Linear Regression Analysis of the Franke Function and Terrain Data

Achraf Atifi
Department of Geosciences, University of Oslo

https://github.com/achat97/FYS-STK4155

(Dated: October 13, 2023)

The linear regression methods OLS, Ridge regression and Lasso regression have been used to fit a model to data generated by the Franke function and to terrain data. We've found that Ridge provides the best fit, with OLS close behind for both cases. For the Franke function, the lowest mean square error was 0.011284 for OLS and 0.011283 for Ridge, with a hyperparameter set to 10^{-9}, for a polynomial of degree 6 for both methods. The best fit for Lasso had a mean square error of 0.01857, with a hyperparameter equal to 10^{-4}, for a polynomial of degree 14. Ridge and OLS are essentially equally good for the Franke function, but Ridge has a lower variance. No resampling provided the best estimate for the Franke function for all methods. For the terrain data, we found that the highest R^2-score for OLS was 0.9075 for a polynomial of degree 10. For Ridge it was 0.9139, with a hyperparameter equal to 10^{-5} and a polynomial of degree 17. Lasso had an R^2-score of 0.806, for a hyperparameter set to 10^{-14} and a polynomial of degree 26. Ridge is slightly better than OLS but is using a much more complex model. Therefore, OLS would be the best overall choice. For the terrain data, we used cross-validation. It was found that 10 folds gave the best result for all cases.

In addition, we analyzed the effect of the hyperparameters on the bias-variance tradeoff. We found that increasing the penalty parameter in Ridge and Lasso will reduce the variance and increase the bias.

I. INTRODUCTION

We're surrounded by data we want to interpret, but it's not trivial for humans to uncover the properties of vast data sets by themselves. This is where machine learning comes in. Machine learning allows us to feed data to an algorithm which unravels the properties of the data without being explicitly programmed to do so for each problem. The simplest form of a machine learning algorithm is linear regression. This method will try to fit a model to input and output data sets. The model will often overfit or underfit the given data, just like if you made a tailored suit for a grown person and gave it to a child. It will not fit. We aim to find a balance between these two. To continue our suit analogy, we want to make a suit which will fit as many people as possible.
Leaving our analogy behind, we can now understand that we want to fit a line to our data that generalizes well to unseen data. The most common type of function you can fit to the data is a polynomial, which we will do in this study, but you can, in practice, fit almost any type of function you want [1].

When delving into the world of machine learning and linear regression, a problem that often arises when gathering data in the real world is that the amount of data is limited [1]. So, we will explore resampling methods which, in essence, imitate a larger pool of data.

In this study, we will fit data from the Franke function, which is generated in-house and real terrain data from the Atlas mountain range in Morocco. We will perform three different linear regression methods: Ordinary least Squares, Ridge regression and Lasso regression.

II. THEORY

Our analysis uses several methods to deduce the best possible model given the data at hand. We will now present the ideas behind these methods and how they are implemented.

A. Linear Regression

The essence of linear regression is that we assume our target data $\mathbf{y} \in \mathbb{R}^{n\times 1}$ can be given as a function

$$\mathbf{y}(\mathbf{x}) = f(\mathbf{x}) + \boldsymbol{\epsilon} \tag{1}$$

where $f(\mathbf{x})$ is a continuous function of our input data $\mathbf{x} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{\epsilon} \in \mathbb{R}^{n\times 1}$ is some noise which we may or may not know. In our case we will assume that $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$. We generally don't know $f(\mathbf{x})$, so we assume that it can be approximated as $f(\mathbf{x}) \simeq \mathbf{X}\boldsymbol{\beta}$, where $\mathbf{X} \in \mathbb{R}^{n\times p}$ is the design matrix. It consists of our input data and is given by

$$\mathbf{X} = \begin{pmatrix} 1 & x_0^1 & x_0^2 & \dots & x_0^{p-1} \\ 1 & x_1^1 & x_1^2 & \dots & x_1^{p-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n-1}^1 & x_{n-1}^2 & \dots & x_{n-1}^{p-1} \end{pmatrix}.$$

$\boldsymbol{\beta} \in \mathbb{R}^{p\times 1}$ is a column vector containing our regression parameters, which are unknown [1]. Given this approximation, we can write (1) as follows

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

or in component form

$$y_i = \mathbf{X}_{i,*}\boldsymbol{\beta} + \epsilon_i. \tag{3}$$

where $\mathbf{X}_{i,*}$ is the $i-$th row in the design matrix. ϵ is the only stochastic component in (2), nonetheless, it also makes $\mathbf{y}$ stochastic with a normal distribution according to our previous assumption. The mean of this distribution is given by

$$\begin{aligned}\mathbb{E}(y_i) &= \mathbb{E}(\mathbf{X}_{i,*}\boldsymbol{\beta} + \epsilon_i) = \mathbb{E}(\mathbf{X}_{i,*}\boldsymbol{\beta}) + \mathbb{E}(\epsilon_i) \\ &= \mathbf{X}_{i,*}\boldsymbol{\beta} + 0 = \mathbf{X}_{i,*}\boldsymbol{\beta}\end{aligned} \tag{4}$$

and its variance is

$$\begin{aligned}\mathrm{Var}(y_i) &= \mathbb{E}[(y_i - \mathbb{E}(y_i))^2] = \mathbb{E}(y_i^2) - \mathbb{E}(y_i)^2 \\ &= \mathbb{E}[(\mathbf{X}_{i,*}\boldsymbol{\beta})^2 + 2\epsilon_i\mathbf{X}_{i,*}\boldsymbol{\beta} + \epsilon_i^2] - (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 \\ &= (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 + 2\mathbf{X}_{i,*}\boldsymbol{\beta}\mathbb{E}(\epsilon_i) + \mathbb{E}(\epsilon_i^2) \\ &- (\mathbf{X}_{i,*}\boldsymbol{\beta})^2 = \mathbb{E}(\epsilon_i^2) = \mathrm{Var}(\epsilon_i) = \sigma^2.\end{aligned} \tag{5}$$

Thus $y_i \sim \mathcal{N}(\mathbf{X}_{i,*}\boldsymbol{\beta}, \sigma^2)$ [2].

Now, since $\boldsymbol{\epsilon}$ is stochastic, we won't be able to have a deterministic description of the target data. We, therefore, drop the error term and are left with

$$\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta} \tag{6}$$

and the error we make is

$$\boldsymbol{\epsilon} = \mathbf{y} - \tilde{\mathbf{y}} \tag{7}$$

which is dubbed as the residuals [1].

We can now clearly see what the aim of linear regression is. Since $\mathbf{y}$ and $\mathbf{X}$ is fixed, we need to find a optimal parameter $\hat{\boldsymbol{\beta}}$ which makes (7) as small as possible. A way to assess this error is by a cost function $C(\boldsymbol{\beta})$. Our choice of cost function gives rise to various regression models, which we will now explore.

1. *Ordinary Least Squares*

In Ordinary Least Squares (OLS), we let the cost function be the mean square error. That is, we let

$$\begin{aligned}C(\boldsymbol{\beta}) &= \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \frac{1}{n}(\mathbf{y} - \tilde{\mathbf{y}})^T(\mathbf{y} - \tilde{\mathbf{y}}) \\ &= \frac{1}{n}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}).\end{aligned} \tag{8}$$

This is a convex function and will, therefore, have a global minima where its derivative is zero. This is given by

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})$$

and solving for $\boldsymbol{\beta}$ gives us

$$\begin{aligned}&\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{y} \\ &\rightarrow \hat{\boldsymbol{\beta}}^{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\end{aligned} \tag{9}$$

which is our optimal parameter [1]. This can be implemented as shown in the pseudocode given by algorithm 1.

