Regression methods

Project 1 on machine learning - Revised

Jolynde Vis

October 11, 2019

1 Abstract

Regression is a machine learning task that is used to determine the strength of the relationship between an outcome variable and different input variables. In this report we study various regression methods, including OLS (Ordinary Least Squares), Ridge regression and LASSO regression. Cross-validation is used to properly assess the MSE- and R^2-scores of the different models. We start by fitting a continuous, two-dimensional function, the so-called Franke function. We found out that for this function, the optimal polynomial degree is a degree of seven. We analysed all three methods and evaluated their MSE- and R^2-scores, and showed that both Ridge and LASSO performed worse than OLS regression for the Franke function. Secondly, the three models are fitted to SRTM terrain data of Norway. We observe that the amount of data makes a big difference, therefore we compare both a big dataset (64.000 points) and a small dataset (648 points). After running and evaluating the regression models on their MSE- and R^2-score, we found that for the small dataset LASSO performs best, while for the big dataset OLS regression with a polynomial up to the 40th degree performs best.

Contents

1 Abstract **1**

2 Introduction **3**

3 Theory and method **4**
3.1 Franke's function 4
3.2 SRTM data 5
3.3 General linear models 5
3.4 Ordinary Least Squares 5
3.5 Ridge Regression 6
3.6 Lasso Regression 6
3.7 Resampling techniques 6
3.8 Bias-variance tradeoff 7
3.9 Model evaluation 8

4 Code implementation **9**
4.1 Regression methods for the Franke function 9
4.2 Regression methods for the terrain data 12

5 Results and discussion **13**
5.1 Franke Function 13
5.1.1 OLS regression 13
5.1.2 Ridge regression 17
5.1.3 LASSO regression 20
5.1.4 Comparison of models 20
5.2 SRTM data 22
5.2.1 OLS regression 22
5.2.2 Bias-variance tradeoff 25
5.2.3 Ridge regression 25
5.2.4 LASSO regression 26
5.2.5 Comparison of the three models 29

6 Conclusion and evaluation **31**

7 Appendix **35**
7.1 Model complexity 35

2 Introduction

Machine learning has been growing over the past couple of years, and is becoming more and more important in our society. Machine learning is widely applicable in many different fields, from self-driving cars to solving high-dimensional differential equations or face recognition in devices [3]. One of the main purposes of machine learning is finding patterns or relationships in data sets.

One of the most common tasks in machine learning dealing with this is regression. Regression is used to determine the strength of the relationship between the outcome variable (or the dependent variable) and the input variables (the independent variables). This relationship can either be linear or nonlinear (e.g. polynomial). Regression methods are widely used, from finance and investing to physics and hydrology. There are several types of regression methods, mostly discriminating on the outcome variable, e.g.: simple or multiple linear regression deals with a continuous outcome variable, whereas logistic regression deals with a binary outcome variable and ordinal regression with a categorical (ordinal) outcome variable [8].

This report focuses on studying various regression methods for a continuous function, including the Ordinary Least Squares (OLS) method, which tries to minimize the the sum of the differences squared of the test data and prediction between. Similarly work Ridge and Lasso regression, where at the same time the coefficients for this linear regression are subject to a condition on the sum of their squares and sum of their absolute values. These regression methods are used to fit the Franke function, which is a weighted sum of four exponentials. To assess the regression models,we will use the Mean Squared Error and R^2 score. Furthermore, resampling techniques such as cross-validation are being used. Afterwards, we will use the same techniques to analyze digital terrain data of Norway trying to approximate the terrain by these regression methods and comparing them with each other.

The structure of this report is as follows. The next section explains the methods and algorithms, section three shows the results of the analysis and presents a critical discussion. The report ends with the conclusions and discussion of the project, and options for future research.

3 Theory and method

This section explains the methods and algorithms used for the analysis.

3.1 Franke's function

Franke's function is a function which has been widely used when testing various interpolation and fitting algorithms [3]. It has two Gaussian peaks of different heights, and a smaller dip, as you can see in figure 1. The function is as follows:

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

where $x, y \in [0,1]$.

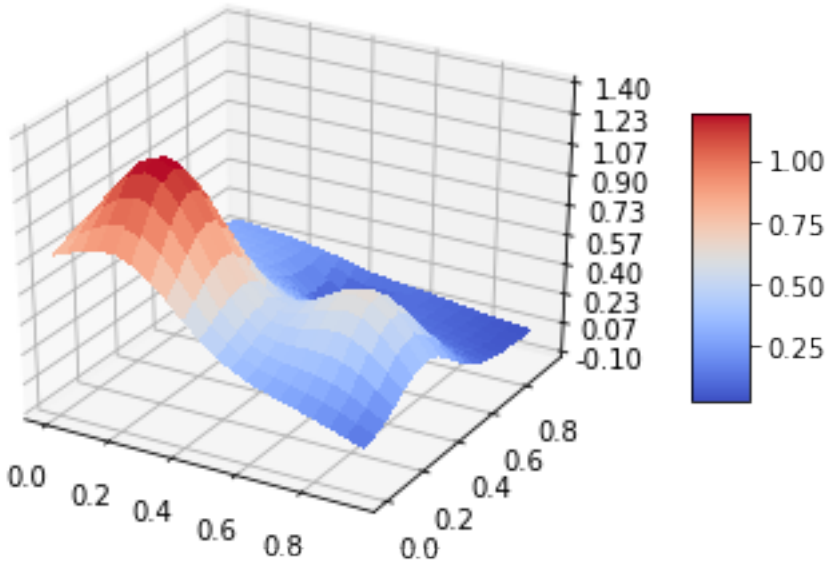


Figure 1: Plot of the Franke function on $[0,1]^2$

3.2 SRTM data

SRTM stands for Shuttle Radar Topography Mission, 'the best quality, freely available digital elevation models worldwide' [4].

The dataset for this research contains the elevation data for a region close to Stavanger, Norway. The data is an 2d-array with length 3601 along the y-axis and 1801 along the x-axis, with the corresponding height values stored inside it.

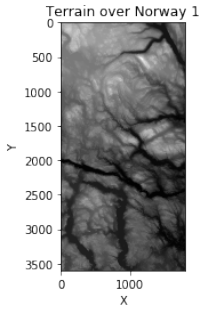


Figure 2: Image of selected terrain data

3.3 General linear models

The general linear model is written as:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Where $\boldsymbol{y}$ is the set of data we are aiming to fit, $\boldsymbol{X}$ is the so-called design matrix, which exists of the input variables, $\boldsymbol{\beta}$ gives the regression parameters and $\boldsymbol{\epsilon}$ contains the errors; representing the noise in the data.

3.4 Ordinary Least Squares

OLS (Ordinary Least Squares) is a method for estimating the unknown parameters in a linear regression model. By defining a function which gives the measure of the spread between the values y_i (the values from the dataset) and $\tilde{y}_i$ (the predicted values), the optimal parameters β_i can be found. This gives the so-called cost function for $\boldsymbol{\beta}$:

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

If the matrix X^TX is invertible, the parameter $\boldsymbol{\beta}$ is defined as follows:

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

3.5 Ridge Regression

When multicollinearity occurs in the design matrix (i.e. a (near-)singular design matrix), the variances of the OLS estimates are large, meaning they may be far from the true value. By adding a regularization parameter, Ridge regression reduces the standard errors. [7]

The multicollinearity leads to that the matrix X^TX is not invertible. By adding the regularization parameter λ as diagonal component the matrix is invertible, i.e. X^TX becomes $X^TX+\lambda I$, where I is the identity matrix. The cost function for $\boldsymbol{\beta}$ then becomes:

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\Big\{(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})\Big\} + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}$$

And the estimate of $\boldsymbol{\beta}$ becomes:

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

3.6 Lasso Regression

Lasso offers another solution to multicollinearity in the design matrix. LASSO stands for Least Absolute Shrinkage and Selection Operator. It shrinks the data values towards a central point, by adding a regularization parameter λ, like in Ridge regression. The difference is that Lasso uses L1 regularization; instead of taking the square of the coefficients, magnitudes are taken into account. This can result in some coefficients becoming eliminated from the model, and therefore helps with feature selection as well. [5]

For lasso regression the cost function for $\boldsymbol{\beta}$ becomes:

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\Big\{(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})\Big\} + \lambda\left\|\boldsymbol{\beta}\right\|_1$$

As λ increases, more coefficients are set to zero and eliminated, and the more the bias increases. As λ decreases, the variance increases. We elaborate more on bias and variance at the end of this section.

