UiO : Department of Physics
University of Oslo

FYS-STK3155/4155 -

APPLIED DATA ANALYSIS AND MACHINE LEARNING

Regression analysis and resampling methods

Authors:
Anders Rudi Bråthen Bakir
Nadia Elise Helene Ørning
Trond Victor Qian

8th October 2024

Abstract

This report examined linear regression methods by fitting two-dimensional polynomials to two datasets. We aimed to evaluate the best fit among Ordinary Least Squares (OLS), Ridge regression, and Lasso regression by applying them to the two-dimensional Franke's function and real terrain data. The models were then optimized using bootstrap resampling and 5-10-fold cross-validation techniques. Error analysis was conducted by examining the Mean square error (MSE), R^2-score, and the bias-variance trade-off.

Our findings indicate that the optimal polynomial degree for OLS and Ridge regression was $m = 5$, and Lasso regression $m = 3$, with an additional regularization parameter $\lambda = 0.0001$ for Ridge and Lasso. Ridge regression demonstrated the best performance for both datasets. For Franke's function, OLS and Ridge achieved an MSE of 0.0105 and 0.0106 and an R^2 score of 0.8870 and 0.8866, respectively, outperforming Lasso (MSE: 0.0149, R^2: 0.8340). Ridge's regularization effectively mitigated overfitting, resulting in a favorable bias-variance balance. Similarly, for the terrain dataset, Ridge regression with a polynomial degree of $m = 8$ and $\lambda = 0.0001$ provided the best results, with an MSE of 0.0098 and an R^2 score of 0.8315. Overall, Ridge regression was identified as the optimal approach for both datasets due to its superior generalization and regularization capabilities.

Table of Contents

1 Introduction

Regression methods are fundamental in data analysis and fit functions to a given data set. Linear regression, in particular, is widely used for estimating the relationship between a dependent variable and one or more independent variables. However, different variants of linear regression can yield varying results depending on the complexity of the data and the presence of noise. Ordinary Least Squares (OLS), Ridge regression, and Lasso regression are three well-known linear regression methods, each with distinct strengths and limitations when applied to different datasets.

This report aims to explore and compare the performance of these three regression techniques by fitting two-dimensional polynomials to two datasets: a dataset generated using Franke's function, which is commonly used for testing regression models, and real-world terrain data from Møsvatn Austfjell in Norway. To ensure good model evaluation, we will optimize the models by applying various resampling techniques, including bootstrap resampling and 5-10-fold cross-validation. We will also use error metrics for comparison, including the Mean Square Error (MSE) and R^2-score. By evaluating these metrics, we aim to identify the optimal polynomial degree and assess the overall performance of each regression technique for both generated and real data.

The report is structured as follows: Section 2 presents the theoretical background, Section 3 outlines the methods used, Section 4 gives the results, and Section 5 discusses the results. The code for this project can be found at: https://github.com/rudi191/FYS_STK_project1/tree/main.

2 Theory

2.1 Linear Regression

Linear regression is an important statistical method for modeling the relationship between a dependent response variable $\boldsymbol{y}$ and one or more independent predictor variables $\boldsymbol{x}$. Suppose you have a dataset consisting of n observations i = 0, 1, 2, ..., n-1 of a response variable $\boldsymbol{y}_i$ and a set of predictor variables $\boldsymbol{x}_i = [x_{i0}, x_{i1}, ..., x_{ip-1}]$. The design matrix of the model is a set of these vectors $X = [x_0 x_1 ... x_{n-1}]$. In a linear regression model, the response variable $\boldsymbol{y}_i$ is linear to the predictors [1]:

$$\mathbf{y}_i = \beta_0 + \beta_1 \mathbf{x}_{i1} + ... + \beta_p \mathbf{x}_{ip} + \varepsilon, \tag{1}$$

which can also be written in matrix notation:

$$\mathbf{y} = X\beta + \varepsilon. \tag{2}$$

β is a vector of unknown parameters, and ε is an error term (noise) assumed to have a mean equal to zero [1]. The approximation $\tilde{\mathbf{y}}$ can be found from $\mathbf{y}(\mathbf{x}_i) = \tilde{\mathbf{y}}_i + \varepsilon_i$. Using Equation 2, we can rewrite the equation as:

$$\tilde{\mathbf{y}}(\mathbf{x}_i) = X\beta. \tag{3}$$

When assuming that the noise ε is normally distributed at zero with a standard deviation of σ, we can calculate the expectation value

$$\mathbb{E}[\mathbf{y}] = X\beta, \tag{4}$$

and the variance

$$\mathrm{Var}[\mathbf{y}_i] = \sigma^2. \tag{5}$$

These calculated values are found in Appendix I.

2.1.1 Ordinary Least Squares (OLS)

The most popular estimation method is called the Ordinary Least Squares (OLS) method. This method is used to choose the unknown parameters in the linear regression model, β_i, which fit the equations ”best”. Thus the optimization problem is [2]:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=0}^{n-1} (\mathbf{y}_i - \tilde{\mathbf{y}}_i)^2, \tag{6}$$

and the cost function for the OLS is:

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(\mathbf{y}_i - \tilde{\mathbf{y}}_i)^2 = \frac{1}{n}[(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}})], \tag{7}$$

which can be rewritten by inserting Equation 3 into Equation 7:

$$C(\beta) = \frac{1}{n}[(\mathbf{y} - X\beta)^T(\mathbf{y} - X\beta)]. \tag{8}$$

The cost function is minimized by the optimal parameter $\hat{\beta}$, that is the derivative $\frac{\delta C(\beta)}{\delta \beta_j}$ equals zero. If we do the derivative of Equation 8, we end up with:

$$\frac{\delta C(\hat{\beta})}{\delta \beta_j} = 0 = X^T(\mathbf{y} - X\hat{\beta}), \tag{9}$$

which can be rewritten, and we get the optimal parameter $\hat{\beta}$ of OLS:

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T \mathbf{y}. \tag{10}$$

The expectation value

$$\mathbb{E}(\hat{\beta}_{OLS}) = \beta \tag{11}$$

and the variance

$$Var(\hat{\beta}_{OLS}) = \sigma^2 (X^T X)^{-1} \tag{12}$$

are calculated in Appendix I.

2.1.2 Ridge

The interpretation of β_i in OLS requires caution, as each coefficient describes the effect of its associated predictor while holding the others constant. This can be complex in the presence of multicollinearity[1]. Multicollinearity, or high correlation between predictors, inflates the variance of the estimated coefficients, making them less reliable and harder to interpret. Ridge regression tries to solve this by adding a regularization parameter λ, hyperparameter, to the optimization problem. If we rewrite the optimization problem (Eq. 6) of OLS where we use $||\mathbf{x}||_2 = \sqrt{\sum_i \mathbf{x}_i^2}$, we get:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n}\sum_{i=0}^{n-1}(\mathbf{y}_i - \tilde{\mathbf{y}}_i)^2 = \frac{1}{n}||\mathbf{y} - X\beta||_2^2. \tag{13}$$

Then we can add the hyperparameter:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n}||\mathbf{y} - X\beta||_2^2 + \lambda||\beta||_2^2. \tag{14}$$

Here, $||\beta||_2^2$ has to be less than or equal to t, where t is a finite number larger than zero [2]. Then we can define the optimized cost function of Ridge regression as:

$$C(X,\beta) = \frac{1}{n}[(\mathbf{y} - X\beta)^T(\mathbf{y} - X\beta)] + \lambda\beta^T\beta. \tag{15}$$

We can now minimize Equation 15 as we did for OLS (Eq. 10) and get:

$$\hat{\beta}_{Ridge} = (X^TX + \lambda I)^{-1}X^T\mathbf{y}. \tag{16}$$

2.1.3 Lasso

Another regression method is the Least Absolute Shrinkage and Selection operator (Lasso). We do the same for Lasso regression, and write the optimization problem as [2]:

$$\min_{\beta\in\mathbb{R}^p} \frac{1}{n}||\mathbf{y} - X\beta||_2^2 + \lambda||\beta||_1, \tag{17}$$

with the norm-1 as $||\mathbf{x}||_1 = \sum_i |\mathbf{x}_i|$. We thus get the cost function as:

$$C(X,\beta) = [(\mathbf{y} - X\beta)^T(\mathbf{y} - X\beta)] + \lambda||\beta||_1. \tag{18}$$

If we derivate with respect to β and that the derivative of the absolute value is:

$$\frac{d|\beta|}{d\beta} = sgn(\beta) = \begin{cases} 1 & \beta > 0 \\ -1 & \beta < 0, \end{cases}$$

then the derivative of the cost function Equation 18 is:

$$\frac{\delta C(X,\beta)}{\delta\beta} = -\frac{2}{n}X^T(\mathbf{y} - X\beta) + \lambda sgn(\beta) = 0, \tag{19}$$

which can be rewritten as:

$$X^TX\beta + \lambda sgn(\beta) = X^T\mathbf{y}. \tag{20}$$

As opposed to OLS and Ridge regression, Equation 20 does not lead to a pretty analytical equation, but can be solved using standard convex optimization algorithms.

2.2 Bias-Variance trade-off and resampling techniques

The Bias-Variance trade-off explains how the complexity of a model affects its prediction accuracy and its capability to perform well on data it hasn't seen during model training. When we increase the complexity of a model, it typically leads to higher variance but lower bias.

