Regression analysis and resampling methods

Elisabeth Strøm

October 9, 2019

Contents

1 Introduction **3**

2 Method **3**

2.0.1 Decomposing the cost function 4

2.1 Ordinary Least Squares regression 5

2.2 Ridge and Lasso regression 5

2.3 Implementation 6

2.3.1 k-fold Cross-Validation 7

2.3.2 Standardizing the data 7

2.3.3 Linear regression 7

3 Results **8**

3.1 The Franke function 8

3.1.1 Cross-validating 9

3.1.2 Bias-Variance trade-off 9

3.1.3 Ridge regression analysis 10

3.1.4 Lasso regression analysis 12

3.2 Terrain analysis 14

3.2.1 Ridge regression 15

3.2.2 Lasso 17

4 Discussion **17**

5 Conclusion **18**

1 Introduction

In this project we test three different linear regression methods: The Ordinary Least Squares (OLS), Ridge regression, and Lasso regression.

This project consists of two parts. First, we try to estimate the Franke function (Franke, 1979), having created a noisy dataset from the function. This is to see how well our algorithm works. We also implement k-fold cross validation.

Next, we sample data taken of the terrain, from somewhere in the world (SOURCE), and try to recreate it by using our regression models.

In Section 2, we go through the theory behind the methods we will use, and also show how we will implement them numerically.

Our results can be seen in Section 3, and we will discuss them in Section 4. Finally, we sum up our conclusions in Section 5.

2 Method

Here we go through our methods of linear regression, while in 2.3, we show how we implement them numerically.

Say we have a dataset consisting of $\{y_i, \boldsymbol{x}_i, i = 0...n-1\}$, where $y_i = y(\boldsymbol{x}_i)$ is the outcome, or the response, and $\boldsymbol{x}_i$ is the predictor, or independent variable. The idea behind linear regression, is determine the true function, $f(\boldsymbol{x})$, from our noisy data

$$y(\boldsymbol{x}_i) = f(\boldsymbol{x}_i) + \epsilon_i. \tag{1}$$

Here, $\boldsymbol{\epsilon}$ is the noise which we assume to be normally distributed with mean $\mu = 0$ and standard deviation σ, $\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$.

Through a regression analysis, we estimate f by the function $\tilde{\boldsymbol{y}}$. For one predictor $\boldsymbol{x}$ this can be written as

$$y_i \approx \tilde{y}_i + \epsilon_i = \sum_{j=0}^{m-1} \beta_j \boldsymbol{x}_{i,j} + \epsilon_i. \tag{2}$$

Here β_j are called the regression coefficients, and $x_{i,j}$ are the terms in the function $\tilde{y}_i$.

In matrix notation,

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}, \tag{3}$$

where $\boldsymbol{y} = [y_0, y_1, \ldots, y_{n-1}]^T$, $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_{n-1}]^T$, and $\boldsymbol{X}$ is the design matrix,

$$\boldsymbol{X} = \begin{bmatrix} x_{00} & x_{01} & \cdots & x_{0,m-1} \\ x_{10} & x_{11} & \cdots & x_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1,0} & x_{n-1,1} & \cdots & x_{n-1,m-1} \end{bmatrix}, \tag{4}$$

The β-coefficients, are found by optimizing the the Mean Squared Error,

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^{1/2}, \tag{5}$$

via the cost function

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^{1/2} = \mathrm{E}[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2]. \tag{6}$$

We will also look at the R^2 score function, defined as

$$R^2(\boldsymbol{y},\tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i-\bar{y})^2}, \tag{7}$$

where $\bar{y}$ is the mean value, $\frac{1}{n}\sum_{i=0}^{n-1} y_i$. Then the closer R^2 is to 1, the better is our fit.

The confidence interval of a parameter X at 95% confidence, assuming a Gaussian distribution, is defined as

$$CI_X^{0.95} = \mathrm{E}[X] \pm 1.96 \cdot \mathrm{STD}[X], \tag{8}$$

where $\mathrm{E}(X)$ is the mean of X, and $\mathrm{STD}[X]$ is the standard deviation of X.

For the β-parameters this becomes,

$$CI_\beta - j = \beta_j \pm 1.96\sqrt{\mathrm{Var}(\beta_j)}. \tag{9}$$

We aim to find $\tilde{y}_i$, through three different methods: The Ordinary Least Square (OLS), Ridge regression, and Lasso regression, which are outlined in the next three subsections.

2.0.1 Decomposing the cost function

The cost function can be decomposed into three terms, the bias squared of $\tilde{\boldsymbol{y}}$, the variance of $\tilde{\boldsymbol{y}}$, and the irreducible error variance, which is just $\mathrm{Var}(\epsilon_i) = \sigma^2$,

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

where in the last term, $\mathrm{Bias}^2(\tilde{\boldsymbol{y}}) = (\boldsymbol{f}-\mathrm{E}[\tilde{\boldsymbol{y}}])^2$, and $\mathrm{Var}(\tilde{\boldsymbol{y}}) = (\mathrm{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})^2$.

