UiO : Department of Technology Systems
University of Oslo

University of Oslo

FYS-STK

Project 1

Authors:
Øien B. Lasse
Nilsen E. Lars

Supervisor:
Prof. Morten
Hjorth-Jensen

October 10, 2022

Abstract

To save the world from judgement day the paper explore ordinary least square, lasso and ridge regression with and without bootstrap and cross-validation. The paper successfully recreate the methods on Franke's function.

The paper then explores the same methods on some digital terrain data close to Stavanger and get the following results: Polynomial of 10th degree yields the best result vs computational time. 15th degree gives overall best results.

The paper also highlights the thoughts of the authors and evaluation of the course, also including what the authors have learned and experienced.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Before 2029 when the terminator is sent back from the future to kill Sara Connor, the machine learning community must have control over the advancing AI Skynet. This assignment is just the first step for the group members to contribute to learning and understanding the world of machine learning and thereby stop Skynet and judgment day.[1]

More specifically, this report will include the primary appliance of various regression methods and methodologies like Ordinary Least Squares, Lasso and Ridge regression, bootstrapping, and cross-validation. It will also highlight the advantages/disadvantages of the chosen methods. The report will start with performing the methodologies on a "made-up test function" called Franke's function. We do this to validate the implementation before we use it on real-life data. We also validate our methods by comparing it with scikit-learn built-in functions. As we have alluded, the report will then use the tested methodologies on some digital terrain data close to Stavanger.

Lastly, the authors of this report will make comments and thoughts on what they have learned and provide remarks on the course. This is meant as constructive feedback and should be interpreted as such. Throughout lectures, lab, and project work, the authors have gained new insights and learned more about concepts they had previously heard of, as well as learning to brand new concepts and broadening the general knowledge and interpretation of them.

[1]The exact date of judgment day is a bit unclear in the terminator series and has been pushed further away into the future with newer releases. Going by the earlier movies it has already happened and is therefore unavailable, but our group remains optimistic! You can read more at this link: https://screenrant.com/terminator-judgment-day-date-year

Chapter 2

THEORY

This chapter aims to introduce the necessary theory used in this report. To answer the problems of this paper, we must first introduce some well-known concepts in linear algebra and statistics

2.1 Coefficients and expected value

Assuming there exists a function $f(x)$ that is able to describe the inspected data, such that $y = f(x) + \epsilon$ where ϵ is normal distributed noise. And that $f(x)$ can be represented through linear regression equations. $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$, where $\boldsymbol{X}$ is the feature/design matrix, and $\boldsymbol{\beta}$ is the unknown parameters/coefficients. $\tilde{\boldsymbol{y}}$ is then the chosen model fitted to describe y.

Verifying the model‘s fit to the actual output can be done through various calculations such as the R^2 and the MSE. The R^2 gives an intuition of how good the model is in terms of percentage I.e how "many" of the observed variations in the model can be explained by the inputs in the model. The R^2 given in terms of the target data y and the model $\tilde{y}$ is given in the equation (2.1)

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{2.1}$$

The Mean Squared Error (MSE) is another typical cost function used to validate a model fitted to actual data. MSE describes the estimated value $\tilde{y}$ deviation from the observed data y squared. the MSE in a general sense and in terms of the cost function applied is presented in equation (2.2)

$$\begin{aligned} MSE &= (\boldsymbol{y} - \tilde{\boldsymbol{y}})^2 \\ C(\boldsymbol{X}, \boldsymbol{\beta}) &= \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \end{aligned} \tag{2.2}$$

Given the assumptions about $\tilde{\boldsymbol{y}}$ to be true, then through derivation of the cost function in equation (2.2) one obtains the optimal coefficient estimate by solving for $\boldsymbol{\beta}$. The full derivation can be obtained from [1]

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} \tag{2.3}$$

The inverse matrix is not always computational I.E; the determinant of the resulting matrix may have a determinant equal to 0. One technique used to compute the inverse is the singular value decomposition. Another widely used inverse is the Moore-Penrose pseudo inverse [2], which will be used throughout this report as it is easy to implement through python.

2.1.1 Ridge and Lasso

From the equation (2.3) one can derive the optimal parameter for Ridge regression. The regression follows the same assumptions that were made earlier about the model ($\tilde{\boldsymbol{y}}$), with an added hyperparameter λ. The cost function is thus defined as.

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda \boldsymbol{\beta}^T\boldsymbol{\beta} \tag{2.4}$$

Through the same procedure and derivation of the cost function, one obtains the optimal beta value for ridge regression.

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^T\boldsymbol{y}, \tag{2.5}$$

One way to interpret this hyperparameter lambda is the restriction of specific beta parameters through equation(2.6). Observed from the equation for large lambda values β_i goes towards zero.

$$\beta_i = \frac{y_i}{1 + \lambda} \tag{2.6}$$

For Lasso regression the cost function following by the l1 norm of the parameter beta can be written as

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda ||\boldsymbol{\beta}||_1 \tag{2.7}$$

The difference between lasso and ridge can be interpreted as; based on the hyperparameter lambda, lasso will remove the unnecessary parameters beta while ridge will dampen them.

$$\frac{d|\beta|}{d\boldsymbol{\beta}} = \text{sgn}(\boldsymbol{\beta}) = \begin{cases} 1 & \beta > 0 \\ -1 & \beta < 0 \end{cases} \tag{2.8}$$

Lasso regression does not lead to a smooth analytical expression; however, due to the sign convention of the l1 norm derivate. The sign of beta changes between -1 and one, but common practice in the machine learning community is to avoid the discontinuity by $\beta = 0$ to set it to 0. Lasso, in general, aims to reduce dimensionality and combines the good features of subset selection and ridge regression [3]. Both ridge and lasso have advantages when it comes to making predictions and reducing the likelihood of over-fitting.

2.1.2 Expectation and variance

Assuming there exists a function $f(x)$ such that $y = f(x) + \epsilon$ where ϵ is normal distributed noise $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ and is independent. $f(x)$ can be represented through

linear regression equations. $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$, where $\boldsymbol{X_{i,*}}\boldsymbol{\beta}$ is a non-random scalar. This implies that y_i follows a normal distribution.

The expected value of y can be calculated through the sample mean. As mentioned in this chapter, it is assumed that y can be written as a linear matrix product of $X\beta$. The estimated value can be calculated by the following equation(2.1.2)

$$E(y_i) = E(\boldsymbol{X}_{i,*}\boldsymbol{\beta}) + E(\epsilon_i) = (\boldsymbol{X_{i,*}}\boldsymbol{\beta})$$

$\mathrm{E}(\mathrm{y}_i)$ (2.9)

The expected value of the noise is zero since the sample mean is zero. Since $(\boldsymbol{X_{i,*}}\boldsymbol{\beta})$ is a non-random scalar, the expected value is equal to its true value I.E "you will always expect the value of 2 to be 2".

Variance describes the expected values square deviation from the mean. And since the expected value would be the sample mean, the variance of each element y_i can be calculated as follows.

$$\begin{aligned} Var(y_i) &= E[(y_i - E(y_i))^2] = E(y_i^2) - [E(y_i)]^2 \\ &= E[(\boldsymbol{X_{i,*}}\boldsymbol{\beta} + \epsilon_i)^2] - (\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2 \\ &= E[(\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2 + 2\epsilon_i \boldsymbol{X_{i,*}}\boldsymbol{\beta} + \epsilon_i^2] - (\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2 \\ &= (\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2 + 2E(\epsilon_i)(\boldsymbol{X_{i,*}}\boldsymbol{\beta}) + E(\epsilon_i^2) - (\boldsymbol{X_{i,*}}\boldsymbol{\beta})^2 \\ &= E(\epsilon_i^2) = Var(\epsilon_i) = \sigma^2 \end{aligned} \tag{2.10}$$

Hence $y \sim \mathcal{N}((\boldsymbol{X_{i,*}}\boldsymbol{\beta}), \sigma^2)$. y follows a normal distribution, with $\boldsymbol{X_{i,*}}\boldsymbol{\beta}$ as the mean and a variance equal to σ^2. That is, the mean is the nonrandom scalar product, and the variance is the noise variance.