Algorithm 1: Ordinary Least Squares

Input Data, Target Data $= x, y$
$X = \mathrm{DesignMatrix}(x)$
$\beta = \mathrm{inverse}(X^TX)X^Ty$
ymodel $= X\beta$

Moving on, we can show that $\hat{\boldsymbol{\beta}}$ has an expectation

$$\begin{aligned}\mathbb{E}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}(\mathbf{y}) \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \mathbf{I}\boldsymbol{\beta} = \boldsymbol{\beta}\end{aligned} \tag{10}$$

and variance by first introducing the following result

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

which we can plug in the following equation and get that the variance is

$$\begin{aligned}&\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \mathbb{E}\{[\hat{\boldsymbol{\beta}} - \mathbb{E}(\hat{\boldsymbol{\beta}})][\hat{\boldsymbol{\beta}} - \mathbb{E}(\hat{\boldsymbol{\beta}})]^T\} \\ &= \mathbb{E}\{[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} - \boldsymbol{\beta}][(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} - \boldsymbol{\beta}]^T\} \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}(\mathbf{y}\mathbf{y}^T)\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\mathbf{X}^T + \mathbf{I}\sigma^2)\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\ &= \boldsymbol{\beta}\boldsymbol{\beta}^T + \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\text{[1].}\end{aligned} \tag{11}$$

2. *Ridge Regression*

We presumed in the last section that $\mathbf{X}^T\mathbf{X}$ is invertible. This is not always the case, and it may be singular. One way to deal with this is to replace $\mathbf{X}^T\mathbf{X}$ with $\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}$, where $\lambda \in [0, \infty)$ is the tuning or penalty parameter. The result is a positive definite matrix, which is invertible [2]. We could also arrive at this result by introducing the following cost function

$$\begin{aligned}C(\boldsymbol{\beta}) &= \frac{1}{n}\|\mathbf{y} - \tilde{\mathbf{y}}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2 \\ &= \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 + \lambda\sum_{i=0}^{p-1}(\beta_i)^2 \\ &= \frac{1}{n}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}\end{aligned} \tag{12}$$

Again, the global minima are where the derivative is zero. This is given by

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}) + 2\lambda\hat{\boldsymbol{\beta}}$$

and solving for $\hat{\boldsymbol{\beta}}$ results in

$$\hat{\boldsymbol{\beta}}^{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}. \tag{13}$$

Recall that we want to minimize (12). Thus, increasing λ drives the coefficients towards zero in order to maintain balance [3].

Also, a result which we will see later on is that

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}^{OLS}) - \mathrm{Var}(\hat{\boldsymbol{\beta}}^{Ridge}) > 0 \tag{14}$$

for $\lambda > 0$ [1].

3. *Lasso Regression*

Lastly, we have Lasso regression. This is similar to Ridge regression in that we add a penalty term to (8). This time we add $\lambda\|\boldsymbol{\beta}\|_1$ which results in the following cost function;

$$\begin{aligned} C(\boldsymbol{\beta}) &= \frac{1}{n}\|\mathbf{y} - \tilde{\mathbf{y}}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 \\ &= \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 + \lambda\sum_{i=0}^{p-1}|\beta_i|. \end{aligned} \tag{15}$$

Due to the absolute value, there's no explicit expression for the optimal parameters [2]. So when we implement Lasso regression, we're going to use the function `sklearn.linear_model.Lasso` from the machine learning library `scikit-learn`.

One note about how Lasso differs from Ridge is that Ridge drives the coefficients towards zero, while Lasso can set coefficients to zero and thus remove less important features [4].

4. *Bias-Variance Tradeoff*

To interpret the value of the mean squared error even further, we can decompose it

$$\begin{aligned} MSE &= \mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \mathbb{E}[(\mathbf{f} + \boldsymbol{\epsilon} - \tilde{\mathbf{y}})^2] \\ &= \mathbb{E}\{[\mathbf{f} + \boldsymbol{\epsilon} - \tilde{\mathbf{y}} + \mathbb{E}(\tilde{\mathbf{y}}) - \mathbb{E}(\tilde{\mathbf{y}})]^2\} \\ &= \mathbb{E}\{[(\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})) + \boldsymbol{\epsilon} + (\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}})]^2\} \\ &= \mathbb{E}\{[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})]^2 + \boldsymbol{\epsilon}^2 + [\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}]^2 \\ &\quad + 2\boldsymbol{\epsilon}[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})] + 2\boldsymbol{\epsilon}[\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}] \\ &\quad + 2[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})][\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}]\} \\ &= \mathbb{E}\{[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})]^2\} + \mathbb{E}\{\boldsymbol{\epsilon}^2\} + \mathbb{E}\{[\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}]^2\} \\ &\quad + 2\mathbb{E}\{\boldsymbol{\epsilon}[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})]\} + 2\mathbb{E}\{\boldsymbol{\epsilon}[\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}]\} \\ &\quad + 2\mathbb{E}\{[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})][\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}]\} \\ &= [\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})]^2 + \mathbb{E}\{[\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}]^2\} + \mathrm{Var}\{\boldsymbol{\epsilon}\} \\ &\quad + 2[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})]\mathbb{E}\{\boldsymbol{\epsilon}\} + 2\mathbb{E}\{\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}\}\mathbb{E}\{\boldsymbol{\epsilon}\} \\ &\quad + 2[\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})]\mathbb{E}\{\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}\} \end{aligned}$$

now we recall that $\mathbb{E}\{\boldsymbol{\epsilon}\} = 0$ and that

$$\mathbb{E}\{\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}\} = \mathbb{E}\{\tilde{\mathbf{y}}\} - \mathbb{E}\{\tilde{\mathbf{y}}\} = 0.$$

Plugging these in gives us

$$\begin{aligned} MSE &= [\mathbf{f} - \mathbb{E}(\tilde{\mathbf{y}})]^2 + \mathbb{E}\{[\mathbb{E}(\tilde{\mathbf{y}}) - \tilde{\mathbf{y}}]^2\} + \mathrm{Var}\{\boldsymbol{\epsilon}\} \\ &= \mathrm{Bias}[\tilde{\mathbf{y}}]^2 + \mathrm{Var}[\tilde{\mathbf{y}}] + \boldsymbol{\sigma}^2. \end{aligned} \tag{16}$$

Recall that we usually don't know the deterministic function f, so we will instead use the data $\mathbf{y}$ we have at hand.

The bias term in (16) is the error that comes from making wrong assumptions about the data, like assuming they're linear when they're not. High-biased models have a tendency to underfit the data, i.e. the model will not learn the underlying properties of the data. The variance is a measure of the spread of our predictions. So, a high-variance model has a tendency to overfit data, meaning that the model tries to traverse as many of the data points as possible with its noise. A common practice which we will do is to split the data points into train and test data. We train our model on the training data and then make a prediction with the test data. Therefore, a high-variance model will perform well if we make a prediction with the training data but will perform poorly when using the test data. The last term is the variance of the error and is due to the data having some noise, as we have mentioned earlier, which we can't control [4].