The cross terms in the fourth line above, disappears as follows,

$$\begin{aligned}
2\mathrm{E}[\boldsymbol{\epsilon}(\boldsymbol{f}-\mathrm{E}[\tilde{\boldsymbol{y}}])] &= 2\mathrm{E}[\boldsymbol{\epsilon}]\mathrm{E}[\boldsymbol{f}-\mathrm{E}[\tilde{\boldsymbol{y}}]] = 0, \\
2\mathrm{E}[\boldsymbol{\epsilon}(\mathrm{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})] &= 2\mathrm{E}[\boldsymbol{\epsilon}]\mathrm{E}[(\mathrm{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})] = 0
\end{aligned}$$

where we have used that the expectation value, or the mean of the noise term $\boldsymbol{\epsilon}$ is 0,

$$\begin{aligned}
2\mathrm{E}[(\boldsymbol{f}-\mathrm{E}[\tilde{\boldsymbol{y}}])(\mathrm{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})] &= 2\mathrm{E}[\boldsymbol{f}-\mathrm{E}[\tilde{\boldsymbol{y}}]]\mathrm{E}[(\mathrm{E}[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})] \\
&= 2(\boldsymbol{f}-\mathrm{E}[\tilde{\boldsymbol{y}}])(\mathrm{E}[\tilde{\boldsymbol{y}}]-\mathrm{E}[\tilde{\boldsymbol{y}}]) \\
&= 2(\boldsymbol{f}-\mathrm{E}[\tilde{\boldsymbol{y}}])\cdot 0 \\
&= 0.
\end{aligned}$$

Generally, as the complexity of our model increases, for instance the polynomial degree where we to fit a polynomial to our data, the bias will decrease, while the variance increases.

A simpler model is easier to train, but have a lower predictive power on more complex problems. A complex model may fit the data better, but will also be prone to overfitting as the complexity increases, sampling the noise instead of the true function f. This means that the variance of $\tilde{\boldsymbol{y}}$ estimated from different training set, will be high. Ideally we would have a low-variance and low-bias model, so we will have to balance the two factors. Linear regression is in general a low variance procedure.

2.1 Ordinary Least Squares regression

We start with the simplest of the regression methods: The Ordinary Least Square.

The idea behind this method, is to fit a polynomial, $\tilde{\boldsymbol{y}}$, to our, usually, noisy data $\boldsymbol{y}$, and try to model the underlying, true function f. In the case of one predictor $\boldsymbol{x}$,

$$y_i \approx \tilde{y}_i + \epsilon_i = \sum_{j=0}^{m-1} \beta_j \boldsymbol{x}_i^j + \epsilon_i. \tag{10}$$

So the elements of the design matrix $\boldsymbol{X}$ (Equation 4) are now $x_{i,j} = x_i^j$.

In our project, we try to recreate the Franke function, which has two independent parameters, $x, y \in [0, 1]$,

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right). \end{aligned} \tag{11}$$

Our estimate of $f(\boldsymbol{x}, \boldsymbol{y})$, is denoted as $\tilde{\boldsymbol{z}}$, and our noisy data we perform an analysis on, is then $z_i = f(\boldsymbol{x}_i, \boldsymbol{y}_i) + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(0, 1)$. The elements of the design matrix will then also assume a slightly different shape, where the order of the polynomial $\tilde{z}_i$ is a result of the polynomial order of both $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ multiplied together. We will get back to how to implement this in Section 2.3.

The $\boldsymbol{\beta}$ coefficients are calculated from

$$\boldsymbol{\beta}^{\text{OLS}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{z}, \tag{12}$$

for the $\tilde{\boldsymbol{z}}$, calculated from $\boldsymbol{X}$ (Equation 3) for different polynomial orders, that minimizes the Mean Squared Error, or the predictive error.

The variance of $\boldsymbol{\beta}$ is,

$$\text{Var}(\boldsymbol{\beta}) = \sigma^2 \cdot \text{diag}(\boldsymbol{X}^T\boldsymbol{X}^{-1}), \tag{13}$$

where σ^2 is still the variance of the noise $\boldsymbol{\epsilon}$.

2.2 Ridge and Lasso regression

In our Ridge and Lasso regression analyses, we still aim to fit a polynomial to f, but now with a penalty parameter, λ. λ will minimize the β parameters.

The β parameters for our Ridge analysis, is calculated from

$$\boldsymbol{\beta}^{\text{Ridge}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{z}, \tag{14}$$

where $\boldsymbol{I}$ is the identity matrix, $\boldsymbol{I} = \text{diag}(1, 1, \ldots, 1)$ of the same shape as $\boldsymbol{X}^T\boldsymbol{X}$, that is, $n \times n$.

The variance of $\boldsymbol{\beta}^{\text{Ridge}}$ is,

$$\text{Var}(\boldsymbol{\beta}^{\text{Ridge}}) = \sigma^2 \cdot \text{diag}([\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{\lambda}\boldsymbol{I}]^{-1}\boldsymbol{X}^T\boldsymbol{X}\{[\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{\lambda}\boldsymbol{I}]^{-1}\}^T), \tag{15}$$

In our Lasso analysis, we do not have an analytical expression for the β coefficients, but they will instead be found by using `sklearn`'s `Lasso` function.

The variance of $\boldsymbol{\beta}^{\text{Lasso}}$ is found by minimizing the Lasso cost function $k = 5$ times through k-fold cross-validation using `sklearn`, and then finding the variance of the k $\boldsymbol{\beta}^{\text{Lasso}}$ that are found.

2.3 Implementation

Here we go through how we implemented our regression analyses.

The first thing we do is create the $\boldsymbol{x}$ and $\boldsymbol{y}$. We make two arrays with evenly spaced points, for a total of a 100 each, between 0 and 1. Then we use the `numpy` command `meshgrid(x,y)` to create a meshgrid. $\boldsymbol{x}$ and $\boldsymbol{y}$ are then used to construct the Franke function $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{y})$. Next we flatten the arrays, and copy $\boldsymbol{z}$ and add the normally distributed noise $\boldsymbol{\epsilon}$. We know have the true function, `z1_true` and the noisy function `z1_noise`,

```
z = FrankeFunction(x, y)
z1_true = np.ravel(z)
# adding noise
z1_noise = np.ravel(z) + np.random.normal(0, stddev, size=z1_true.shape)
```