3.7 Resampling techniques

Resampling techniques involve repeatedly drawing samples from a training set and refitting the model on each sample, to obtain additional information about the fitted model [6]. There are several resampling methods, for

this project specifically the k-fold cross-validation is used. In k-fold cross-validation the dataset is randomly divided into k subsets (i.e. the number of folds), each equally sized. The model is fitted on the k-1 training sets, and the remaining subset is used as test set. This is done multiple times, so that every subset is used as test data. In the end, the model is summarized by using the model evaluation scores for each sample [3].

3.8 Bias-variance tradeoff

As we saw before, we find the parameters $\boldsymbol{\beta}$ via the cost-function:

$$C(\tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2$$

We can rewrite this as follows:

$$\begin{aligned}
&\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] \\
&= \mathbb{E}[(\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^2] \\
&= \mathbb{E}[(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}] + \boldsymbol{\epsilon} - (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]))^2] \\
&= \mathbb{E}\Big[(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \boldsymbol{\epsilon}^2 + (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \boldsymbol{\epsilon}(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}]) \\
&\qquad + \boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}) + (\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})\Big] \\
&= \mathbb{E}[(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] \\
&\quad + \mathbb{E}[\boldsymbol{\epsilon}(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])] + \mathbb{E}[\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})] + \mathbb{E}\big[(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})\big] \\
&= \mathbb{E}[(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])] \\
&\quad + \mathbb{E}[\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})] + (\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])\mathbb{E}[\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}] \\
&= \mathbb{E}[(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + (\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])\mathbb{E}[\boldsymbol{\epsilon}] \\
&\quad + (\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})\mathbb{E}[\boldsymbol{\epsilon}] + (\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])(\mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}]) \\
&= \mathbb{E}[(\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] \\
&= \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \sigma^2
\end{aligned}$$

In this formula, the left side gives the total error, and the first part of the right side is the bias squared, the second part is the variance and the last part the variance of the error, or the irreducible error in the data [2].

The bias is the difference between the average prediction of the model and the correct value which we are trying to predict. Models with a high bias oversimplify the model which leads to underfitting. Variance is the variability of model prediction for a given data point. Models with a high variance do not generalize and have high error rates on unseen (test) data; i.e. overfitting [2]. The bias-variance tradeoff deals with finding the right balance between this bias and variance.

3.9 Model evaluation

The regression models are evaluated using two different criteria: the MSE (Mean Squared Error) and the R^2 score. The MSE is a measure of the quality of an estimator, and the smaller the value the better the fit (where zero is a perfect fit). The MSE is calculated by:

$$\text{MSE}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2$$

The R^2 score is the coefficient of determination, it indicates how well observed outcomes are replicated by the model. The best possible score is 1.0, whereas a negative R^2 score indicates that the model performs worse than just taking the mean. The R^2 score is given by:

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_i (y_i - \tilde{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \quad \text{,where} \quad \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

4 Code implementation

In this section we will explain some parts of our code implementation. The code for the Franke function ('Regression methods Franke Function) and for the terrain data ('Regression methods Terrain Data') can be found in the Jupyter Notebooks (see email).

4.1 Regression methods for the Franke function

For the Franke function the design matrix is set up as follows:

```
def CreateDesignMatrix_X(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)
    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
```

Where x and y are given by a number of points between 0 and 1, and n is the degree of the polynomial. The true function is the Franke function without the added noise. Normally not available, but since we have it we might as well use it.

The MSE and R2-score are given by the following functions:

```
def MSE(z_data, z_model):
    n = np.size(z_model)
    return np.sum((z_data-z_model)**2)/n
print(MSE(z_true, ztilde))

def R2(z_data, z_model):
```

```
    return 1 - np.sum((z_data - z_model) ** 2) / np.sum((z_data -
        np.mean(z_model)) ** 2)
print(R2(z_true, ztilde))
```

The cross-validation is implemented with the following function:

```
def cross_validation(x, y, k):
    n = len(x)
    indexes = np.arange(y.shape[0])
    np.random.shuffle(indexes)
    x = x[indexes]
    y = y[indexes]

    r2_train = []
    r2_test = []
    mse_train = []
    mse_test = []
    bias = []
    variance = []

    for i in range(k):
        x_train = np.concatenate((x[:int(i*n/k)], x[int((i +
            1)*n/k): ]), axis = 0)
        x_test = x[int(i*n/k):int((i + 1)*n/k)]
        y_train = np.concatenate((y[:int(i*n/k)], y[int((i +
            1)*n/k): ]), axis = 0)
        y_test = y[int(i*n/k):int((i + 1)*n/k)]

        beta =
            np.linalg.pinv(x_train.T.dot(x_train)).dot(x_train.T).dot(y_train)
        ytilde = x_train @ beta
        ypredict = x_test @ beta

        mse_train.append(MSE(y_train, ytilde))
        mse_test.append(MSE(y_test, ypredict))
        r2_train.append(R2(y_train, ytilde))
        r2_test.append(R2(y_test, ypredict))
        bias.append(np.mean((y_test - np.mean(ypredict))**2))
        variance.append(np.mean(np.var(ypredict)))
```

```
    r2_train = np.array(r2_train)
    r2_test = np.array(r2_test)
    mse_train = np.array(mse_train)
    mse_test = np.array(mse_test)
    bias = np.array(bias)
    variance = np.array(variance)

    return r2_test, mse_train, mse_test, bias, variance, r2_train
```

Where k is the number of folds. This function returns everything we need to cross-validate. The same cross-validation function is used for Ridge regression, where beta is replaced with the definition that gives the beta for Ridge. Ridge regression is implemented as follows:

```
betaridge = np.linalg.inv(X_train_r.T.dot(X_train_r)+
       (_lambda*np.eye(len(X_train_r[0])))).dot(X_train_r.T).dot(z_new_train)
zridge = X_train_r @ betaridge
ridge_predict = X_test_r @ betaridge
```

Where the following code with GridSearchCV is used to find the optimal value for lambda.

```
nlambdas = 50
lambdas = np.logspace(-7, 2, nlambdas)
param_grid = {'alpha': lambdas}

ridge = skl.Ridge()

ridge_regressor = GridSearchCV(ridge, param_grid,
    scoring='neg_mean_squared_error', cv=kfold)
ridge_regressor.fit(X_new[:,1:], z_1)
```

The LASSO regression is implemented as follows:

```
clf_lasso = skl.Lasso(alpha=l_lambda, precompute = True, tol = 10,
    max_iter = 10e3).fit(X_new_train, z_new_train)
pred_lasso = clf_lasso.predict(X_new_test)
```

To validate our own algorithms they are compared with the corresponding sk-learn functionalities. To verify the results sometimes an simplified plot or calculation is used (for example one without cross-validation and one with cross-validation, to compare the results.