B. Resampling Techniques

We often have a limited data set, and gathering new data is often time-consuming or expensive. A way to slightly work around this problem is to use resampling techniques. These methods involve drawing samples from a training data set several times, which is used to refit the model each time. When this is done, we can compare how the fits differ [1].

We're going to use two different resampling techniques, Bootstrap and Cross-Validation.

1. Bootstrap

Let $\mathcal{D} = \{z_0, z_1, \ldots, z_{n-1}\}$ be our original training data set. Then, the bootstrap works as follows

a) Draw a bootstrap sample $\mathcal{D}^* = \{z_0^*, z_1^*, \ldots, z_{n-1}^*\}$ randomly from $\mathcal{D}$ with replacement.

b) Calculate the regression parameters $\boldsymbol{\beta}^*$ using $\mathcal{D}^*$ and fit it with the test data yielding $\tilde{\mathbf{y}}^*$.

c) Repeat this process B times, yielding $\tilde{\mathbf{y}}_1^* \cdots \tilde{\mathbf{y}}_B^*$.

d) Now, we can get a better approximation of e.g. the error by finding the mean of the errors of the different fits [1].

To show how this is implemented, pseudocode is shown in algorithm 2, where bootstrap is used to generate B-sets of model predictions. These can then be used to calculate e.g. the MSE for each case, and then simply find the average of all those MSE values.

Algorithm 2: Ordinary Least Squares with Bootstrap

Input Data, Target Data $= x, y$
Number of bootstraps $= B$
$X =$ DesignMatrix(x)
ymodel $=$ empty(len(y_{test}), B)
$X_{train}, X_{test}, y_{train}, y_{test} =$ split(X, y)

for $i = 0$ **to** $i = B - 1$ **do**
  $X^*, y^* =$ resample(X_{train}, y_{train})
  $\beta =$ OLS(X^*, y^*)
  ymodel$[:, i] = X_{test}\beta$
end

2. Cross-Validtation

Some issues that bootstrap gives rise to is that some values may be drawn more frequently than others, giving them a higher weight. So, a resampling technique which deals with this issue is k-fold cross-validation. What we do is that we

a) Shuffle the dataset randomly.

b) Split the dataset into k groups.

c) For each group:

1. Decide which group to be test data and let the rest be training data (all groups must at a point be test data).
2. Fit the model to the training data.
3. Retain an error estimate using the test data.

d) Take the average of the errors calculated [1].

C. Error Estimate

We have already introduced an error estimate that we're going to use in our analysis. Namely the mean square error

$$MSE(\mathbf{y}, \tilde{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2. \tag{17}$$

Another method we're going to use to assess the model accuracy is the R^2-score

$$R^2(\mathbf{y}, \tilde{\mathbf{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{18}$$

where

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i.$$

The best possible score is 1, which we want our model to be close to.

III. RESULTS AND ANALYSIS

We will perform linear regression on two types of data sets. The first is data from the Franke function, while the other is terrain data found at https://earthexplorer.usgs.gov/.

Both cases are two-dimensional, and we would like to fit a polynomial of degree p to the data. In the case of $p = 2$, the polynomial would look like the following;

$$\tilde{z}(x, y) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + xy\beta_4 + \beta_5 y^2$$

From this, we see that for a polynomial of degree p, we will have $(p + 1)(p + 2)/2$ regression parameters. In addition, if we have N x and y-values, we will have a mesh grid of N^2 points. This results in a design matrix with dimensions $N^2 \times ((p + 1)(p + 2)/2)$.

A. Franke Function

The Franke function is given by

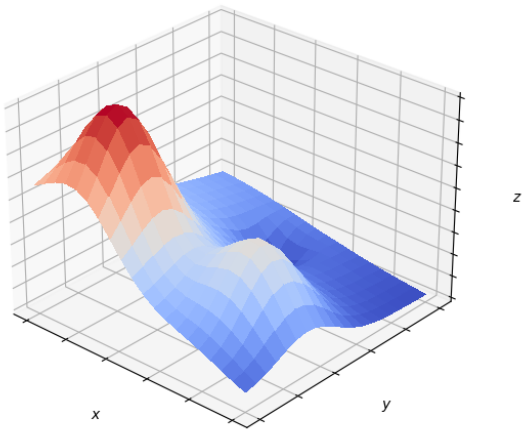

Figure 1: Plot of (19) with $N = 20$ and $c = 0$

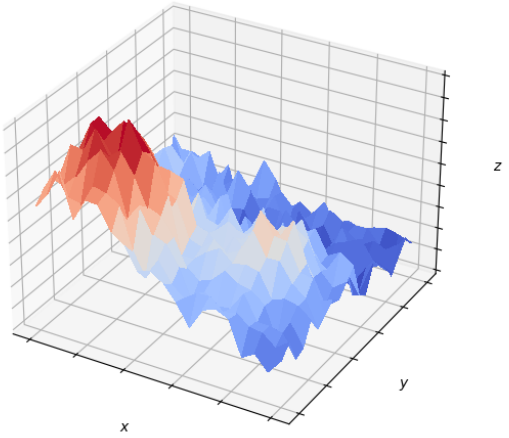

Figure 2: Plot of (19) with $N = 20$ and $c = 0.1$

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

where $x, y \in [0,1]$. We will generate N-values randomly for each x and y from a uniform distribution. In addition, we're going to add a small normally distributed noise $c\mathcal{N}(0,1)$, where c is a tuning parameter. So the full description of our target data is

$$z(x,y) = f(x,y) + c\mathcal{N}(0,1). \tag{19}$$

When performing one of the regression methods without resampling, or when we use bootstrap, we will use 20% of the data as test data, while the rest is going to be used as training data. There's no need to split data when performing cross-validation because it's an inherent part of it.

When it comes to scaling, we know that the input data is already between 0 and 1. Furthermore, if we would like to standardize the data, we're just going to increase the spread in the data since we're dividing by a standard deviation, which is less than one. So what we're going to do is centre our data; that is, we're going to subtract by the mean as follows

$$\mathbf{z}^*_{train} = \mathbf{z}_{train} - \mu^z_{train}$$

$$\mathbf{z}^*_{test} = \mathbf{z}_{test} - \mu^z_{train}$$

$$\mathbf{X}^*_{j,train} = \mathbf{X}_{j,train} - \mu^{\mathbf{X}_j}_{train}$$

$$\mathbf{X}^*_{j,test} = \mathbf{X}_{j,test} - \mu^{\mathbf{X}_j}_{train}$$

where j corresponds to the column number. An important consequence of this is that the intercept gets removed. This may lead to a reduced error when performing Ridge or Lasso regression [1].

1. *Ordinary Least Squares*

We will perform ordinary least squares as described in section II.1 with data generated by (19). From figure 3, we see that as we increase the complexity of our model, the more accurate it becomes. It's pretty trivial to see why this is the case. From figure 1 and 2, we see that the shape is rather complex and definitely not linear. So, it makes sense that a higher degree would fit the data better.