Variance refers to how much a model's predictions fluctuate for different data sets; high variance means the model is sensitive to small changes in the data, resulting in a wide spread of predictions around the mean. Conversely, bias represents the model's systematic error—how much the model's predictions consistently deviate from the true values. A model with high bias oversimplifies the data, failing to capture underlying patterns. An ideal model strikes a balance between low bias and low variance to avoid both under- and overfitting. Overfitting is when the training model follows the noise too closely and then we have high variance and low bias, while underfitting is when the model has less complexity so that it does not fit the data well enough to capture the trend of the data, and then we have low variance and high bias [1]. When we perform a Bias-Variance trade-off, where we try to minimize the error of bias and variance simultaneously, we keep the test data fixed while new training is performed. In other words, we calculate the Mean Squared error (MSE), variance, and bias from the same test data.

The cost function for the ordinary least squares method is given by:

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

We can show that this cost function can be rewritten in terms of the bias and variance of the model. First, the cost function can be rewritten as:

$$\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \mathbb{E}[\mathbf{y}^2 - 2\mathbf{y}\tilde{\mathbf{y}} + \tilde{\mathbf{y}}^2] = \mathbb{E}[\mathbf{y}^2] + 2\mathbb{E}[\mathbf{y}\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}^2]. \tag{21}$$

$\mathbb{E}[\mathbf{y}^2]$ can be written as:

$$\mathbb{E}[\mathbf{y}^2] = \mathbb{E}[(f + \varepsilon)^2] = \mathbb{E}[f^2 + 2f\varepsilon + \varepsilon^2] = \mathbb{E}[f^2] + 2\mathbb{E}[f\varepsilon] + \mathbb{E}[\varepsilon^2]. \tag{22}$$

Here, we assumed that the true data is generated from a noisy model $\mathbf{y} = f(x) + \varepsilon$, that $\mathbb{E}[f] = f$ because $f(x)$ is assumed unbiased, that $\mathbb{E}[\varepsilon] = 0$, and that $\mathbb{E}[\varepsilon^2] = \sigma^2$. We then get:

$$\mathbb{E}[\mathbf{y}^2] = f^2 + 2f\mathbb{E}[\varepsilon] + \sigma^2 = f^2 + \sigma^2. \tag{23}$$

$\mathbb{E}[\mathbf{y}\tilde{\mathbf{y}}]$ can be written as:

$$\mathbb{E}[\mathbf{y}\tilde{\mathbf{y}}] = \mathbb{E}[(f + \varepsilon)\tilde{\mathbf{y}}] = \mathbb{E}[f\tilde{\mathbf{y}} + \varepsilon\tilde{\mathbf{y}}] = \mathbb{E}[f\tilde{\mathbf{y}}] + \mathbb{E}[\varepsilon\tilde{\mathbf{y}}] = f\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\varepsilon]\mathbb{E}[\tilde{\mathbf{y}}] = f\mathbb{E}[\tilde{\mathbf{y}}]. \tag{24}$$

If we use that $\mathrm{Var}[\mathbf{x}] = \mathbb{E}[\mathbf{x}^2] - (\mathbb{E}[\mathbf{x}])^2$, we can write $\mathbb{E}[\tilde{\mathbf{y}}^2]$ as:

$$\mathbb{E}[\tilde{\mathbf{y}}^2] = \mathrm{Var}[\tilde{\mathbf{y}}] + (\mathbb{E}[\tilde{\mathbf{y}}])^2, \tag{25}$$

and if we combine all the terms, we get:

$$\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = f^2 + \sigma^2 - 2f\mathbb{E}[\tilde{\mathbf{y}}] + \mathrm{Var}[\tilde{\mathbf{y}}] + (\mathbb{E}[\tilde{\mathbf{y}}])^2 \tag{26}$$

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

Here, we assumed that since f is unbiased that $\mathbb{E}$[f] = $\mathbb{E}$[$\mathbf{y}$], and that Bias[$\tilde{\mathbf{y}}$] = $\mathbb{E}$[($\mathbf{y}$ - $\mathbb{E}$[$\tilde{\mathbf{y}}$])^2].

The first term represents the bias, the second term accounts for the variance, and the final term is the variance of the error, ε. Since σ^2 represents the error variance (dataset), it cannot be controlled or reduced. Therefore, only the bias and variance terms contribute to the Mean Squared Error (MSE), and minimizing these components is crucial for reducing the overall MSE.

2.2.1 Bootstrap

The bootstrap method is used to determine statistical accuracy. The approach involves randomly drawing datasets from the training data with replacement, ensuring that each drawn sample has the same size as the original training set. This process is repeated B times to generate B bootstrap datasets. The model is then refitted to each of these bootstrap datasets, allowing us to analyze the behavior of the model across all B replications [3]. This makes the bootstrap method particularly effective when dealing with poorly behaved data or small sample sizes, enhancing the accuracy of the analysis.

The steps for the independent bootstrap are as follows: first, draw n samples with replacement from the observed variables, denoted as $\mathbf{x} = (x_1, x_2, ..., x_n)$. Next, define a new vector $\mathbf{x}^*$ containing the values drawn from $\mathbf{x}$. Then, use the vector $\mathbf{x}^*$ to compute $\hat{\beta}^*$ by evaluating $\hat{\beta}$ with the observations from $\mathbf{x}^*$. Finally, repeat this entire process k times.

2.2.2 Cross-validation (k-fold)

Cross-validation is used to estimate the prediction error of a model. Ideally, we would have an abundance of data, but in practice, data is often limited. To address this, k-fold cross-validation divides the available data into k subsets, using some of these subsets to train the model while reserving the remaining ones for testing [3].

The steps for cross-validation with different values of k are as follows: first, shuffle the dataset randomly. Then, divide the dataset into k groups. For each group, designate it as the test set while using the remaining groups as the training set. Fit the model on the training set, evaluate

it on the test set, and record the evaluation score, discarding the model afterward. Finally, summarize the model using the sample of model evaluation scores [4].

3 Methods

The code we developed can be found in the Github repository https://github.com/rudi191/FYS_STK_project1/tree/main. See README for information on how to reproduce the results in the report. The methods were implemented in Python using the libraries numpy, matplotlib, imageio and scikit-learn. Github copilot, version: 1.168.0, and GPT-UIO were used to aid in the implementation of the code. Much of the code was inspired by the lecture notes and the weekly excercises by Hjorth-Jensen, M, see: Hjorth-Jensen, M. Applied Data Analysis and Machine Learning — Applied Data analysis and Machine learning. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.

3.1 Datasets

3.1.1 Franke Function

To evaluate and test various regression methods, we used the Franke Function, which is a weighted sum of four exponential terms and takes two input arguments:

$$\begin{aligned} f(\mathbf{x},\mathbf{y}) = &\frac{3}{4}\exp\left(-\frac{(9\mathbf{x}-2)^2}{4}-\frac{(9\mathbf{y}-2)^2}{4}\right)+\frac{3}{4}\exp\left(-\frac{(9\mathbf{x}+1)^2}{49}-\frac{(9\mathbf{y}+1)}{10}\right) \\ &+\frac{1}{2}\exp\left(-\frac{(9\mathbf{x}-7)^2}{4}-\frac{(9\mathbf{y}-3)^2}{4}\right)-\frac{1}{5}\exp\left(-(9\mathbf{x}-4)^2-(9\mathbf{y}-7)^2\right). \end{aligned} \tag{27}$$

The equation is defined for $\mathbf{x}$, $\mathbf{y} \in [0, 1]$. This function allowed us to verify that our developed algorithms were functioning correctly before applying them to the terrain data.

The data was split using the train_test_split functionality in scikit-learn into 0.2 test and 0.8 train sets. The data was only centered by subtracting the mean of the training data from the training data and test data since the data was already scaled between 0 and 1.

3.1.2 Terrain Data

The data is from Møsvatn Austfjell in Norway. The specific radar map we used can be downloaded from https://github.com/CompPhysics/MachineLearning/blob/master/doc/Projects/

2023/Project1/DataFiles/SRTM_data_Norway_2.tif. This was imported using the library imageio with the method imread. The shape of the data was assessed and linearly spaced values were defined between 0 and 1 for x and y. The z-values were normalized between 0 and 1.

The data was split into smaller train and test sets due to the large amount of data affecting the runtime. Especially in bootstrapping and cross-validation, where the data was reduced to even smaller amounts to run a high number of resampling at higher complexity. We used the sample size 0.01.

3.1.3 Preprocessing of Data

The preprocessing of data was done to ensure good performance across all model types and to reduce multicollinearity. In addition, it is important for Ridge and Lasso regression because the magnitude of the coefficients, which these methods depend on, is affected by the scale of the data. Another reason is to improve numerical stability for higher order polynomials.