Through the expression for $\hat{\beta}$ from equation(2.3). It is possible to derive the estimated value.

$$\begin{aligned} E(\boldsymbol{\beta}) &= E[(\boldsymbol{X^T X})^{-1}\boldsymbol{X^T Y}] \\ &= (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T} E(\boldsymbol{Y}) \\ &= (\boldsymbol{X^T X})^{-1}\boldsymbol{X^T X}\boldsymbol{\beta} \\ &= \boldsymbol{\beta} \end{aligned} \tag{2.11}$$

This result shows that the estimator of the parameterβ is unbiased. Meaning on average, it should produce correct parameters. The Variance can be calculated as:

$$\begin{aligned} \mathrm{Var}(\boldsymbol{\beta}) &= E\{[\boldsymbol{\beta} - E(\boldsymbol{\beta})][\boldsymbol{\beta} - E(\boldsymbol{\beta})]^T\} \\ &= E\{[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} - \boldsymbol{\beta}]\,[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} - \boldsymbol{\beta}]^T\} \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\, E\{\mathbf{Y}\,\mathbf{Y}^T\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\,\{\mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T + \sigma^2\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T \\ &= \boldsymbol{\beta}\,\boldsymbol{\beta}^T + \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T \;=\; \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1} \end{aligned} \tag{2.12}$$

The variance of each β coefficient can be estimated through: $\sigma^2(\beta_j) = \sigma^2\sqrt{(X^TX)^{-1}_{jj}}$ The variance of each beta coefficient can further be developed into the confidence interval of beta.

$$Ci = \mu_\beta \pm \frac{2\sigma_\beta}{\sqrt{n}} \tag{2.13}$$

2.2 Bias-variance trade-off.

One problem that consists throughout any analysis of real-world problems is the lack of data. This section aims to introduce concepts widely used to deal with such problems.

The bootstrap method is the first to be introduced; the process resamples from the training data. The way bootstrap works is: It takes a new sample from the training data, which can be the exact data point multiple times, and computes the statistical value (mean, median, etc..). Then the idea is to store this statistical value, then repeat the process. One obtains the confidence interval over the statistical value by analyzing the stored values. And can be further used to compute the data's probability density function (PDF). The probability density function given by the bootstrap can be viewed in equation (2.14)

$$\tilde{p}(z) = \frac{1}{\sqrt{2\pi}(\sigma/\sqrt{m})} \exp\left(-\frac{(z-\mu)^2}{2(\sigma/\sqrt{m})^2}\right) \tag{2.14}$$

The bootstrap can be viewed in more rigorous terms through the paper by Efron[4].

Another common problem with splitting the data into training and test sets is that the data model is calculated on the training set and then applied to the test set. This can be misleading since the model created may not be the best fit. It raises the question, "what if other points were selected for training and testing?".

The cross-validation method addresses this problem. Utilizing the cross-validation method ensures that all data points have been used for training and testing. It works intuitively like this; divide the entire dataset into equal size blocks. Then select one block for testing and the rest for training. Keep track of how well the parameters were fitted to the test data, then choose a new block for testing and the others for training. Repeat the process for all blocks chosen. The cross-validation method can also be utilized to see which classifying algorithm has the best fit for one's data.

2.2.1 Mathematics of Bias Variance trade off

The cost function that is sought to be minimized is given as:

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

Earlier in this chapter, there was introduced some assumptions for y_i. Based on the same assumptions, the cost function can be rewritten as.

$E[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \frac{1}{n}\sum_i (f_i + \epsilon_i - E[\tilde{\boldsymbol{y}}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - E[\tilde{\boldsymbol{y}}])^2 + \sigma^2$.

The f_i comes from the assumption that a function exists that describes the data plus some noise ϵ_i. Thus the first term is defined as the bias term, thought of as faults in the model given by the assumptions made in the model. The second term describes the variance of the model $\tilde{\boldsymbol{y}}$, how much each point in the model deviates from the sample mean of the model. Lastly, the σ^2 is the variance of the noise. Also, it was shown that this was the variance of the data y.

Utilizing a more compact notation:

$E[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = E[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^2]$

Then adding and subtracting the expected value of the mode $E(\tilde{\boldsymbol{y}})$. One obtains through implementing the assumptions given previously in this chapter.

$$E[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = E[(\boldsymbol{y} - E[\tilde{\boldsymbol{y}}])^2] + Var[\tilde{\boldsymbol{y}}] + \sigma^2 \tag{2.15}$$

This can be written in terms of the bias squared and the variance of the model.

$$E[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = (Bias(\tilde{\boldsymbol{y}}))^2 + Var[\tilde{\boldsymbol{y}}] + \sigma^2 \tag{2.16}$$

A common problem in machine learning is to find the model that is not too under-fitted and not over-fitted; this is the aim of the bias-variance trade-off. Minimize the cost function so that the variance does not get too large and the bias gets too large.

Chapter 3

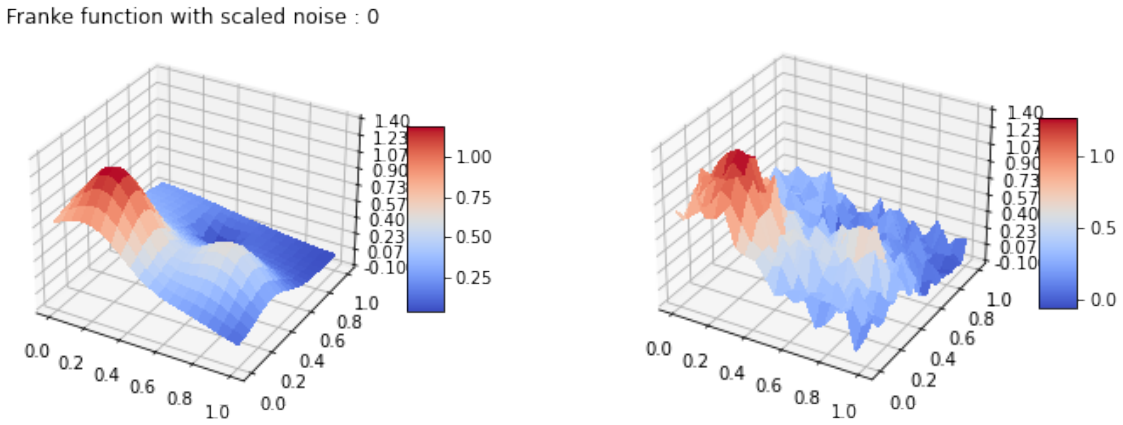
METHOD

This chapter will detail what has been done during the project and relate the theory introduced to the solution provided. The project task is split into two parts. The first part uses uniform "simple" data to analyze the different techniques, while the second will apply these techniques to real-world data. For the method, the uniform created data fitted to a Franke function will be provided, and the actual data will be discussed in the result chapter.

3.1 Analyzing different regression methods fitted to Franke function

The first part of a given project is to analyze the data collected; this section, as mentioned, is the "vanilla" data and the Franke function introduces the concepts.

The data for the first part of the project is a 20x20 mesh of uniform distribution between 0 and 1. This data is fitted to a Franke function. The target function is observed with and without noise in figure(3.1)



(a) Smooth Franke function

(b) Franke function with noise

Figure 3.1: Output function

The figure illustrates the output/target that the various regression methods are

implemented and fitted. The rest of this section will deliver the methods used for each regressions. Common features and limitations in all models are listed as follows:

1. R^2 and MSE is observed over a model complexity of up to five polynomials
2. The same design/feature matrix is used in all regression methods.
3. When analyzing bias-variance trade-off with bootstrap, the complexity is increased to 12, so noticeable change can be viewed.
4. analyzing the cross-validation of each method is done with a complexity of 12 polynomials to see the change noticeably.

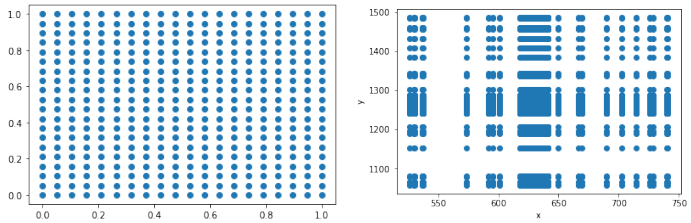
This ensures that all methods are analyzed and discussed under the same assumptions and such that none of the methods has a biased advantage.