Figure 4 shows us the values of the optimal parameters as we increase the polynomial degree. We see that as we increase the polynomial degree, the larger the parameters become. Again, this is because as the complexity increases, the better it can fit the data given. But this gives rise to what was discussed in II.4. That is, the variance will increase. This can be seen in figure 5. β_i for i between 6 and 14 have a high variance, and thus we are starting to overfit. This is also shown in figure 6. When we increase the complexity, the test error will increase, while the training error will decrease because it's overfitting the training data.

Moving on, we will implement the bootstrap method discussed earlier and use this to calculate the bias-variance tradeoff to see what is happening in detail. The result can be seen in figure 7. So, as we've pointed out, the variance does actually increase. The bias will, of

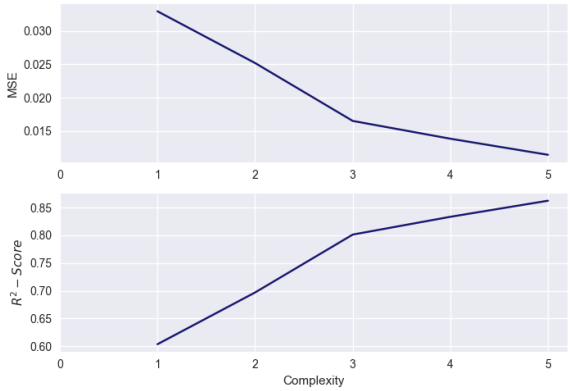

Figure 3: The MSE and R^2-score as a function of a polynomial degree p for OLS with $N = 20$ and $c = 0.1$. Here plotted up to $p = 5$.

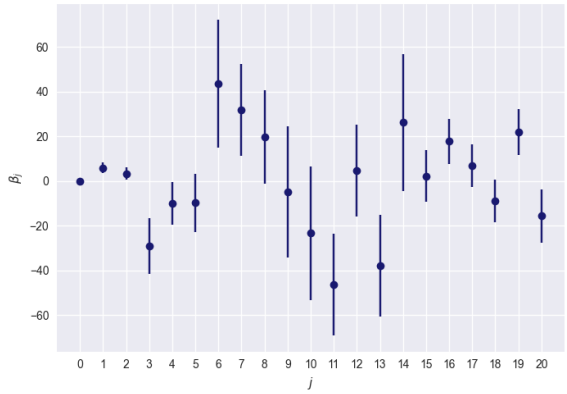

Figure 5: Values of the optimal regression parameters for $p = 5$, $N = 20$ and $c = 0.1$ within one standard deviation.

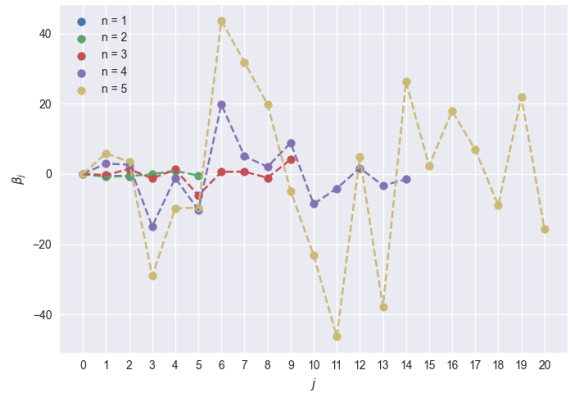

Figure 4: The values of the different optimal regression parameters generated for OLS with $N = 20$ and $c = 0.1$ for degrees 1–5.

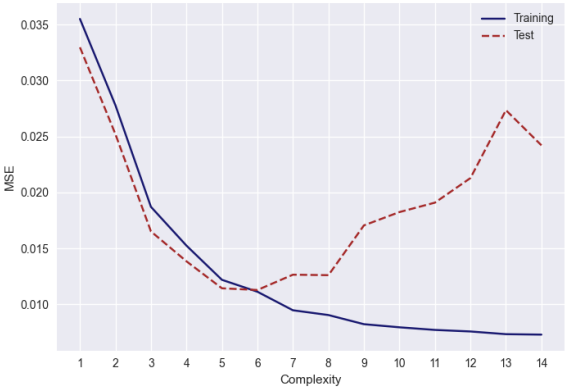

Figure 6: The MSE a as a function of a polynomial degree p for OLS with $N = 20$ and $c = 0.1$. Here plotted up to $p = 14$.

course, decrease to a certain point as we increase complexity due to the fact that equation (19) has complex properties. For all the results we have laid out so far, it seems that the most correct fit for our data is a polynomial of degree five.

We've also chosen to look at a case with twice as much data. This is shown in figure 8. We see that the model has enough data to learn the most important properties and thus create a more correct fit. So, when increasing the complexity, the variance will not increase that much.

Lastly, we'll implement k-fold cross-validation for $k =$ 5–10, which can be seen in figure 9. We see that the error generally decreases as we increase the number of folds. We also see that the best estimate is for $k = 10$ and $p = 8$. If we compare the error when using no resampling, bootstrap and cross-validation, as seen in figure 10, we see that the best estimate is actually the case with a regular train-test split and no resampling. This is usually not what we would expect since regular splitting can have an unbalanced influence on either model building or model evaluation [1]. Cross-validation usually takes care of this, but in this instance, it has a negative effect. Tuning the amount of folds did not change this effect. In figure 11, we've increased the amount of data points by three times. In this case, we see that the cross-validated model is slightly better at a higher complexity. This suggests that the training sets in each fold may be too small, making it harder for the model to make accurate predictions.

2. *Ridge Regression*

Now, we're going to implement Ridge regression. This method adds another complexity, which is the penalty parameter. So, not only are we going to find the right

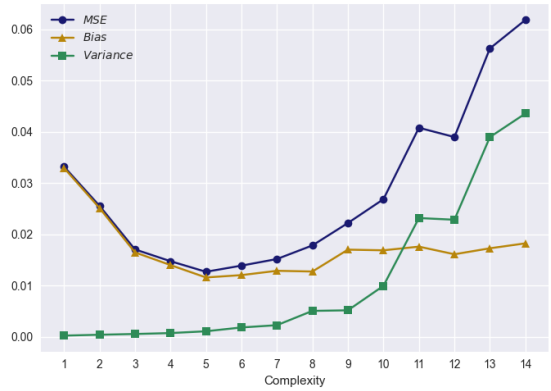

Figure 7: Bias-Variance tradeoff for OLS using the bootstrap method with $N = 20$, $c = 0.1$ and $B = N^2$ bootstrap iterations..

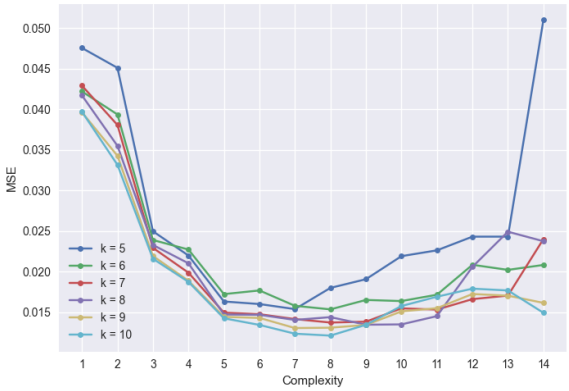

Figure 9: The MSE a as a function of a polynomial degree p for OLS with $N = 20$ and $c = 0.1$. Here, plotted up to $p = 14$ using cross-validation where the number of folds ranges from 5 to 10.