3.2 Implementation in Python

This part presents how we implemented the methods described in the Theory section using Python. We include some code examples in the report of critical components. For calculations and verification of the manual regression methods, see: https://github.com/rudi191/FYS_STK_project1/blob/main/Supplemental%20material/MethodVerificationCode.py

3.2.1 Functions

We defined a function that creates a design matrix for a given polynomial degree with inputs x, y, and degree. This function creates a design matrix that includes the intercept column. We decided to include the intercept in our models. Although including the intercept will have little effect on the overall fit of the model, it ensures that any residual bias is properly managed, after centering the data. The function uses a nested for loop with i and j indexes to create all terms, including the interaction terms. MSE and R^2 functions were created for the evaluation of the models. See code example under.

```
# Defining a function for creating the design matrix
def create_design_matrix(x, y, degree):
    num_terms = int((degree + 1) * (degree + 2) / 2)
```

```
    X = np.zeros((len(x), num_terms))
    idx = 0
    for i in range(degree + 1):
        for j in range(degree + 1 - i):
            X[:, idx] = (x ** i) * (y ** j)
            idx += 1
    return X
def MSE(y_true, y_pred):
    return np.mean((y_true - y_pred)**2)
def R2(y_true, y_pred):
    return 1 - (np.sum((y_true - y_pred)**2) / np.sum((y_true - np.mean(y_true))**2))
```

3.2.2 Regression methods

3.2.3 OLS

The ordinary least square regression was implemented using the normal equation and taking the inverse to find the beta parameters. We then estimated the z values for the train and the test sets and finally calculated the mean square error and coefficient of determination. This was done inside a for loop that iterated over different polynomial degrees, appending MSE, R^2, and coefficients, to evaluate model performance across different complexities. The polynomial degree was 5 for the Franke data and 7 for the terrain data. The results were plotted in two line plots. Code example is given below.

```
X_ = create_design_matrix(x, y, Maxpolydegree)
beta = inv(X_.T @ X_) @ X_.T @ z_
z_pred = X_ @ beta
```

3.2.4 Ridge Regression

For the implementation of Ridge regression, 5 lambda values were used between 0.0001 and 1. The model was fitted using the modified normal equation adding the L2 regulation term by adding lambda times the identity matrix. Then the model was fitted for different polynomial degrees and different lambda values inside a nested for loop. The polynomial degree was 5 for the Franke data and 10 for the terrain data. The results were plotted in line plots. See code example below.

```
beta_ridge = inv(X_.T @ X_ + lmbda * np.eye(X_.shape[1])) @ X_.T @ z_
```

3.2.5 Lasso

Lasso regression was implemented using the lasso method of sckikit-learn. We used the same lambda-values as for Ridge regression. Then the model was fitted for different polynomial degrees and lambdas in the same way as for Ridge regression. Example is given below.

```
lmbda = 0.1
lasso = Lasso(alpha=lmbda, fit_intercept=False, max_iter=10000)
lasso.fit(X_design_train, z_train)
beta_lasso = lasso.coef_
```

3.3 Resampling

3.3.1 Bootstrap

Bootstrapping was used as a resampling technique to study the bias-variance tradeoff with the OLS model. This was implemented with the scikit-learn resample method in a nested for loop. In this way the data was resampled 100 times for each polynomial degree. The prediction values were stored as vectors in a numpy array, with each vector corresponding to all the prediction for the given resampling run. The error, bias and variance was then calculated and stored in arrays. We used 100 bootstraps for polynomial degrees up to 9 for the Franke data and degree 21 for the terrain data. The bias, variance, and error were plotted. Code example of the nested loop is given below.

```
for degree in range(1, Maxpolydegree + 1):
    z_pred = np.empty((z_test.shape[0], n_boostraps))
    for i in range(n_boostraps):
        x_, y_, z_ = resample(x_train, y_train, z_train)
        design_m = create_design_matrix(x_, y_, degree)
        beta_m = inv(design_m.T @ design_m) @ design_m.T @ z_
        z_pred[:, i] = create_design_matrix(x_test, y_test, degree) @ beta_m
```

3.3.2 *K*-fold Cross-validation

Cross-validation was used on OLS, Ridge, and Lasso. Here, we used the scikit-learn methods for the implementation of cross-validation and all three models. The *k*-fold was set to k = 10 and was tested for a max polynomial degree of 9 for the Franke data and degree 10 for the terrain data. The lambda values were between 0.00001 and 0.01, and there were 5 lambda values. The MSE was plotted for different polynomial degrees and different lambda values in one plot. Code example below.

```
Maxpolydegree = 9
k = 10
kfold = KFold(n_splits=k, shuffle=True, random_state=315)
ols_mse = np.zeros(Maxpolydegree)

for degree in range(1, Maxpolydegree + 1):
    X_design_temp = create_design_matrix(x_train, y_train, degree)
    model = LinearRegression()
    scores = cross_val_score(model, X_design_temp, z_train,
                             scoring='neg_mean_squared_error', cv=kfold)
    ols_mse[degree-1] = -np.mean(scores)
```

4 Results

4.1 Franke Data

In this section, we analyzed the performance of OLS, Ridge, and Lasso regression models when fitted to the Franke function, varying the polynomial degree from 1 to 5. The Franke function was plotted as a 3D model in Figure 4.1.

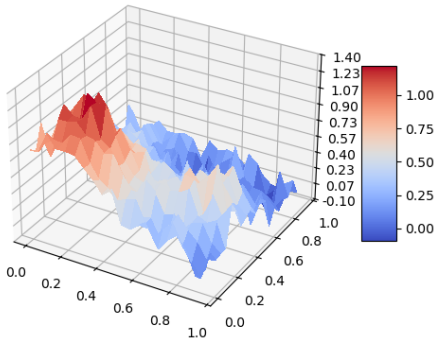

Figure 4.1: 3D-plot of Franke Function with added noise

4.1.1 OLS

First, we plotted the Mean Squared Error (MSE), the coefficient of determination (R^2), and β_i against the different polynomial degrees to analyze their relationship for OLS-regression, which is shown in Figure 4.2 and Figure 4.3. We observed a consistent decline in MSE for both training and testing datasets as the polynomial degree increases. We also observed an increase in R^2 for both training and test datasets. At polynomial degrees four to five the plots might have started to plateau for both MSE and R^2.

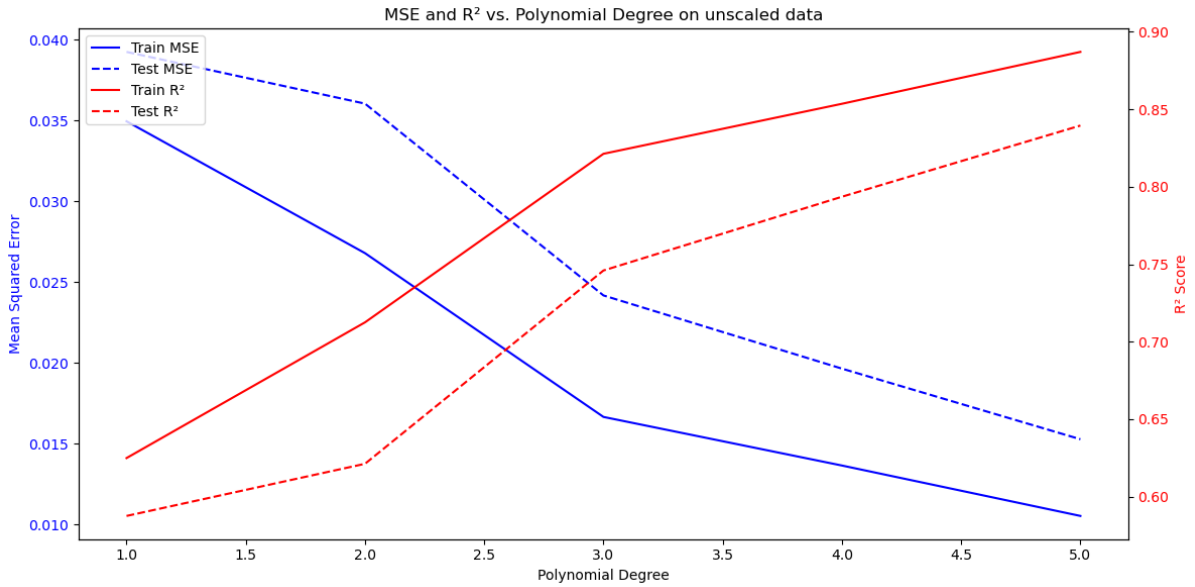

Figure 4.2: MSE and R^2 vs. Polynomial Degree on unscaled data

The β-values in Figure 4.3 tended to increase with model complexity. Notably, polynomial

degree 5 had the highest variance in β.

