Regression analysis and resampling methods applied to Franke's Function and Topographical data

Femtehjell, Hoel, Otterlei and Steeneveldt

October 2023

UiO : University of Oslo

Contents

List of Figures

Abstract

In this project, we aim to employ various regression methods, including Ordinary Least Squares, Ridge, and Lasso, to analyze a set of terrain data and Franke's function. We explore the use of two-variable polynomials of varying orders and optimize hyperparameters for Ridge and Lasso. To assess the effectiveness of the models, we utilize resampling techniques such as non-parametric bootstrap and k-fold cross-validation. These techniques allow us to evaluate each model's performance by dividing the data into test and training subsets and training the model multiple times on the training subset while evaluating it on the testing subset. Although our ultimate goal is to identify the best model and evaluate its performance, we also explore and compare several regression techniques to gain valuable insights into their strengths and weaknesses. We found our best results using Lasso regression.

1 Introduction

Early in our education, we learn how to connect points by drawing a straight line through them. However, when such a line does not exists, how should one proceed? You may perhaps try to draw a wave through the points, or maybe draw a straight line through which is the 'closest' to all the points. Later we learn how to utilize tools to automatically create these lines, which are as mathematically close to the points as possible, but how does this work? How does the computer decide which type of line to draw, and how does it calculate how 'far' the line is from the points?

Suppose all of your points lie along a line, except one which is far away. Should the best line be the one which passes straight through most of the points, or should we penalize it for being far from one of the points. Maybe we should draw a line which goes straight from point to point, deviating far to catch the deviant. What if we know that we only have a handful of all the points, and that we want to place the line such that future points will also lay close to it. Does your answer change then?

We find the answers to these questions within the art of regression. One common way to calculate the deviation of points from the line, is to calculate the mean sum of the square of the errors (MSE), a method often attributed to Galileo Galilei [Hald, 1986]. This method is what we find at the core of our first regression method, namely ordinary least squares (OLS), which tries to minimize this metric. This however, may cause problems where our model is too closely fitted our line, such that new points lay far away. In order to alleviate some of these issues, there are more complex methods such as Ridge and Lasso regression, which build on OLS, which add a penalty in order to make sure that the most relevant aspects of the line are prioritized.

In the last decade the concept of machine learning has become wildly popular. It is used to some degree in nearly all aspect of society and keeps growing in popularity. Since regression lies at the core of many machine learning methods,

studying how these different methods mentioned above behaves becomes key for moving on to more complex methods in machine learning.

We will apply these methods to the Franke function, a method commonly used to test interpolation problems [Franke, 1982]. This function consists of two peaks, which is ideal as we will later attempt to analyse real world terrain data.

2 Theory

2.1 Ordinary Least Squares

In order to have a basis to work from, we assume that there exists a continuous function $f : X \to Y$ such that our data points $\boldsymbol{y} \in Y$ can be described as

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon} \sim N(0, \sigma^2)$. We are then looking for an approximation of f which gives $\tilde{\boldsymbol{y}}$ such that the MSE $\frac{1}{n}(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2$ is minimized. In order to approximate f, we consider n samples of $\boldsymbol{y}$ and assume that there are p characteristics which define each of the samples, such that $y_i = f(\boldsymbol{x}_i) + \varepsilon_i$ where $\boldsymbol{x}_i = \big[x_{i,0}, x_{i,1}, \ldots, x_{i,p-1}\big]$.

We gather this information in a matrix $\mathbf{X}$, called the design matrix, such that

$$\mathbf{X} = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,p-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & x_{n,1} & \cdots & x_{n,p-1} \end{bmatrix}.$$

We are seeking to find a causal relationship between $\boldsymbol{y}$ and $\boldsymbol{x}$, and as we have no knowledge of what type of function f is, we assume there is a linear relationship such that $y_i = \boldsymbol{x}_i \cdot \boldsymbol{\beta} + \varepsilon_i$, for some weights β_i. This can be written as

$$\boldsymbol{y} = \begin{bmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,p-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,0} & x_{n,1} & \cdots & x_{n,p-1} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{p-1} \end{bmatrix} + \begin{bmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where $f(\boldsymbol{x})$ is now equal to $\mathbf{X}\boldsymbol{\beta}$. Our approximation $\tilde{\boldsymbol{y}}$ then becomes $\tilde{\boldsymbol{y}} = \mathbf{X}\boldsymbol{\beta}$. We call $\boldsymbol{x}_i$ the explanatory variables, $\boldsymbol{\beta}$ the regression parameter, and $\boldsymbol{y}$ the response variable.

Our goal is now to find $\hat{\boldsymbol{\beta}}$ such that $(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2$ is minimized, i.e.,

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\arg\min} \frac{1}{n} (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^2 = \underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\arg\min} \frac{1}{n} (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}).$$

As then

$$(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}) = \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\boldsymbol{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

is a non-negative quadratic equation, the minimum must exist. We may therefore find it by equating the partial derivatives with respect to the p components of $\boldsymbol{\beta}$ with zero. In finding the derivative, we can ignore the scaling factor $\frac{1}{n}$.

$$\frac{\partial}{\partial \boldsymbol{\beta}} \left(\boldsymbol{y}^T \boldsymbol{y} - 2\boldsymbol{\beta}^T \mathbf{X}^T \boldsymbol{y} + \boldsymbol{\beta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\beta} \right) = 0$$
$$-2\mathbf{X}^T \boldsymbol{y} + 2\mathbf{X}^T \boldsymbol{X} \boldsymbol{\beta} = 0$$
$$2\mathbf{X}^T \boldsymbol{X} \boldsymbol{\beta} = 2\mathbf{X}^T \boldsymbol{y}$$
$$\boldsymbol{\beta} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \boldsymbol{y}$$

When the matrix $\mathbf{X}^T\mathbf{X}$ is invertible, this solution $\hat{\boldsymbol{\beta}}$ which minimizes the MSE can be found precisely. Often, especially when working numerically, the matrix is singular, so we instead apply the Moore-Penrose generalised inverse defined from the Singular Value Decomposition, often just called the pseudoinverse [Süli and Mayers, 2003, p. 74–82].

2.1.1 Expectation and variance of OLS

The expectation value of a single element y_i of $\boldsymbol{y}$ is found as

$$\mathbb{E}[y_i] = \mathbb{E}\left[\mathbf{X}_{i,*}\boldsymbol{\beta} + \varepsilon_i\right] = \mathbf{X}_{i,*}\boldsymbol{\beta} + \mathbb{E}\left[\varepsilon_i\right] = \mathbf{X}_{i,*}\boldsymbol{\beta},$$

as only ε_i is a random variable with expectation 0, while $\mathbf{X}_{i,*}\boldsymbol{\beta}$ is a non-random scalar. The variance of y_i is then found as

$$\begin{aligned}
\mathrm{Var}(y_i) &= \mathbb{E}[y_i^2] - \mathbb{E}[y_i]^2 \\
&= \mathbb{E}\left[(\mathbf{X}_{i,*}\boldsymbol{\beta})^2 + 2\varepsilon_i \mathbf{X}_{i,*}\boldsymbol{\beta} + \varepsilon_i^2\right] - (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 \\
&= (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 + 2\mathbb{E}[\varepsilon_i]\mathbf{X}_{i,*}\boldsymbol{\beta} + \mathbb{E}[\varepsilon_i^2] - (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 \\
&= \mathbb{E}[\varepsilon_i^2] = \mathbb{E}[\varepsilon_i^2] - \mathbb{E}[\varepsilon_i]^2 = \mathrm{Var}(\varepsilon_i) \\
&= \sigma^2.
\end{aligned}$$

Thus, we can gather that $y_i \sim N\left(\mathbf{X}_{i,*}\boldsymbol{\beta}, \sigma^2\right)$, and that $\mathbb{E}[\boldsymbol{y}] = \mathbf{X}\boldsymbol{\beta}$. In order to analyze how this effects our solutions of $\hat{\boldsymbol{\beta}}$, we performs the same analysis as with y_i.

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

Further, we find the variance of $\hat{\boldsymbol{\beta}}$, noting that $\mathbf{X}^T\mathbf{X}$ is symmetric, as

$$\begin{aligned}
\mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}[\hat{\boldsymbol{\beta}}^2] - \mathbb{E}[\hat{\boldsymbol{\beta}}]^2 \\
&= \mathbb{E}\left[\left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y}\right)\left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y}\right)^T\right] - \boldsymbol{\beta}\boldsymbol{\beta}^T \\
&= \mathbb{E}\left[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y}\boldsymbol{y}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\right] - \boldsymbol{\beta}\boldsymbol{\beta}^T \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}\left[\boldsymbol{y}^2\right]\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T
\end{aligned}$$

Then as

$$\begin{aligned}
\mathrm{Var}(\boldsymbol{y}) &= \mathbb{E}\left[\boldsymbol{y}^2\right] - \mathbb{E}\left[\boldsymbol{y}\right]^2 \\
\mathbb{E}\left[\boldsymbol{y}^2\right] &= \mathrm{Var}(\boldsymbol{y}) + \mathbb{E}\left[\boldsymbol{y}\right]^2 \\
\mathbb{E}\left[\boldsymbol{y}^2\right] &= \sigma^2 + \mathbf{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\mathbf{X}^T,
\end{aligned}$$

we get that

$$\begin{aligned}
\mathrm{Var}(\hat{\boldsymbol{\beta}}) &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\left(\sigma^2 + \mathbf{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\mathbf{X}^T\right)\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} + \boldsymbol{\beta}\boldsymbol{\beta}^T - \boldsymbol{\beta}\boldsymbol{\beta}^T \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}
\end{aligned}$$

2.2 Ridge

When conducting Ordinary Least Squares (OLS) regression, overfitting can be a challenge. Overfitting occurs when our model closely conforms to the training data, not only capturing the genuine underlying pattern but also incorporating the random fluctuations present in our data. This can result in a model that exhibits exceptional performance on the training data it has seen but performs poorly when tasked with predicting unseen data.

