FYS-STK4155

REGRESSION ANALYSIS & RESAMPLING METHODS

October 7, 2019

Marius Havgar

&

Åsmund Danielsen Kvitvang

Abstract

The machine learning landscape is rapidly expanding, and the models are becoming increasingly complicated. Are simple regression models obsolete, or are they still an important class of simple but effective tools? In this paper we analyze the performance of three simple regression models, applied to a variety of different data sets. We consider their performance on theoretical surfaces, noisy data, and we apply the models to complicated terrain data. We provide suggestions of how to suitably tune the hyper parameters of each model, and we show that simple regression models may indeed be very effective tools for prediction under the right circumstances.

Contents

1 Introduction

We will start by introducing some basic definitions and preliminaries needed later in this paper. We will also look into the essentials of the different regression models we are going to use. In the next section we will look further into some analysis of the methods before we start looking at the experiment results.

1.1 Definitions and Preliminaries

1.1.1 Design Matrix

The design matrix is a matrix that contains combinations of powers of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in a systematic order. The design matrix is constructed as

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_1^1 & y_1^1 & x_1^2 & x_1 y_1 & y_1^2 & x_1^3 & x_1^2 y_1 \dots \\ 1 & x_2^1 & y_2^1 & x_2^2 & x_2 y_2 & y_2^2 & x_2^3 & x_2^2 y_2 \dots \\ \vdots & \vdots & \ddots & & & & & \end{pmatrix},$$

where

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}.$$

The procedure to create the design matrix may be summarized in three steps and is as followed.

- Let k = 1.
- Begin with x^k and exhaust all configurations of $x^{k-j} y^j$ for $j = 0, 1, \dots, k$.
- Now let $k = 2, 3, \dots, n$ and repeat the previous step.

This makes out the rows of our design matrix. Full details on this algorithm may be found in the appendix, see Code.

1.1.2 Mean Squared Error & R^2-score

Let $\boldsymbol{Y}$ denote the set of observations. For each choice of polynomial degree we have an associated estimator $\hat{\boldsymbol{\beta}}$, and a set of predicted values $\hat{\boldsymbol{Y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$. A measure of how well our

model performs, called the *mean squared error* (MSE), is defined as

$$MSE(\hat{\boldsymbol{\beta}}) = \frac{C(\hat{\boldsymbol{\beta}})}{n} = \frac{1}{n}(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}})^T(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}).$$

The MSE increases as the deviation between the predicted and observed values increases. Another such measure of how well the model fits to the true values is the R^2-score, defined as

$$\mathrm{R2}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}.$$

The R^2 score is often referred to as the coefficient of determination. We can interpret it as a measure of how much of the variability is accounted for by the model.

1.1.3 Bias and Variance

Bias is a quantity that describes how much an estimator tends to over- or under- a parameter. If for an estimator $\hat{\theta}$ of θ we have that $E[\hat{\theta}] = \theta$, we say that $\hat{\theta}$ an *unbiased* estimator of θ

The variance is a numerical value used to indicate how widely individuals in a group vary. If each of the observations vary greatly from the mean value the variance gets big, while small variance correspond to small variation from mean. Distinguishing between the variance of a sample and population is important as they are computed differently. We denote the variance of a population by σ^2, while the sample variance is denoted s^2. They are defined as

$$\sigma^2 := \sum_i \left(\frac{X_i - \bar{X}}{n}\right), \quad s^2 := \sum_i \left(\frac{x_i - \bar{x}}{n-1}\right), \tag{1}$$

but since we are going to work with big loads of data we divide the latter by n. In Equation (1) $\bar{x}$ is the sample mean, x_i is the ith element of the sample and n is the number of elements in the sample.

1.2 Ordinary Least Squares

Ordinary least squares (OLS) is a *linear regression* model used to estimate the unknown coefficients of a polynomial. Specifically, given a set of data points, typically observations from an experiment, the OLS method aims to find the coefficients of a linear polynomial that minimizes the sum of squares between the observed points and the points estimated

by the OLS polynomial.

1.3 Ridge Regression

The *Ridge regression* model is very similar to the OLS, It uses a slightly altered cost function where it is added an additional term using the $\mathscr{L}^2$-norm, which yields a similar minimization problem as in the OLS-case. As we will discover, this method will have a regularizing effect on the predicted outputs, compared to the OLS.

1.4 Train-Test-Split

To split the data into training and testing sets we use train_test_split function from SciKit Learn. The function shuffles the data randomly and devides the sets into training and testing sets.

1.5 Lasso Regression

The Lasso regression is very similar to Ridge regression. It also features a penalty term in its cost function, however, the penalty term in Lasso regression uses the $\mathscr{L}^1$-norm.

1.6 k-fold Cross-Validation

A crucial part of machine learning is to evaluate the performance of the model. If we have an abundance of data, we could ideally split the data into a training and a test set without suffering from a lack of data in either category. In reality we may only have access to a data set of limited size, or it may be the case that it is very computationally expensive to generate more data. In either case we may use a cross validation technique such as *k-fold cross validation* to get a good estimate on the performance, without taking too much data away from training or testing.