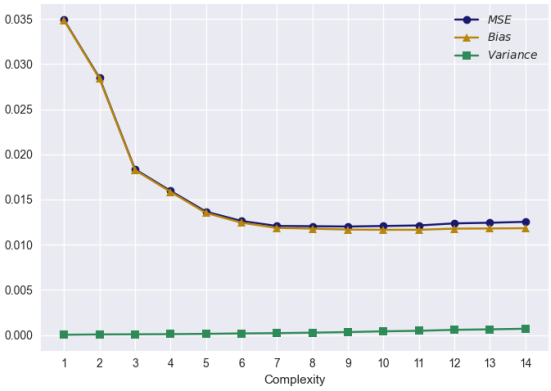

Figure 8: Bias-Variance tradeoff for OLS using the bootstrap method with $N = 50$, $c = 0.1$ and $B = N^2$ bootstrap iterations.

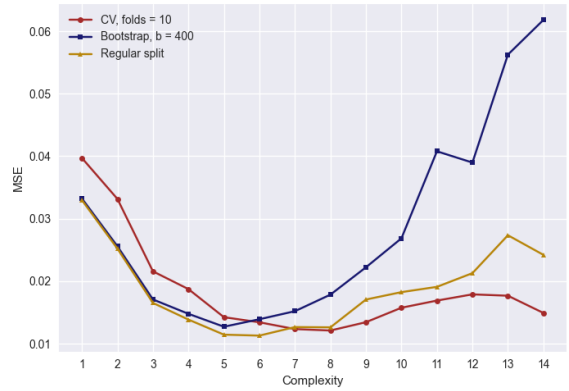

Figure 10: The MSE a as a function of a polynomial degree p for OLS with $N = 20$ and $c = 0.1$. Here, plotted up to $p = 14$ for three cases. a) No resampling, b) bootstrap with $B = N^2$ iteration and c) cross-validation with 10 fold.

polynomial degree but also the right hyperparameter. Firstly, we implement the bootstrap method and plot a bias-variance tradeoff as above, but now for two different hyperparameters. We start with $\lambda = 10^{-1}$, which is shown in figure 12. The figure clearly shows what we stated in equation (14), which is that the penalty parameter tends to reduce the variance and increase the bias. If we now let $\lambda = 10^{-9}$, we see that the variance will increase again, in addition to the error being reduced, as shown in figure 13. Increasing the number of data points will, as in the OLS case, reduce the variance, as can be seen in figure 14. Notice that the last two cases are quite similar to their equivalents using OLS. This is due to Ridge regression being close to OLS when λ is small.

Moving on, we will implement k-fold cross-validation. We plot the error for the different numbers of folds to help decide how many to use. We choose something in between the penalty parameters used above. So we let $\lambda = 10^{-3}$. The result is shown in figure 15. We see that the optimal number of folds is $k = 10$.

So, now that we are equipped with the right number of folds, we want to find the polynomial degree and hyperparameter, which gives the lowest error. This is shown in figure 16. We see that the lowest error is achieved when $\lambda = 10^{-9}$ and $p = 8$. We're now ready to compare the three methods again. This is shown in figure 17. This is equivalent to what we got for OLS. The best model is the case with no resampling, with cross-validation close behind(but for a higher polynomial degree). Again, this could be due to the lack of data.

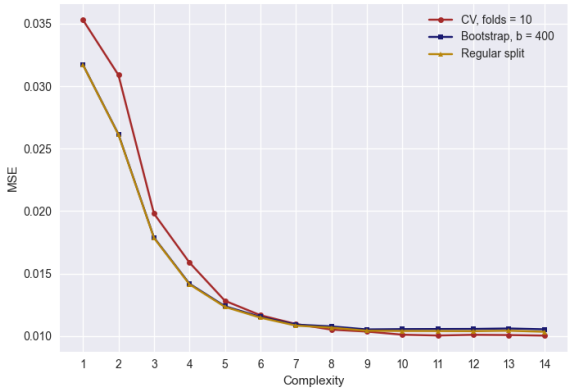

Figure 11: The MSE a as a function of a polynomial degree p for OLS with $N = 80$ and $c = 0.1$. Here, plotted up to $p = 14$ for three cases. a) No resampling, b) bootstrap with $B = N^2$ iteration and c) cross-validation with 10 fold.

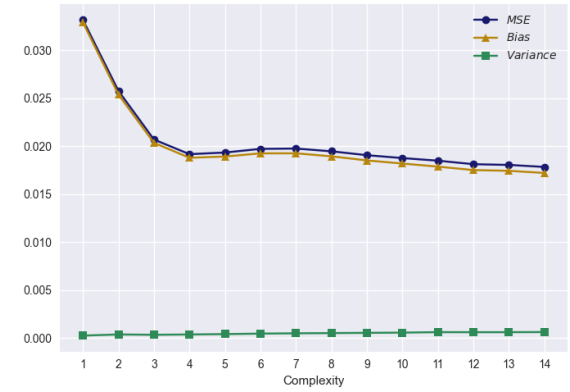

Figure 12: Bias-Variance tradeoff for Ridge using the bootstrap method with $N = 20$, $\lambda = 10^{-1}$, $c = 0.1$ and $B = N^2$ bootstrap iterations.

3. Lasso regression

Lastly, we're going to look at Lasso regression. Our analysis will be the same as we did for Ridge regression, but this time we will perform a bias-variance tradeoff with $\lambda = 10^{-1}$ and $\lambda = 10^{-6}$. The first case is shown in figure 18. We see that for this case, the mean square error will actually stay constant, but why? Like in Ridge regression, λ will tend to increase bias and reduce variance as it gets larger. So, in this case, the reduction of variance is so large that it got set to zero. Since the variance is zero, there is no interplay with the bias, which is, therefore, constant since the MSE is constant.

If we decrease the penalty parameter and set it to $\lambda = 10^{-6}$, we get the result shown in figure 19. Since the hyperparameter is smaller, the variance doesn't get

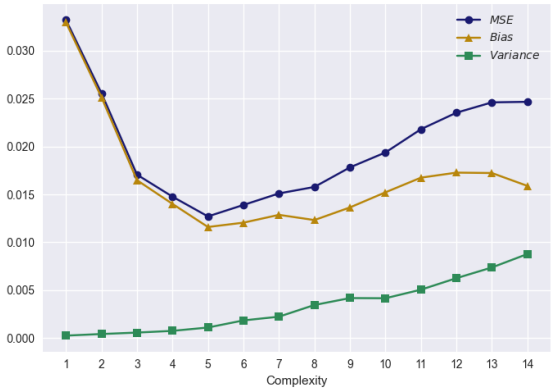

Figure 13: Bias-Variance tradeoff for Ridge using the bootstrap method with $N = 20$, $\lambda = 10^{-9}$, $c = 0.1$ and $B = N^2$ bootstrap iterations.