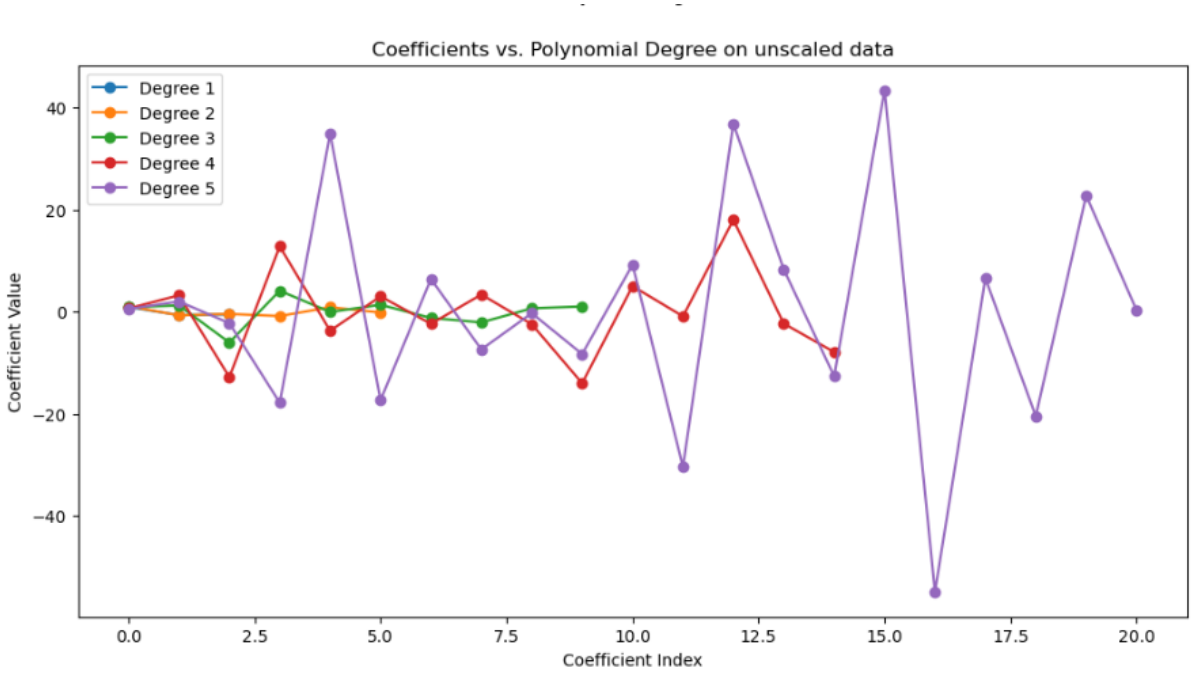

Figure 4.3: β values as a function of coefficients indices and polynomial degree

To examine the effects of scaling, we implemented centering by subtracting the mean from each feature, i.e., `x_train` = `x_train` - `x_mean`. The reason for centering rather than full scaling was based on the fact that our data already lay within the range [0, 1]. The centering, however, did not affect the results, as they remained largely unchanged for both scaled and non-scaled data. Despite this, we chose to continue centering the data. Centering ensures that each feature has a mean of zero, reducing potential biases arising from differences in feature means. As noted in Jupyter Notebook 5.6, "If our predictors represent different scales, then it is important to standardize the design matrix (X) by subtracting the mean of each column from the corresponding column."[2]

4.1.2 Ridge regression

Next, we plotted the MSE (Figure 4.4) and R^2 (Figure 4.5) as a function of polynomial degree for different values of λ in Ridge regression. In Figure 4.4, the MSE consistently decreased as the polynomial degree increased for all values of λ, with a more pronounced decline observed at lower λ values. Similarly, Figure 4.5 shows that R^2 increased with polynomial degree across all λ values, with a steeper increase for smaller values of λ. As the polynomial degree rose and λ decreased, the R^2-score approached the optimal value of 1, indicating improved model fit.

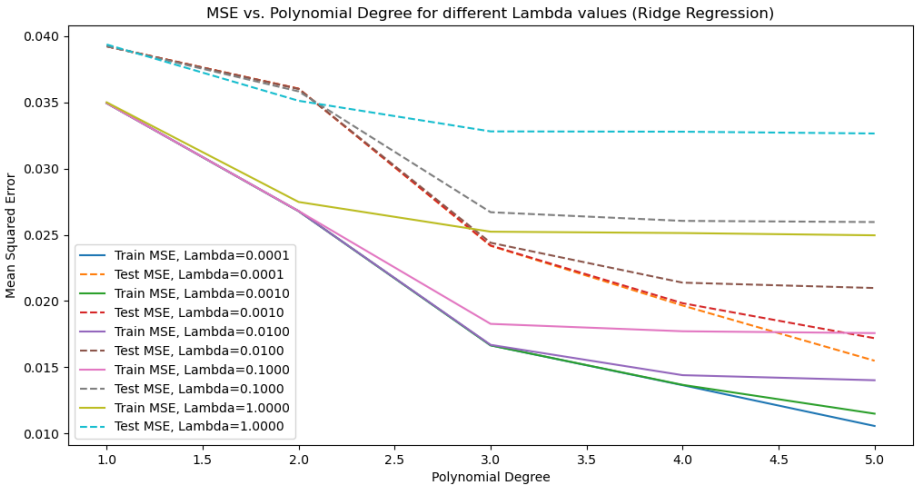

Figure 4.4: MSE vs. Polynomial Degree for Ridge with different lambdas

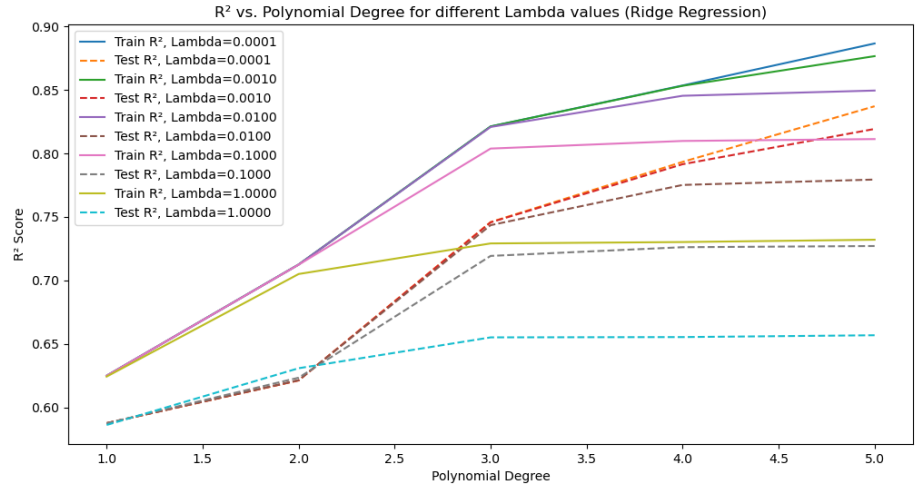

Figure 4.5: R^2 vs. Polynomial Degree for Ridge with different lambdas

4.1.3 Lasso regression

We also plotted the MSE (Figure 4.6) and R^2-score (Figure 4.7) as functions of polynomial degree for various λ values in Lasso regression. Similar to the Ridge regression results, both MSE decreased and R^2 increased as the polynomial degree increased when $\lambda = 0.0001$. However, for larger λ values, both MSE and R^2 remained relatively unchanged.

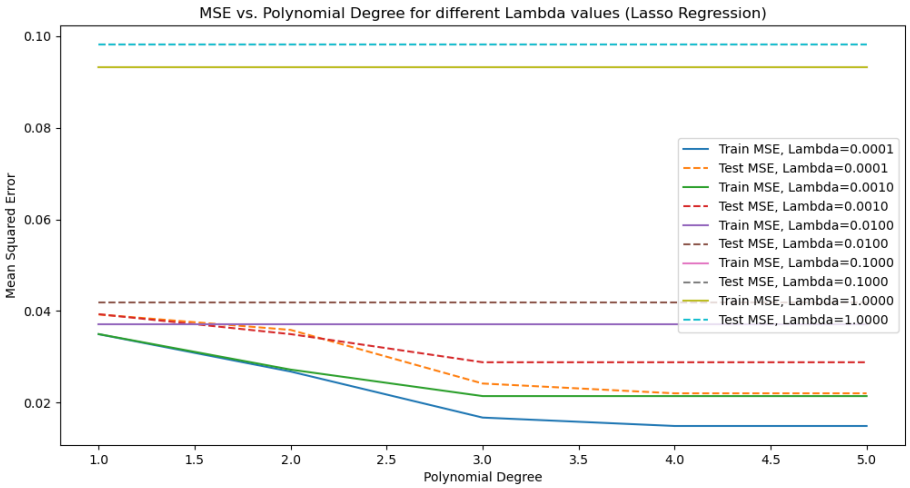

Figure 4.6: MSE vs. Polynomial Degree for Lasso with different lambdas

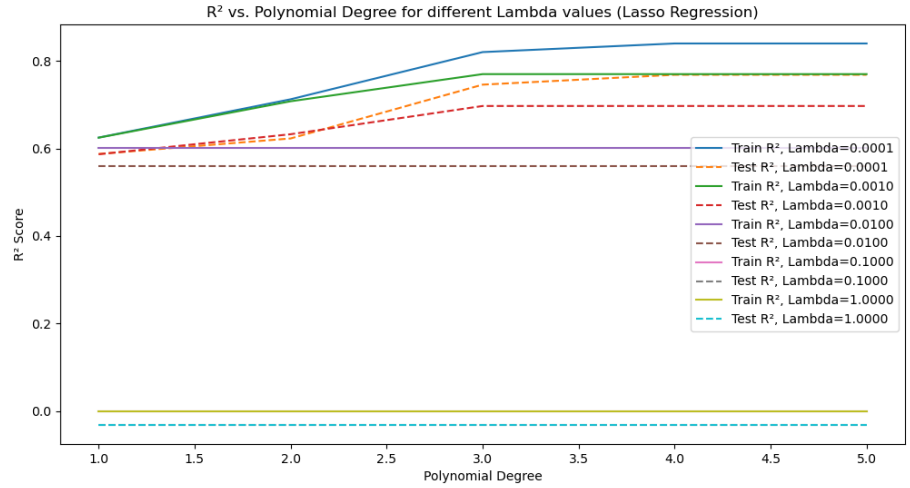

Figure 4.7: R^2 vs. Polynomial Degree for Lasso with different lambdas

The MSE and R^2 values for the OLS, Ridge, and Lasso regression models are presented in Table 4.1.