Overfitting is influenced by two closely related factors: the density of our training data and the complexity of our model. Ideally, with an infinitely dense training dataset, we could employ an arbitrarily complex model. Acquiring such a dataset is in practice often impossible. Our model must therefore deliver reliable performance even when trained on a limited number of data points.

Ridge regression is a method which aims to address this by taking into account the size of the values in the regression parameter $\boldsymbol{\beta}$. The cost function is now defined as

$$\begin{aligned}
C\left(\mathbf{X}, \boldsymbol{\beta}\right) &= \frac{1}{n}\|\boldsymbol{y} - \tilde{\boldsymbol{y}}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2 \\
&= \frac{1}{n}\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\right)^2 + \lambda\boldsymbol{\beta}^2
\end{aligned}$$

The right addend ($\lambda\|\boldsymbol{\beta}\|_2^2$) is called the regularization term, and we require that $\|\boldsymbol{\beta}\|_2^2 \leq t$ for some finite constant $t > 0$. It penalizes large regression

coefficients, discouraging the model from assigning excessive importance to any single feature. This skews our predictions closer to zero. Resulting in higher bias, but lower variance.

The strength of the regularization is determined by λ. When $\lambda = 0$, we recover OLS regression. By increasing the value of the hyperparameter λ, we enforce stronger penalization on large coefficient values, effectively skewing our predictions more towards zero. The optimal value of λ can be determined through resampling techniques.

Given a λ our optimal regression parameter $\hat{\boldsymbol{\beta}}$ minimizes the cost function. That is

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\arg\min} \frac{1}{n} \left(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\right)^2 + \lambda\boldsymbol{\beta}^2$$

The right hand side multiplies out to

$$\frac{1}{n}\left(\boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\boldsymbol{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}\right) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta},$$

which again is a non-negative quadratic, meaning the minimum exists. We proceed similarly as with OLS, by taking the partial derivatives with respect to each component of $\boldsymbol{\beta}$ and equating it with zero. As $\frac{1}{n}$ is just a scaling factor, we can safely ignore it while finding the derivative.

$$\begin{aligned}
\frac{\partial}{\partial\boldsymbol{\beta}}\left(\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\right)^2 + \lambda\boldsymbol{\beta}^2\right) &= 0 \\
-2\mathbf{X}^T\boldsymbol{y} + 2\mathbf{X}^T\boldsymbol{X}\boldsymbol{\beta} + 2\lambda\boldsymbol{\beta} &= 0 \\
\mathbf{X}^T\boldsymbol{X}\boldsymbol{\beta} + \lambda\boldsymbol{\beta} &= \mathbf{X}^T\boldsymbol{y} \\
\left(\mathbf{X}^T\mathbf{X} + \boldsymbol{I}_p\lambda\right)\boldsymbol{\beta} &= \mathbf{X}^T\boldsymbol{y} \\
\boldsymbol{\beta} &= \left(\mathbf{X}^T\mathbf{X} + \boldsymbol{I}_p\lambda\right)^{-1}\mathbf{X}^T\boldsymbol{y}
\end{aligned}$$

Again, we apply the pseudoinverse when finding $(\mathbf{X}^T\mathbf{X} + \lambda\boldsymbol{I})^{-1}$. This gives us that the optimal parameter $\hat{\boldsymbol{\beta}}$ for ridge regression is given by

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

2.3 Lasso

LASSO is an abbreviation of Least Absolute Shrinkage and Selection Operator, and performs both regularization as in Ridge regression, and predictor selection.

Here, the cost function is defined as

$$\begin{aligned}
C\left(\mathbf{X}, \boldsymbol{\beta}\right) &= \frac{1}{n}\|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 \\
&= \frac{1}{n}\left(\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\right)^2 + \lambda\sum_i |\beta_i|.
\end{aligned}$$

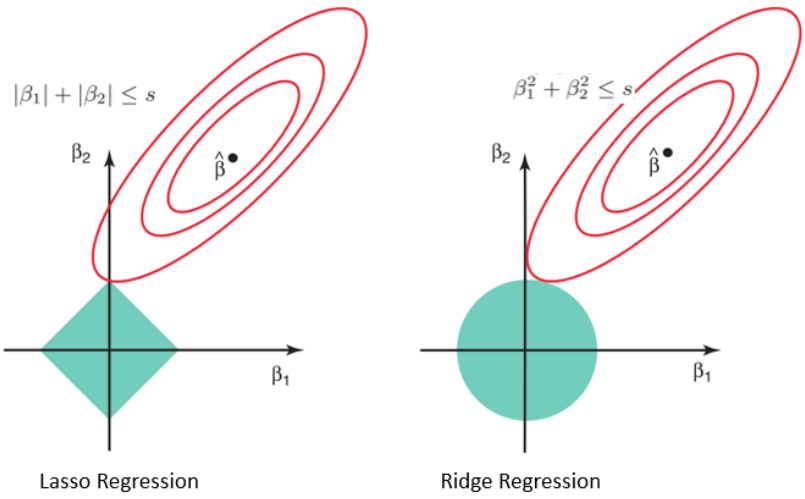

Figure 1: Lasso vs Ridge regression for constrained values. (Figure 3.11 in [Hastie et al., 2009])

To see why predictor selection happens with Lasso regression it's helpful to have a look at a simple model with two features and two corresponding β values

Here, $\hat{\boldsymbol{\beta}}$ is the OLS solution, along with the contour lines for other values of $\boldsymbol{\beta}$. As we relax the constraint t (s in the figure), we grow the "diamond" generated by the constrained ℓ_1 norm until it meets with the target ellipse, and the corners of the diamond are more likely to intersect before the edges. This is especially the case when working higher dimensions, as the corners stick out further [Murphy, 2012, p. 432–436]. This is not the case with the constrained ℓ_2 norm, as it has no "corners" along the axes.

Due to the nature of the ℓ_1 norm, there exists no neat analytical solution, as absolute values do not perform nicely under derivation. We get that

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\beta}} \left(\frac{1}{n} (\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta})^2 + \lambda \sum_i |\beta_i| \right) &= 0 \\
-2\mathbf{X}^T\boldsymbol{y} + 2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + \lambda \operatorname{sgn}(\boldsymbol{\beta}) &= 0 \\
2\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} + \lambda \operatorname{sgn}(\boldsymbol{\beta}) &= -2\mathbf{X}^T\boldsymbol{y},
\end{aligned}$$

which we've yet to find a closed, generalised, form of.

2.4 Scaling

Scaling data is an essential concept in regression modeling, particularly in regularized regression. The process involves adjusting the values of numeric features

in your dataset to a common scale, ensuring each feature has comparable magnitudes. This standardization or normalization of the features can prove highly beneficial in several ways.

One of the primary advantages of data scaling is in the realm of computational performance. As there exists no analytical solution for Lasso regression, we instead approximate it numerically. This is done through coordinate decent, which converges faster when the features are scaled.

Another important advantage of scaling is the ease it brings in model interpretation. When the ranges of different features vary widely, it's hard to compare the weights that a model assigns to individual features. However, with scaling, the weights can be interpreted on a comparable basis, providing meaningful insights [James et al., 2021, p. 237].

To see how scale impacts the accuracy of regularized regression let $\hat{\boldsymbol{\beta}}$ denote the OLS solution to some design matrix X. Then multiply column j in our design matrix by 1000. The value of $\hat{\beta}_j$ to our new OLS problem will have shrunk by the same factor of 1000. In these two problems the relationship between our response variable ($\boldsymbol{y}$) and explanatory variable (X_j) is the same, but regularized regression treat them very differently. As in the first case the penalty of including our explanatory variable is 1000 times larger [James et al., 2021, p. 237]. Thus the importance of features with a small variance, but high predictive power will be underestimated, and features with large variance, but small predictive power will be overestimated.

Scaling is performed through the formula

$$\tilde{x}_{i,j} = \frac{x_{i,j}}{\sqrt{\frac{1}{n}\sum_{i=0}^{n-1}\left(x_{i,j} - \bar{x}_j\right)^2}},$$

where

$$\bar{x}_j = \frac{1}{n}\sum_{i=0}^{n-1} x_{i,j}$$

is the mean value of the column. Thus, all of the scaled predictors will have a standard deviation of one [James et al., 2021, p. 237].

2.5 Bias-Variance tradeoff

In order to measure our models, we split our dataset into two difference sets. One for training, and one for testing. The variance of a model refers to the amount our approximation $\tilde{\boldsymbol{y}}$ would change for varying sets of training data, compared to the true value of $\boldsymbol{y}$ [James et al., 2021, p. 34]. Ideally, we would not want our estimate of $\tilde{\boldsymbol{y}}$ to change too much based on our sample of training data.

The bias of a model refers to how our model handles the approximations necessary to work with a physical problem. Errors are generated when working with an idealised version of a problem, as we may be unable to capture the complexity present. A high bias indicates that our method is oversimplifying

our data, in effect missing relevant features. This is called underfitting, where the model is too simple to capture patterns in the data.