4.2 Regression methods for the terrain data

For the terrain data we used the same functions, with a few tweaks. Firstly, the terrain data has been standardised, because the values are fairly big. This has been done by subtracting the mean and dividing by the standard deviation. Secondly, since we have a lot of data, we made the array which gives the data smaller, to make computations easier. Therefore, we remove the last column and row to get a 3600 by 1800 array which we now can make smaller. This is done by setting the new value as the average over a small array inside our 3600 by 1800 array, to get an 360 by 180 array. For these purposes, we used the following code, where k the factor is by which each axis will get shorter.

```
def rebin(a, shape):
    sh = shape[0],a.shape[0]//shape[0],shape[1],a.shape[1]//shape[1]
    return a.reshape(sh).mean(-1).mean(1)

terrain1 = terrain1[:3600, :1800]
terrain1 = (terrain1 - np.mean(terrain1)*np.eye(terrain1.shape)) /
    np.std(terrain1)
terrain1 = rebin(terrain1, (3600//k, 1800//k))
```

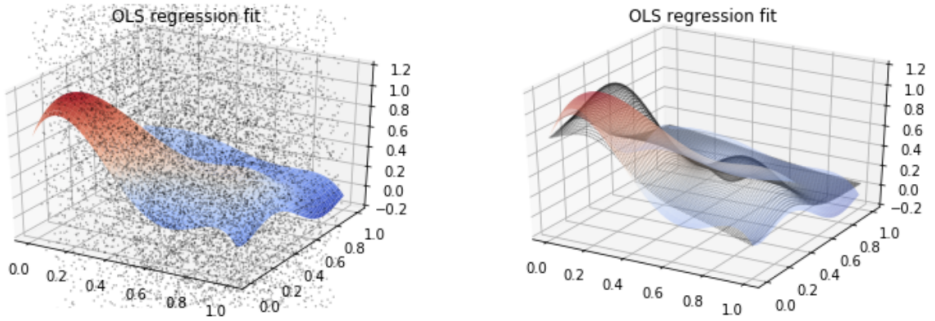
5 Results and discussion

This section starts with presenting the results of the three regression methods on the Franke function, and a discussion of these results. The section continues with the results of the regression mehtods on the SRTM data, and ends with a discussion of which regression method fits the SRTM data the best.

5.1 Franke Function

5.1.1 OLS regression

After creating the data with 250 data points in x and y and a polynomial degree of 5, we start with fitting an OLS regression model to the data. The data is split into 80% training data and 20% test data.



(a) OLS regression fit with the noisy data(b) OLS regression fit with the real data

Figure 3: Plot of OLS regression model fit

Figure 3a shows the noisy data (Franke function with added noise), and a colored surface plot of the fitted OLS regression on this (noisy) data. Figure 3b shows the scattered data (the black dots) of the real data (the Franke function without the noise), and a colored surface plot of the same fit of the OLS regression (on the noisy data). All the models are always trained on the noisy data, where the goal is to predict the real function as good as possible.

Next we can compute the MSE-score and R^2-score for the training, test and the real data. The results are in table 1.

We observe a big difference between the MSE-score for the training/test data and the MSE-score for the true data. This is due to the noise we've

Table 1: MSE- and R^2-scores for OLS regression with polynomial degree 5.

	MSE-score	**R2-score**
Training data	1.010	0.071
Test data	1.020	0.077
Real function data	0.002	0.971

added to the Franke function. The noise is added with a multiplication of 1, which is why the MSE-score is around 1. The R^2-score also differs greatly for the training and test data and the true data. However, seeing that we want the model to predict the 'real' data (i.e. the true function), the R^2-score for the real function is most relevant one. The MSE- and R^2-score for the real data are both very good, showing that the OLS regression model with polynomial degree 5 is a pretty good fit. In the next sections we will see if we can optimize our OLS fit further, by trying to minimize the MSE-score.

K-fold cross-validation

Instead of using a train-test split, which tests our data on a small test set (20% of the dataset), we will use cross-validation to assess the model. We use a k of 10, meaning that we do a 10-fold cross-validation, where the data is split into 10 parts (i.e. folds, randomly picked), the model is trained on 9 parts (90% of the data) and is tested on the remaining part (10% of the data), which is done 10 times. That means that all the data has been used as both training and once as test data. The MSE- and R^2-scores for the 10 CV-folds are averaged and displayed in table 2, together with the standard deviation of these 10 scores.

Table 2: MSE- and R^2-scores for cross-validated OLS regression with polynomial degree 5.

	MSE-score	**MSE-score CV (std)**	**R2-score**	**R2-score CV (std)**
Training data	1.010	1.012 (+/- 0.003)	0.071	0.072 (+/- 0.002)
Test data	1.020	1.013 (+/- 0.025)	0.077	0.071 (+/- 0.010)
Real function data	0.002	N/A	0.971	N/A

We see that the standard deviations are fairly low, which means that the model has a consistent performance on different test sets. We also see that the scores for OLS without cross-validation fall in the range of the CV-scores and their standard deviation.

The MSE- and R^2-score for the real data is not available anymore unfortunately, because otherwise we would have to test the model on the real data as well. We will optimize the MSE-score for the test data.

The SVD and pseudo-inverse

A standard matrix inversion might lead to (near-) singular matrices. To prevent this, we can use the SVD (Singular Value Decomposition). However, we found out fairly late that the reason that our MSE-score exploded after the 13th/14th polynomial degree was probably due to this singularity (the higher the polynomial degree, and the bigger the design matrix, the bigger the chance on near-singularity). Seeing that we did not have a lot of time left, we decided not to implement the SVD-algorithm from scratch, but we used Numpy's pinv, which computes the pseudo-inverse of a matrix, using its singular value decomposition (tip given at the data lab).

Complexity of the model

Using our code for cross-validation, we will start to optimize our OLS regression model. We start by looking at which polynomial degree (i.e. the complexity of the model) fits the data the best. This is plotted in figure 4.

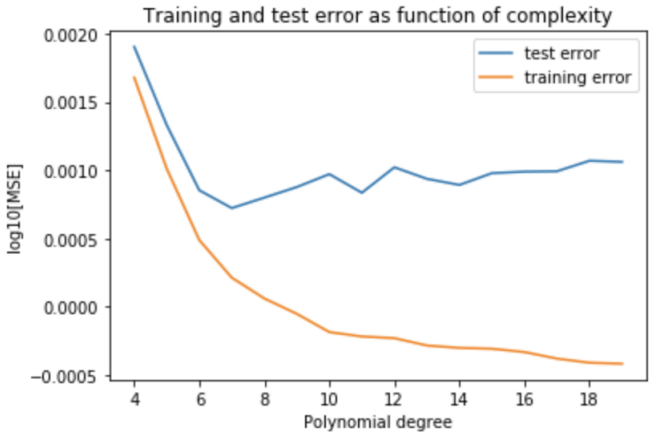


Figure 4: Training and test error as a function of complexity

The lines are plotted starting at a polynomial degree of 4 (the graph starting at degree 1 can be found in appendix 1). Figure 4 shows that the training error will decrease when the complexity increases, this is because the model will start to fit every point in the training data. However, if we would let the model do this, it would not be able anymore to perform on unseen data, because it has fitted all the noise in the training data as well. This is called overfitting. We can see that our model performs best for a polynomial degree of 7, because from this point the test error starts rising again. Appendix 1 also shows a plot for the error on the true data, which confirms this.

Bias-variance tradeoff

Now we can plot the bias and the variance for the OLS regression model, as seen in figure 5.

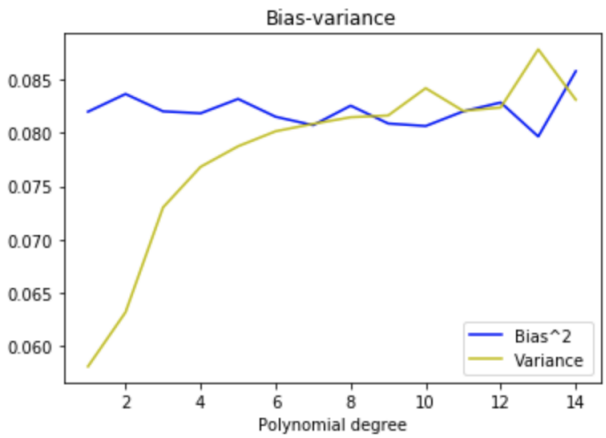


Figure 5: Bias and variance plotted against complexity

We see that for a polynomial of 7, there is a good tradeoff between the bias and the variance. It is where they first cross, and where the variance is not that high yet and the bias is not too low. However, we can also see that for our data, the bias-variance tradeoff looks a bit different. The variance shows a logarithmic growth instead of an exponential growth. This could happen if the different data points in the data are far from each other. If we look at the Franke function, we see that it has a relatively high peak and a few dips, meaning that when the complexity of the model increases it will go towards these peaks and dips, which will cause the variance to rise quickly.