Table 4.1: The MSE and R^2 values for OLS, Ridge, and Lasso regression on both the train and test datasets.

Train/Test	Regression method	Polynomial degree	λ	MSE	R^2
Train	OLS	5	-	0.0105	0.8870
Test	OLS	5	-	0.0153	0.8395
Train	Ridge	5	0.0001	0.0106	0.8866
Test	Ridge	5	0.0001	0.0155	0.8372
Train	Lasso	3	0.0001	0.0149	0.8340
Test	Lasso	3	0.0001	0.0220	0.7684

4.1.4 Bias-Variance trade-off of OLS

Next, we performed the bootstrap method on the OLS model. Figure 4.8 illustrates the Bias-Variance trade-off for the OLS model, using 100 bootstrap samples to estimate the bias, variance, and overall error as a function of polynomial degree. The plot demonstrates that as the polynomial degree increased, the bias decreased while the variance simultaneously increased. This is a classic pattern of the bias-variance trade-off; the model became increasingly flexible and capable of capturing complex patterns in the data, thereby reducing bias. However, this also resulted in greater sensitivity to fluctuations in the data, leading to an increase in variance.

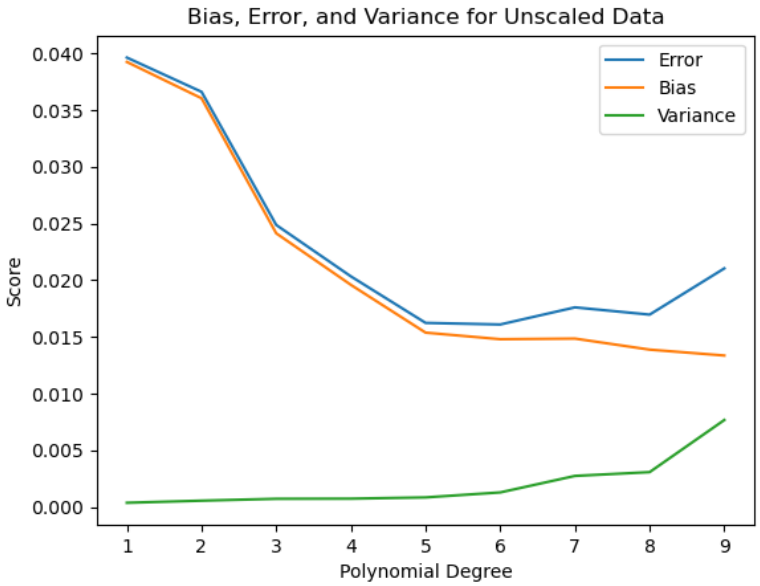

Figure 4.8: Bias-variance plot for OLS

We also plotted our training and test datasets as a function of model complexity, as seen in Figure 2.11 in the book The Elements of Statistical Learning [3]. This figure can be found in Appendix II.

4.1.5 *K*-fold cross-validation

Lastly, we performed *k*-fold cross-validation on both the OLS, Ridge, and Lasso regression models. Figure 4.9 shows the MSE as a function of polynomial degree using 10 fold. The MSE for OLS decreased with increased degree, but at degree 5 the OLS fit started to increase again, which could have been due to overfitting. Ridge and Lasso however stayed constant at around 0.01 after degree 5.

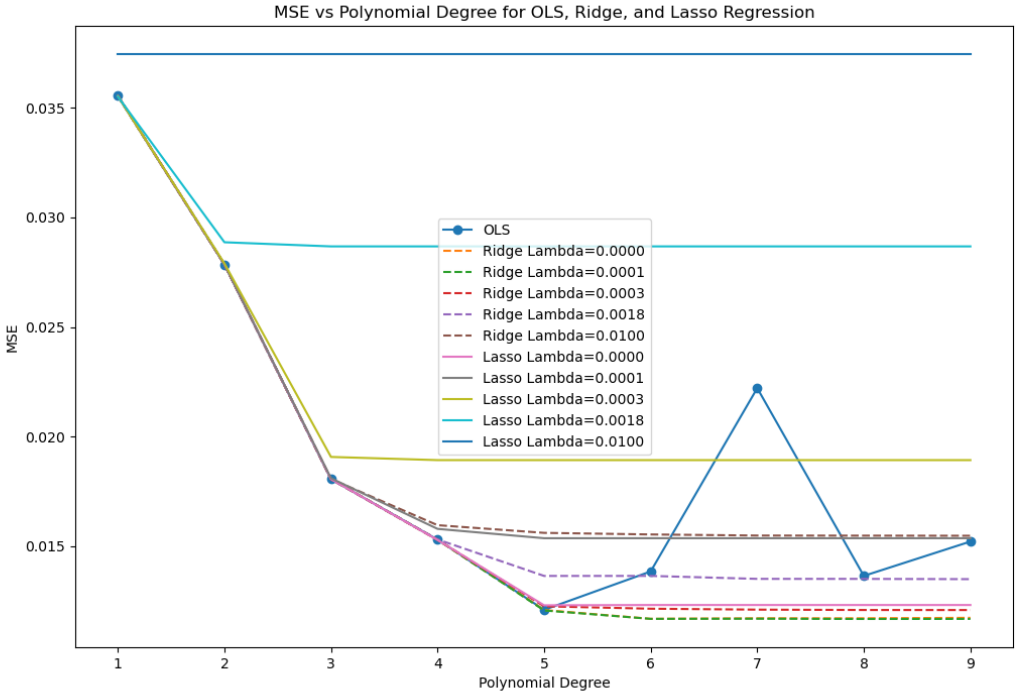

Figure 4.9: MSE plot for Ridge, Lasso, and OLS with cross-validation (10 folds)

4.2 Terrain Data

In this section, we analyzed the performance of OLS, Ridge, and Lasso regression models when fitted to real terrain data. The terrain data was plotted as a 3D-model in Figure 4.10. It also shows the estimated terrain surface.

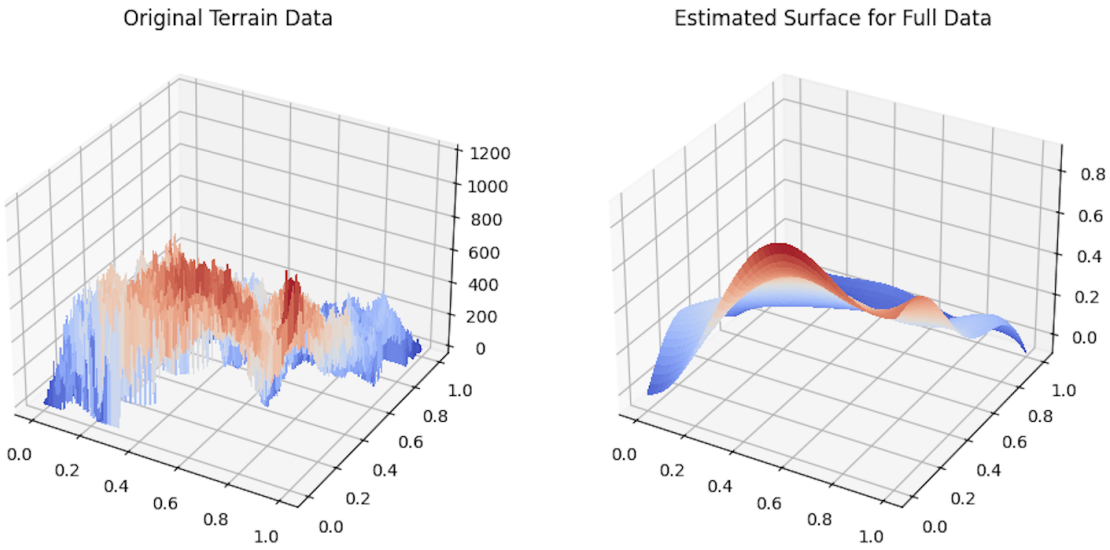

Figure 4.10: 3D-model and estimated surface of terrain data

4.2.1 OLS

Figure 4.11 presents the MSE and R^2 values as a function of polynomial degree for the OLS model, while Figure 4.12 shows the corresponding β-values versus their indices.

As the polynomial degree increased, we observed a decline in MSE for both the training and test datasets, indicating improved prediction accuracy. However, from degree 6 onward, the MSE and R^2 values began to plateau.