The expected mean squared error of our model indicates how well we can expect the model to perform. It is defined as $\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right]$ [James et al., 2021, p. 34]. As we assume that $\boldsymbol{y}=f(\boldsymbol{x})+\boldsymbol{\varepsilon}$, we get that

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

Where we used that $\boldsymbol{\varepsilon}\sim N(0,\sigma^2)$, and $\mathrm{Var}[\boldsymbol{\varepsilon}]=\mathbb{E}\left[\boldsymbol{\varepsilon}^2\right]+\mathbb{E}[\boldsymbol{\varepsilon}]^2$, in the final step. Furthermore,

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

As $\mathbb{E}\left[\mathbb{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}}\right]=\mathbb{E}[\tilde{\boldsymbol{y}}]-\mathbb{E}[\tilde{\boldsymbol{y}}]=0$

$$\begin{aligned}&= \mathbb{E}\left[(f(\boldsymbol{x})-\mathbb{E}[\tilde{\boldsymbol{y}}])^2\right]+\mathbb{E}\left[(\tilde{\boldsymbol{y}}-\mathbb{E}[\tilde{\boldsymbol{y}}])^2\right] \\ &= \mathrm{Bias}\left[\tilde{\boldsymbol{y}}\right]+\mathrm{Var}\left[\tilde{\boldsymbol{y}}\right]\end{aligned}$$

Combining this with our previous result gives us finally that

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

The σ^2 term is called the irreducible error [Hastie et al., 2009, p. 223], and cannot be improved by our models. Ideally, we want to choose a method such that we have both low variance and low bias. However, typically as the complexity of our model increases, the variance increases. Conversely, the same applies in the other direction. As the complexity is decreased, we see a higher bias and a lower variance. This is called the Bias-Variance tradeoff [Hastie et al., 2009, p. 223].

2.6 Resampling

Resampling methods allow us to repeatedly draw random samples from our training data, in order to learn more about our model. This allows us to better estimate the error and variance of the model, granting us information we would not have had, should we have trained on the entirety of our data at once. The two most commonly used methods of resampling is cross-validation and bootstrapping [James et al., 2021, p. 197].

2.6.1 Cross-validation

Cross-validation consists of splitting our initial dataset into multiple sets. To motivate this, we first consider the validation set approach. This consists of splitting our data into one training set, and one for testing or *validation*. This allows us to estimate how well our model will perform after training on future data. Note that splitting our data can cause the estimated test error rate to vary widely, as we split our data randomly. It can also cause us to overestimate our test error, as models tend to perform better when trained on more data, and we are in this case only training on a subset of the total data we have available [James et al., 2021, p. 198–200].

When cross-validating, we attempt to alleviate some of these issues. One method is called Leave one out cross-validation (LOOCV), were we for each data point train the model on the remaining values, and then calculate the test error with the data point we left out. We then take the average of all of these values, giving us the LOOCV estimate. This is for most models quite computationally expensive, especially when we have a lot of data.

The upside of this method is that the mean squared error estimate were we excluded the observation (x_i, y_i), where $\text{MSE}_i = (y_i - \tilde{y}_i)^2$, is approximately unbiased [James et al., 2021, p. 201]. However, we expect a high variance as the training sets are so similar [Hastie et al., 2009, p. 242].

A more sophisticated approach which alleviates some of the previous problems, is called K-Fold cross-validation. In this method, we split our data into k roughly equal parts. We then reserve each fold as a validation set, training our model on the remaining $k-1$ folds. This has the benefit of being less computationally expensive, as we are only training our model k times, while also having a lower variance. If our folds contain too few points, our model might suffer from high bias. We therefore typically choose either $k = 5$ or $k = 10$ as a compromise [Hastie et al., 2009, p. 243].

Our estimated error for predicting unseen data is then

$$\text{Err}_{\text{CV}} = \frac{1}{K}\sum_{i=1}^{K} MSE(\boldsymbol{y_i}, \boldsymbol{f}_{\kappa(i)}(\boldsymbol{x_i})) = \frac{1}{K}\sum_{i=1}^{K} \text{FoldErr}_{\text{i}}$$

The sample variance of our model cost is given by:

$$\underset{\text{CV}}{\text{Var}} = \frac{1}{K-1}\sum_{i=1}^{K}(\text{FoldErr}_{\text{i}} - \text{Err}_{\text{CV}})^2$$

2.6.2 Bootstrapping

Instead of cross-validation we could use the bootstrap to asses our model. The bootstrap resampling technique allows us to estimate the sampling distribution of any statistic. If our training data is $\mathbf{X} = (x_1, x_2....x_N)$ we draw N samples with replacement from $\mathbf{X}$ uniformly B times creating $\boldsymbol{Z}_1, \boldsymbol{Z}_2....\boldsymbol{Z}_B$ new training sets. The error of prediction of unseen data can then be estimated by

$$\text{Err}_{\text{boot}_1} = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{N}\sum_{i=0}^{N-1}(y_i - f_b(x_i))^2$$

Where $f_b(x_i)$ is our prediction for x_i by the model fit on $\boldsymbol{Z}_b$. Unfortunately, this is not a particularly good estimator because bootstrap samples used to produce $f_b(x_i)$ may have contained x_i [Hastie et al., 2009, p. 270]. To rectify this problem, the leave-one-out bootstrap estimator offers an improvement by emulating cross-validation.

$$\text{Err}_{\text{boot}_2} = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{|C^{-i}|}\sum_{i\in C^{-i}}(y_i - f_b(x_i))^2$$

Where C^{-i} is the set of indices of bootstrap sample b that do not contain observation i. However this estimate is upward biased. To combat this we can use that the average number of distinct observations in each bootstrap sample is about 0.632N ([Hastie et al., 2009, p. 270]) which gives the .632 bootstrap

$$\text{Err}_{\text{boot}_{.632}} = .368\bar{\text{err}} + .632\text{Err}_{\text{boot}_2}$$

Where

$$\bar{\text{err}} = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - f(x_i))a$$

That is the training error of our model trained on all the data. Yet, as our degree of overfitting increases the more this estimate becomes optimistically biased, since $\bar{\text{err}}$ approaches 0. This leads us to the .632+ estimator

$$\text{Err}_{\text{boot}_{.632+}} = (1-w)\bar{\text{err}} + w\text{Err}_{\text{boot}_2}$$
$$w = \frac{0.632}{1 - 0.368R} \qquad R = \frac{\text{Err}_{\text{boot}_2} - \bar{\text{err}}}{\gamma - \bar{\text{err}}}$$

Where γ is the no-information error rate. That is the error of all combinations of input-output on a model trained on all our training data

$$\gamma = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(y_i - f(x_j))^2$$

The .632+ bootstrap estimate is most useful when the sample size is small, or when the data is imbalanced or noisy and is rarely used for large sample sizes, as it gets very computationally expensive [Hastie et al., 2009, p. 271].

2.7 Model assessment and selection

The goal of our project is to identify the best model that accurately predicts the target variable on unseen data. Two commonly used metrics for measuring the accuracy of a model are the Mean Squared Error (MSE) and the R^2 score.

MSE is defined as the average of the squared difference between the predicted and actual values for all data points in the testing set, as shown below:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and N is the total number of data points.

The R^2 score, also known as the coefficient of determination, measures how well the model fits the data relative to the baseline model. It is calculated as:

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

where y_i is the actual value, $\hat{y}_i$ is the predicted value, $\bar{y}$ is the mean of the actual values, and n is the total number of data points. The numerator represents the residual sum of squares, and the denominator represents the total sum of squares. Essentially, R^2 measures the proportion of the variance in the target variable that can be explained by the model, with higher scores indicating a better fit.

In our project, we primarily rely on the MSE metric to compare and evaluate different models. While both metrics are commonly used to evaluate models, neither one is inherently better than the other. The choice of which metric to use depends on the project's specific requirements and goals.

When attempting to minimize the MSE of our predictions on unseen data it is tempting to simply perform a train-test split and see which models produces the lowest MSE on the test data. This could be fine for model selection, that is picking the best model. However, this will undoubtedly lead to some fitting of the test data, giving us an overly optimistic model assessment.

Instead we will utilize our resampling techniques for tuning and selecting the best model then train the model on all of our training data, before finally evaluating it's performance on truly unseen data to estimate how well it performs.

3 Method

3.1 Data sets

In this project we focus on the approximation of two datasets using 2D polynomial regression methods: OLS, Ridge, and LASSO regression. To validate the effectiveness of our techniques, we first utilize a self-generated dataset known as the Franke Function, which is a well-established smooth function. Following this validation, we proceed to analyze terrain data from a specific location in Norway, collected by NASA's Shuttle Radar Topography Mission (SRTM).

3.2 Data Holdout

In this project, we have chosen to use a soft 2-way holdout approach for evaluating the performance of our different models. In this approach, we do not perform a strict holdout of our test data, but use it instead to evaluate the performance of our models. It's also worth noting that we change the split when performing cross-validation. While this approach works well for ranking different models, it is important to recognize that it may not provide an accurate estimate of the model's performance on truly unseen data.

3.3 Two-dimensional polynomial regression

Franke's function is given by

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

for $x, y \in [0,1]$, and consists of two gaussian peaks and is thus apt for our use in estimating topographical data.

As $f : [0,1] \times [0,1] \to \mathbb{R}$ is a continuous function, and the set of all polynomials $\mathcal{P}$ is dense in $C([0,1] \times [0,1], \mathbb{R})$, we know by the Stone-Weierstrass Theorem that there exists a sequence of polynomials $\{p_n\}$ converging uniformly to f [Lindstrøm, 2017, p. 116–129]. In our case, this means that it is sensible to try to construct a polynomial which estimates our function f. As a polynomial function $p : \mathbb{R}^2 \to \mathbb{R}$ is a function

$$p(x,y) = \sum_{\substack{0 \le n \le N \\ 0 \le m \le M}} c_{n,m} x^n y^m,$$

we construct our explanatory variables $\boldsymbol{x}_i$ of all combinations $x_i^n y_i^m$ of $n, m \in \mathbb{N}$ such that $0 \le n + m \le N$ for some polynomial degree N. The regression parameter $\boldsymbol{\beta}$ then corresponds to the weights $c_{n,m}$.