The bias for the model seems to be decreasing, however very gentle. This could be due to the amount of noise in the model, and to the number of data points. The added noise is normal distributed, meaning it is everywhere. So when the model is not complex, it does not capture the function points, which cause the bias. But when the model gets more complex, it captures the function better, but not the noise in the middle anymore. Because we have a lot of data points, the model is not overfitting yet at a polynomial degree of 15. Eventually, when the complexity goes towards infinity, the bias will decrease because the model will capture both the data points and the noise.

In table 3 are the results for the OLS regression model for a polynomial degree of 5 and a polynomial degree of 7.

Table 3: MSE- and R^2-scores for OLS regression with polynomial degree 5 and polynomial degree 7.

	Polynomial degree = 5	**Polynomial degree = 7**
MSE-score	1.013 (+/- 0.020)	1.012 (+/- 0.023)
R2-score	0.071 (+/- 0.020)	0.072 (+/- 0.010)
MSE real function	0.0024	0.0014
R2 real function	0.971	0.983

We see that for a polynomial degree of 7 all the scores are better.

Now we will have a look at Ridge and LASSO regression to see if these methods can optimize our fit more.

5.1.2 Ridge regression

For Ridge we need to find the optimal value of the hyperparameter λ. We can find this value by plotting the MSE-score and the value of λ. This is done in figure 6.

We can see in figure 6 that as λ goes to zero, the MSE-score gets lower. When λ is zero, we end up with a regular OLS regression. Meaning, this graph indicates that Ridge will not perform better, because it keeps decreasing for a lower value of λ.

Next, we plot the MSE-scores against the model complexity, for different values of λ, see figure 7.

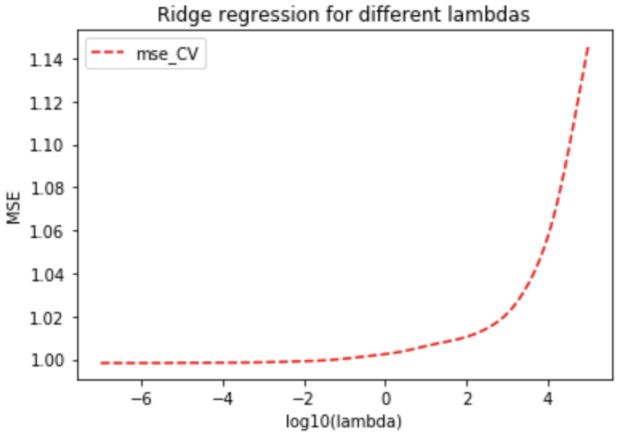
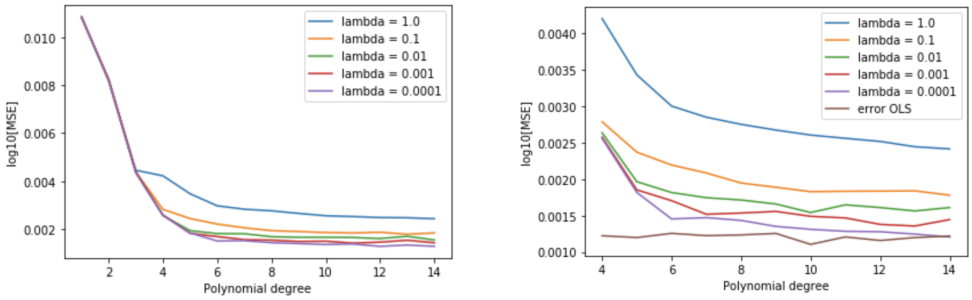


Figure 6: Ridge regression, the MSE-score and λ



(a) Model complexity and lambda (b) Complexity and lambda with OLS

Figure 7: Relation between complexity, λ and MSE-score

We see that the MSE-scores for the Ridge regression steadily go down, and that some (the lower λ values) seem to flatten out after the 14th degree. When we compare this to the MSE of the OLS regression (figure 7b), we see that the MSE-score for the OLS regression is still lower than the MSE-scores for Ridge regression.

Bias-variance tradeoff Ridge

Next we have a look at the bias-variance tradeoff for Ridge regression, for different λ's (see figure 8 and 9).

Ridge regression shrinks the coefficients towards zero, so it decreases the variance of beta, so when λ increases the variance decreases. If we look at

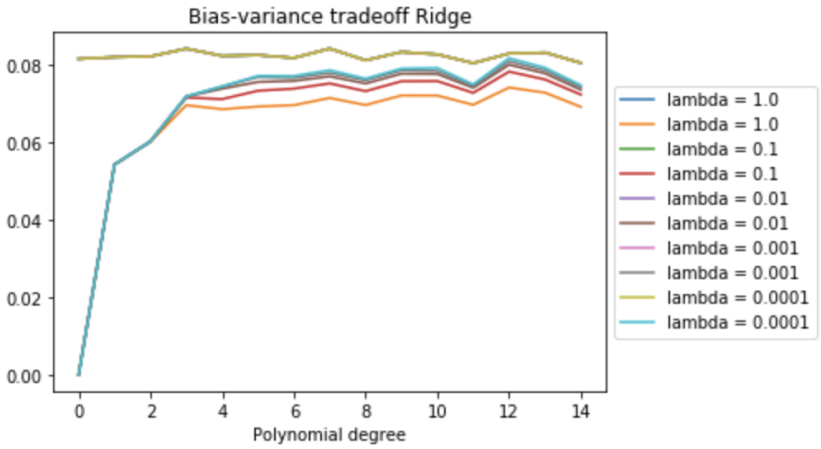
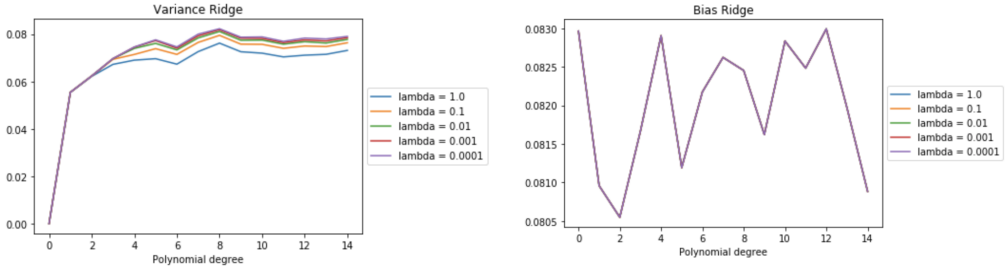


Figure 8: Error plotted against complexity



(a) Variance for different λs

(b) Bias for different λs

Figure 9: Variance and bias depending on model complexity and λ

figure 7b, we see that for $\lambda = 1.0$ the variance is the lowest. Furthermore, the graph of the variance is comparable to the variance graph for OLS. The bias, on the other hand, should increase when a greater value for λ is used. Because the coefficients are shrunk more towards zero for a greater value of λ, the estimated values will be further away from the actual values. Figure 7b is a plot of the bias for different values of λ, however the graph does not explain this theory, seeing that the bias does not change for different values of λ here. In the bias-variance graph in figure 8, we see the same result as for OLS regression, the bias seems to decrease very slowly.