The design matrix is provided in the code below and will be the same design matrix for all the topics discussed. The design matrix was computed in the following order.

```
def X_generator(x, y, n ):
if len(x.shape) > 1:
    x = np.ravel(x)
    y = np.ravel(y)

N = len(x)
l = int((n+1)*(n+2)/2) # Number of columns in beta
X = np.ones((N,l))

for i in range(1,n+1):
    q = int((i)*(i+1)/2)
    for k in range(i+1):
        X[:,q+k] = (x**(i-k))*(y**k)

return X
```

This ensures that the design matrix includes all the cross points I.e $z(x, y) = b_0 + b_1x + b_2xy + b_3x^2 + b_4y^2 + b_5x^2y + ...$

Given the data is uniformly distributed between 0 and 1, there is no need for scaling. Scaling is an important feature, given that the dataset includes outliers and is unevenly distributed. Illustrated in figure(3.2) it is observed scaled vs unscaled data. In the case with scaled data, figure(3.2a) is the case this report where data is created for the sake of analysis. However, for the sake of good general practice, the data will be scaled utilizing sickit-learns standards scaler[5].



(a) Scaled data (b) Unscaled data

Figure 3.2: Data points

3.2 Various regression fitted to the Franke function.

This section will include the various codes in pseudo format to describe the techniques used. The codes will not include the functions and creating of empty lists, etc. But rather show the structure, full code is provided in the git hub link.

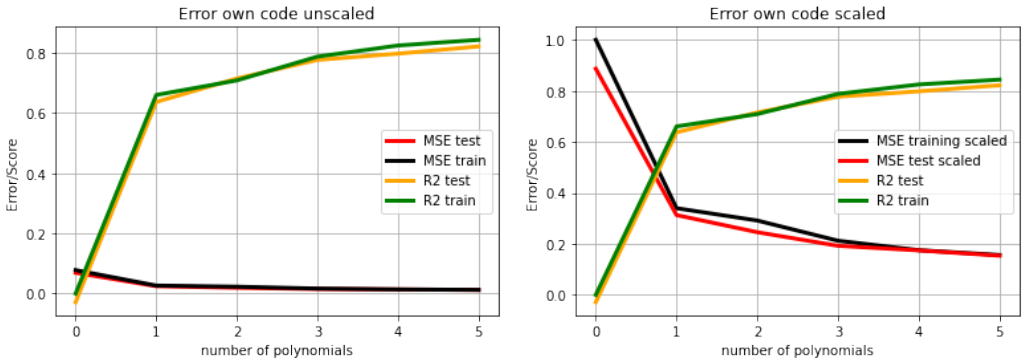
3.2.1 Finding the Polynomial degree

An excellent approach when it comes to regression is to figure out at what polynomial the model performs best. This is validated by computing the R^2 and MSE equations (2.1& 2.2). The code can be viewed:

```
for i in range(numOfPoly_max +1 ):
poly = i
print(numOfPoly)
X = X_generator(x_flat,y_flat, poly)
Xs = X_generator(x_flat,y_flat, poly)
print(Xs)
if i > 0:
    Xs = np.delete(Xs,0,1)

X_train, X_test, z_train, z_test = traningDataGenerator(X,z)
X_train_s, X_test_s, z_train_s, z_test_s = traningDataGenerator(Xs,z)
X_offset = np.mean(X_train,axis=0)
z_offset = np.mean(z_train,axis=0)

#scaling the data
X_train_scaled, X_test_scaled, z_train_scaled, z_test_scaled = scale(X_train_s
X_offset_scaled = np.mean(X_train_scaled,axis=0)
z_offset_scaled = np.mean(z_train_scaled,axis=0)
# matrix inversion to find beta
beta = np.linalg.pinv(X_train.T @ X_train) @ X_train.T @ z_train
intercept = np.mean(z_offset - X_offset @ beta)
beta_scaled = np.linalg.pinv(X_train_scaled.T @ X_train_scaled) @ X_train_scal
intercept_scaled = np.mean(z_offset_scaled - X_offset_scaled @ beta_scaled)

# and then make the prediction
```

```
ztilde = X_train @ beta
ztilde_scaled = X_train_scaled @ beta_scaled + z_offset_scaled #scaled
zpredict = X_test @ beta
zpredict_scaled = X_test_scaled @ beta_scaled + z_offset_scaled  #scaled

R2_test.append(R2(z_test,zpredict))
R2_test_scaled.append(R2(z_test_scaled,zpredict_scaled)) #scaled

R2_train.append(R2(z_train,ztilde))
R2_train_scaled.append(R2(z_train_scaled,ztilde_scaled)) #scaled

MSE_train.append(MSE(z_train,ztilde))
MSE_train_scaled.append(MSE(z_train_scaled,ztilde_scaled)) #scaled

MSE_test.append(MSE(z_test,zpredict))
MSE_test_scaled.append(MSE(z_test_scaled,zpredict_scaled)) #scaled

numOfPoly_index.append(poly)
beta_index.append(beta)
beta_index_scaled.append(beta_scaled) #scaled
```

The code compares bot scaled and unscaled data; this is to highlight how subtracting the mean and dividing by the standard deviation impacts the results. Notice also the use of pinv in the code; from NumPy, this is the pseudo inverse as introduced in theory chapter [6].



(a) unscaled cost functions (b) Scaled cost functions

Figure 3.3: Cost functions up to polynomial degree 5

From the figure, it is observed that a polynomial degree of 5 would yield the best results, and the lowest MSE comes from the unscaled case. However, when further analyzing the parameter β within the confidence interval. It is observed that the scaled case has less variance of the parameters, which in turn can be interpreted as not overfitted. This could explain the low value of the arbitrary low MSE of the unscaled case. Throughout the project, the data will be scaled as is standard practice.

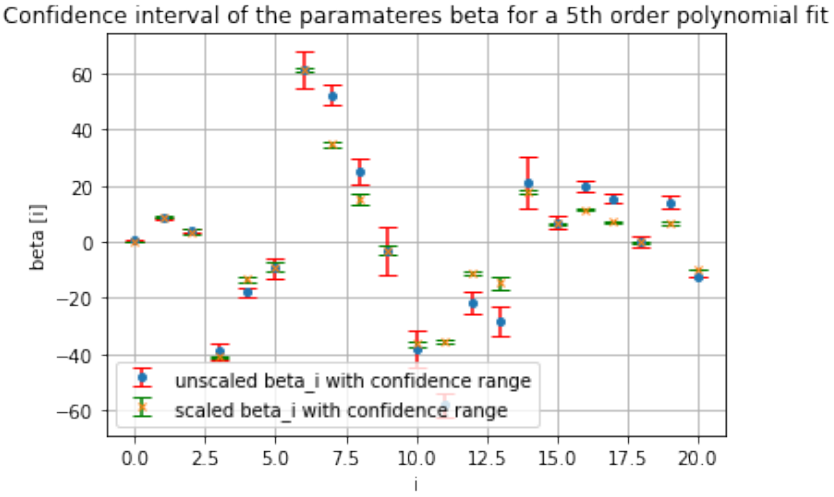


Figure 3.4: Confidence interval of β

From the figure discussed, the beta parameters, when scaling, has a better confidence interval compared to the unscaled. This provides a better argument for why utilizing scaling even on the vanilla dataset as it gives less uncertainty in the selection of beta parameters.

3.2.2 Bootstrap

First to highlight the bias-variance trade-off is the bootstrap. As explained in the theory chapter, this method aims to "create" more data. So before discussing other techniques, bootstrap will be applied to the OLS. Since figure (3.3) shows that increasing the polynomial degree up until 5 has a promising result, the interesting discussion would be to increase the polynomial further. The behaviour can be observed in figure (3.5).

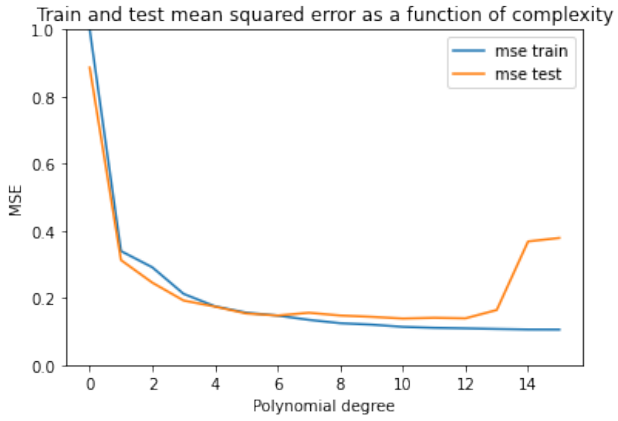


Figure 3.5: Train vs test MSE over complexity

From the figure, it is observed that the test MSE is worse after polynomial degree 6 and best around 5 to 6. This could be due to the lack of data (20x20 mesh). What it is interpreted as by the authors is that as the complexity increase, the variance

increase, and the bias decreases. in other words, the model goes from underfitting to overfitting and should be discussed more closely.