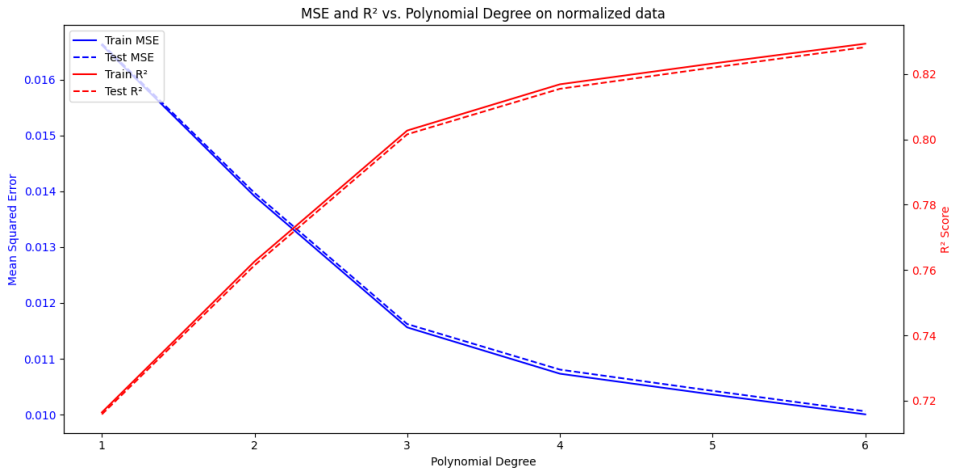

Figure 4.11: MSE and R^2 vs polynomial degree (train/test) for OLS

Moreover, Figure 4.12 shows that the β-values tended to grow in magnitude as model complex-

ity increased, reflecting greater variance at higher polynomial degrees. The variance of β-values was notably highest at degree 6.

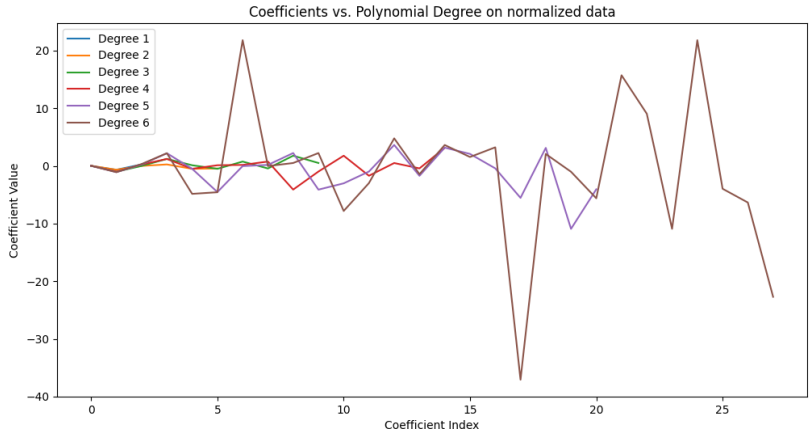

Figure 4.12: β values as a function of coefficient indices for OLS

4.2.2 Ridge regression

Figure 4.13 depicts the MSE as a function of polynomial degree for different λ values in Ridge regression, while Figure 4.14 depicts the corresponding R^2 values. As the polynomial degree increased, the MSE decreased, and the R^2 increased for all values of λ, with the most significant changes occurring at lower λ values. The optimal polynomial degree appeared to be 8, where the MSE reached 0.0098 and the R^2 reached 0.8315 for $\lambda = 0.0001$.

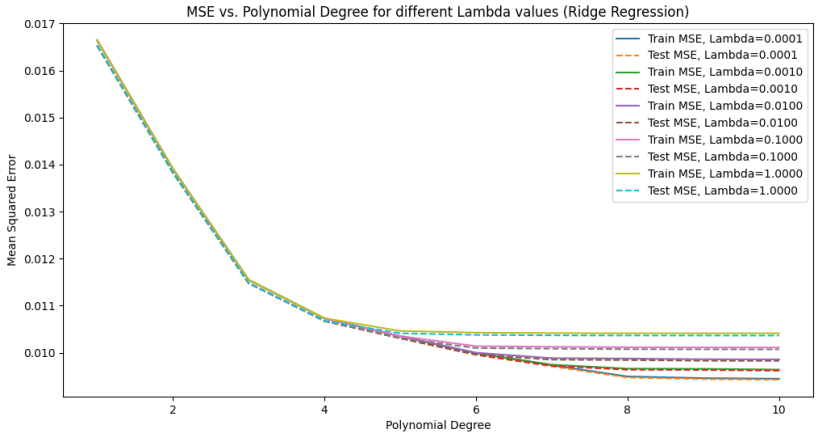

Figure 4.13: MSE vs polynomial degree for different lambda values (Ridge regression)

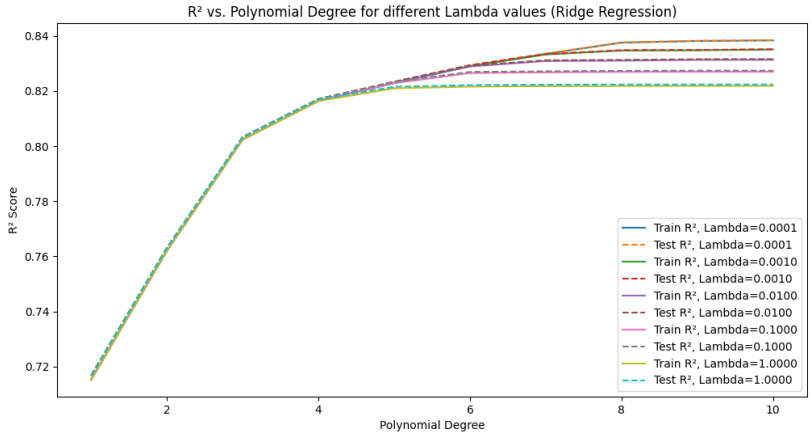

Figure 4.14: R^2 vs polynomial degree for different lambda values (Ridge regression)

4.2.3 Lasso regression

Figure 4.15 presents the MSE as a function of polynomial degree for different λ values in Lasso regression, while Figure 4.16 shows the corresponding R^2 values. Similar to the OLS and Ridge regression models, the MSE decreased and the R^2 increased with higher polynomial degrees, but this trend was only observed for λ values below 0.01. For λ values equal to or greater than 0.01, the MSE and R^2 remained constant, indicating limited model flexibility. The optimal polynomial degree was approximately 3, where the MSE was 0.0115 and the R^2 was 0.8016.

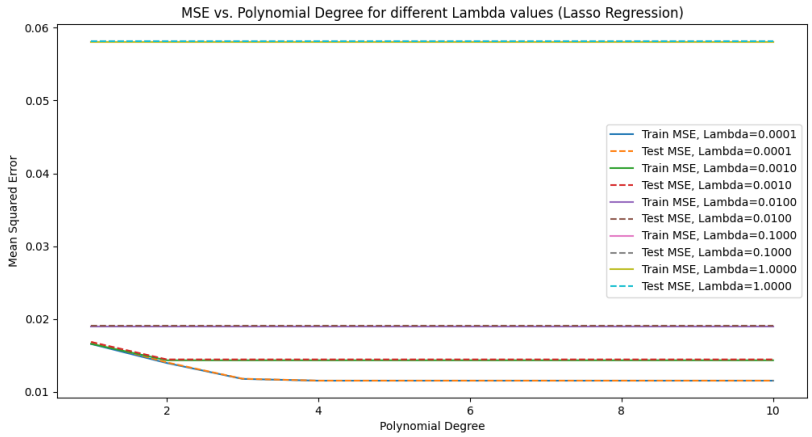

Figure 4.15: MSE vs polynomial degree for different lambda values (Lasso regression)

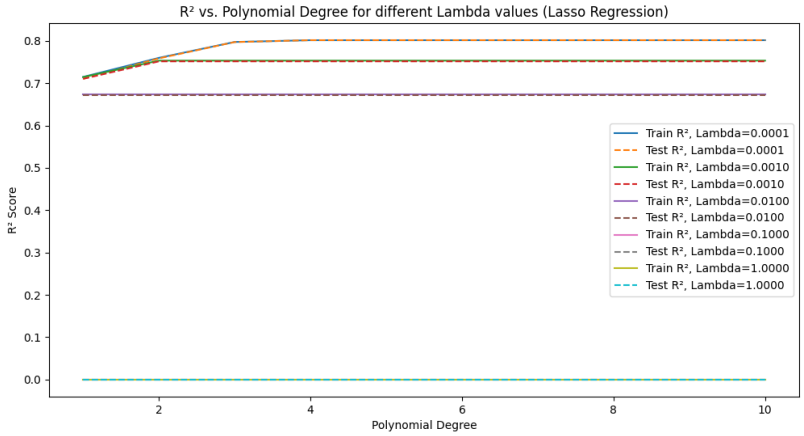

Figure 4.16: R^2 vs polynomial degree for different lambda values (Lasso regression)

The MSE and R^2 values for the OLS, Ridge, and Lasso regression models are presented in Table 4.2.

Table 4.2: The MSE and R^2 values for OLS, Ridge, and Lasso regression on both the train and test datasets.

Train/Test	Regression method	Polynomial degree	λ	MSE	R^2
Train	OLS	6	-	0.0100	0.8256
Test	OLS	6	-	0.0100	0.8274
Train	Ridge	8	0.0001	0.0098	0.8315
Test	Ridge	8	0.0001	0.0097	0.8333
Train	Lasso	3	0.0001	0.0115	0.8016
Test	Lasso	3	0.0001	0.0115	0.8017

4.2.4 Bias-Variance trade-off of OLS

Figure 4.17 shows the plot of error, bias, and variance versus polynomial degree for the OLS model. The error and bias decreased with increasing model complexity until the polynomial degree was around 19, where the error significantly increased together with the variance.