Next thing to do is create the design matrix $\boldsymbol{X}$. It is made so that the polynomial degree increases with the columns. Say we have a 2nd order polynomial, the first row then looks like

$$x_{0,*} = [1, x_0, y_0, x_0^2, x_0y_0, y_0^2].$$

To calculate $\boldsymbol{\beta}$ for OLS and Ridge regression, we need to be able to invert $\boldsymbol{X}^T\boldsymbol{X}$. We do this through a singular value decomposition

```
U, s, Vt = np.linalg.svd(X)
Sigma = np.zeros((len(U),len(Vt)))

for i in range(0,len(Vt)):
Sigma[i,i] = s[i]
Sigma_inv = np.linalg.pinv(Sigma)
X_inverted = np.matmul(Vt.T, np.matmul(Sigma_inv,U.T))
```

Now, finding $\tilde{\boldsymbol{z}}$ is straight forward. Note that we will train our algorithm on the noisy data $\boldsymbol{z}^{\text{noise}}$, but assert the fit by calculating MSE with the true Franke function $\boldsymbol{z}^{\text{true}}$.

2.3.1 k-fold Cross-Validation

We implement the k-fold Cross-Validation. Here our data is shuffled and split into $k = 5$ equal parts. 1/5 is used as test data, wile the rest is training data. We then perform a regression analysis k times, where we change which is the test set every time. For each analysis, we estimate the MSE, and then, finally we take the mean of the k error estimates. MSE is calculated both for the test, and training set.

2.3.2 Standardizing the data

We standardize the datasets in the following way: For the training set $\boldsymbol{X}^{\mathbf{train}}$, we subtract the mean of each column from that column, and then we divide each column with the standard deviation of each column,

$$\boldsymbol{X}_{*,j}^{\text{train,c}} = \frac{\boldsymbol{X}_{*,j}^{\text{train}} - E[\boldsymbol{X}_{*,j}^{\text{train}}]}{\text{STD}(\boldsymbol{X}_{*,j}^{\text{train}})}, \qquad \text{for } j = 1, 2, \ldots, m-1. \tag{16}$$

For z, we subtract the mean only, $\boldsymbol{z}^{\text{train,c}} = \boldsymbol{z}^{\text{train,c}} - \text{E}[\boldsymbol{z}^{\text{train,c}}]$

We apply these same transformations to the test set, that is,

$$\boldsymbol{X}_{*,j}^{\text{test,c}} = \frac{\boldsymbol{X}_{*,j}^{\text{test}} - E[\boldsymbol{X}_{*,j}^{\text{train}}]}{\text{STD}(\boldsymbol{X}_{*,j}^{\text{train}})}, \qquad \text{for } j = 1, 2, \ldots, m-1. \tag{17}$$

and $\boldsymbol{z}^{\text{test,c}} = \boldsymbol{z}^{\text{test,c}} - \text{E}[\boldsymbol{z}^{\text{train,c}}]$. The columns now have a mean equal to 0, and a standard deviation equal to 1.

We add the mean again when calculating $\tilde{\boldsymbol{z}}$, that is

$$\tilde{\boldsymbol{z}}^{train} = \boldsymbol{z}^{train,c} + \text{E}[\boldsymbol{z}^{\text{train,c}}] \tag{18}$$

$$\tilde{\boldsymbol{z}}^{test} = \boldsymbol{z}^{test,c} + \text{E}[\boldsymbol{z}^{\text{train,c}}]. \tag{19}$$

Standardizing the data is not that important for the OLS method, but rather for Ridge and Lasso, where we have the penalty λ. λ puts constraints on the β-coefficients, but the coefficients size depends on the magnitude of each variable. It is therefore important to get the data on the same scales, so that no variable is punalized unfairly, and to be able to see which factors truly contribute the most to $\tilde{\boldsymbol{z}}$. Also, we no longer have an intercept, but it can be interpreted as being the $\text{E}[\boldsymbol{z}^{\text{train,c}}]$.

2.3.3 Linear regression

We first run the three regression analyses on the Franke Function data. Given that we know the true function already, we will train our algorithm with the noisy data, but calculate the MSE using the true function.

Next we move onto the terrain data, where we used the files found together with the project instructions. This is a quite large dataset, so we chose a small cutout, just the first 1000 pixels in x and y direction. We down-sampled this image, by only including every 30th pixel in either direction.

For the OLS, we calculate MSE for several polynomial degrees, to find which one offers the best fit.

For Ridge and Lasso, we must also find the best fit λ parameter that together with the right polynomial, minimizes the MSE.

The result of our analyses can be found in Section 3.

3 Results

Here we show the results of our regression analyses on the Franke function, and on the terrain data.

3.1 The Franke function

The best fit polynomial order, λ, the minimum predictive error MSE, and R^2 for each of the regression models, including cross-validation, can be found in Table 1. This will be discussed at the end of this section, but first we will go through the intermediate results.

Having produced the true and noisy data as stated in Section 2.3, we start by fitting a 5th order polynomial to `z1_noise`. Unless stated otherwise, assume $\boldsymbol{X}$ is standardized, and $\boldsymbol{z}^{noise}$ is mean-centered.

For an Ordinary Least squares analysis, the β parameters along with their confidence intervals at 95%, can be seen in Figure 1. We see that the confidence intervals are quite large in places, but we were not able to ascertain as to why.