The number of parameters for a polynomial degree of N is then the sum of the combination for how many ways we can write $n + m = d$, for $0 \leq n, m \leq d$, $d = 0, 1, \ldots, N$. We get

$$\#\ \boldsymbol{\beta}\text{ parameters} = \sum_{d=0}^{N}\sum_{n=0}^{d} 1 = \sum_{d=0}^{N} d + 1 = \frac{(N+1)(N+2)}{2},$$

meaning that the number of parameters for our model increases quadratically as we increase the polynomial degree.

In order to generate our data set, we use `numpy.linspace` in order to get N evenly spaced values between 0 and 1. This is then used to generate a mesh grid of values, which is shuffled whenever relevant. We added our error term $\varepsilon \sim N(0, 1)$ to each of our data points using `randn` from `numpy.random` in order to simulate real world data, fixing a seed to get reproducible results.

3.4 Scaling

As our values of x and y, for both Franke's function and the terrain data, lie in $[0, 1]$, our explanatory variables might have vastly different scales. Suppose we are using polynomials of degree up to 5. If our value of x_i lies even in the middle of our interval, i.e. $x_i = 0.5$, then $x_i^5 = 0.03125$. This does not affect OLS regression, however this will cause unwanted bias for regularized regression. We thus have to scale in order to get good results from ridge and lasso regression. To achieve this, we use `StandardScaling` from `sklearn.preproccesing`.

3.5 Creating our design matrix

In order to set up our design matrix with the polynomial explanatory variables, we initialize our matrix with the number of rows equal to the number of data points and length equal to the number of $\boldsymbol{\beta}$ parameters we calculated earlier. The code looks as following. We calculate each combination of $x^n y^m$ in much the same fashion we used to count the number of parameters β_i.

3.6 Ordinary least squares

We found our analytical solution $\hat{\boldsymbol{\beta}}$ for OLS using `pinv` from `numpy.linalg` in order to calculate the pseudo-inverse mentioned in the theory section, as we cannot guarantee that $\mathbf{X}^T\mathbf{X}$ is non-singular.

3.7 Ridge

The code for calculating $\hat{\boldsymbol{\beta}}_{\text{Ridge}}$ is almost identical as that for OLS. The difference is that the function now takes in a λ parameter, and we use $\mathbf{X}^T\mathbf{X} + \lambda \boldsymbol{I}_p$ inside `pinv`.

3.8 Lasso

As there exists no analytical solution for Lasso, we opted for using the built-in `Lasso` regression from `sklearn.linear_model`.

3.9 Resampling

3.9.1 Cross Validation

In this project, we have utilized both our own method and the `cross_val_score` function from the `sklearn.model_selection` module to perform Cross Validation.

The `cross_val_score` function from `sklearn.model_selection` automatically performs the necessary steps to split the data into folds and evaluates the model on each fold using a specified scoring metric, we choose MSE.

In addition to using `cross_val_score`, we have also implemented our own method for splitting the data into folds using the our own `KFold` class. We have sorted our data for plotting purposes and thus we need to set the `shuffle` parameter equal to `True` in our method as well as `cross_val_score` to ensure a randomized order of the data when creating the folds.

3.9.2 Bootstrapping

For our project, we performed Bootstrapping on the training set of our train-test split. This means that we generated multiple bootstrap samples by randomly selecting indices from the training data with replacement. We then trained our regression models on each bootstrap sample and evaluated their performance on the test set.

While this Bootstrapping technique was not explicitly specified in the theory section, it is the method presented to us in the lectures, and we thus choose to stick with it.

It's worth noting that our train-test split (4/5, 1/5) is somewhat akin to performing Cross Validation with 5 folds, as each fold represents approximately 1/5 of the data. This allows for a comparison of the two techniques, which can be seen in the results section below.

4 Results

4.1 Results for Franke's function

We started with asserting how well our model performs when trained on Franke's function. Our data looked as following, after generating our values in x and y direction from the uniform distribution between 0 and 1. We then structured our data as a mesh grid. The random noise was added with $0.2 \cdot \varepsilon_{i,j} \sim N(0, 1)$.

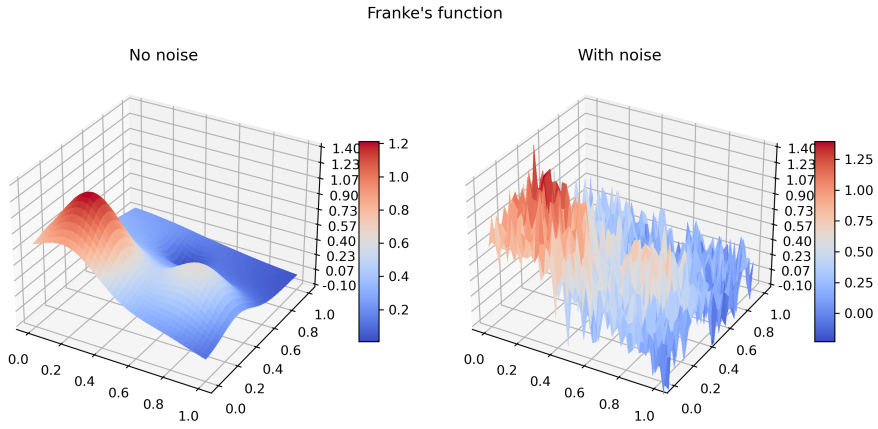

Figure 2: Franke's function with and without noise.

4.1.1 Results from Ordinary least squares

The same data was used for each plot, unless otherwise specified.

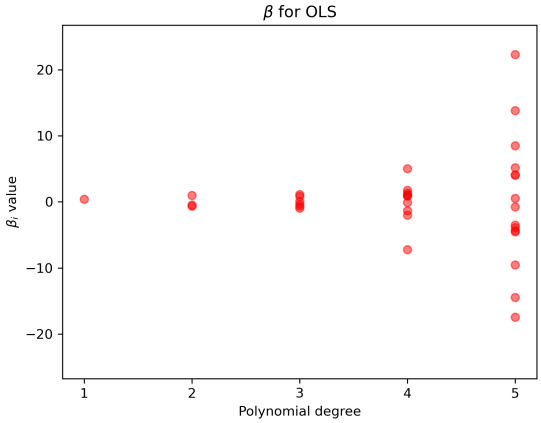

Figure 3: Values of $\boldsymbol{\beta}$ from ordinary least squares for polynomial degree up to 5.

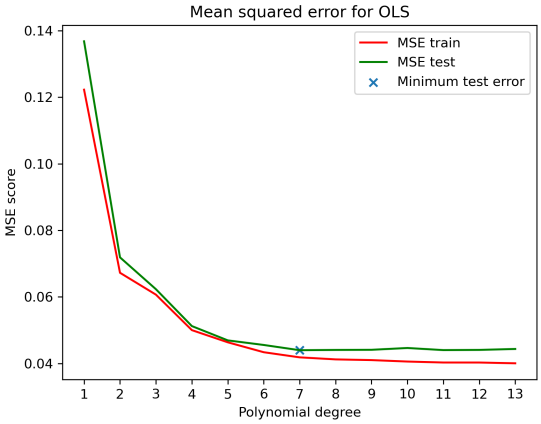

Figure 4: MSE for ordinary least squares of polynomial degree up to 13.

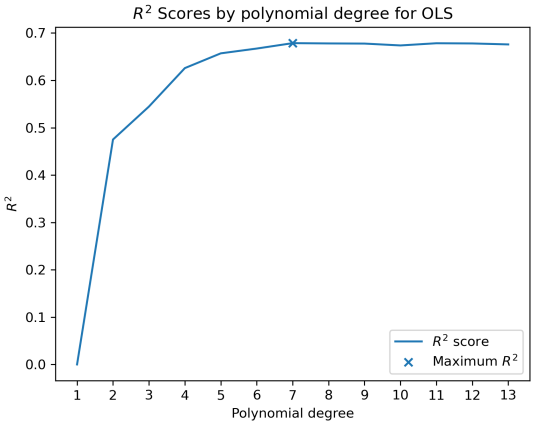

Figure 5: R^2 for ordinary least squares of polynomial degree up to 13.

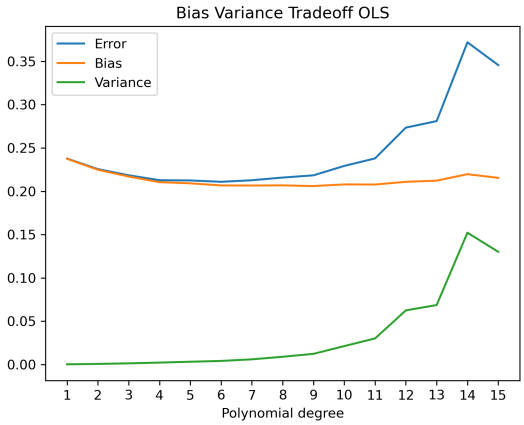

Figure 6: Bias variance tradeoff for OLS up to polynomial degree of 15. Resampling with bootstrap.