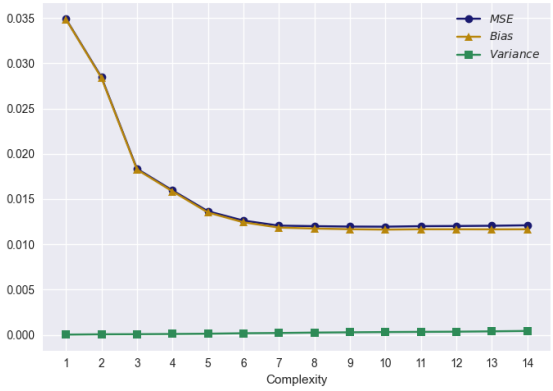

Figure 14: Bias-Variance tradeoff for Ridge using the bootstrap method with $N = 50$, $\lambda = 10^{-9}$, $c = 0.1$ and $B = N^2$ bootstrap iterations.

dampened as much as in the last case, but it's still very small. The fit is also more accurate, and the error decreases with complexity, as can be seen.

Moving on to cross-validation, we want to first find the number of folds which give the best estimate. This time, we'll choose the hyperparameter which has gotten us the smallest error so far, which is $\lambda = 10^{-6}$. The result is shown in figure 20. This case differs a little from the others in that the optimal fold is $k = 9$. So, using this fold to find the optimal polynomial degree and hyperparameter results in the heatmap shown in figure 21. The optimal hyperparameter and polynomial are the smallest and largest values we have plotted, respectively. In fact, if we decrease the hyperparameter even further or increase the degree, the optimal will move to those values. This makes Lasso tedious and taxing on the computational resources.

Lastly, we'll look at how the different methods per-

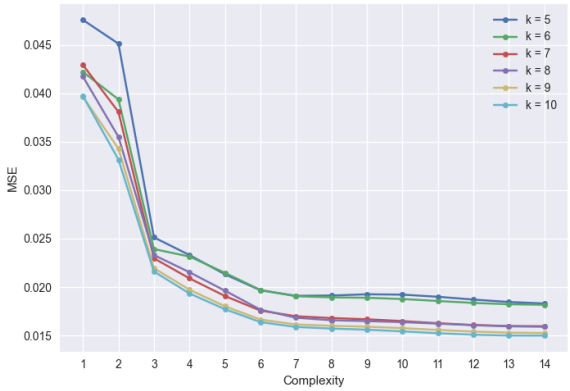

Figure 15: The MSE a as a function of a polynomial degree p for Ridge with $N = 20$, $\lambda = 10^{-3}$ and $c = 0.1$. Here, plotted up to $p = 14$ using cross-validation where the number of folds ranges from 5 to 10.

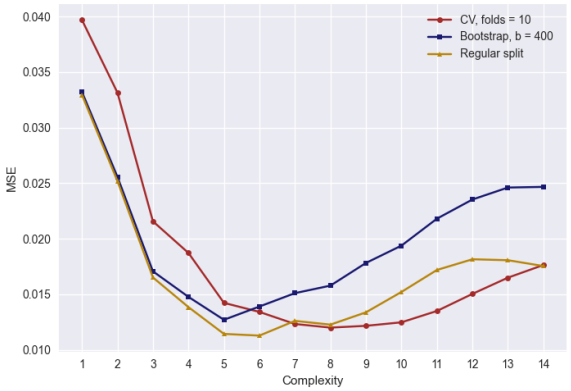

Figure 17: The MSE a as a function of a polynomial degree p for Ridge with $N = 20$, $\lambda = 10^{-9}$ and $c = 0.1$. Here, plotted up to $p = 14$ for three cases. a) No resampling, b) bootstrap with $B = N^2$ iteration and c) cross-validation with 10 folds.

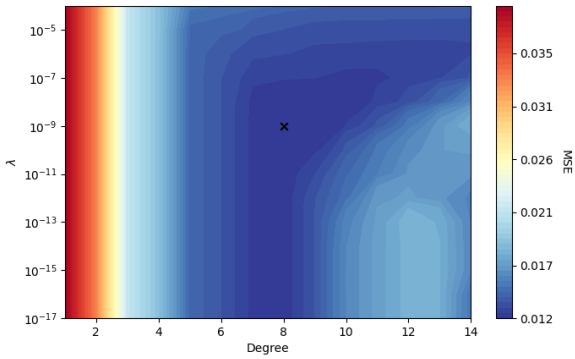

Figure 16: The MSE a as a function of a polynomial degree p and penalty parameter λ for Ridge with $N = 20$ and $c = 0.1$. Here plotted up to $p = 14$ using cross-validation with $k = 10$.

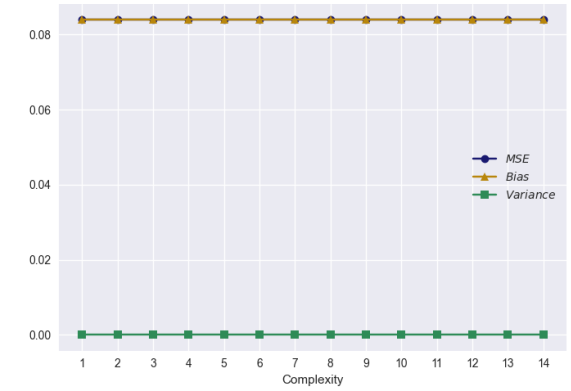

Figure 18: Bias-Variance tradeoff for Lasso using the bootstrap method with $N = 20$, $\lambda = 10^{-1}$, $c = 0.1$ and $B = N^2$ bootstrap iterations.

form compared to each other. This can be seen in figure 22. Previously, cross-validation has performed close to the performance of the case with no resampling, which is the best model this time as well. Now, it performs substantially worse.

4. Comparison of Regression Methods

We conclude that either OLS or Ridge would produce the best fit for the data from (19). They're both the same polynomial degree and provide, in practice, the same error, but keep in mind equation (14). The variance of the optimal parameters for OLS is larger than for Ridge. So we would prefer to go with Ridge.

	Degree(p)	Hyperparameter(λ)	MSE
OLS	6	-	0.01128402
Ridge	6	10^{-9}	0.01128348
Lasso	14	10^{-4}	0.01857086

Table I: The lowest MSE for each regression method achieved with no resampling and for which polynomial degree p and hyperparameter λ

B. Terrain Data

The terrain data we've chosen is from the Atlas mountain range in Morocco. The whole data set we've downloaded is shown in figure 23, but this is a set with dimen-

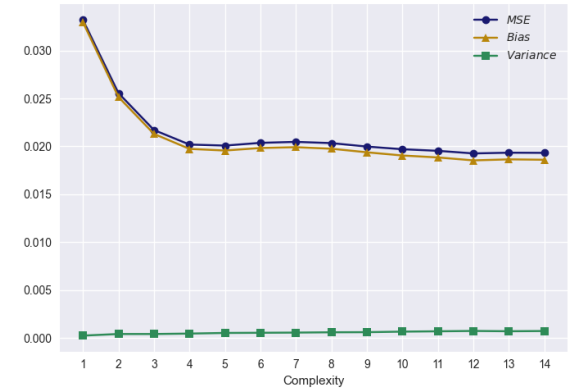

Figure 19: Bias-Variance tradeoff for Lasso using the bootstrap method with $N = 20$, $\lambda = 10^{-6}$, $c = 0.1$ and $B = N^2$ bootstrap iterations.