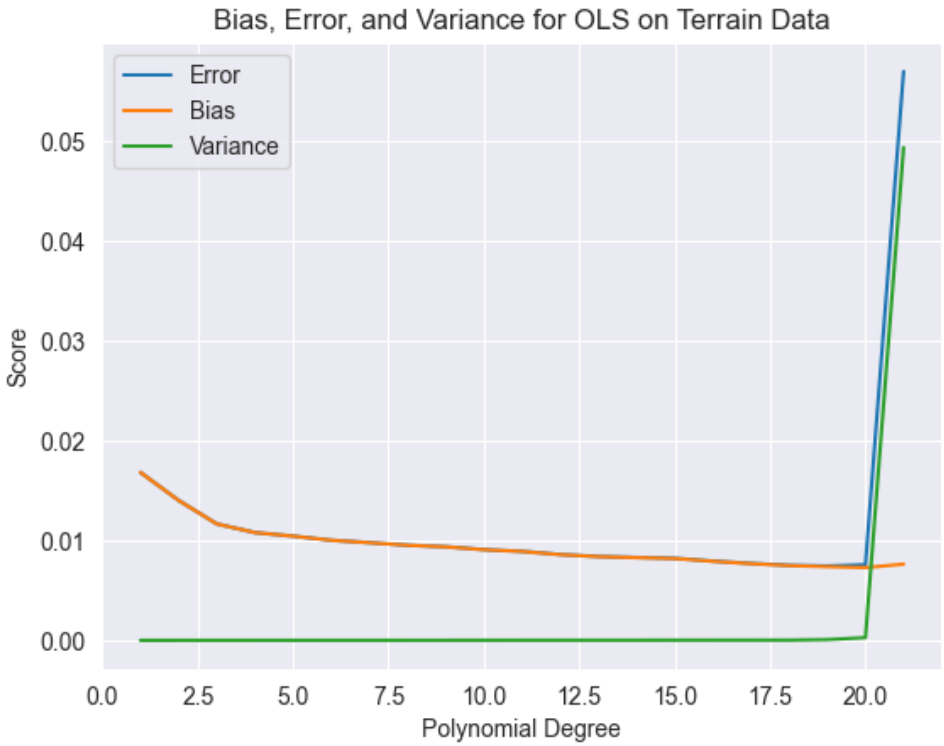

Figure 4.17: Error/Bias/Variance vs polynomial degree for OLS

4.2.5 *K*-fold cross-validation

Figure 4.18 gives the MSE versus polynomial degree with $k = 10$ for OLS, Ridge, and Lasso. The MSE of all the models decreased with increased polynomial degrees.

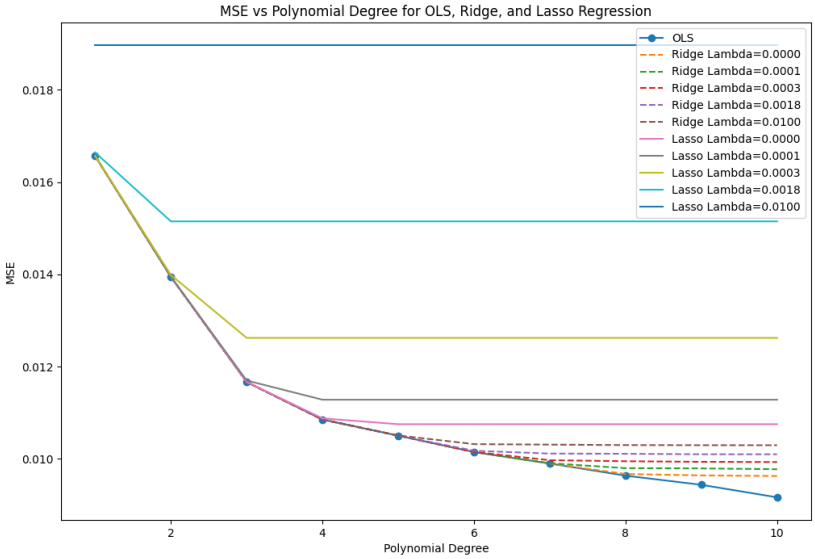

Figure 4.18: MSE vs polynomial degree with $k = 10$ for OLS, Ridge, and Lasso

5 Discussion

In this study, we evaluated the performance of OLS, Ridge, and Lasso regression models on both Franke and terrain datasets to determine the best regression approach for fitting and generalizing the data.

5.1 Franke Data

The OLS model achieved its optimal performance at a polynomial degree of 5, with an MSE of 0.0105 for the training set and 0.0153 for the test set which can be found in Table 4.1. The corresponding R^2 values were 0.8870 and 0.8395, respectively. These results indicate that the OLS model was able to fit the training data effectively and generalize relatively well to the test data, with only minimal signs of overfitting at this complexity level. However, as seen in Figure 4.3, increasing the polynomial degree beyond 5 led to overfitting, as the model began to capture noise rather than true underlying patterns.

The bias-variance analysis of the OLS model, depicted in Figure 4.8, further supports this conclusion. Initially, increasing the polynomial degree led to a reduction in total error, indicating an improved fit. However, beyond degree 6, the variance began to increase substantially, leading to an overall rise in error. This is a clear sign of overfitting, as the model's complexity began

to outweigh its ability to generalize. This behavior is consistent with the theory discussed in Section 2.2.

For Ridge regression, the best performance was observed with a regularization parameter of λ = 0.0001 and a polynomial degree of 5. The MSE values were 0.0106 for the training set and 0.0155 for the test set, with R^2 values equal to 0.8866 and 0.8372, respectively. These results indicate that Ridge regression performed comparably to OLS but provided a slight advantage due to its ability to mitigate overfitting through regularization. The regularization term in Ridge effectively constrained the model complexity, allowing it to retain generalization capability even at higher polynomial degrees.

The Lasso regression model also performed good, although not as good as OLS and Ridge. Its optimal performance was achieved at $\lambda = 0.0001$ and a polynomial degree of 3, with an MSE of 0.0149 for the training set and 0.0220 for the test set, and R^2 values of 0.8340 and 0.7684, respectively. Additionally, for λ values greater than 0.001, both MSE and R^2 remained constant, suggesting that Lasso suffered from excessive regularization, which limited its capacity to capture meaningful relationships in the data.

Cross-validation results, presented in Figure 4.9, further reinforced our findings. The cross-validated error for OLS began to increase at polynomial degree 5, similar to the trend observed in the bootstrap analysis in Figure 4.8, highlighting the onset of overfitting at this point. Meanwhile, the error for Ridge regression remained relatively stable, likely due to the influence of the regularization term, which controlled the model complexity effectively. Lasso was also quite stable, but the MSE values were not as good.

Overall, while OLS and Ridge regression models demonstrated similar performance in terms of MSE and R^2, Ridge regression showed a distinct advantage due to its ability to control overfitting more effectively. The regularization parameter in Ridge allowed the model to achieve a balance between fit quality and generalization capability, which was not observed in OLS.

5.2 Terrain Data

The OLS model demonstrated optimal performance at a polynomial degree of 6, with no distinguishable difference between test and train data, achieving an MSE of 0.0100 and an R^2 of 0.8256 (Fig. 4.11). The MSE and R^2 values can be found in Table 4.2. This implies that the model effectively captured the underlying patterns of the data without overfitting at this level of complexity. However, the bias-variance trade-off plot (Fig. 4.17) indicated a potential issue of overfitting emerging at much higher polynomial degrees (around 19 and beyond), as

the variance began to increase sharply, leading to an overall error rise. This discrepancy with Figure 4.12, which suggested overfitting from degree 6, points to a potential coding error or inconsistency that requires further examination.

Ridge regression, on the other hand, achieved the best results with a regularization parameter of $\lambda = 0.0001$ and a polynomial degree of 8, producing an MSE of 0.0098 for both training and test datasets and R^2 values equal to 0.8315. While these metrics were similar to those of the OLS model, Ridge had the advantage of added regularization, which helped mitigate overfitting at higher degrees of complexity.

The Lasso model showed relatively weaker performance compared to OLS and Ridge. Its optimal configuration was found at $\lambda = 0.0001$ and a polynomial degree of 3, yielding an MSE of 0.0115 and an R^2 value of 0.8016. Lasso's performance was generally inferior, with higher λ values resulting in excessive regularization that restricted the model's ability to capture essential patterns in the data. This outcome suggested that the Lasso model's sensitivity to the features was limited, which reduced its effectiveness relative to the other methods.

Cross-validation results, presented in Figure 4.18, did not reinforce our findings since we only plotted to polynomial degree 10. The cross-validated error for OLS continued to decrease past polynomial degree 6 (degree from Fig. 4.12) all the way to degree 10. If we had plotted to degree 20, we might have seen an increase in error for OLS again similar to the bias-variance trade-off plot (Fig. 4.17). Ridge, however, maintained its optimal performance at a polynomial degree of 8, consistent with the MSE plot for Ridge (Fig. 4.13). This consistency and lack of clear overfitting issues provide additional evidence that Ridge regression is the overall better fit for the terrain dataset.

Comparing the three models, Ridge regression emerged as the most suitable approach for the terrain data. Both OLS and Ridge demonstrated similar MSE and R^2 values at their respective optimal polynomial degrees, but Ridge had a clear advantage due to its regularization term. This allowed Ridge to maintain a better generalization capability and reduced the risk of overfitting, particularly at higher polynomial degrees where OLS showed a sharp increase in variance. Thus, Ridge regression at $\lambda = 0.0001$ and a polynomial degree of 8 is the recommended model for this dataset, achieving a favorable balance between fitting accuracy and generalization.