5.1.3 LASSO regression

For the LASSO regression we use the build-in functionality of sk-learn. We can make the same plot for λ as we did for the Ridge regression. The graph for LASSO also shows that for a bigger value of λ the error will increase, indicating that LASSO is not a better method than OLS for this dataset.

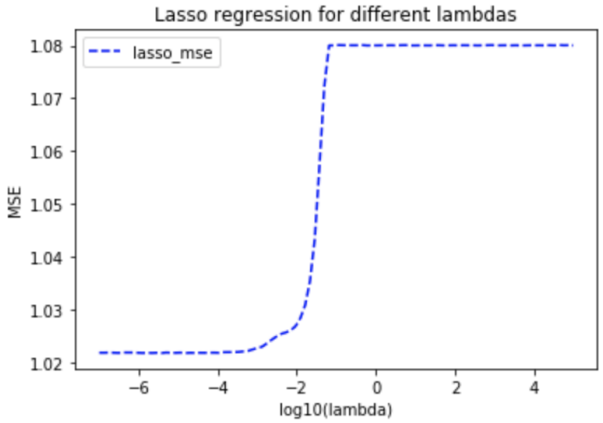


Figure 10: Lasso regression for different values of λ

For LASSO, when λ increases, the bias increases as well. When λ decreases, the variance increases.

5.1.4 Comparison of models

Finally we can compare the scores of the different models. To find the final optimal values of λ for Ridge and LASSO we use GridSearchCV from sk-learn, which uses a cross-validated grid search to find the lowest MSE-score for a certain value of λ. Using this, the optimal values for λ for both Ridge and LASSO are 10^7. The values of λ to search between were from 10^7 to 10^2, given that GridSearchCV gives the lowest value here indicates that it would go further to zero, towards OLS regression, if possible. The results are in table 4.

We see that OLS and Ridge perform almost equally, while the LASSO regression is notably worse.

The reason for LASSO regression to perform bad on this dataset could be explained by that apparently the features in the dataset are not correlated.

Table 4: Regression results, K-fold CV with k=10.

	OLS	**Ridge** ($\lambda = 10^{-7}$)	**LASSO** ($\lambda = 10^{-7}$)
MSE-score	1.012 (+/- 0.023)	1.012 (+/- 0.044)	1.027 (+/- 0.032)
R2-score	0.072 (+/- 0.010)	0.072 (=/- 0.010)	0.058 (+/- 0.008)

Since LASSO applies an 'automatic feature selection' by reducing some coefficients to zero, some variables could be eliminated. It also regulates the parameters. So it eliminates and shrinks features at the same time. Because our dataset is quite large, the risk of overfitting is fairly low. When applying LASSO, the model is most likely underfitting the data, due to the elimination and shrinkage of the coefficients. For LASSO the same applies as with Ridge, when λ is zero we have a regular OLS regression.

The scores for OLS and Ridge are almost the same, however, the standard deviation scores are slightly lower for OLS. Because λ is very small, and Ridge does not set coefficients to zero, but shrinks them towards zero, the results are almost the same as for OLS. The reason why Ridge is not better than OLS could be; for a large value of λ the variance decreases but the bias increases. Since we have a lot of data, the model won't overfit quickly. So when applying Ridge, λ goes almost to zero for the lowest error, seeing that otherwise the bias of the model will increase. We saw in the graph of the bias-variance tradeoff for OLS (figure 8) that the bias is very low for this model, so when applying Ridge the bias will be greater for every value of λ then it is for OLS, explaining that OLS is a better fit for this data.

Lastly, we can compare the scores for the real data for the three models, see table 5. Here we see the same results, OLS and Ridge are almost the same (the difference is further down in the scores), while LASSO is arbitrarily worse.

Table 5: Regression results on the real function.

	OLS	**Ridge** ($\lambda = 10^{-7}$)	**LASSO** ($\lambda = 10^{-7}$)
Real function: MSE-score	0.001	0.001	0.019
Real function: R^2-score	0.983	0.983	0.765

5.2 SRTM data

For the SRTM data we start by normalizing the terrain data. The values for the elevation data are between 50 (min) and 1865 (max), which would give large MSE-scores otherwise. The x and y axis, as seen in figure 11, are given by points from 0 to 1800 and 0 to 3600, respectively. Since a grid of 1800 by 3600 points is fairly big, we (randomly) downsample the data so that we have 10% of the original data. This gives a 360 by 180 point grid. Figure 11 shows a plot of the data.

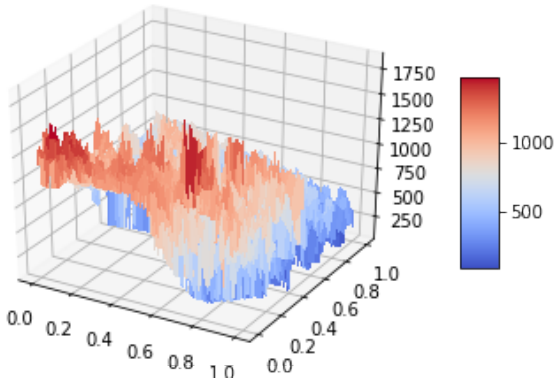


Figure 11: Plot of the terrain data

5.2.1 OLS regression

Same as for the Franke function, we started with fitting an OLS regression model with polynomial degree 5 on the terrain data, see figure 12.

We split the data in 80% training and 20% test data again. We use the pseudo-inverse (numpy.pinv) again to prevent problems with (near-) singularity. We fit the model with polynomial degree 5 on the training data and test it on the unseen test data, and apply cross-validation (with 10 folds) to assess the model. The results can be seen in table 6. We see that the scores do not differ much, but the standard deviations of the cross-validation are very low, which is because the amount of data is so big.

After this we optimize the OLS regression model again. We start by plotting the MSE-score against the complexity of the model.

In figure 13a we see that both the training and test error steadily go down when the complexity increases. We can even plot this graph up until

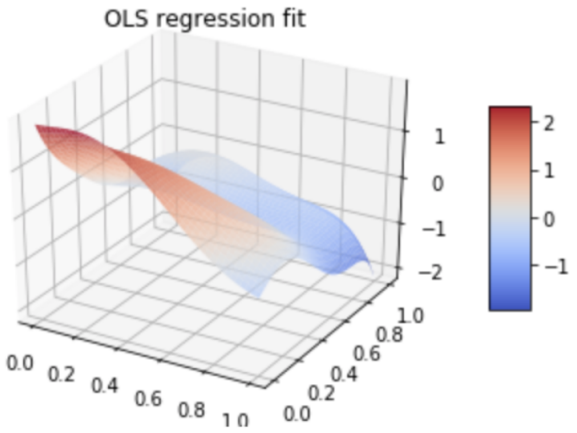
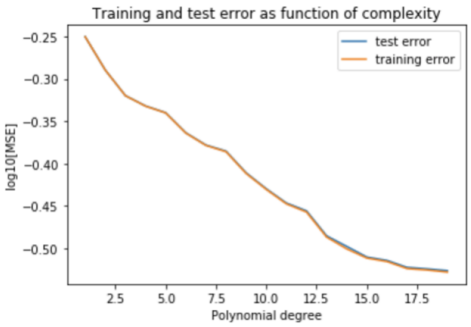


Figure 12: Fit of the OLS regression on the normalized terrain data

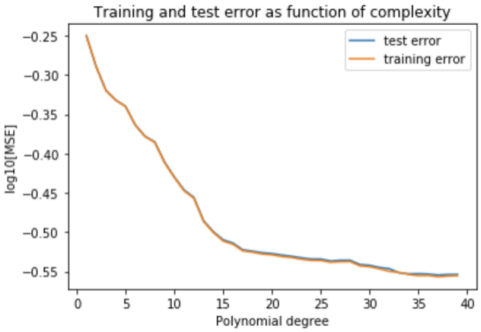
Table 6: MSE- and R^2-scores for cross-validated OLS regression with polynomial degree 5.