From the Theory, it was introduced that the bootstrap could be used to more accurately give the actual expected value of statistical properties; variance and bias are such properties. There would also be interesting to see how the bias-variance changes with an increased number of data points.

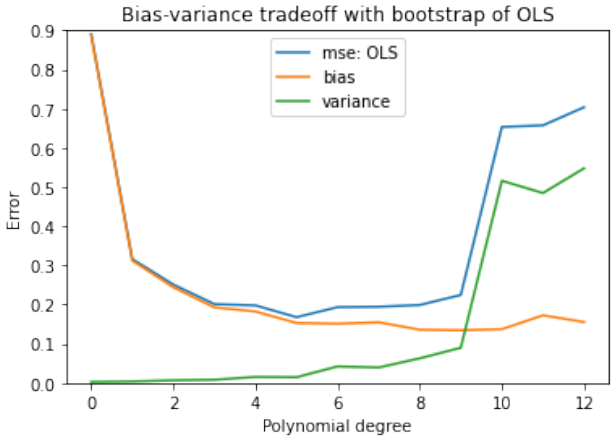


Figure 3.6: Bias variance trade-off OLS.

Figure(3.6) illustrates that after polynomial degree 4, the variance increases as the bias decreases, and after polynomial 9, the variance is dominant. Thus the bootstrap of OLS indicates that the MSE is at its lowest around 5 polynomial degrees

For illustrative purposes, there was made a test run with 55 data points and an increased polynomial degree to be observed in figure(3.7)

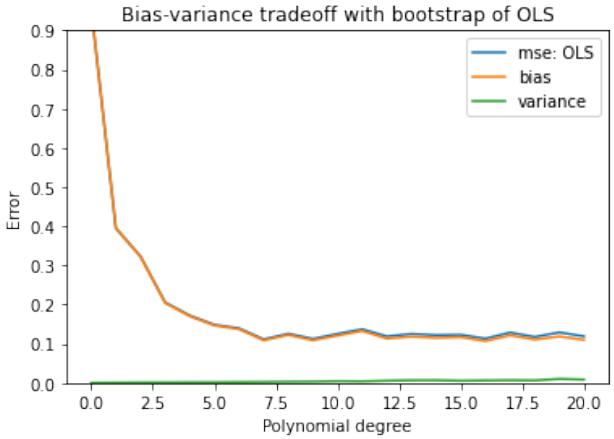


Figure 3.7: Bias variance trade-off OLS 55 data points.

As expected, with an increased number of data points, one can achieve higher polynomial degrees at more stable bias and variance. Thus a different polynomial degree would five a better prediction.

The code below illustrates the structure of the bias-variance code with bootstrap for all regression methods. The next subsection will go more in-depth on the Ridge and lasso

```
for poly in range(numOfPoly_max +1):

 i = poly
 X = X_generator(x_flat,y_flat, poly)

 if poly > 0: # Remove the intercept colum from the
     X = np.delete(X,0,1)

 X_train, X_test, z_train, z_test = traningDataGenerator(X,z)
 poly_degree = int((poly+1)*(poly+2)/2) # Trenger en forklaring

 I = np.eye(poly_degree,poly_degree)
 if poly > 0: # Fjerner intercept kolonnenn og første rad fra identitess matris
     I = np.eye(poly_degree-1,poly_degree-1)


 X_train, X_test, z_train, z_test = scale(X_train, X_test, z_train, z_test)


 #
 z_pred = np.empty((z_test.shape[0], n_bootstraps))
 z_pred_ridge = np.empty((z_test.shape[0], n_bootstraps))
 z_pred_lasso = np.empty((z_test.shape[0], n_bootstraps))
 z_tilde = np.empty((z_train.shape[0], n_bootstraps))
 z_tilde_ridge = np.empty((z_train.shape[0], n_bootstraps))
 z_tilde_lasso = np.empty((z_train.shape[0], n_bootstraps))
 z_pred_ridge_s = np.empty((z_test.shape[0], n_bootstraps))
 for j in range(n_bootstraps):
     #Resampling data
     x_bs,z_bs = resample(X_train, z_train)

     #Caluclating the parameters beta for OLS and Ridge
     beta_ols = np.linalg.pinv(x_bs.T @ x_bs) @ x_bs.T @ z_bs
     beta_ridge = np.linalg.pinv(x_bs.T @ x_bs + lmb_ridge[i] * I) @ x_bs.T @ z

     #Making predictions
     z_pred[:,j] = X_test @ beta_ols + np.mean(z_bs,axis = 0)
     z_tilde[:,j] = X_train @ beta_ols + np.mean(z_bs, axis = 0)
     z_pred_ridge[:,j] = X_test @ beta_ridge + np.mean(z_bs, axis = 0)

     z_tilde_ridge[:,j] = X_train @ beta_ridge + np.mean(z_bs, axis = 0)

     #validating that right usage of sickit learn
     Ridge_model = linear_model.Ridge(lmb_ridge[i],fit_intercept = True)
     Ridge_model.fit(x_bs,z_bs)
```

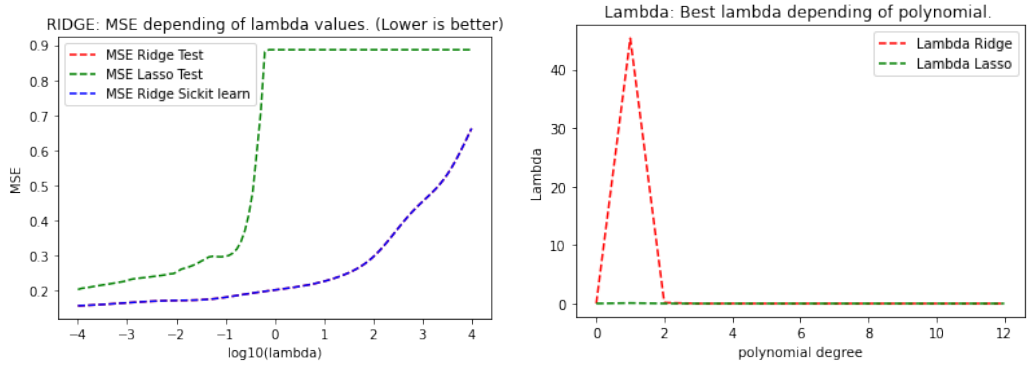
```
    z_pred_ridge_s[:,j] = Ridge_model.predict(X_test)


    #Same for lasso
    RegLasso = linear_model.Lasso(lmb_lasso[i],max_iter=1e5, tol=0.1,fit_inter
    RegLasso.fit(x_bs,z_bs)
    z_pred_lasso[:,j] = RegLasso.predict(X_test)
    z_tilde_lasso[:,j] = RegLasso.predict(X_train)
```

3.2.3 Ridge and Lasso Regression

To sum it up briefly the implementation of Ridge and Lasso regression is done by adding a lambda value to each polynomial and see if the MSE is improved. The best lambda value is then stored. The implementation is done with 100 lambda values spanning from 0.0001 to 10000. Ridge is implemented manually in the beta calculation. There is also an implementation of Ridge using scikit-learn, this is just done for verification purposes. Lasso is implemented with scikit-learn.

```
numberOfLambdas = 100 #number of lamdas
lambdas = np.logspace(-4, 4, numberOfLambdas)

    for i in range(numberOfLambdas):
    lmb = lambdas[i]
    Ridge_beta = np.linalg.pinv(X_train.T @ X_train + lmb*I) @ X_train.T @ z_train
    Ridge_model = linear_model.Ridge(lmb,fit_intercept = True)
    Ridge_model.fit(X_train,z_train)
    z_pred_ridge_s = Ridge_model.predict(X_test)
    # and then make the prediction

    ypredictRidge = X_test @ Ridge_beta  + z_intercept
    MSERidgePredict[i] = MSE(z_test,ypredictRidge)
    RegLasso = linear_model.Lasso(lmb,max_iter=1e5, tol=0.1,fit_intercept=True)

    RegLasso.fit(X_train,z_train)
    ypredictLasso = RegLasso.predict(X_test)

    MSELassoPredict[i] = MSE(z_test,ypredictLasso)
    MSE_Ridge_sklearn[i] = MSE(z_test,z_pred_ridge_s)
    if i==0:
        RidgeBestLambdaIndex = 0
        LassoBestLambdaIndex = 0

    else :
        if MSERidgePredict[i] < MSERidgePredict[RidgeBestLambdaIndex] :
            RidgeBestLambdaIndex = i
        if MSELassoPredict[i] < MSELassoPredict[LassoBestLambdaIndex] :
            LassoBestLambdaIndex = i
```

Plotting the best lambda values for both Ridge and Lasso it becomes clear that, for the most part, the lower the value of lambda, the better the implementation. This tells us that the best implementation was OLS and adding any additional lambda values mostly just made the fit worse. This is expected from Franke's function as it is best described with OLS.