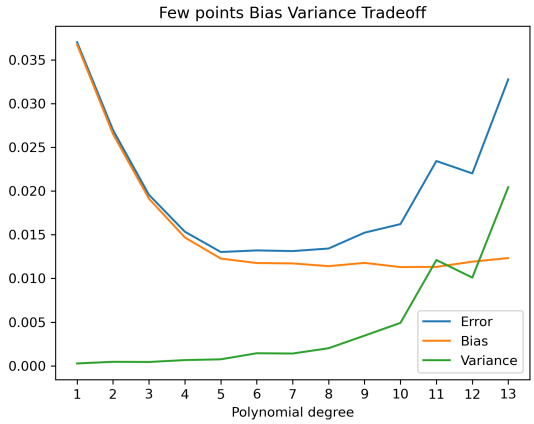

Figure 7: Bias variance tradeoff for OLS with 400 points instead of 1600 for polynomial degree up to 13. Resampling with bootstrap.

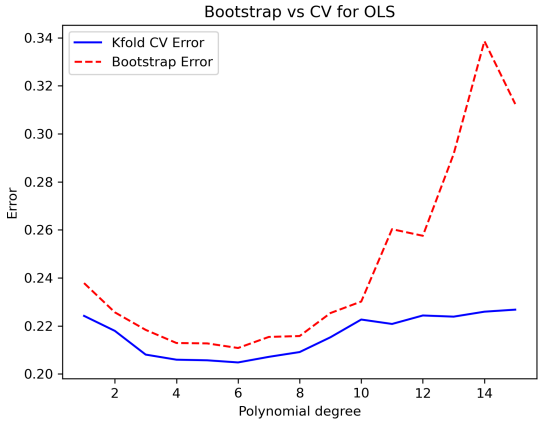

Figure 8: Comparison of MSE for bootstrapping and cross validation for ordinary least squares for polynomial degree up to 15.

4.1.2 Results from Ridge

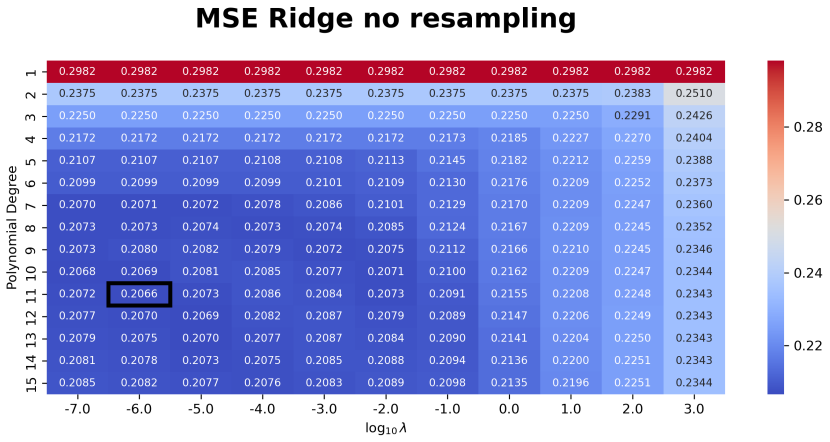

Figure 9: MSE for Ridge regression of different values for λ up to degree 15. No resampling.

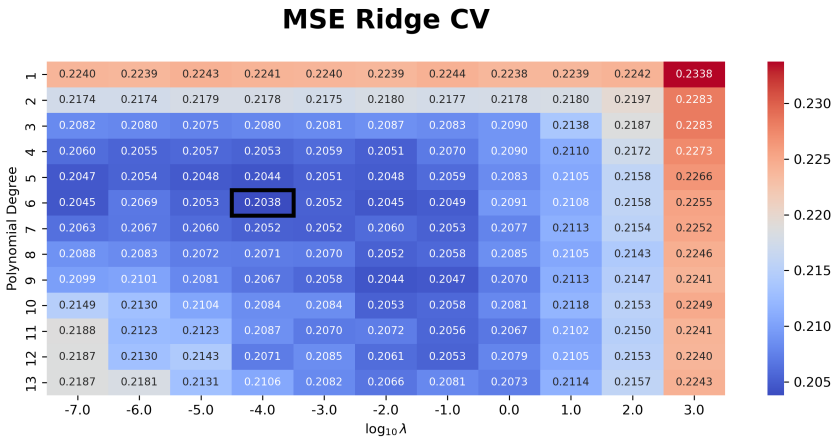

Figure 10: MSE for Ridge regression of different values for λ up to degree 13. Resampling with cross validation.

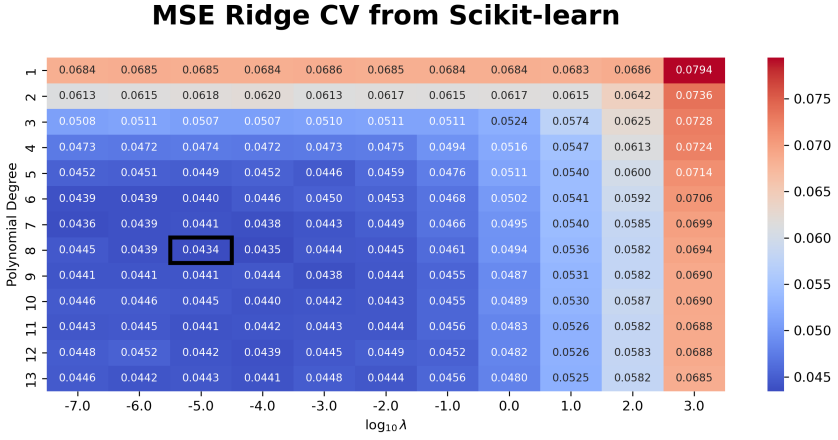

Figure 11: MSE for Ridge regression of different values for λ up to degree 13. Resampling with cross validation. Functions from Scikit-learn.

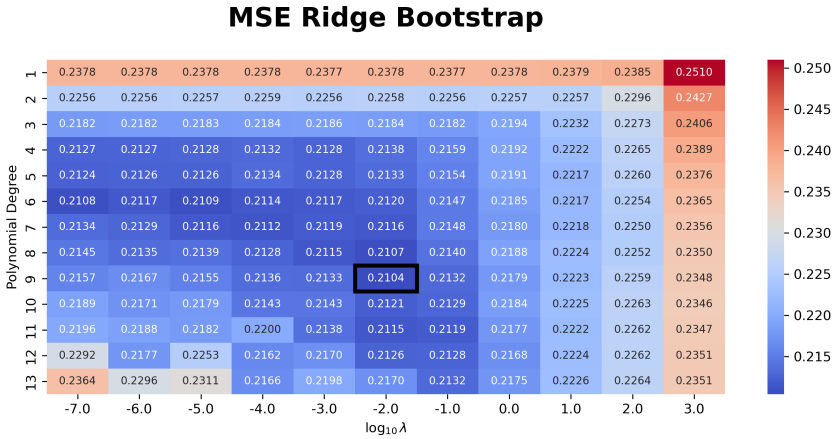

Figure 12: MSE for Ridge regression of different values for λ up to degree 13. Resampling with bootstrap.

4.1.3 Results from Lasso

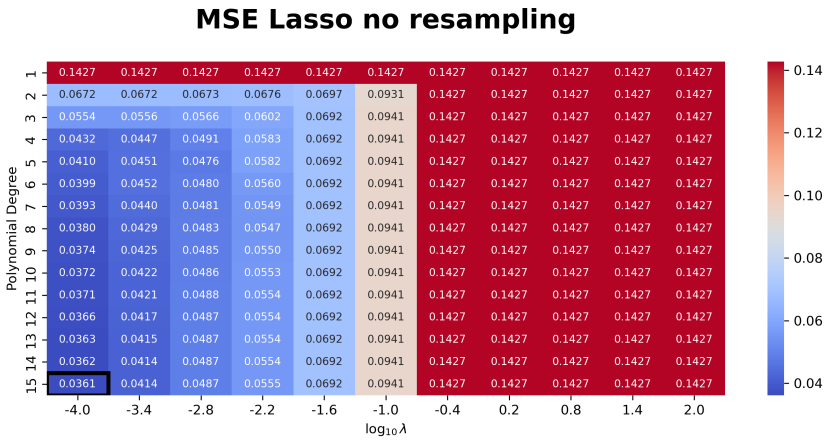

Figure 13: MSE for Lasso regression of different values for λ up to degree 15. No resampling.

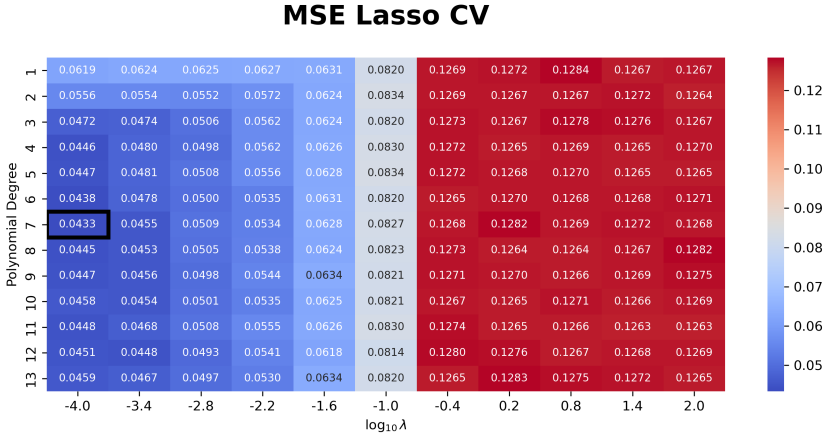

Figure 14: MSE for Lasso regression of different values for λ up to degree 13. Resampling with cross validation.