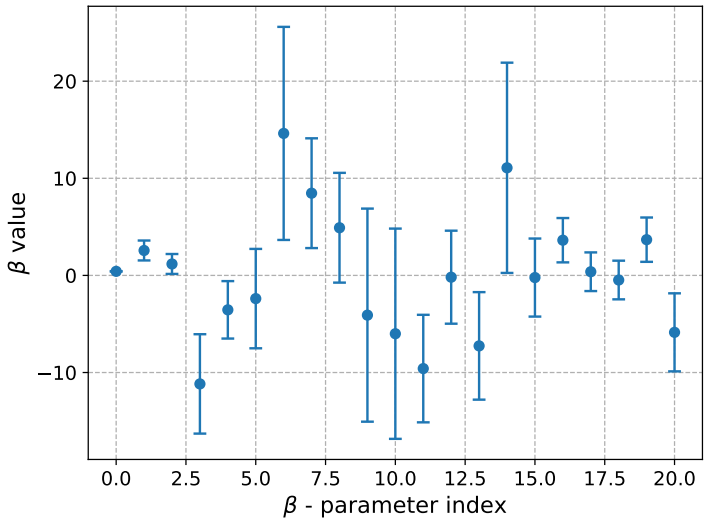

Figure 1: The values of the parameters β found from an Ordinary Least Squares regression analysis, are shown on the y-axis, while the index of the parameters are on the x-axis. The errorbars are calculated at 95% confidence

Still, the MSE score is quite good, at $MSE(\boldsymbol{z}^{\text{true,test}}, \tilde{\boldsymbol{z}}) = 4.09{\cdot}10^{-3}$, and $R^2(\boldsymbol{z}^{\text{true,test}}, \tilde{\boldsymbol{z}}) = 0.95$

Double checking our results against that found by `sklearn` using `LinearRegression`, showed

that they were approximately the same.

3.1.1 Cross-validating

Next we used $k = 5$-fold cross validation on our data. We find now a very similar error as before, $MSE = 4.62 \cdot 10^{-3}$, and $R^2 = 0.94$.

Assuming we did not know the true underlying function, and we used our noisy model instead to calculate the error, we find $MSE(\boldsymbol{z}^{\text{noise,test}}, \tilde{\boldsymbol{z}}) = 1.19$, which should be the irreducible error variance, that is, σ^2, which in our case we had chosen as 1, so this too is a good result.

3.1.2 Bias-Variance trade-off

We try to create a figure similar to Figure 2.11 of Hastie, Tibshirani, and Friedman, which shows the test and training error as a function of the complexity of the model. In our case, the complexity is the polynomial degree.

To reproduce this figure, we will pretend we do not know the true model, and use the noisy data alone. We then get the diagram in Figure 2. Lowest MSE is still found at the 5th polynomial order, like before. Here we see that the test error reaches a minimum, but the training error continuously decreases.

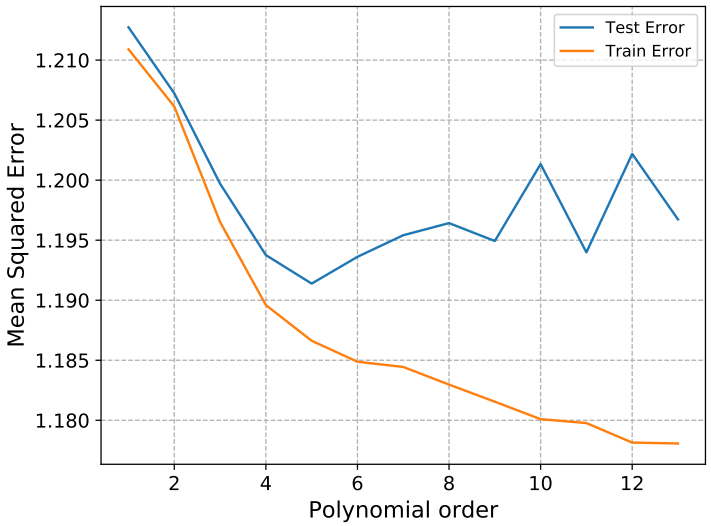

Figure 2: The training and test MSE for increasing model complexity, using Ordinary Least Squares and k-fold Cross Validation and the noisy dataset. While the test error reaches a minimum before increasing again, the training error will continuously decrease with increasing complexity.

If we instead calculate the MSE based on the real data, we get Figure 3. Here too we see that the 5th order polynomial fits the data the best.

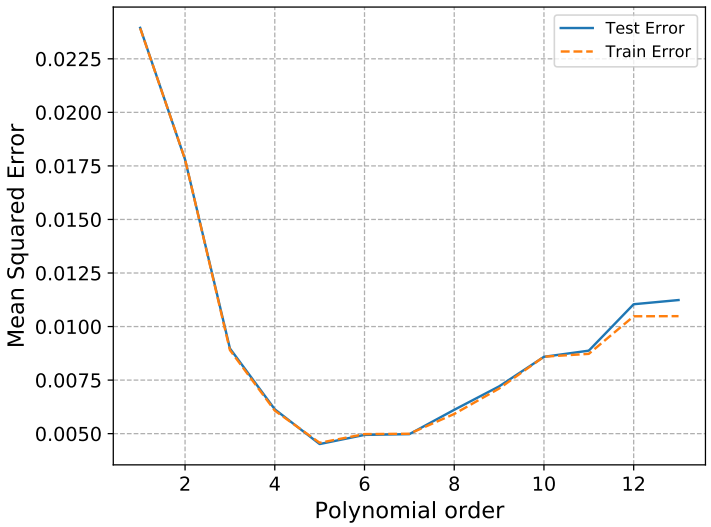

Figure 3: The training and test MSE for increasing model complexity, using Ordinary Least Squares and k-fold Cross Validation and the true dataset. While the test error reaches a minimum before increasing again, the training error will continuously decrease with increasing complexity.

3.1.3 Ridge regression analysis

Now that we try a new method, we go back to starting with fitting a 5th order polynomial to the data. But here we must also take into account the penalization parameter λ value.

If we plot polynomial degrees against the MSE for various values of λ, we get the diagram in Figure 4, and the close-up in Figure 5. What we see is that a 5th order polynomial is favored for $\lambda = 10^{-7} - 10^{-2}$, and the best fit value is $\lambda = 10^{-2}$. However, we note that the difference in MSE is extremely small for $\lambda < 10^{-2}$, making it uncertain which exact value of λ is the best fit. We see this also in Figure 6, where the Test error is quite noisy, and the best fit value becomes uncertain.