(a) Mse as a function of lambda polynomial degree 10

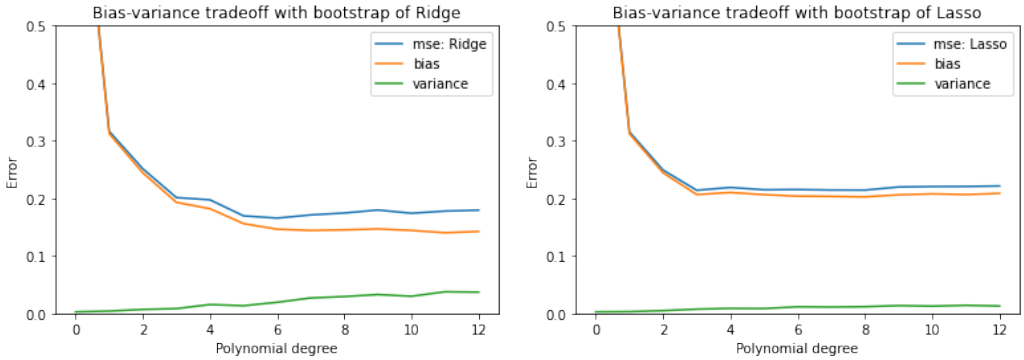
(b) Lambda values as a function of complexity

Figure 3.8: Lambda values for ridge and lasoo

Table 3.1: Lambda values for lasso and ridge as a function of complexity

Complexity	Lasso_lambda	Ridge_lambda
0	0.000100	0.000100
1	0.081113	45.348785
2	0.001630	0.141747
3	0.000100	0.000100
4	0.000100	0.005995
5	0.000100	0.000100
6	0.000534	0.000100
7	0.000774	0.000100
8	0.000368	0.000100
9	0.000305	0.000100
10	0.000305	0.000100
11	0.000368	0.000100
12	0.000534	0.000100

Using the best lambda value for each polynomial the paper then explore the bias-variance trade-off of Ridge and Lasso implemented with bootstrap and a polynomial degree of maximum 12.



(a) Bias variance trade-off Ridge. (b) Bias variance trade-off Lasso

Figure 3.9: Bias and variance tradeoff for Ridge and Lasso

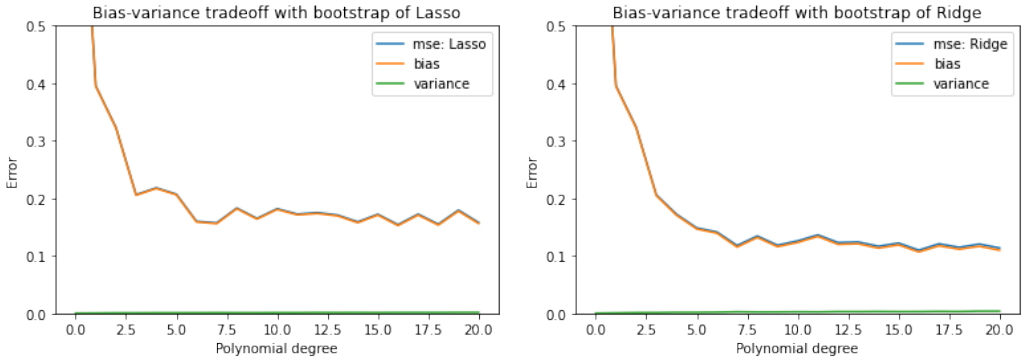
As seen from the figures both Ridge and Lasso experience lower variance and slightly higher bias. that is, the change in bias and variance happens at a slower rate when compared with the ordinary least squares. This is interpreted as the OLS best fit the Franke function as both ridge and lasso seem to flatten more towards the increasing complexity. Another noticeable difference from the OLS, as mentioned in theory, is that the shrinking of unnecessary parameters prevents overfitting. Thus leading to the flattening of the curve. A deeper analysis of the 20x20 mesh can be viewed in the table.

Table 3.2: Bias-variance trade-off with bootstrap

MSE_OLS	MSE_Ridge	MSE_Lasso	Bias_OLS	Bias_Ridge	Bias_Lasso	Var_OLS	Var_Ridge	Var_Lasso
0.889927	0.889927	0.889927	0.887159	0.887159	0.887159	0.002768	0.002768	0.002768
0.316316	0.316316	0.315262	0.312284	0.312284	0.312146	0.004032	0.004032	0.003116
0.251269	0.251269	0.249056	0.244303	0.244303	0.244247	0.006965	0.006965	0.004809
0.201287	0.201287	0.213961	0.192935	0.192936	0.206513	0.008353	0.008351	0.007448
0.198027	0.197369	0.218814	0.182475	0.181940	0.210119	0.015552	0.015429	0.008695
0.167989	0.169492	0.214784	0.153176	0.156093	0.206305	0.014813	0.013399	0.008479
0.193589	0.165612	0.215212	0.151218	0.146295	0.203707	0.042371	0.019317	0.011505
0.194579	0.171126	0.214362	0.154970	0.144232	0.203289	0.039610	0.026894	0.011073
0.199112	0.174453	0.214241	0.136107	0.145156	0.202510	0.063005	0.029297	0.011730
0.224623	0.179630	0.219637	0.134658	0.146809	0.206057	0.089965	0.032822	0.013580
0.653433	0.174029	0.220400	0.137357	0.144204	0.207751	0.516076	0.029825	0.012649
0.657368	0.177795	0.220584	0.172691	0.140178	0.206494	0.484677	0.037617	0.014090
0.703545	0.179355	0.221419	0.155713	0.142354	0.208525	0.547832	0.037001	0.012894

The table is computed on the same complexity where each row describes the complexity of the polynomial and corresponding mse, bias, and variance for each regression in its respective columns.

The same calculation is done with a mesh grid of 55 values. The result is shown below:



(a) Bias Variance trade-off Lasso 55 data points.

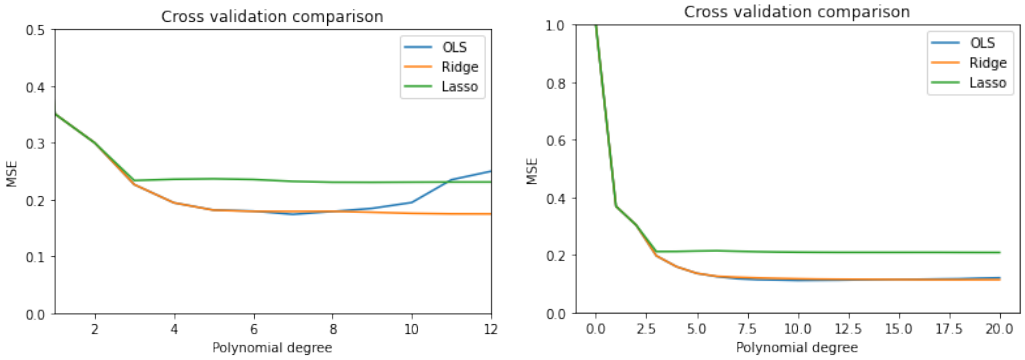
(b) Bias Variance trade-off Ridge 55 data points.

Figure 3.10: Bias variance trade ff with 55 data points Ridge and Lasso

Observed as in the OLS case, there is more signs of underfitting than overfitting in this region, ranging up to a complexity of 20.

3.2.4 Cross validation

To ensure that the selection of training and test data is not biased towards the best fit. The theory chapter introduced the cross-validation method, which has been utilized and analyzed to verify which method best fits the Franke function.