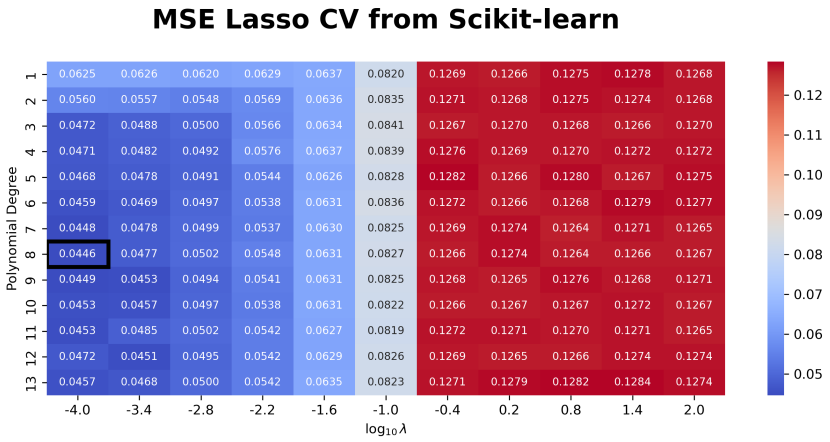

Figure 15: MSE for Lasso regression of different values for λ up to degree 13. Resampling with cross validation. Functions from Scikit-learn.

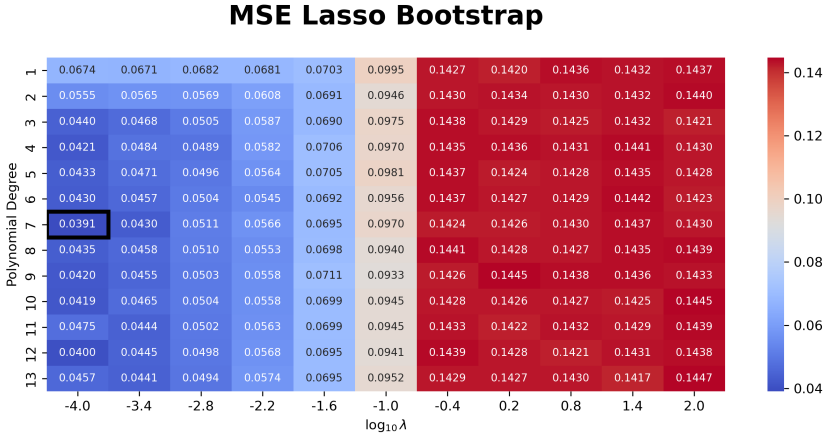

Figure 16: MSE for Lasso regression of different values for λ up to degree 13. Resampling with bootstrap.

4.2 Results for topographical data

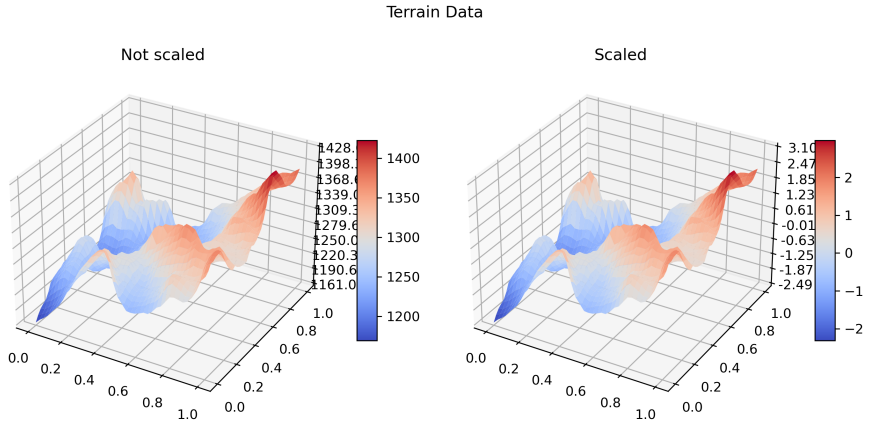

Figure 17: Data used before and after scaling. Topographical data.

4.2.1 Results from Ordinary least squares

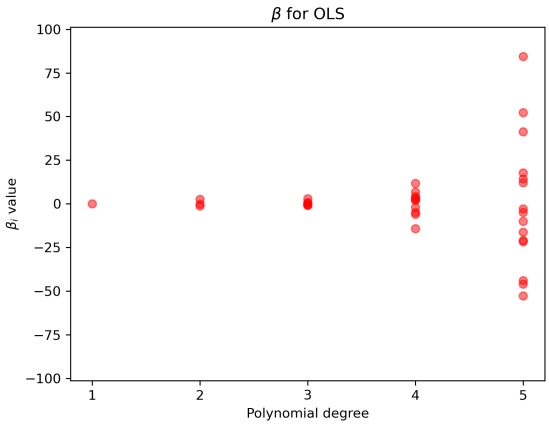

Figure 18: Values of $\boldsymbol{\beta}$ from ordinary least squares for polynomial degree up to 5. Topographical data.

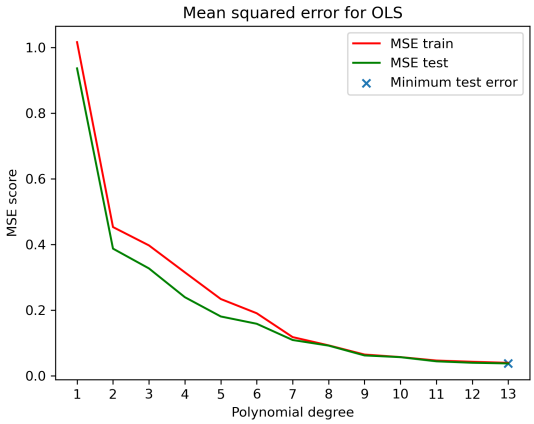

Figure 19: MSE for ordinary least squares of polynomial degree up to 13. Topographical data.

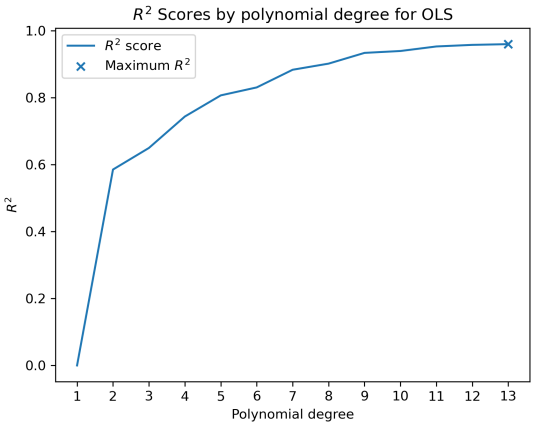

Figure 20: R^2 for ordinary least squares of polynomial degree up to 13. Topographical data.

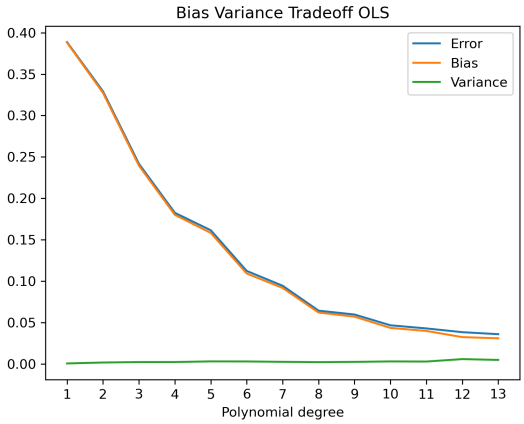

Figure 21: Bias variance tradeoff for OLS up to polynomial degree of 13. Resampling with bootstrap. Topographical data.

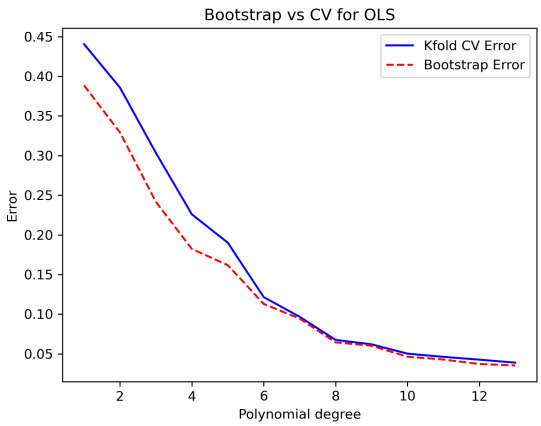

Figure 22: Comparison of MSE for bootstrapping and cross validation for ordinary least squares for polynomial degree up to 13. Topographical data.

4.2.2 Results from Ridge

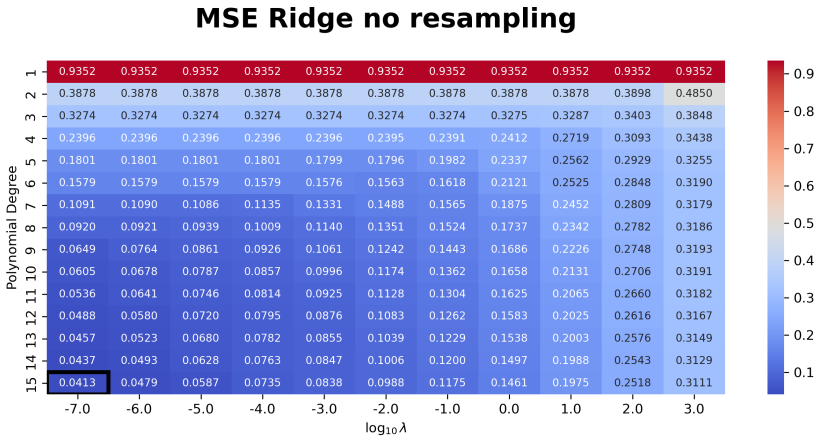

Figure 23: MSE for Ridge regression of different values for λ up to degree 15. No resampling. Topographical data.