Nonetheless, we choose to continue on with a 5th order polynomial and $\lambda = 1e - 2$. The test and training error for this pair of variables can be seen in Figure 6.

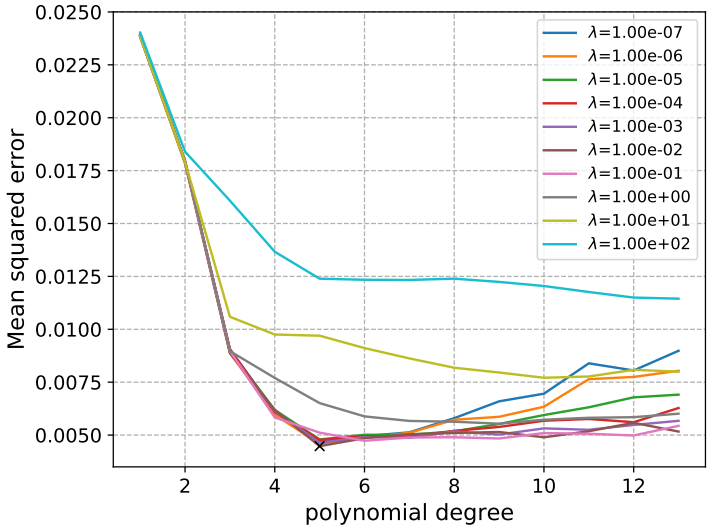

Figure 4: The MSE, calculated using the true model, for the test data for various λ values and polynomial degrees, while using cross validation and Ridge regression. The minimum MSE value has been marked by a black $\times$.

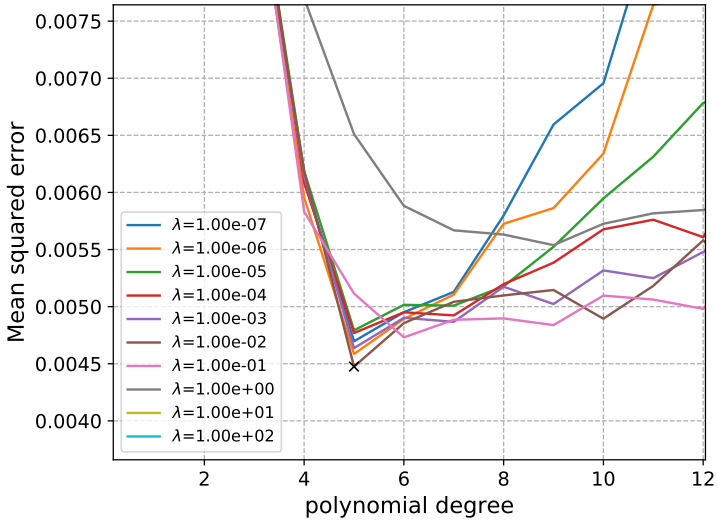

Figure 5: Close-up of the MSE, calculated using the true model, for the test and training data for various λ values, while using cross validation and Ridge regression, for a fifth order polynomial. The minimum MSE value has been marked by a black $\times$.

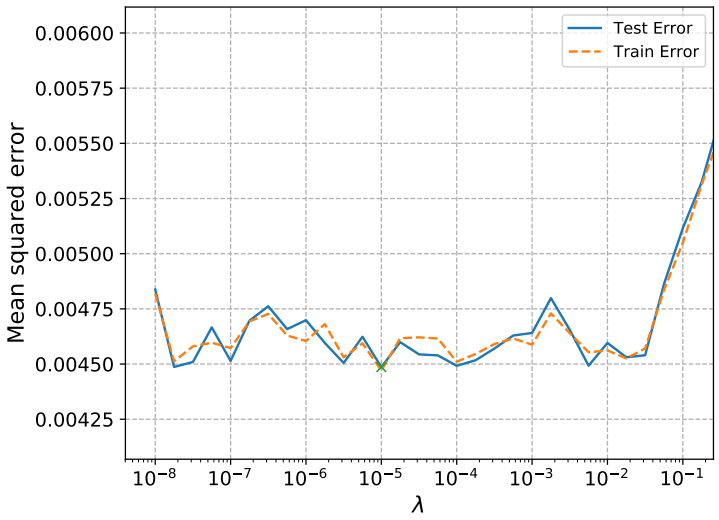

Figure 6: The training and test error, calculated using the true model, for various λ values, while using cross validation and a fifth order polynomial. The minimum MSE value has been marked by a yellow $\times$.

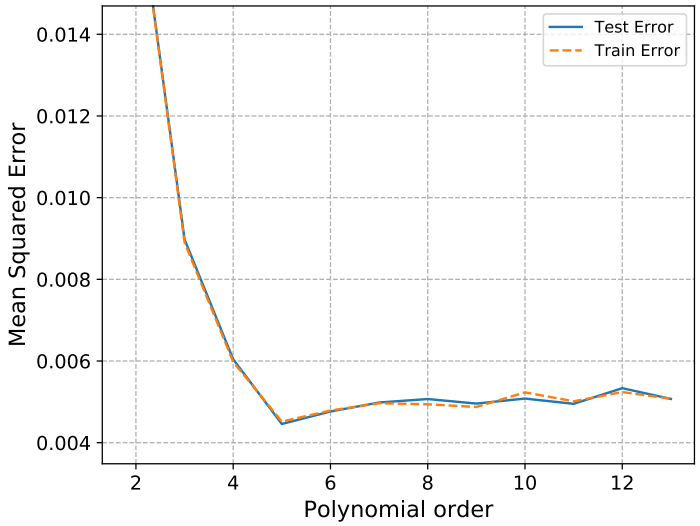

Figure 7: The training and test MSE for increasing model complexity, using Ridge regression and k-fold Cross Validation. While the test error reaches a minimum before increasing again, the training error will continuously decrease with increasing complexity.