(a) Cross validation 20 datapoints

(b) Cross validation 55 datapoints

Figure 3.11: Cross validation

As observed from the figures, the increase of data points constitutes that there is some difference between ridge and OLS for higher complexities which is to be expected as the Franke function is best described by ordinary least squares. For the original 20x20 mesh, the OLS slightly outperforms the ridge before the variance increases, and OLS enters a region of overfitting. Compared to the bootstrap, the minimum MSE is here found at polynomial 7 and not 5; this leads to believe that the mix of training and testing data is important for selecting the proper optimal parameters. More clearly, this can be viewed in the table (3.3)

Table 3.3: Cross validation score compariosn 20 datapoints

Complexity	mse_cval_OLS	mse_cval_Ridge	mse_cval_Lasso
0.0	1.015795	1.015795	1.015795
1.0	0.350721	0.350721	0.350720
2.0	0.299986	0.299986	0.299928
3.0	0.226429	0.226428	0.233562
4.0	0.194314	0.194266	0.235754
5.0	0.181533	0.181340	0.236584
6.0	0.179721	0.179224	0.235342
7.0	0.173971	0.179144	0.232013
8.0	0.178988	0.179271	0.230483
9.0	0.184461	0.177590	0.230335
10.0	0.194783	0.175727	0.230617
11.0	0.234875	0.174810	0.230821
12.0	0.249836	0.174668	0.230871

The cross-validation was computed accordingly:

```
for degree in range(0,max_degree + 1):
degree_list[degree] = degree
X = X_generator(x_mesh,y_mesh, degree)
if degree > 0:
    X = np.delete(X,0,1)
mse_OLS_split = np.zeros(k)
mse_Ridge_split = np.zeros(k)
mse_Lasso_split = np.zeros(k)
cv_split = 0


for train_ind, test_ind in kfold.split(X):
    X_train = X[train_ind]
    X_test = X[test_ind]
    z_train = z[train_ind]
    z_test = z[test_ind]

    X_train, X_test, z_train, z_test = scale(X_train, X_test,z_train,z_test)

    #Creating models
    OLS_model = linear_model.LinearRegression(fit_intercept = True)
    OLS_model.fit(X_train, z_train)

    Ridge_model = linear_model.Ridge(lmb_ridge[degree],fit_intercept = True)
    Ridge_model.fit(X_train,z_train)

    Lasso_model = linear_model.Lasso(lmb_lasso[degree],max_iter=1e5, tol=0.1,f
```

```
        Lasso_model.fit(X_train,z_train)

        #Predicting
        z_pred_ols = OLS_model.predict(X_test)
        z_pred_ridge = Ridge_model.predict(X_test)
        z_pred_lasso = Lasso_model.predict(X_test)

        #Calculating MSE for each fold
        mse_OLS_split[cv_split] = mean_squared_error(z_test, z_pred_ols)
        mse_Ridge_split[cv_split] = mean_squared_error(z_test, z_pred_ridge)
        mse_Lasso_split[cv_split] = mean_squared_error(z_test, z_pred_lasso)

        cv_split += 1

    mse_Ridge_cval[degree] = np.mean(mse_Ridge_split)
    mse_OLS_cval[degree] = np.mean(mse_OLS_split)
    mse_Lasso_cval[degree] = np.mean(mse_Lasso_split)
```

Chapter 4

Result and Discussion

This chapter will present and discuss results around the data on the terrain in Norway. The methods used are the same as those shown in the method chapter. And the code implementation is the same.

4.1 The data

Firstly the data selected is data given by the professor of this course. The data represent the terrain in Norway. The file selected is the Norway 1 file and can be viewed in figure(5.1)

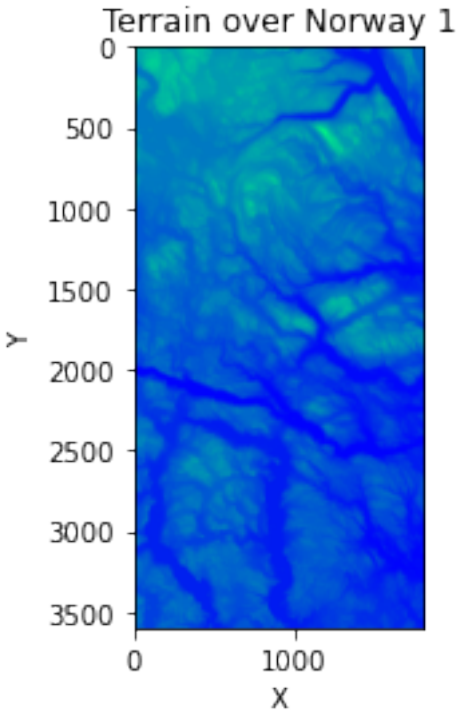
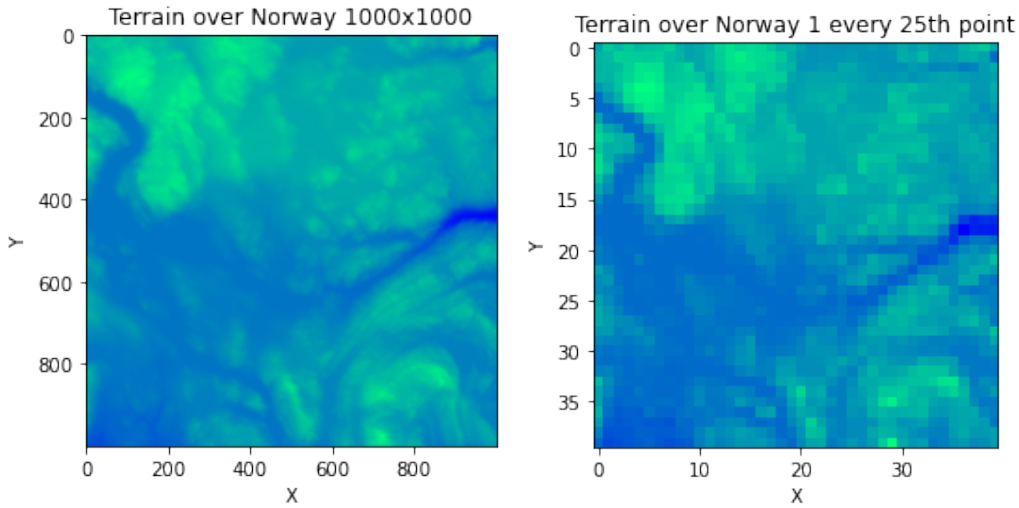


Figure 4.1: Terrain data, target data

This file is large, leading to an increased computational time for re-sampling techniques. The data file was scaled down; the chosen location was selected from

(X =1000 and Y=1000). The argument for choosing this data portion is that it will incorporate a spread variation between low and high curves.

It was chosen to minimize the number of pixels to further reduce computational time.



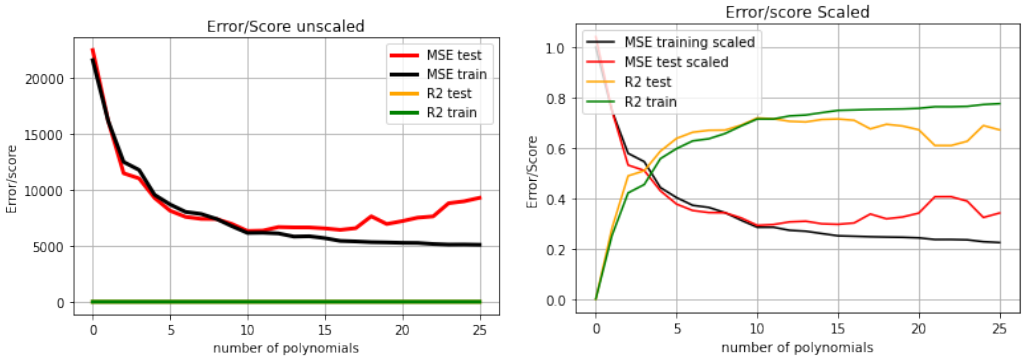
(a) Terrain data for $X, Y = 0 - 1000$ (b) every 25th point $X, Y = 0 - 1000$

Figure 4.2: Terrain data selected and applied

By selecting every 25th point, the terrain is still readable and, most crucial; computational ease is achieved. Further, the terrain data is scaled using the standard scaler; this is to handle some noise and further smoothen the target data as the input data to the design matrix is uniformly distributed data points between 0 and 1.

4.2 Ordinary least squares and optimal polynomial

As introduced in the method, the first analysis is of the scaled vs. unscaled case to decide the best-order polynomial.



(a) Unscaled OLS on real data (b) Scaled OLS on real data

Figure 4.3: Scaled vs unscaled OLS as a function of complexity

As observed from the figure, the unscaled case gives an unacceptable MSE. Therefore the scaled case is the only viable option. From the figure, it is also observed from

the selected data points that a 10th-degree polynomial would describe the terrain the best for 1 simple run on the specified train and test data.

4.3 Bias variance trade off

To be able to view the changes in the bias-variance trade-off, it was chosen to run for 25 polynomial degrees. Also to see if the effect of resampling would give a better estimate for higher polynomial degrees than the first run in figure(4.3).