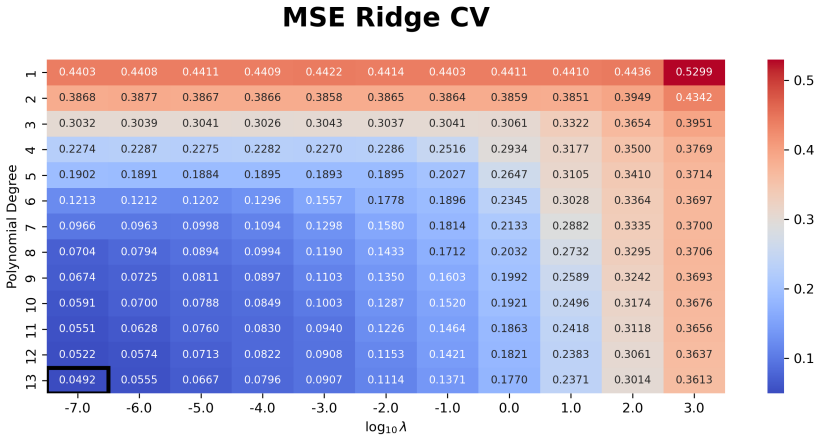

Figure 24: MSE for Ridge regression of different values for λ up to degree 13. Resampling with cross validation. Topographical data.

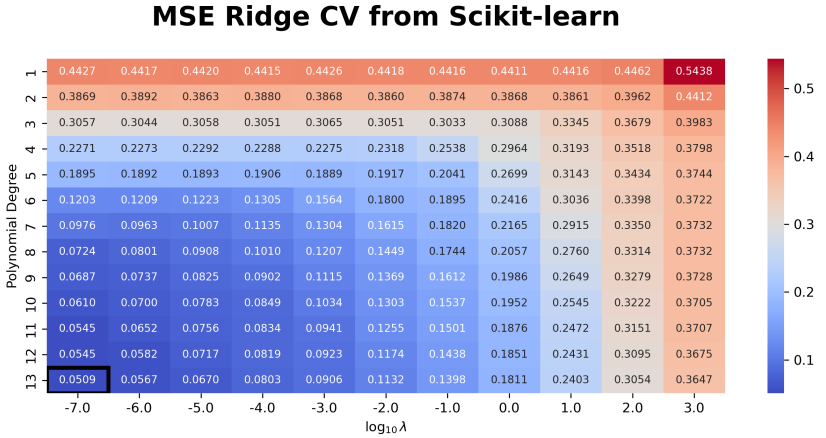

Figure 25: MSE for Ridge regression of different values for λ up to degree 13. Resampling with cross validation. Functions from Scikit-learn. Topographical data.

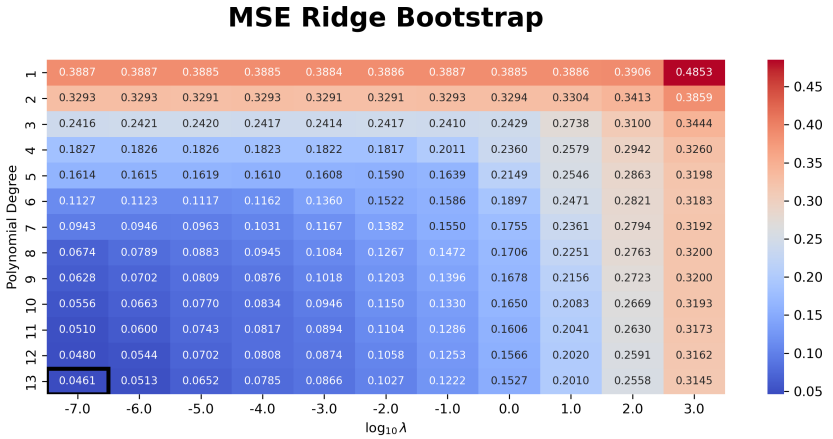

Figure 26: MSE for Ridge regression of different values for λ up to degree 13. Resampling with bootstrap. Topographical data.

4.2.3 Results from Lasso

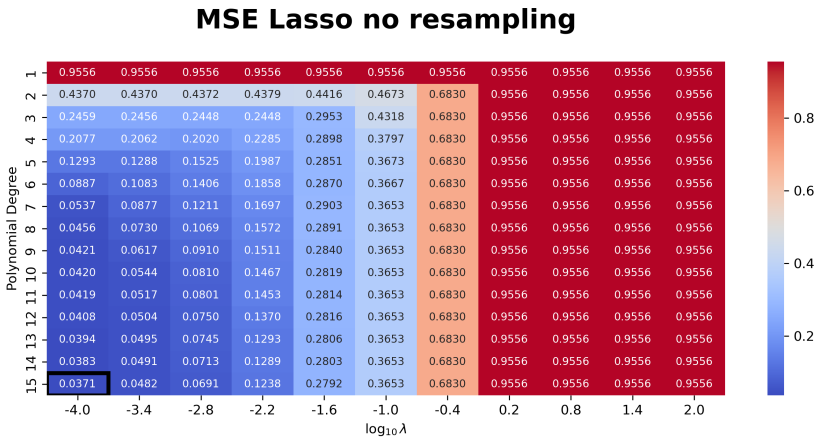

Figure 27: MSE for Lasso regression of different values for λ up to degree 15. No resampling. Topographical data.

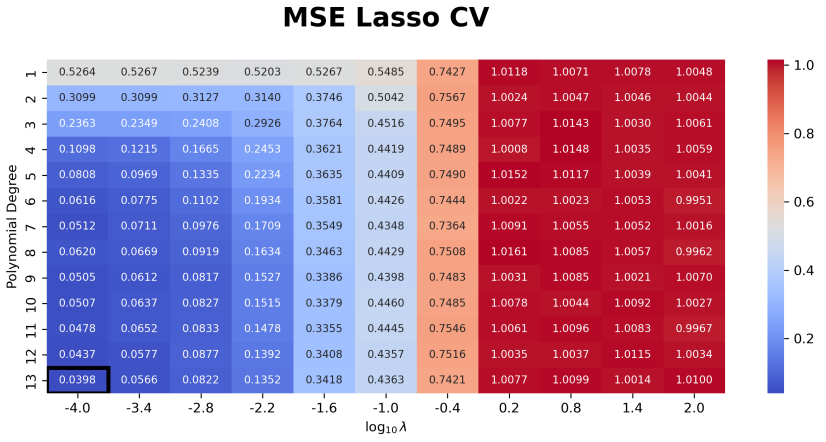

Figure 28: MSE for Lasso regression of different values for λ up to degree 13. Resampling with cross validation. Topographical data.

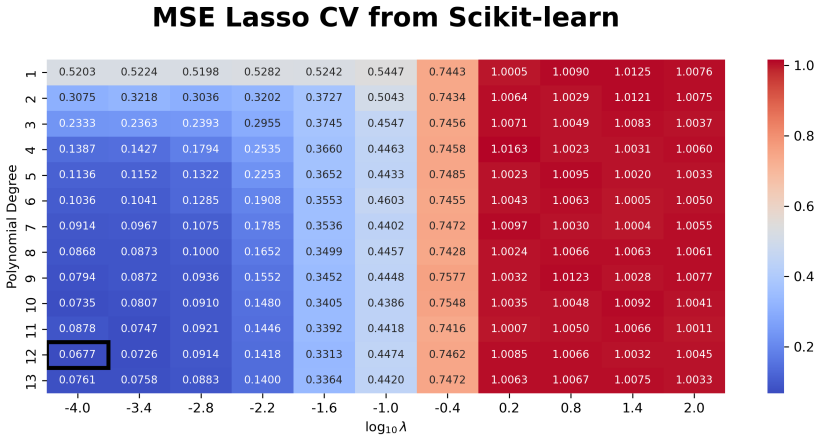

Figure 29: MSE for Lasso regression of different values for λ up to degree 13. Resampling with cross validation. Functions from Scikit-learn. Topographical data.

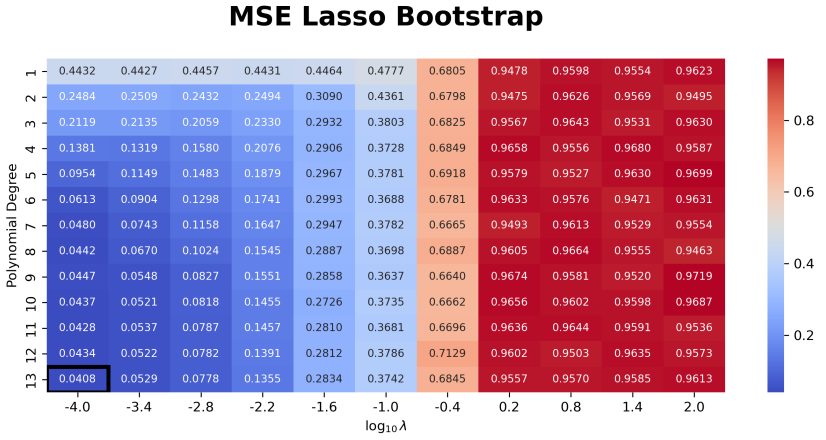

Figure 30: MSE for Lasso regression of different values for λ up to degree 13. Resampling with bootstrap. Topographical data.