3.1.4 Lasso regression analysis

Finally, we use the Lasso method to find $\tilde{z}$. First we calcaulte the MSE as a function of polynomial order and λ. This can be seen in Figure 8.

We again start from a 5th order polynomial, and find that, with cross-validation, $\lambda = 2 \cdot 10^{-5}$ fit data best, as it had an predictive error at $MSE = 4.68 \cdot 10^{-3}$ and $R^2 = 0.943$. For any lower λ values, we get a warning that there is no converging. Having already standardized the data and increased the number of iterations and tolerance in `sklearn`'s `Lasso`, we elected not to test this further. This can be seen in Figure 9. Here we find also that a 5th order polynomial fits the best, this time with $\lambda = 10^{-5}$. We note that the increase in MSE is very small for increasing polynomial orders, for all λ.

While running the calculations for $\lambda = 10^{-6}$ and $\lambda = 10^{-5}$ we did however get a no-convergence-warning. Seeing as $\lambda = 10^{-4}$ provides an almost as good fit, we will use that instead.

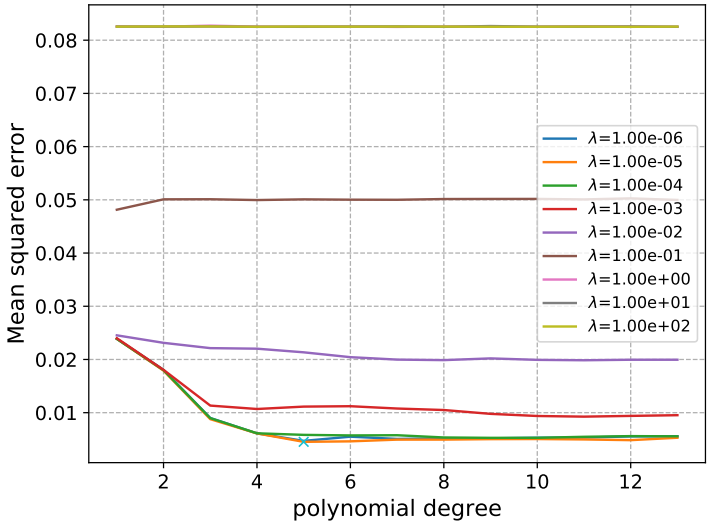

Figure 8: The MSE, calculated using the true model, for the test and training data for various λ values, while using cross validation and Lasso regression, for a fifth order polynomial. The minimum MSE value has been marked by a blue $\times$.

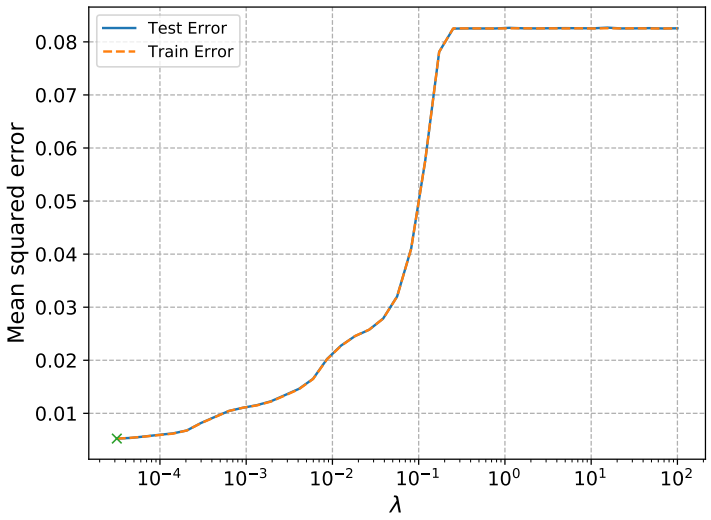

Figure 9: The MSE, calculated using the true model, for the test data for various λ values, while using cross validation and Lasso regression on a fifth order polynomial. The minimum MSE value has been marked by a yellow $\times$, although due to convergence issues, we will not use that one.

3.2 Terrain analysis

Here are the results from our linear regression analyses on data of a terrain in Norway. The original image, and the result of our chosen sampling, is seen in Figure 10

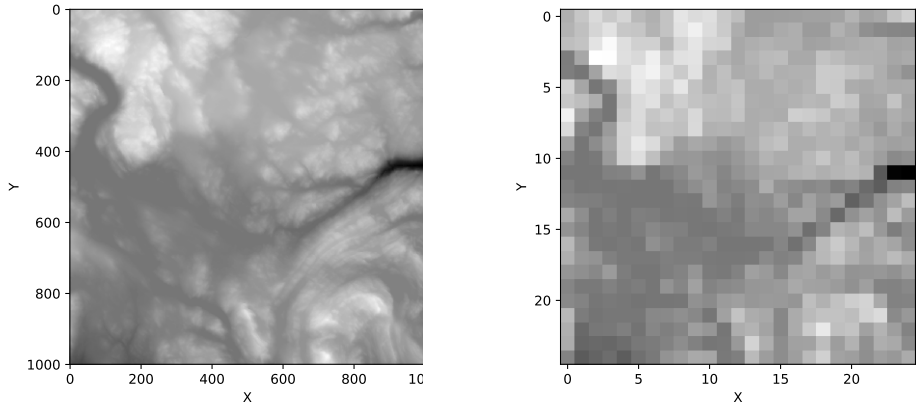

Figure 10: To the left is the small cutout of the original data. To the right is the down-sampled image that we will work on.