4.3.1 Finding optimal lambda

Table 4.1: Optimal lambdas for Ridge and Lasso

Complexity	Lasso_lambda	Ridge_lambda
0	0.000100	0.000100
1	0.015199	0.000100
2	0.000100	0.000100
3	0.000175	0.170735
4	0.000100	0.000643
5	0.000100	0.000100
6	0.000100	0.000100
7	0.000100	0.000100
8	0.000100	0.000100
9	0.000100	0.000100
10	0.000100	0.000100
11	0.000100	0.000100
12	0.000100	0.000100
13	0.000100	0.000100
14	0.000100	0.000100
15	0.000100	0.000100
16	0.000100	0.000100
17	0.000100	0.000100
18	0.000100	0.000100
19	0.000100	0.000100
20	0.000100	0.000100
21	0.000100	0.000100
22	0.000100	0.000100
23	0.000100	0.000100
24	0.000100	0.000100
25	0.000100	0.000100

From the table above and figure 4.4 below it is observed that for some of the early polynomials it is best to add lambda values for both Ridge and Lasso. This is clearly not the case for the higher polynomials. Both ridge and lasso in the case with real-world data, aims to shrink/remove the first parameters but then revert to that OLS or close to it is the best fit for the target data.

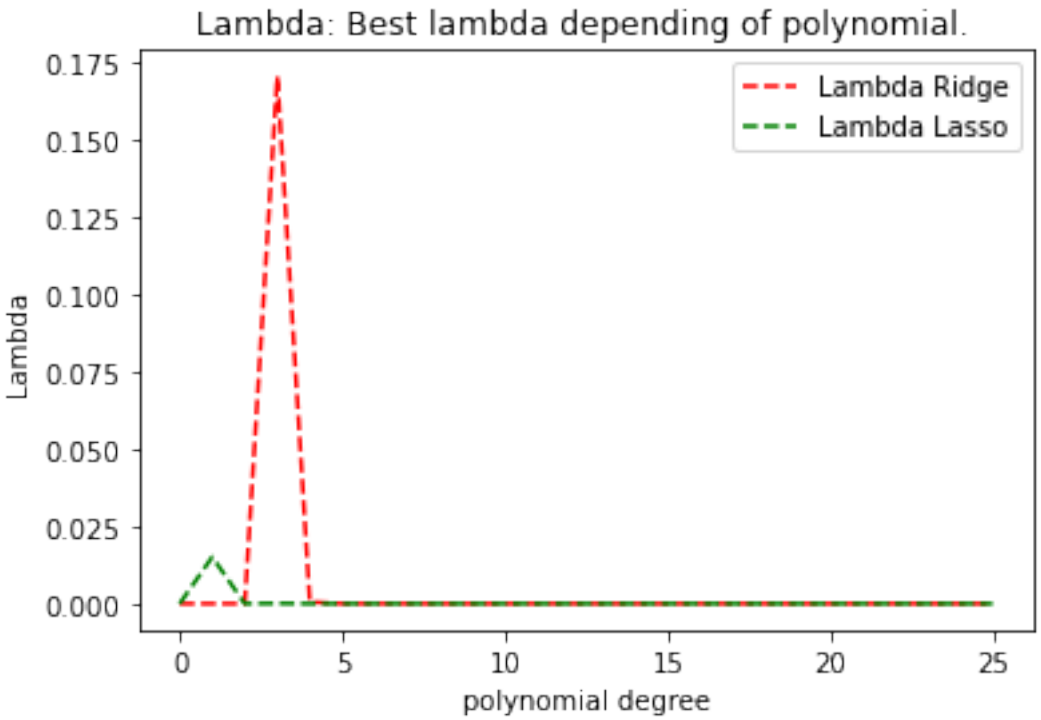


Figure 4.4: Best lambda values for ridge and lasso on target data

4.3.2 bootstrapped Bias variance trade-off

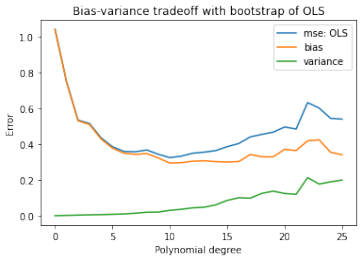


Figure 4.5: OLS

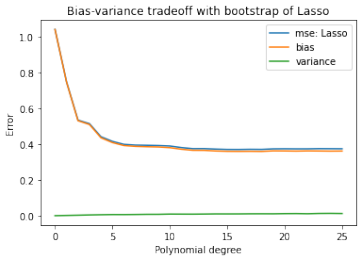


Figure 4.6: Lasso

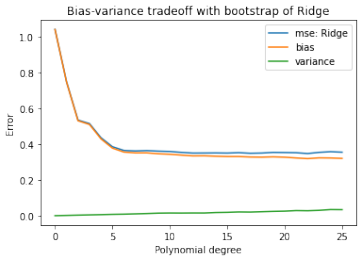


Figure 4.7: Ridge

Figure 4.8: Bias variance trade off on terrain data

As seen from the figure, the behavior of OLS is equal to that of the first run for a polynomial degree of 25. Up until degree 10, OLS Ridge and lasso experienced the same behavior before the variance increased drastically for the OLS case but remained somewhat stable. Compared to the Franke function in the method chapter, the polynomial degree on the terrain data requires more complexity and results in worse(higher) MSE. The results indicate that the OLS is somewhat the best fit for the terrain data up until degree 10. Whereas after degree 10, the ridge seems to outperform the lasso for the higher polynomials slightly.

Table 4.2: Bias variance and MSE bootstrap

MSE_OLS	MSE_Ridge	MSE_Lasso	Bias_OLS	Bias_Ridge	Bias_Lasso	Var_OLS	Var_Ridge	Var_Lasso
1.042158	1.042158	1.042158	1.041526	1.041526	1.041526	0.000632	0.000632	0.000632
0.750477	0.750477	0.749894	0.748436	0.748436	0.748011	0.002041	0.002041	0.001884
0.534973	0.534973	0.534743	0.531386	0.531386	0.531539	0.003587	0.003587	0.003204
0.515985	0.515594	0.515035	0.510721	0.510608	0.510344	0.005264	0.004986	0.004691
0.436665	0.436598	0.442056	0.430308	0.430278	0.436194	0.006357	0.006320	0.005862
0.386258	0.386184	0.416112	0.378232	0.378243	0.409321	0.008026	0.007941	0.006791
0.359697	0.365179	0.399487	0.349627	0.356191	0.392839	0.010070	0.008987	0.006648
0.357956	0.362418	0.395329	0.343353	0.351708	0.388119	0.014602	0.010709	0.007210
0.367869	0.364370	0.394234	0.347875	0.351860	0.386163	0.019994	0.012510	0.008071
0.343859	0.361370	0.392911	0.323180	0.346662	0.384759	0.020679	0.014707	0.008152
0.325264	0.359470	0.390308	0.294710	0.344001	0.380823	0.030553	0.015469	0.009486
0.333493	0.354483	0.381399	0.296991	0.339333	0.372139	0.036502	0.015149	0.009261
0.349893	0.350646	0.375253	0.305276	0.335002	0.366141	0.044617	0.015644	0.009112
0.355830	0.350995	0.375456	0.307829	0.335456	0.365798	0.048001	0.015539	0.009657
0.364389	0.351425	0.372782	0.302967	0.333126	0.362299	0.061421	0.018299	0.010483
0.386543	0.350673	0.370856	0.300483	0.331313	0.360461	0.086060	0.019360	0.010395
0.404029	0.352866	0.370665	0.303324	0.331369	0.360150	0.100705	0.021497	0.010515
0.440983	0.349522	0.371601	0.342773	0.328694	0.360509	0.098210	0.020828	0.011092
0.454990	0.350771	0.371140	0.329861	0.327945	0.359851	0.125129	0.022826	0.011289
0.467431	0.354409	0.374081	0.329578	0.329667	0.362975	0.137853	0.024742	0.011106
0.496454	0.353701	0.374625	0.371611	0.327662	0.362553	0.124843	0.026039	0.012072
0.485202	0.352644	0.374097	0.364893	0.323385	0.361668	0.120309	0.029259	0.012430
0.632733	0.348048	0.374255	0.419211	0.319672	0.362782	0.213522	0.028376	0.011474
0.602635	0.354603	0.375311	0.425076	0.324005	0.362243	0.177559	0.030598	0.013068
0.544901	0.358296	0.374998	0.355244	0.323302	0.361503	0.189657	0.034994	0.013495
0.540351	0.355474	0.374684	0.340729	0.321024	0.361961	0.199622	0.034450	0.012723

4.3.3 Cross Validation

Utilizing the cross-validation with 10 folds to compare the different methods.