	MSE-score	**MSE-score CV (std)**	**R2-score**	**R2-score CV (std)**
Training data	0.455	0.457 (+/- 0.017)	0.541	0.540 (+/- 0.001)
Test data	0.463	0.457 (+/- 0.017)	0.540	0.540 (+/- 0.012)

the 40th polynomial degree, and still the test error does not start to increase (it does however flattens out). Moreover, the training and test error are very similar. This result can be explained by the size of the data. Because we are using a grid with 64.000 points, and we have 64.000 points in the terrain data as well, we have a lot of data, all within 0 to 1. This essentially means that overfitting is nearly impossible, at least not for polynomial degrees who are equal to or smaller than 40. There will be some point where it starts to overfit, and in figure 13b it seems that the test error will start to rise, but our computational power (and time) is not enough to create a graph with a polynomial degree above 40. To prove this explanation, figure 14a shows a plot with a grid of 2592 (72 x 36) points, where we observe that the test error starts being different from the training error. However, the test error still seems to decrease up until the 40th degree. Downsampling to 648 datapoints gives figure 14b, where we observe that from a polynomial degree of 15 the



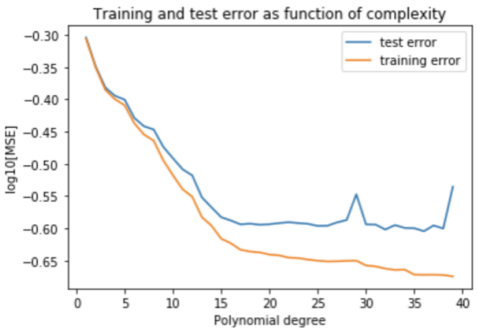
(a) Complexity up to degree 20



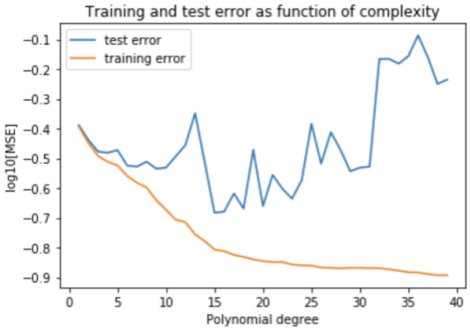
(b) Complexity up to degree 40

Figure 13: MSE-score and model complexity for OLS regression

test error starts rising.



(a) Number of datapoints = 2592



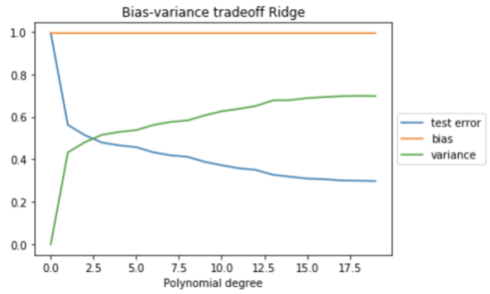
(b) Number of datapoints is 648

Figure 14: MSE-score and model complexity for OLS regression with down-sampling

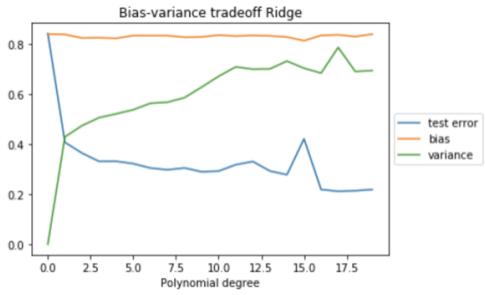
We will continue with both samples (dataset 1 = 64.000 datapoints, dataset 2 = 648 datapoints), to analyse the effect of Ridge and LASSO regression. However, for the 10% data sample, we will use a polynomial degree of 15, seeing that this is the point where the error starts flattening out (and because computing power and time do not allow us to use a polynomial degree of 40 or higher).

5.2.2 Bias-variance tradeoff

Now we can plot the bias-variance tradeoff for Ridge, for both dataset 1 and dataset 2.



(a) Number of datapoints = 64.000



(b) Number of datapoints is 648

Figure 15: MSE-score and model complexity for OLS regression with down-sampling

The bias-variance tradeoff shows in figure 15a that for the 64.000 datapoints the variance rises very slowly, while the bias remains one. The variance rises so slow due to the fact that there are so many data points. When we decrease the number of datapoints (figure 15b we see that the bias rises much quicker. The bias remains one, because there are so many datapoints (figure 15a). For dataset 2 we see that the bias changes a little (figure 15b).

We can now compare the scores for the OLS regression models, see table 7.

We see in table 7 that for both data sets the higher order polynomial performs better. We also see that for a smaller dataset, the OLS model performs better. However, the standard deviations of the cross-validations on the test set are almost 10 times higher, meaning that the regression model is less stable. We will now have a look at Ridge and LASSO.

5.2.3 Ridge regression

We start by plotting different values for λ against the MSE-score.

We see in figure 16a that for dataset 1 a smaller lambda gives a smaller error. In figure 16b we see that for dataset 2 the graph looks almost the same. Using GridSearchCV we find for both datasets an optimal lambda value of 1e-07.

Table 7: MSE- and R^2-scores OLS regression with polynomial degree 5 and 15.

	Training data (64.000 points)	**Test data (64.000 points)**	**Training data (648 points)**	**Test data (648 points)**
MSE-score OLS (degree = 5)	0.457 (+/- 0.003)	0.457 (+/- 0.029)	0.300 (+/- 0.013)	0.322 (+/- 0.115)
R2-score OLS (degree = 5)	0.540 (+/- 0.003)	0.540 (+/- 0.023)	0.642 (+/- 0.012)	0.615 (+/- 0.120)
MSE-score OLS (degree = 15)	0.308 (+/- 0.002)	0.310 (+/- 0.011)	0.157 (+/- 0.006)	0.225 (+/- 0.127)
R2-score OLS (degree = 15)	0.690 (+/- 0.002)	0.688 (+/- 0.014)	0.812 (+/- 0.012)	0.726) (+/- 0.127

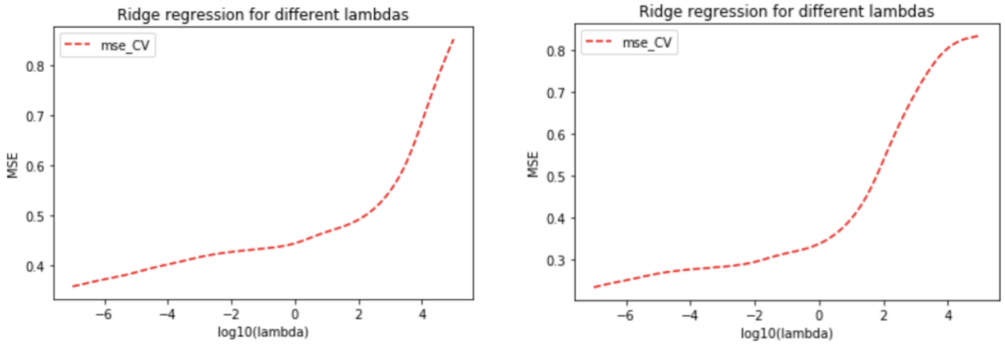
When we compare different values for lambda with the model complexity, we see in figure 17 that for both dataset 1 and 2 the lowest value for lambda gives the lowest score. The MSE-scores for dataset 1 are more smooth due to the bigger amount of data in that dataset. The MSE-scores seem to flatten out from around the 6th polynomial degree.

We will compare the model scores for Ridge regression with the model scores for OLS and LASSO in table 8, after the section on LASSO regression.

