lasso with 5-Fold Cross-Validation	138.531	15
Lasso with 10-Fold Cross-Validation	137.647	15

Table 2: Mean Squared Error (MSE) and Best Model Degree across various regression techniques. The minimum MSE value is highlighted in red. The techniques marked with an asterisk (*) indicate that a less complex model was arbitrarily selected than the one resulting from the analysis.

For real-data, the OLS is the best performing method with a degree of 21. By training this algorithm on the entire training dataset and testing on the test data, a value of MSE = 18,249 is obtained. Plot 17 shows the predicted values generated from test data by the OLS model with respect to the original sample extracted from real data.

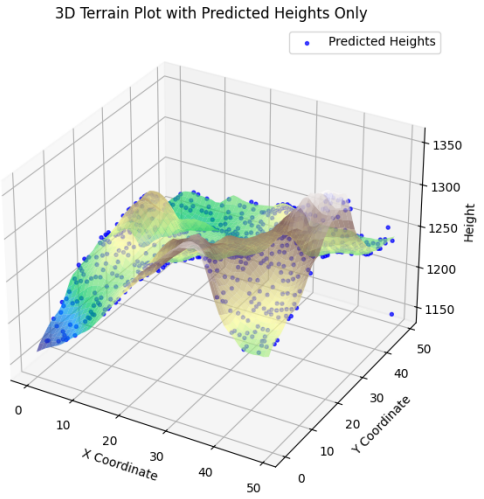

Figure 17: 3D terrain plot to compare predicted values from test data (blue points) to original sample.

4 Discussions

The results obtained allow us to have a general understanding of each model's behavior.

In the synthetic data the performance of each model is assessed around low values of MSE, which is optimal. Ordinary least squares is a good performing method but favors a relatively high complexity of the model. In this context the efficiency of the regularization methods is well highlighted by the results. Lasso and Ridge select a very simple model keeping an error close to the one of OLS. Cross validation is demonstrated to be an efficient method to obtain better results, it allows to lower the model error and their complexity. Generally results on synthetic data follow the expectations for each technique applied.

For real data the situation is not as accurate. Overall the performance is less satisfactory, with particularly high errors for Lasso that were not expected. It is quite evident how OLS outperforms every other method in this context. By looking at best performing models for each algorithm, regularization doesn't seem so effective.

By considering a compromise between performance and complexity as mentioned in the results, regularization might show its effects also on this analysis. Cross validation offers better results in lowering model complexity but not in increasing performance.

In synthetic data having values in a small range [0,1] and a controlled noise helped in allowing models to operate close to their theoretical characteristics.

With real terrain data the situation is more challenging: a more complex relationship, more noise and outliers are expected. OLS might perform well but with a higher risk of overfitting. Regularization methods on the other hand might not be effective due to complexity or noiseness of the data. The real-data sample might have also been not representative or particularly noisy leading to a biased and less accurate prediction. We would like to mention that enlarging the data space did not actually improve the results.

5 Conclusions

This project provided insights into the different linear regression methods and their behaviors. A key takeaway from this analysis is the necessity of understanding the underlying principles of each method to interpret results and to choose the appropriate model for different data.

The role of complexity in model performance has been examined, showing the need to balance it, avoiding overfitting while reducing bias. Choosing a more complex model is not always convenient. Lasso and Ridge demonstrated efficacy in managing this tradeoff, with the drawback of sacrificing some predictive accuracy.

Additionally it is clear from the analysis the necessity of carefully selecting evaluation metrics, while train MSE is useful to identify underfitting or overfitting it is clearly not appropriate for comparing models. The value of data-preprocessing steps and intercept handling is also evident. The study reveals the need of examining different models as slightly less accurate ones might offer significantly more interpretable solutions.

Furthermore, synthetic data analysis helped understand the behavior of each technique, but as shown in the real-data analysis the applications of these to noisy real-world scenarios is a challenging task.

Overall this project established a solid workflow for applying linear regression techniques that can be adapted and exported to different contexts and datasets. Moving forward it may be worth trying to expand the considerations made by testing on more datasets and with different parameters. Exploring more sophisticated approaches could improve the results on the terrain data, for example expanding beyond linear regression could better explain the relationship present in the data.

6 Appendix

6.1 Source Code

All codes are available in the GitHub repository: Project-1-Proetto-Ruffoni-FYS-STK3155.

The code used to implement and test the functions described in this paper can be found on GitHub at the GitHub repository for testing code.

All analyses on synthetic data, including the corresponding results and plots, are available at the GitHub repository for synthetic data analysis.

Additionally, the analysis of real-world data, along with its associated outputs, can be found at the GitHub repository for real data analysis.

6.2 ChatGPT Usage

ChatGPT was used to improve the clarity and precision of the language, as English is not the authors' native language. Its assistance was limited to suggesting synonyms and refining sentence structures. It was never used to generate paragraphs or contribute directly to the text. Additionally, ChatGPT was not involved in the creation of the functions implemented in the analysis. The use has been sparse and occasional so not direct chat will be provided.

References

[1] Jerome Friedman Trevor Hastie Robert Tibshirani. *ELements of Statistical Learning, Data Mining, Inference, and Prediction, 2ed*. Springer, 2009.

[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. "Deep Learning". In: *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 5.

[4] Pankaj Mehta et al. "A high-bias, low-variance introduction to Machine Learning for physicists". In: *Physics Reports* 810 (May 2019), 1–124. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2019.03.001. URL: http://dx.doi.org/10.1016/j.physrep.2019.03.001.

[5] Cheng Soon Ong Marc Peter Deisenroth A. Aldo Faisal. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[6] V. Sharma. "A Study on Data Scaling Methods for Machine Learning". In: *International Journal for Global Academic Scientific Research* (2022).

[7] Trevor Hastie Robert Tibshirani Jonathan taylor Gareth James Daniela Witten. *An introduction to statistical learning with applications in python*. Springer, 2023.

[8] Wessel N. van Wieringen. *Lecture notes on ridge regression*. 2023. arXiv: 1509.09169 [stat.ME]. URL: https://arxiv.org/abs/1509.09169.

[9] Morten Hjorth-Jensen. *Data analysis and machine learning: From ordinary least squared to Ridge ansd Lasso Regression*. URL: https://github.com/CompPhysics/MachineLearning/blob/9c19ea9bfdb1ed64ab2106e43d19b4b75c324802/doc/LectureNotes/week35.ipynb.

[10] Morten Hjorth-Jensen. *Data analysis and machine learning: Statistical interpretation and Resampling methods*. URL: https://github.com/CompPhysics/MachineLearning/blob/9c19ea9bfdb1ed64ab2106e43d19b4b75c324802/doc/LectureNotes/week37.ipynb.

[11] Morten Hjorth-Jensen. *Introduction to the course, Logistics and Practicalities*. URL: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week34.ipynb.