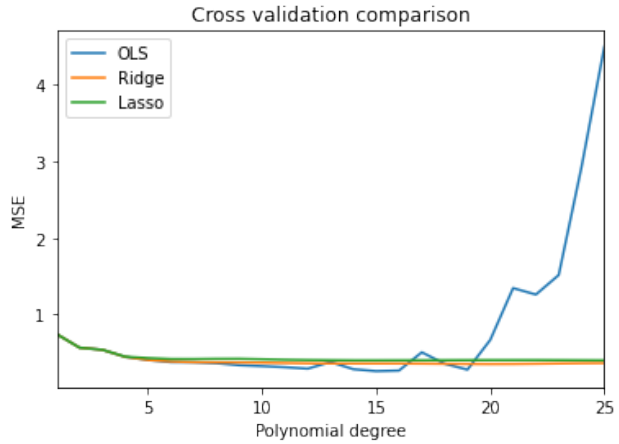


Figure 4.9: Cross validation for OLS, Ridge and Lasso

From the figure above, there is observed a shift in the optimal polynomial compared to the bootstrapped case. As in the bootstrap, the lowest MSE of OLS was found with a degree of 10, but the cross-validation gives a minimum MSE of 15th.

Table 4.3: Cross validation of the different methods.

Complexity	mse_cval_OLS	mse_cval_Ridge	mse_cval_Lasso
0.0	1.002413	1.002413	1.002413
1.0	0.745855	0.745855	0.746329
2.0	0.568951	0.568951	0.568945
3.0	0.541883	0.541781	0.542086
4.0	0.448969	0.448877	0.452040
5.0	0.408915	0.408776	0.433817
6.0	0.380902	0.389544	0.421615
7.0	0.378592	0.380729	0.421202
8.0	0.366699	0.378180	0.424128
9.0	0.340770	0.376964	0.424715
10.0	0.329017	0.374387	0.418470
11.0	0.314862	0.370061	0.412089
12.0	0.295952	0.365768	0.408331
13.0	0.382151	0.363387	0.405827
14.0	0.288879	0.362865	0.403952
15.0	0.263879	0.363011	0.403866
16.0	0.271200	0.362677	0.404271
17.0	0.509201	0.361250	0.404960
18.0	0.360271	0.358969	0.405843
19.0	0.283073	0.356823	0.406790
20.0	0.670585	0.355937	0.407459
21.0	1.345660	0.356931	0.407517
22.0	1.261438	0.359657	0.407626
23.0	1.515456	0.363331	0.406383
24.0	2.923026	0.366866	0.405206
25.0	4.488122	0.369212	0.404022

From the table above, the optimal polynomial seems to be around 15th. This is different from our previously observed 10th. However, the difference is rather minute. This tells us that the group was slightly unlucky with our test/train data split in the first run, but the error is rather minute. Still, it is the group's opinion that the increased run-time cost of 15th vs. 10th polynomial complexity is probably not worth it. So 10th order polynomial is the recommendation. the same table can be see plotted in figure 4.9. The reason for OLS being the lowest can be explained by the number of observations is largely higher than the number of variables.

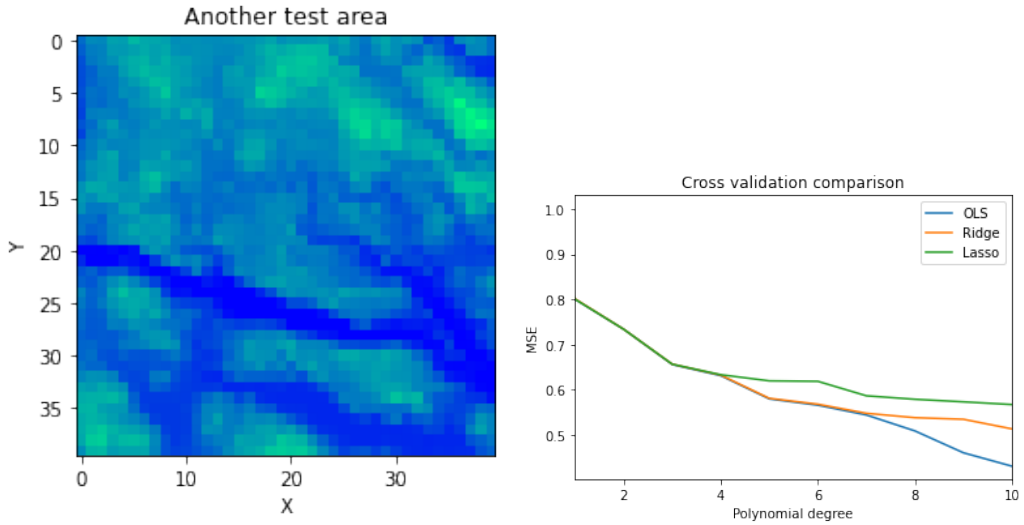
If Ridge and Lasso outperform the OLS, some of the predictors need to be insignificant, or the predictors are highly correlated. Comparing the minimum MSE in both bootstrap and cross-validation can be observed in the table (4.4)

Table 4.4: Minimum MSE across resampling and regresion

Cval_OLS	Cval_Ridge	Cval_Lasso	boot_OLS	boot_ridge	boot_lasso
0.263	0.355	0.403	0.325	0.348	0.370

The comparison yields the lowest MSE for both bootstrap and cross-validation is indeed the OLS. This is the same as observed in the franke function in the method. Although the method chapter produced lower MSE it is not surprising when predicting height above a mesh that a polynomial fit with the ordinary least squares will yield the best result. As discussed earlier, if the data contained super high correlation or more non-significant predictors. One would expect the lasso to perform better, although for increasingly high complexity, the lasso and ridge manage to remove/reduce the insignificant predictor parameters. While the OLS skyrockets as it does not select or reduce the dimensionality of the feature matrix.

To ensure that the OLS is the better option, the authors decided to run for a polynomial degree 10 on another part of the map.



(a) Terrain3 data[1500:2500,0:1000] (b) Cross validation on terrain data 3

Figure 4.10: Validating OLS performance

More clearly, in this plot, the OLS outperforms both ridge and lasso to lower the cost function. From the second test run, there should be no doubt that the ordinary least squares are indeed the best description of the terrain.

Chapter 5

CONCLUSION

The paper has been successful in recreating some of scikit-learn's functions and applying them to real-world data. The terrain data, even with a massive reduction in data points, was still good enough to produce models. A polynomial degree of 10 was the best fit with the real-world data. With further complexity, it was observed that the MSE of "train" and "test" started to diverge. Over-fitting was observed for increased complexity in both method and results. The paper also concludes that OLS is the best-fit regression method for this problem. Through the discussion on computational time vs MSE decrease, the authors conclude that a polynomial of 10 is better than a polynomial of 15 due to the heavy increase in computational time vs. result.

The Authors of this report have had a whole learning experience with the project, and as the project developed and the paper was written, more knowledge was gained. From the perspective of workload, yes, there is much to be covered and much to be considered; luckily, the professor provides good material and suggestions. The lab exercises and discussion through them have had the most learning value for the project, and the lecture notes provide a good foundation for understanding. It is with a consensus of the project group that working on these projects has been a good and fun experience, although the work combined with other heavy tech courses makes for long days and busy weekends. major key points learned are:

1. broader understanding of statistics.
2. better understanding of python in general.
3. demystified some of the topics within machine learning.
4. progression group planning and work ethics have been improved through the project.

Future research: The group has discovered that after plotting both Franke's function, the real map data and some other real life energy-data (not related to this project) for different polynomials vs the original data, seemingly unscaled versions were more identical than the scaled ones. This is in stark contrast to when we compare metrics, as scaled score considerably better on MSE-values. This somewhat baffles the authors and could be interesting to dwell further into.

Finally, the world is slightly safer after this report. Our project group is more optimistic about saving the world from AI Skynet, given our new-developed insight. But more work is needed. See you in the following report.

I'll be back.

Figure 5.1

Bibliography

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "The elements of statistical learning: data mining, inference, and prediction. Springer series in statistics". In: (2009).

[2] R. Penrose. "A generalized inverse for matrices". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 51.3 (1955), pp. 406–413. DOI: 10.1017/S0305004100030401.

[3] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288.

[4] B Efron. "The 1977 RIETZ lecture". In: *The Annals of Statistics* 7.1 (1979), pp. 1–26.

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

[6] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.