5.2.4 LASSO regression

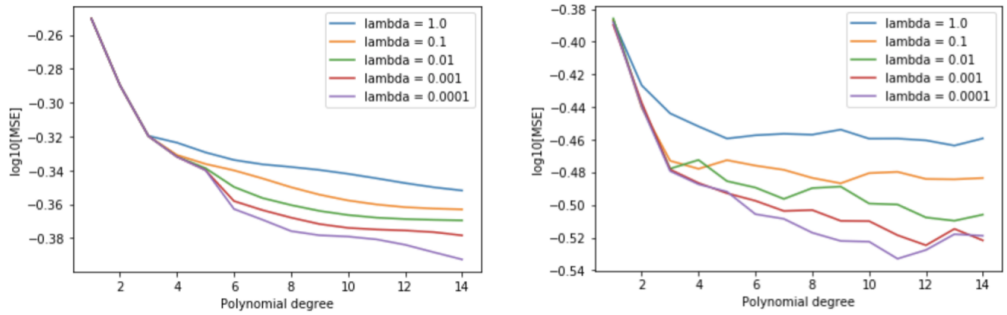
We start by plotting different values for λ against the MSE-score.

Figure 18a shows, for dataset 1, that for LASSO, as well as for Ridge, the MSE goes down when the value for lambda goes down. However, when using GridSearchCV for the LASSO model, we find an optimal value for lambda = 0.0039, and if we zoom in (see figure 18b), we see that there is a dip in in the graph. For dataset 2 we observe the same (see figure 19a), and GridSearchCV gives the same value for lambda (0.0039). When we zoom in on this graph (figure 19b) we see that the MSE-scores are much more scattered though, due to the low amount of data.



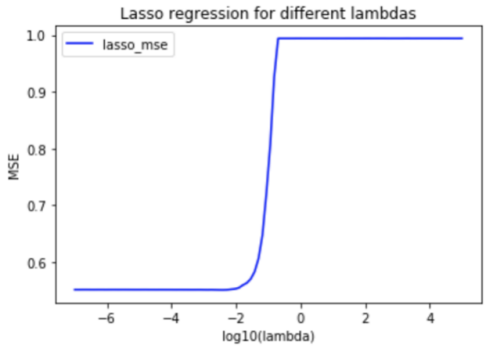
(a) Number of datapoints = 64.000 (b) Number of datapoints = 648

Figure 16: Ridge regression for different lambdas, for different amounts of datapoints

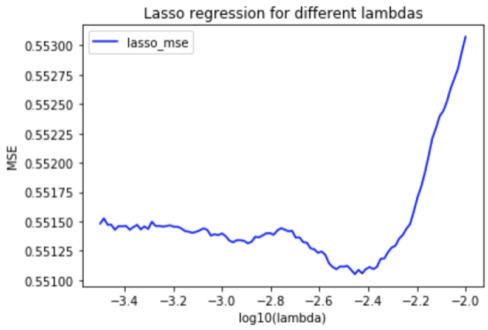


(a) Number of datapoints = 64.000 (b) Number of datapoints = 648

Figure 17: Ridge regression and model complexity for different lambdas, for different amounts of datapoints

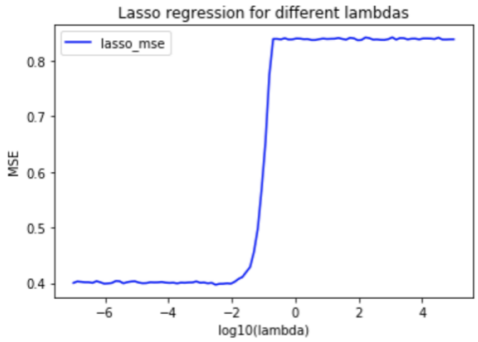


(a) log10(lambda) from -7 to 5

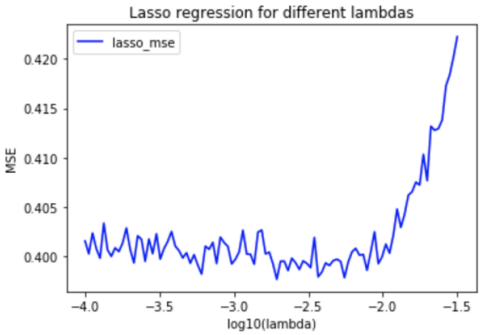


(b) log10(lambda) from -3.5 to -2

Figure 18: LASSO regression for different lambdas, for dataset 1 (64.000 points)



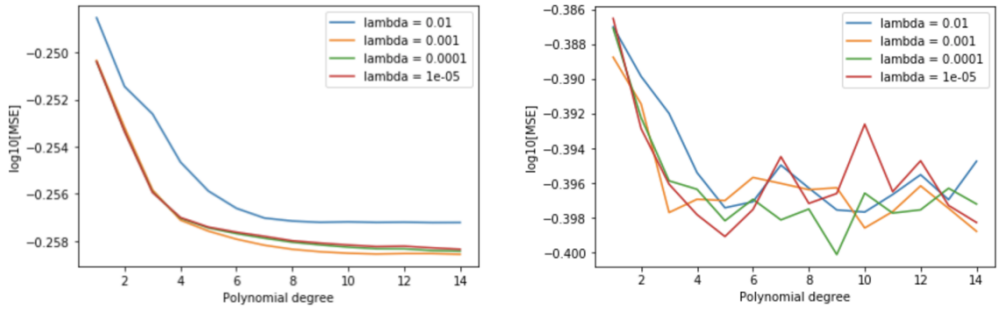
(a) log10(lambda) from -7 to 5



(b) log10(lambda) from -3.5 to -2

Figure 19: LASSO regression for different lambdas, for dataset 2 (648 points)

Then we compare different values for lambda with the model complexity, see figure 20.



(a) Number of datapoints = 64.000 (b) Number of datapoints = 648

Figure 20: LASSO regression and model complexity, for different lambdas, for different amounts of data

We see in figure 20 that for dataset 1 the MSE-score for LASSO first descents, but then remains the same for higher polynomial degrees. For dataset 2 the same seems to happen, however this data is somewhat harder to interpret because it is not as smooth. However, when we compare the values on the Y-axis for the MSE-score, we see that the MSE-score for LASSO on dataset 2 is much lower.

Next we will compare the three models on the two datasets.

5.2.5 Comparison of the three models

We see that OLS regression with a polynomial degree of 15 performs better than Ridge and LASSO for dataset 1, where it has the lowest MSE-score and the highest R2-score. Would we increase the polynomial degree even further, these scores will improve further due to the high amount of data. For dataset 2, the most interesting observation is that the MSE-score for the LASSO model is lower than the MSE-score for the OLS regression. However, the standard deviation is a lot bigger. Remarkable is also that the R2-score for the LASSO is the worst of the three models. However, this can be explained. An R2-score will never decrease, and most of the time even increase, when predictors are added to a model. This means that a model with more terms may appear to have a better fit, simply because it has more predictors [1]. Since LASSO penalizes the coefficients and sets some of them

Table 8: MSE- and R^2-scores for the different regression models.

	Dataset 1 (64.000 data points) CV 10-fold (std.)	**Dataset 2 (648 data points) CV 10-fold (std.)**
MSE-score OLS	0.309 (+/- 0.009)	0.207 (+/- 0.089)
MSE-score Ridge (λ = 1e-07)	0.358 (+/- 0.013)	0.289 (+/- 0.110)
MSE-score LASSO (λ = 0.0039)	0.551 (+/- 0.028)	0.152 (+/- 0.152)
R2-score OLS	0.689 (+/- 0.012)	0.745 (+/- 0.097)
R2-score Ridge (λ = 1e-07)	0.640 (+/- 0.009)	0.649 (+/- 0.148)
R2-score LASSO (λ = 0.0039)	0.446 (+/- 0.015)	0.512 (+/- 0.211)

to zero, it effectively reduces the number of coefficients (predictors) in the model. Therefore, the R2-score is not a good way to compare a LASSO model with other models. In conclusion, for a small amount of data, the LASSO model performs better than OLS and Ridge. For the large batch of data, OLS outperforms Ridge and LASSO, seeing that overfitting is (almost) impossible.