6 Conclusion

The goal of this report was to estimate the most suitable model comparing the performance of different linear regression methods (OLS, Ridge, and Lasso) on two types of data. In order

to get a better reliability of our results, we also studied and implemented resampling methods, in particular boostrapping and k-fold cross validation. Our results indicate that Ridge regression consistently provided the best balance between fitting accuracy and generalization for both datasets. Ridge's use of regularization allowed it to mitigate overfitting, particularly at higher polynomial degrees, where OLS demonstrated increased variance and overfitting issues. Lasso regression, however, exhibited reduced accuracy due to excessive regularization, which limited its ability to capture the complexity of the data. Overall, Ridge regression, with appropriate selection of the regularization parameter ($\lambda = 0.0001$), emerged as the most reliable model, achieving favorable results for both the synthetic Franke function and the real terrain dataset. To further assess model performance, especially in the case of potential overfitting, additional experiments could be conducted with higher polynomial degrees, particularly for cross-validation and bias-variance analysis. This would help clarify the discrepancies observed in the current results.

Bibliography

1. James G, Witten D, Hastie T and Tibshirani R. An Introduction to Statistical Learning: With applications in R. p. 22, 71, 101. New York: Springer, 2017
2. Hjorth-Jensen M. Week 36: Linear Regression and Statistical interpretations. 2024. Available from: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week36.ipynb
3. Hastie T, Tibshirani R and Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed. p. 11, 241-249. New York: Springer, 2016
4. Hjorth-Jensen M. Week 37: Statistical interpretations and Resampling Methods. 2024. Available from: https://github.com/CompPhysics/MachineLearning/blob/master/doc/LectureNotes/week37.ipynb

Appendix

I Paper and pencil

We can show that the expectation value of $\mathbf{y}$ for a given element i is $\mathrm{X}_{i*}\beta$:

$$\mathbb{E}[y_i] = \mathbb{E}[\sum_j x_{ij}\beta_j + \varepsilon_i] = \mathbb{E}[\sum_j x_{ij}\beta_j] + \mathbb{E}[\varepsilon_i] = \mathbb{E}[\sum_j X_{ij}\beta_j] \tag{28}$$

Here, $\mathbb{E}(\varepsilon_i) = 0$ because ε_i is a normally distributed error. Further, we assume that $\mathrm{X}_{ij}\beta_j$ are not stochastic variables. Then we get:

$$\mathbb{E}[\sum_j X_{ij}\beta_j] = X_{i*}\beta \tag{29}$$

We can also show that the variance of $\mathbf{y}$ of a given element i is σ^2:

$$\begin{aligned}\mathrm{Var}[y_i] &= \mathbb{E}[y_i^2] - \mathbb{E}[y_i]^2 = \mathbb{E}[(X_{i*}\beta + \varepsilon_i)^2] - (X_{i*}\beta)^2 \\ &= \mathbb{E}[(X_{i*}\beta)^2 + \varepsilon_i^2 + 2X_{i*}\beta\varepsilon_i] - (X_{i*}\beta)^2 \\ &= \mathbb{E}[(X_{i*}\beta)^2] + \mathbb{E}[\varepsilon_i^2] + \mathbb{E}[2X_{i*}\beta\varepsilon_i] - (X_{i*}\beta)^2 \\ &= \mathbb{E}[\varepsilon_i^2] + 2X_{i*}\beta\mathbb{E}[\varepsilon_i] = \mathbb{E}[\varepsilon_i^2] = \mathrm{Var}[\varepsilon_i] + \mathbb{E}[\varepsilon_i]^2 = \mathrm{Var}[\varepsilon_i] = \sigma^2\end{aligned} \tag{30}$$

Here we have again used that the expectation of ε_i equals zero, and that the variance $\mathrm{Var}[\varepsilon_i] = \mathbb{E}[\varepsilon_i^2]$ - $(\mathbb{E}[\varepsilon_i])^2 = \mathbb{E}[\varepsilon_i^2] = \sigma^2$.

We would also like to show that $\mathbb{E}[\hat{\beta}_{OLS}] = \beta$. The optimal parameter $\hat{\beta}$ for the ordinary least square is $\hat{\beta} = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{y}$. By putting this expression in $\mathbb{E}[\hat{\beta}_{OLS}]$ we have:

$$\mathbb{E}[\hat{\beta}_{OLS}] = \mathbb{E}[(X^TX)^{-1}X^Ty] = (X^TX)^{-1}X^T\mathbb{E}[y] = (X^TX)^{-1}X^TX\beta = \beta \tag{31}$$

We have assumed that X is non-stochastic and used what we showed in Equation 29 that $\mathbb{E}[y_i]$ = $\mathrm{X}_{i*}\beta$.

Lastly, we can show that the variance of $\hat{\beta}_{OLS}$ equals $\sigma^2(\mathrm{X}^T\mathrm{X})^{-1}$:

$$\begin{aligned}\mathrm{Var}[\hat{\beta}_{OLS}] &= \mathbb{E}[\hat{\beta}_{OLS}^2] - \mathbb{E}[\hat{\beta}_{OLS}]^2 = \mathbb{E}[([X^TX]^{-1}X^Ty)^2] - \beta^2 \\ &= \mathbb{E}[([X^TX]^{-1}X^Ty)([X^TX]^{-1}X^Ty)^T] - \beta^T\beta \\ &= \mathbb{E}[(X^TX)^{-1}X^Tyy^TX(X^TX)^{-1}] - \beta^T\beta\end{aligned} \tag{32}$$

$$
\begin{aligned}
&= (X^T X)^{-1} X^T \mathbb{E}[yy^T] X (X^T X)^{-1} - \beta^T \beta \\
&= (X^T X)^{-1} X^T \mathbb{E}[(X\beta + \varepsilon)(X^T \beta^T + \varepsilon^T)] X (X^T X)^{-1} - \beta^T \beta \\
&= (X^T X)^{-1} X^T \mathbb{E}[X\beta\beta^T X^T + \varepsilon\varepsilon^T + X\beta\varepsilon^T + \varepsilon X^T \beta^T] X (X^T X)^{-1} - \beta^T \beta \\
&= (X^T X)^{-1} X^T [(X\beta\beta^T X^T) + \mathbb{E}[\varepsilon\varepsilon^T] + X\beta\mathbb{E}[\varepsilon^T] + \mathbb{E}[\varepsilon] X^T \beta^T] X (X^T X)^{-1} - \beta^T \beta \\
&= (X^T X)^{-1} X^T [X\beta\beta^T X^T + \mathbb{E}[\varepsilon\varepsilon^T]] X (X^T X)^{-1} - \beta^T \beta \\
&= (X^T X)^{-1} X^T [X\beta\beta^T X^T + \sigma^2 I] X (X^T X)^{-1} - \beta^T \beta \\
&= (X^T X)^{-1} X^T X\beta\beta^T X^T X (X^T X)^{-1} + \sigma^2 (X^T X)^{-1} X^T X (X^T X)^{-1} - \beta^T \beta \\
&= \beta\beta^T + \sigma^2 (X^T X)^{-1} - \beta^T \beta \\
&= \sigma^2 (X^T X)^{-1}
\end{aligned}
$$

II 2.11 figure from Hastie et al.

Figure II.1 shows the MSE of the train and test dataset of OLS regression as a function of polynomial degrees for the Franke function. It shows that at low polynomial degrees, we have high bias and low variance, i.e. underfitting, and at high polynomial degrees we have low bias and high variance, i.e. overfitting. So we would benefit from staying between polynomial degrees 4 and 6.

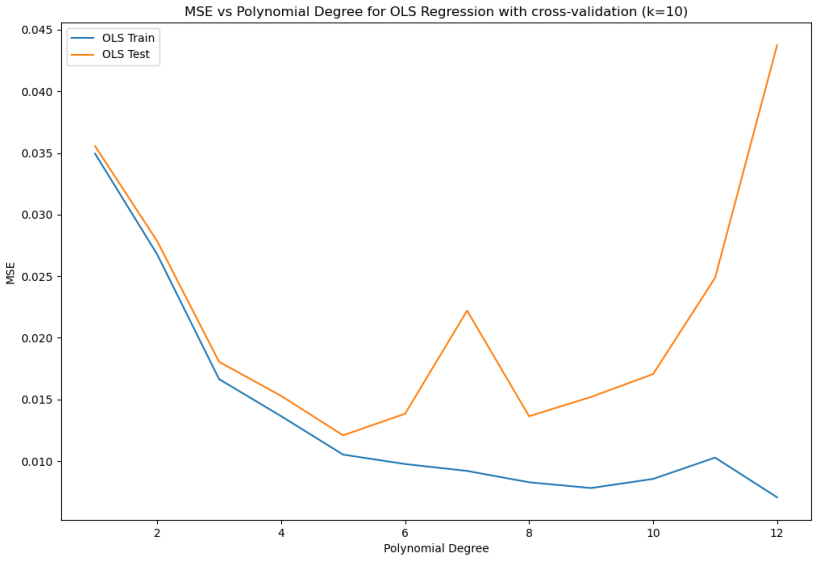

Figure II.1: Our version of Figure 2.11 found in The Elements of Statistical Learning by Hastie et al. [3]. The test and train error as a function of model complexity for the Franke function.