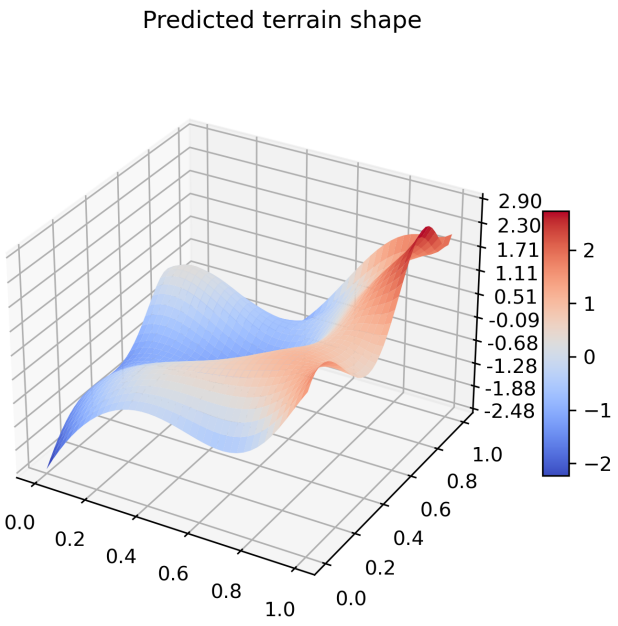

Figure 31: Predicted surface using Lasso, polynomial degree of 13, $\lambda = 10^{-4}$.

5 Discussion

In general, the results we were able to produce were heavily effected by a lack of computing power. Ideally, we would have preferred to utilize lower values for λ where we see that the optimal values found lie on the lower range of the attempted values, a higher number of bootstraps, and a higher max iteration for solving Lasso. In some places, we decided to compare our own methods with those we would have gotten, had we utilized scikit-learn to a greater extent.

5.1 Discussion of Franke

We started out with a simple validation set approach (4, 5) for Ordinary least squares, in order to get a feeling for what kind of results we might expect. The resulting figures indicate that we may get a high degree of overfitting when attempting to use polynomial linear regression with a high polynomial degree with this kind of problem. We previously found that the number of $\boldsymbol{\beta}$ parameters grows quadratically with the polynomial degree, but we did not state how the size of an individual parameter β_i would behave. We see in figure 3 that most points lie around zero, gaining high values in comparison when degree is increased. We decided to only plot the values for the first five polynomial degrees, as when extending the graph the outliers make it difficult to pick up the details around the lower degrees. This is highly indicative of overfitting, which explains our previous figures.

We were also interested in exploring more about the Bias-Variance trade off. We initially produced figure 6, however due to the high number of points used $(40 \cdot 40)$, we saw only a small increase in variance. To illustrate the effect better, we decided to reproduce the plot with fewer points $(20 \cdot 20)$ in figure 7. This behaviour is expected, as variance is highly correlated with the number of samples we have at our disposal.

We ended off our results from ordinary least squares by comparing two different methods of resampling in figure 8. Here, we compared bootstrapping and K-Fold, seeing almost identical results. We found this a bit surprising, as we found bootstrapping a lot more computationally heavy. With OLS we found the best results with a polynomial degree of 7.

When attempting to compare how the mean squared error is effected for varying degrees of polynomials and different values for the regularization parameter λ, we initially used a 3D plot. This was highly informative, however difficult to place into a report such as this, due to not being able to rotate the graph. We instead therefore ended up using heat maps, were cells are shaded based on their values. As this ends up capturing a lot of detail, we highlight the box with the lowest value.

Over all, it seems like the lowest error for Ridge regression seems to lie in the range of polynomial degree 6 – 11 and $\lambda \in [10^{-5}, 10^{-2}]$. Due to limited computing power, we were unable to narrow this interval down further. When using no method of resampling 9 we got suspiciously similar results for most of the values, perhaps indicating a fault in the code.

With a varying range of values, we found smallest to be 0.2038 with cross validation 10 when $\lambda = 10^{-4}$ with a polynomial degree of 6. With bootstrapping 12 the optimal value was found as approximately 0.2104 when $\lambda = 10^{-2}$ and the polynomial degree was 9. Interestingly, Scikit-learn wildly outperformed our methods 11, with a score of 0.0434. We have been unable to figure out exactly why, but our leading theory is that we are either scaling poorly, or using a different method of inverting $\mathbf{X}^T\mathbf{X}$.

The results from Lasso regression are all characterized by the optimal values landing on our lowest value for λ. Upon finding this, we attempted to shift our range of values down, however we were constantly met with exceedingly slow run times and convergence errors. We increased the maximal number of iterations for Lasso to 100000, which gave the results above, however this did not fix the issue of convergence.

With no resampling 13, Lasso produced an MSE of 0.0361 with the lowest measured value $\lambda = 10^{-4}$ and polynomial degree 15. An almost identical value of 0.0391 was found with bootstrapping 16, only with a polynomial degree of 7. Promisingly, cross validating gave almost identical values of 0.0433 and 0.0446 with respectively our own KFold algorithm 14 and Scikit-learn 15. This was with polynomial degrees of respectively 7 and 8, and $\lambda = 10^{-4}$.

This last result further corroborates that the issue with our implementation of Ridge lies in the fitting or predicting stage, as everything else except the choice of model is identical between Lasso and Ridge. It is interesting to note that we might be seeing the zeroing out of the parameters for $\boldsymbol{\beta}$ as λ increases, as the MSE becomes nearly constant across the polynomial degrees.

5.2 Discussion of Terrain data

The data utilized for this section was of terrain data from Møsvatn Austfjell in Norway. The data set consists of $3601 \cdot 1801$ points, which given our previous performance issues is way too much. In our initial attempt, we choose 40 evenly spaced points in x and y direction from the total data. This resulting in an extremely high bias, giving us mostly useless results. We therefore opted to select every other point in the first 80×80 grid, resulting in a more modest $40 \cdot 40$ points. We also scaled the data such that it has a mean of 0 and standard deviation of 1. This is shown in figure 17, and note that this does not change the overall shape.

We saw the same trends for the values of $\boldsymbol{\beta}$ with the terrain data 18 as with Frank's function 3 when applying Ordinary least squares. Interestingly, we did not see a high degree of overfitting for higher complexity 19, the model generally seemed to improve. This might imply that our data set can predicted nicely using polynomials 20.

The graph of the bias and variance further supports this hypothesis 21. This is likely due to focusing on a much smaller area of the total terrain, giving less noise. We find it unlikely that our predictions would generalize well to the remaining terrain. Previously, we found that bootstrapping turned unstable for

higher polynomial degrees, but for this data set it seems to correspond well with k-fold 22.

In general, we found our best values for the MSE with a high polynomial degree and a low value for λ. We were large limited by our computing power, reducing our capability of getting more accurate estimates. With no resampling 23, we got our lowest value for MSE of 0.0413 with a polynomial degree of 15 and $\lambda = 10^{-7}$.

For cross validation, we got similar results with our own implementation 24 as with that from Scikit-learn 25. This corroborates that the differences we saw with Franke's function might be from our method of scaling, however we did not have time to explore this connection further. We again got similar results bootstrapping 26, all giving our lowest value of MSE with the highest tested polynomial degree and lowest value for λ.

The results from Lasso regression show signs of the parameters $\boldsymbol{\beta}$ zeroing out, as the MSEs are largely identical across polynomial degrees for high values of λ. With no resampling 27, we got an MSE of 0.0371. Comparing our results 28 with those from Scikit-learn 29, our estimates were generally lower. Our implementation of k-fold gave the best result of 0.0398 with a polynomial degree of 13 and $\lambda = 10^{-4}$. Scikit-learn on the other hand found the best result of 0.0677 with a polynomial degree of 12, meaning that we might not be far from the ideal parameters. Bootstrapping 30 again gave similar results, while being a lot more computationally intensive.

Finally, we decided to plot the predicted surface from out best estimate 31, which was found as Lasso with a polynomial degree of 13 and a regularization parameter of $\lambda = 10^{-4}$. The graph shows a striking similarity to the true data 17, seeming to just be a smoothed out version. It is worth noting that we might have been lucky in our choice of points, seeing that the highest and lowest points are respectively on opposite sides of our data range. We guess that this is easier to estimate using polynomial regression.

6 Conclusion

Throughout this project, we spent a lot of time changing our code as we moved through different parts. This illustrated the importance of planning out our codebase, such that we were less reliant on changing arguments everywhere for each small iteration. We also got our first introduction to writing reports, as we are all math majors, being more familiar with writing calculations and carrying out proofs. We also saw the importance of choosing and handling our data correctly, as this lead to wildly different results. Had we focused on this initially, we would have saved a lot of time on computation and model selection, perhaps giving us more time to figure out the mote minute details. It might also be beneficial to write some of the heavier computations in another language like C, when we are unable to apply pre built methods.

7 References

References

[Franke, 1982] Franke, R. (1982). Scattered Data Interpolation: Tests of Some Methods. *Mathematics of Computation*, 38.

[Hald, 1986] Hald, A. (1986). Galileo's statistical analysis of astronomical observations. *International Statistical Review / Revue Internationale de Statistique*, 54(2):211–220.

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer Verlag, Berlin.

[James et al., 2021] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2021). *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer US.

[Lindstrøm, 2017] Lindstrøm, T. L. (2017). *Spaces: An introduction to real analysis*, volume 29. American Mathematical Soc.

[Murphy, 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambdridge, Massachusetts.

[Süli and Mayers, 2003] Süli, E. and Mayers, D. F. (2003). *An Introduction to Numerical Analysis*. Cambridge University Press.

8 Appendix

The code is publicly available on github.com/augustfe/FYSSTK, written in python.