6 Conclusion and evaluation

In this report we started by looking at the Franke function, a continuous, two-dimensional function. We fitted three different regression models to this function; OLS regression, Ridge regression and LASSO regression. Cross-validation was used as resampling technique to properly assess the different models. The three models are compared on their MSE-score and R^2-score, both for the training and test data, and in the end also for the real data without the noise. We started by assessing the MSE-score for OLS as a function of the model complexity, where we showed that the optimal polynomial degree, which gives the lowest MSE-score, is a degree of seven. We also assessed the bias-variance tradeoff as a function of the complexity of the model, which confirmed our initial optimal polynomial degree of seven.

Using this polynomial degree, we looked at the values for λ for the Ridge regression. It turned out that the smaller λ gets, the lower the MSE-score for Ridge, already indicating that Ridge might not be a better fit. We also discussed the MSE-score and bias-variance tradeoff for different values of λ and model complexity. Here we also saw that the smaller λ, the lower the MSE-score. We found that the optimal value for λ is 10e-7. We also saw that the MSE-score of OLS was still lower than the MSE-score for Ridge regression.

Then we looked at LASSO regression, where we also found that the smaller the value for λ, the lower the MSE-score. We found that the optimal value for λ is 10e-7.

When comparing the MSE-scores and R^2-scores for the three different models, with their optimal hyperparameters, we saw that for the Franke function, the OLS regression gave the best scores. However, Ridge regression gave almost the same results.

Next we fitted the three regression methods on terrain data of a region close to Stavanger, Norway. We found out that our first sample, which was only 10% of the data, was still very big, which meant that overfitting was nearly impossible and we could go up to 40th degree polynomial without the MSE-score for the test data rising again. We then used two sets of the data, one with the 10% sample (dataset 1, consisting of 64.000 points) and one with only 648 points (dataset 2). When using dataset 2, we found out that the optimal polynomial degree for OLS was 15.

We then fitted a Ridge regression and a LASSO regression on both dataset 1 and dataset 2. For Ridge the optimal value of λ was 1e-07, and for LASSO

this optimal value was 0.0039.

We compared the three different models on both dataset 1 and dataset 2, and found out that for dataset 1 (with 64.000 points), OLS regression performs the best. Simply because overfitting with this many datapoints is nearly impossible. We used a polynomial degree of 15 for the OLS regression, but if we would use a polynomial degree of 40 the scores for the OLS regression would be even lower, without overfitting the data. For dataset 2 (with 648 points), we saw that LASSO gives the lowest MSE-score, which remarkable goes together with the lowest R2-score of the three models. This comes from the fact that models with more parameters have a higher R2-score, and since LASSO shrinks some of the coefficients to zero, the model contains less parameters. Therefore the LASSO model is the best fit for dataset 2 of the terrain data.

Future research

Options for future research would be to look into downsampling the data for the Franke function, and see if Ridge and LASSO start to perform better when there is less data, and the risk of overfitting becomes higher. For the terrain data it would be useful to look at how far the terrain data can be downsampled, without losing information. Where is the optimal point between enough data, but also fast computing speed (so not too much data).

Project feedback

Lastly, I would like to give some feedback on the project. I think the project is really interesting and useful, and I've learned a lot. It was very hard at the beginning, seeing that I do not have a physics or computational sciences background (my background is Industrial Design Engineering (Bsc.) and Innovation Sciences (Msc.) and that all of my courses so far always dealt with more 'social' problems. Therefore at the beginning I was really confused by using regression methods for 'some 3d function', and for terrain data, while I've only used regression methods to measure relations between for example number of patents and company size. However, in the end I really understood what I was doing, even though it took some time, and I learned a lot during this project. I think that for people who do not have a lot of experience with programming in Python (like me), this assignment takes even more time. Because even though you know what you want to do and why you have to do that, you don't know how to do it in Python. This takes up a lot of extra time in the beginning, where you cannot work on

your project. Maybe it would be an idea to do a 2 or 3 hours crash-course in Python somewhere in the first week, for people who are not very experienced.

References

[1] Minitab Blog Editor. Multiple regression analysis: Use adjusted r-squared and predicted r-squared to include the correct number of variables., 2013. Accessed: 10.10.2019.

[2] et al. Hastie, T. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics, 2008.

[3] Morten Hjorth-Jensen. Lectures notes in fys-stk4155. data analysis and machine learning: Linear regression and more advanced regression analysis, 2019. Accessed: 13.09.2019.

[4] et al. K. G. Nikolakopoulos. Srtm vs aster elevation products. comparison for two regions in crete, greece. *International Journal of Remote Sensing*, 27(21):4819–4838, 2006.

[5] Towards Data Science. Ridge and lasso regression: A complete guide with python scikit-learn., 2018. Accessed: 13.09.2019.

[6] Towards Data Science. How good is your model? — intro to resampling methods., 2019. Accessed: 13.09.2019.

[7] NCSS Statistical Software. Chapter 335 ridge regression, n.d. Accessed: 13.09.2019.

[8] Statistics Solutions. What is linear regression, 2013. Accessed: 13.09.2019.

7 Appendix

7.1 Model complexity

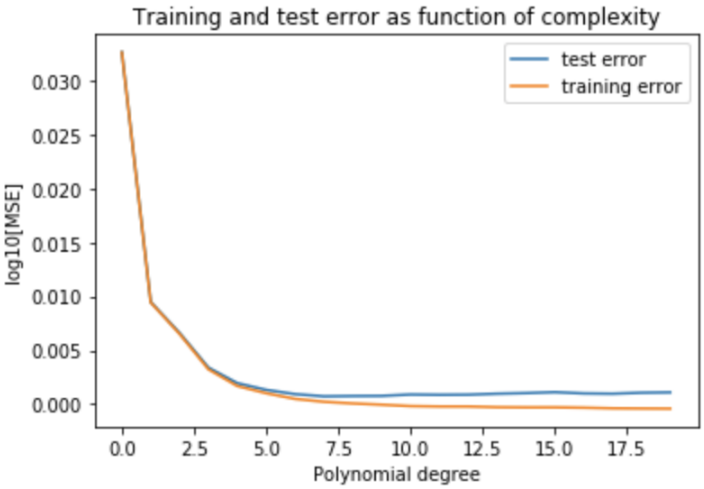


Figure 21: Comparing training and test error

In figure 21 we see that for a polynomial degree in the order of zero and one the error is relatively so big that the rest of the graph is not readable anymore.

The graph in figure 22 shows that for a polynomial of 7 we also have a low MSE-score for the real function. However, interpreting the results of this graph should be with caution, since there is no cross-validation used this graph is based on a small sample of the data, and on one run. Even though, it confirms our results.

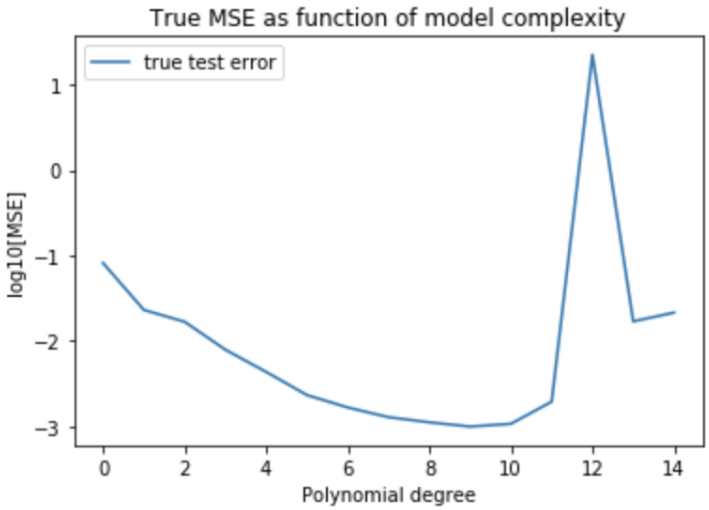


Figure 22: Error on test data