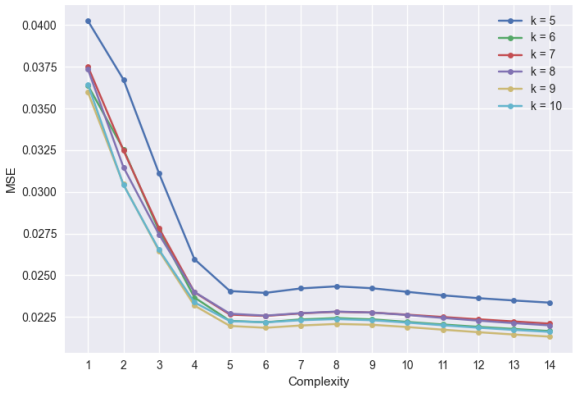

Figure 20: The MSE a as a function of a polynomial degree p for Lasso with $N = 20$, $\lambda = 10^{-6}$ and $c = 0.1$. Here, plotted up to $p = 14$ using cross-validation where the number of folds ranges from 5 to 10.

sions (3601×3601), so we're just going to limit ourselves to a smaller region within the region we've chosen. This is shown in figure 24. The size of this smaller region is (30×30), so we will always have $N = 30$. To get a broader understanding of what we're trying to fit, we plot a 3D figure of the data, which is shown in figure 25.

The target data is now the height of the terrain, but the input data we're going to use is still going to be between 0 and 1. Therefore, we scale as we did for the Franke function. Since our prediction and target data are large, the MSE will have a large number even though it's a good fit. Therefore, to make it more intuitive, we will mainly use the R^2-score. So now, we will again apply all the regression methods and cross-validation as a resampling technique.

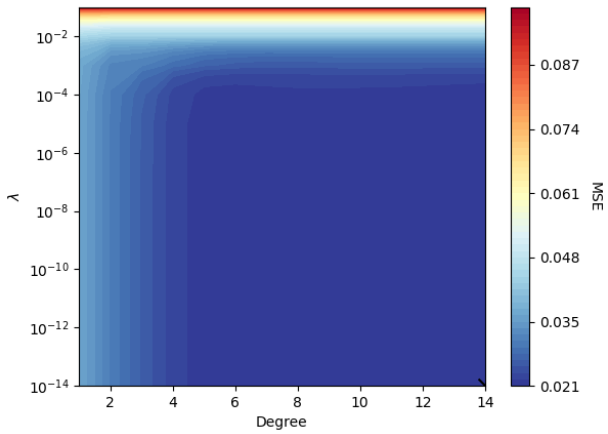

Figure 21: The MSE a as a function of a polynomial degree p and penalty parameter λ for Lasso with $N = 20$ and $c = 0.1$. Here plotted up to $p = 14$ using cross-validation with $k = 9$.

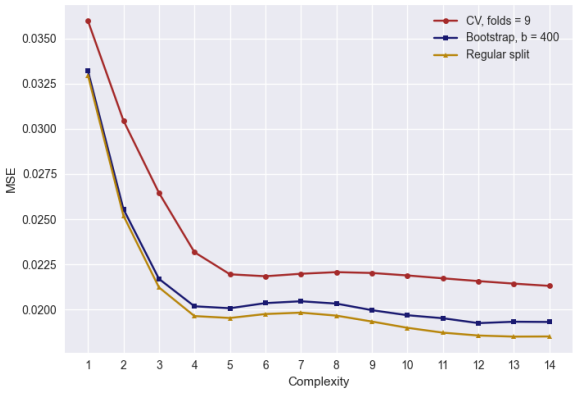

Figure 22: The MSE a as a function of a polynomial degree p for Lasso with $N = 20$, $\lambda = 10^{-14}$ and $c = 0.1$. Here, plotted up to $p = 14$ for three cases. a) No resampling, b) bootstrap with $B = N^2$ iteration and c) cross-validation with 9 folds.

1. *Regression models*

We're starting with OLS as usual, and we're going to plot the R^2-score using cross-validation with different amounts of folds. This can be seen in figure 26. We see that for 5 folds, the model performs worse than a straight line. Furthermore, the best fit is achieved for $k = 10$ with a polynomial degree of 10.

We will repeat this for both Ridge and Lasso regression. For both cases, we let $\lambda = 10^{-9}$. The results can be seen in figure 27 and 28. Both cases have the best performance at $k = 10$. The best fit is achieved at a

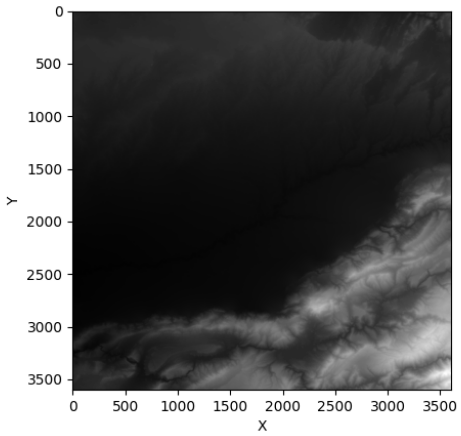

Figure 23: Image of a part of the Atlas mountain range

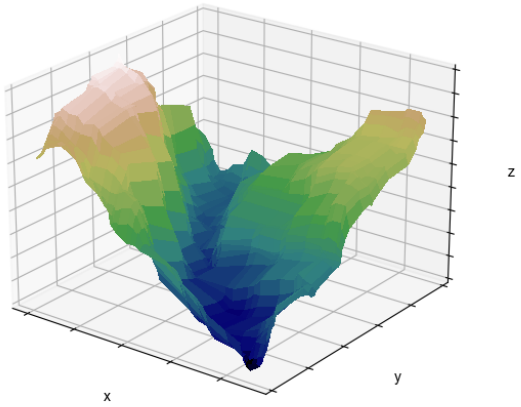

Figure 25: The terrain data we are going to fit.

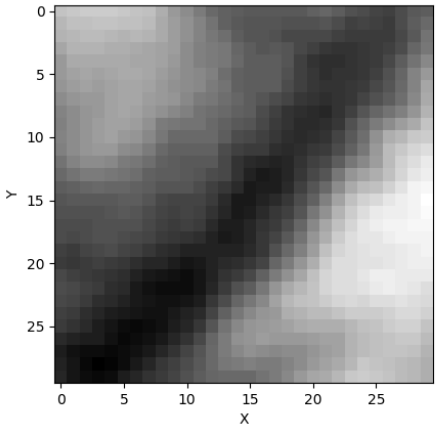

Figure 24: The terrain data we are going to fit.

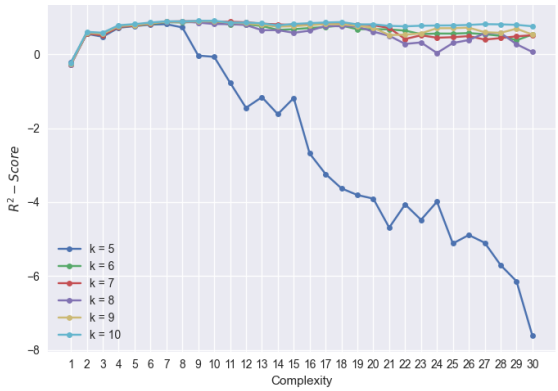

Figure 26: The R^2 a as a function of a polynomial degree p for OLS. Here, plotted up to $p = 30$ using cross-validation where the number of folds ranges from 5 to 10.

polynomial degree of 12 for Ridge and 27 for Lasso.