Now, to find the best fit polynomial for an OLS, and also the best fit λ for the Ridge and Lasso models, before we discuss which method fits the data the best.

The results of our analyses can be seen in Table 2

We first performed an Ordinary Least Squares regression analysis, which included Cross-Validation. We did this for the training and testing set for various polynomial degrees. This can be seen in figure 8.

We find that a 10th order polynomial minimizes the MSE test error the best, although this depends heavily on the amount of data we use. We found as expected, that the more data we used, the higher order polynomial was found to minimized MSE. In these cases, the test and training error were almost the same, meaning it is more difficult to overfit and start sampling the noise, but the higher the variance would become.

In the end the fit is a quite poor one, with $MSE = 8413.40$, and $R^2 = 0.609$. It is unclear why the MSE is so high, but we suspect there is something wrong with our calculations. However we obtained the same result using our own algorithm, and that by `sklearn`.

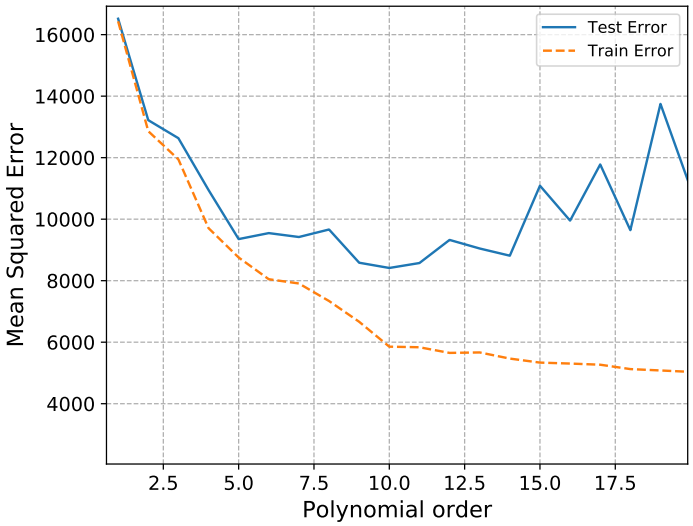

Figure 11: The testing and training error for various polynomial degrees, while using k-fold cross validation

3.2.1 Ridge regression

When performing a Ridge regression, we found again that a 10th order polynomial fit the data the best.

Fitting both λ and the polynomial degree at the same time, we get the plot in figure 12. We see that the curves are quite erratic, only the two highest $\lambda = 1, 10^{-1}$ have a somewhat smooth descent before increasing again. This makes our estimate of both polynomial order and λ uncertain. It seems that $\lambda = 10^{-9}$ and $d = 12$ might work as well, as the curve seem more "calm", but the MSE is slightly higher.

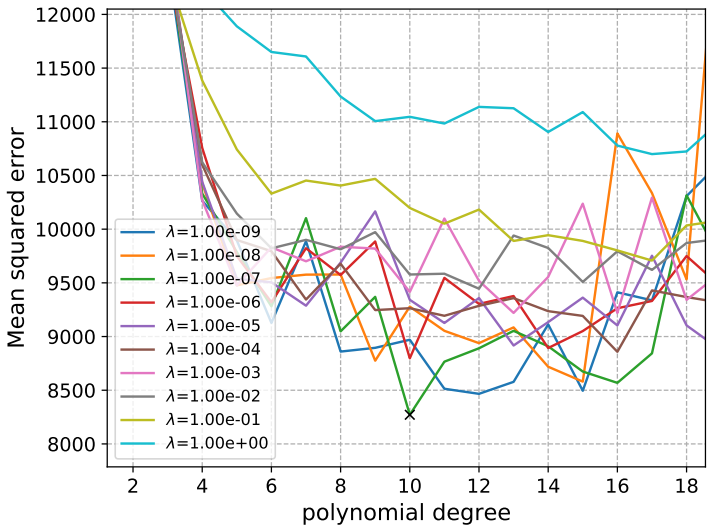

Figure 12: The Mean Square error calculated for various values of λ and polynomial orders. The black $\times$ marks the best fit.

Like before, we plot the MSE against λ for the polynomial we found that reduced the MSE the most. The test error seems quite erratic, and there is no minimum, as the training and test error is increasing. Can be due to overfitting for the test error, but what it means for the training data, we are not certain. It flattens out at very small λ, meaning an OLS will propbably fit better.

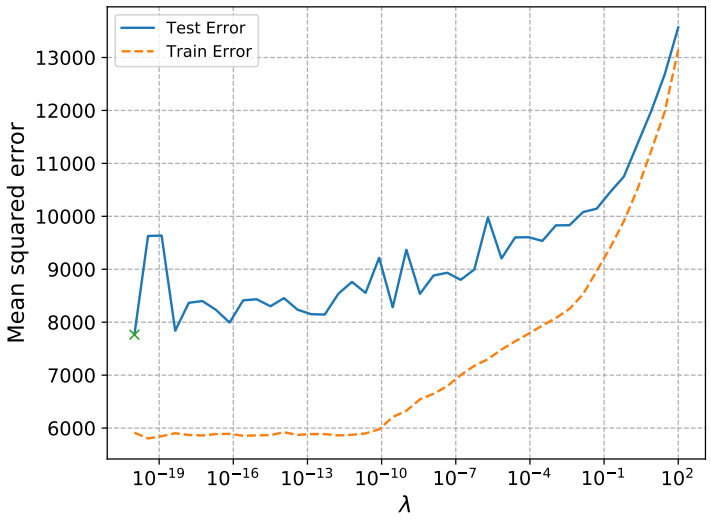

Figure 13: The MSE, calculated for various λ values, while using cross validation and Ridge regression on a 10th order polynomial. The minimum MSE value has been marked by a yellow $\times$

3.2.2 Lasso

We have a hard time performing the Lasso regression analysis. The computations take a very long time, and the small λ that seems to be favored, throw out a no-convergence warning. We therefore assume that a 10th order polynomial is still the best option, and calculate the MSE for various λ, as seen in Figure 14. Here too, the curve is erratic, and there is no well-defined minimum.

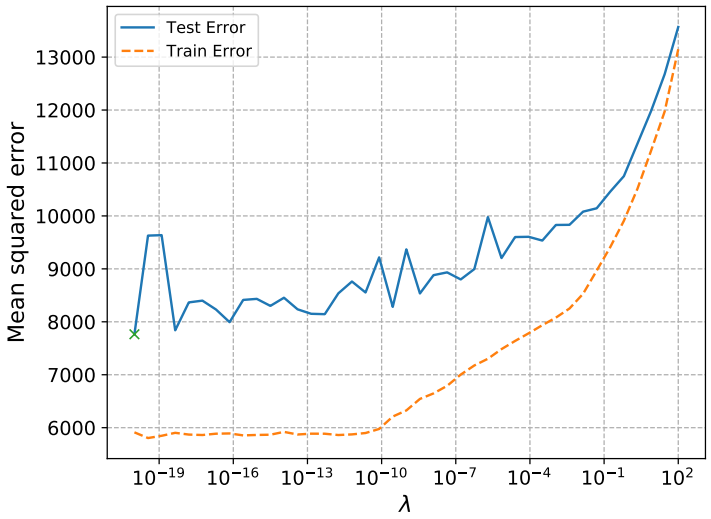

Figure 14: The MSE, calculated for various λ values, while using cross validation and Lasso regression on a 10th order polynomial. The minimum MSE value has been marked by a yellow $\times$

min error: 8333.226101945776 min error poly: 29 max r2 score: 0.6207354507501407

Need a higher polynomial

4 Discussion

For our data generated from the Franke function, we find that, irregardless of method, a 5th order polynomial fits our data the best. The β-coefficients, do however, have very large confidence intervals.

The MSE increases slightly as we change from OLS to Ridge and to Lasso, but only at the fourth decimal spot. Similarly, the R score does not really change, staying at approximately $R^2 = 0.95$ This makes it difficult to say if one method is more appropriate than another. In Table1 The R^2 and MSE scores are shown for the best fit models. We can see from this, that the OLS method wins out by a small margin.

Initially, we had fewer points, but this proved to make our analyses difficult, as the noise was of the same order as the Franke Function, making it hard to sample the true function. Our MSE was then approximately 0.02, and R^2 lied around ± 0.1. We found that this could be remedied by lowering the noise, by, for instance lowering its standard deviation, or by increasing the amount of data points. We chose the latter. We suspect there may be something wrong in the way the

Table 1: The R^2 and MSE scores for the OLS, Ridge, and Lasso regression for the best fitted polynomials, d and λ.

Model	d	λ	MSE	R^2
OLS	5	-	$4.62 \cdot 10^{-3}$	0.944
Ridge	5	10^{-2}	$4.71 \cdot 10^{-3}$	0.943
Lasso	5	10^{-4}	$5.24 \cdot 10^{-3}$	0.936

Table 2: The R^2 and MSE scores for the OLS, Ridge, and Lasso regression for the best fitted polynomials, d and λ.

Model	d	λ	MSE	R^2
OLS	10	-	8413.40	0.609
Ridge	10	10^{-7}	8271.37	0.618
Lasso	10	-	7998.89	0.628

confidence intervals of the β parameters are calculated, as they are very large at certain data points, but we are not able to find it.

Performing an analysis on the terrain data proved difficult. The minimum MSE and corresponding R^2-scores can be seen in Table 2. It would seem that the Lasso managed to recreate the terrain the best, but this does not say much, as they all have an $R^2 \approx 0.6$, and the λ-values being as uncertain as they are.

As for the reason for this, we are not certain. I suspect that if, instead of just skipping over pixels, we calculated the median, or average of neighboring pixels, applyed a Gaussian blur, then the data to fit would have been smoother, and our regression methods might have had it easier estimating a good $\tilde{\boldsymbol{y}}$. We initially tried to do this, but time ran short near the end. The challenge then is not undersampling the original terrain image to much so that it was still recognizable, but not oversampling it either, so that wedo not get a high variance, very higher order polynomial.

Personal note at the end. I found this project somewhat confusing at times. It seemed that different instructors would endorse, or emphasize certain things quite differently. The foremost example being whether or not to penalize the intercept, where there were quite conflicting replies on piazza. Beyond that, I am aware that this report could have been more filled in in places, but I have done the best with the time I had.

5 Conclusion

In this project we have used three different linear regression models: The Ordinary Least Squares, Ridge regression, and Lasso regression.

We first created noisy data from the known Franke function, that we used to train our algorithm, and make a prediction as to the true function f. We then calculated the Mean Square Error(MSE) and R^2 score to find the model and the parameters that minimized the MSE. We concluded that

the OLS recreated the true underlying function the best, with an $MSE = 4.62 \cdot 10^{-3}$ and 0.944.

We performed also linear regression on data taken of a terrain in Norway. We scaled down the image simply by skipping over pixels, and then tried to model the image. In this case, we found that all three models made very poor predictions, where $R^2 \approx 0.6$, and a very high $MSE \sim 10^3$. The reason for the high MSE was not clear, most likely there is a mistake somwhere in our algorithm. In the future, to improve the fit, we could instead, or in addition to, skipping pixels, blur the image with a Gaussian filter.

All code can be found at https://github.com/sefthus/FYS-STK4155/tree/master/project1

References

Richard Franke. A Critical Comparison of Some Methods for Interpolation of Scattered Data, 1979. URL https://calhoun.nps.edu/handle/10945/35052.