Lastly, we're going to find the optimal hyperparameter and polynomial degree combo. Starting with Ridge, a heatmap is plotted in figure 29. We find that we achieve the lowest error when $\lambda = 10^{-6}$ and $p = 17$. For the Lasso case, which is shown in figure 30, the lowest error is achieved when $\lambda = 10^{-14}$ and $p = 26$.

	Degree(p)	Hyperparameter(λ)	R^2
OLS	10	-	0.907532
Ridge	17	10^{-5}	0.913974
Lasso	26	10^{-14}	0.806012

Table II: The highest $R^2 - score$ for each regression method achieved using cross-validation with 10 fold, and for which polynomial degree p and hyperparameter λ

2. Comparison of Regression Methods

By looking at table 2, we see that Ridge provides the best fit with OLS right behind it. However, it's worth noting that Ridge has a significantly higher polynomial degree. So, if we want to save computational power, OLS

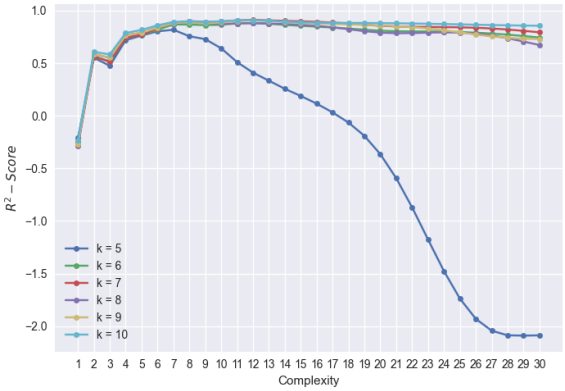

Figure 27: The R^2 a as a function of a polynomial degree p for Ridge with $\lambda = 10^{-9}$. Here, plotted up to $p = 30$ using cross-validation where the number of folds ranges from 5 to 10.

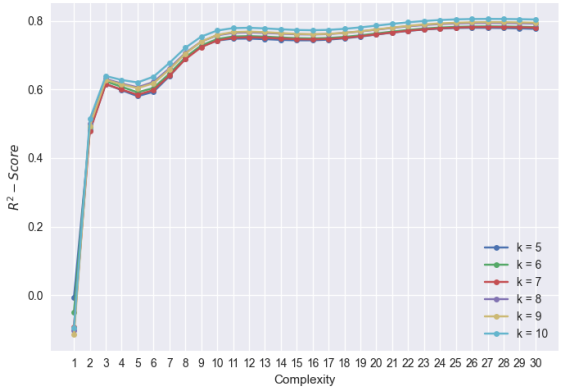

Figure 28: The R^2 a as a function of a polynomial degree p for Lasso with $\lambda = 10^{-9}$. Here, plotted up to $p = 30$ using cross-validation where the number of folds ranges from 5 to 10.

is a perfectly safe choice. In figure 31 we have plotted the predicted terrain using OLS. We see that it generalizes well, but the fine details are missing.

IV. CONCLUSION

We have applied three types of linear regression methods: Ordinary Least Squares, Ridge and Lasso. These methods were used to fit two data sets, one generated from the Franke function with some added noise (19), and the other is real terrain data. We've found that the model which gave the best fit for both data sets was Ridge, with OLS being close behind. Lasso, on the other hand, was computationally expensive and provided a far worse error. For all regression methods evaluating the data from

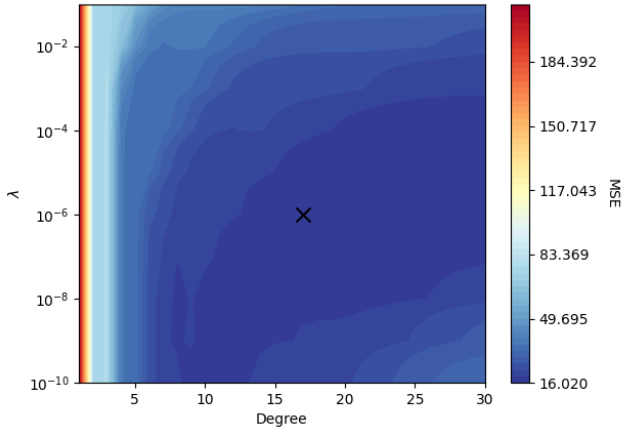

Figure 29: The MSE a as a function of a polynomial degree p and penalty parameter λ for Ridge. Here plotted up to $p = 30$ using cross-validation with $k = 10$.

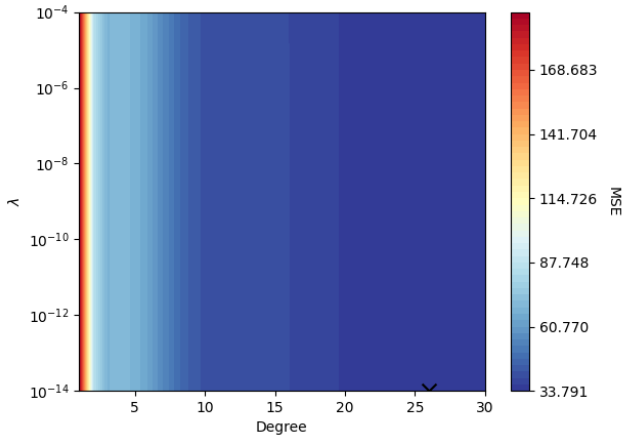

Figure 30: The MSE a as a function of a polynomial degree p and penalty parameter λ for Lasso. Here plotted up to $p = 30$ using cross-validation with $k = 10$.

(19), we experimented with the regular train-test split of the data, bootstrapping and cross-validation. It was found that a regular split provided the smallest error for all methods. We found that the lowest mean square error was 0.011284 for OLS and 0.011283 for Ridge for a polynomial of degree 6. For Lasso the MSE was found to be 0.01857 for a polynomial of degree 14. Ridge used a hyperparameter equal to 10^{-9}, while Lasso used 10^{-4}.

For the terrain data, we used cross-validation for all cases and experimented with the number of folds. We found that 10 folds gave the best result for all methods. We found the R^2-score to be 0.9075 for OLS, 0.9139 for Ridge and 0.806 for Lasso for a polynomial of degree 10,

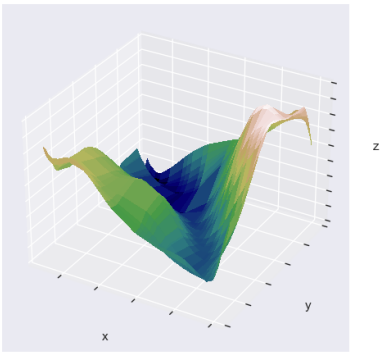

Figure 31: Predicted terrain data using OLS.

17 and 26, respectively. Ridge used a hyperparameter of size 10^{-5} and Lasso used 10^{-14}.

In addition, we explored the bias-variance tradeoff and how the penalty parameters affect these. We concluded that it dampens the variance and increases the bias as the parameter increases.

[1] M. Hjorth-Jensen. (2021). *Applied Data Analysis and Machine Learning.* Available at: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html (Accessed: 21 Oct 2022).

[2] W.N. van Wieringen. (2021). *Lecture notes on ridge regression.* arXiv:1509.09169v7.

[3] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning,* 2nd edt. New York: Springer, 2009.

[4] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow,* 2nd edt. Sebastopol, California: O'Reilly Media, 2019.