The k-fold cross validation is a resampling teqhnique that splits the data set into k folds. In an iteration, $k-1$ folds are used to train the model, and the k-th fold is used to evaluate the performance of the model. This is repeated until each fold has been used for testing exactly once. The estimated performance of the model will then be the mean of the k tests. The most commonly used partitioning is a split of data into 5 to 10 equally sized parts. In Figure 1 $k = 10$ is used as an example.

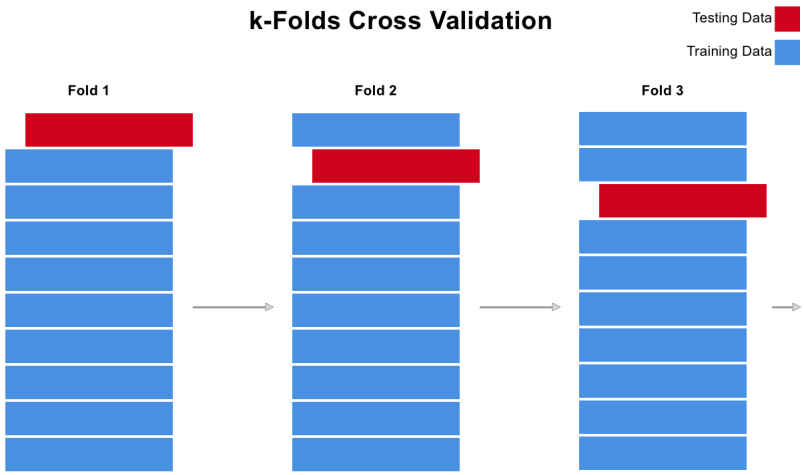


Figure 1: Illustration of how the k-fold cross-validation algorithm works. Red nodes are the sets of testing data, while the blue nodes are the training data.

1.7 Bias-Variance Tradeoff

The *bias-variance tradeoff* yields a relation between the mean-squared error (MSE) and the bias and variance, and gives a way to decompose the MSE,

$$\begin{aligned}\mathbb{E}\left[(y-\tilde{y})^2\right] &= (f-\mathbb{E}[\tilde{y}])^2 + \mathbb{E}\left[(\mathbb{E}[\tilde{y}]-\tilde{y})^2\right] + \mathbb{E}[\epsilon^2] \\ &= (\mathrm{BIAS}[\tilde{y}])^2 + \mathrm{VAR}[\tilde{y}] + \sigma^2.\end{aligned}$$

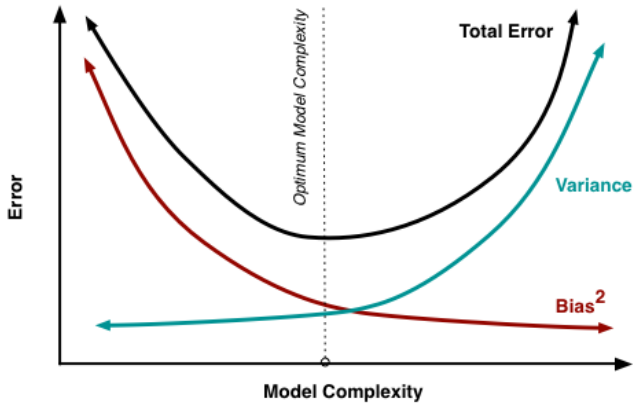


Figure 2: Bias-variance trade-off. The black graph is the variance and squared bias added together. The dotted line is the optimal choice of complexity for the model.

2 Theory

2.1 Ordinary Least Squares

Let there be $p+1$ unknown parameters denoted by $\beta_0, \beta_1, \cdots, \beta_p$. Then, the true OLS regression model yields

$$Y_j = \beta_0 + \sum_{i=1}^{p} \beta_i X_{i,j} + \varepsilon_j,$$

while the estimated model is

$$\hat{Y}_j = \hat{\beta}_0 + \sum_{i=1}^{p} \hat{\beta}_i X_{i,j}.$$

Written in matrix notation we have that $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where

$$\boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}, \quad \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}, \quad \boldsymbol{X} = \begin{pmatrix} 1 & X_{11} & X_{21} \dots & X_{p1} \\ 1 & X_{12} & X_{22} \dots & X_{p2} \\ \vdots & \vdots & \dots & \vdots \\ 1 & X_{1n} & X_{2n} \dots & X_{pn} \end{pmatrix}.$$

No matter the source of the variables in the design matrix, the model is linear in the parameters [HTF01, p. 44].

We define the cost function

$$C(\boldsymbol{X}, \boldsymbol{\beta}) := \boldsymbol{\epsilon^2} = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = \| \boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta} \|_2^2$$

which we want to minimize to estimate our parameters $\hat{\boldsymbol{\beta}}$. This yields

$$\frac{\partial}{\partial \hat{\boldsymbol{\beta}}} \left(C\left(\hat{\boldsymbol{\beta}}\right) \right) = -\boldsymbol{Y}^T \boldsymbol{X} - \boldsymbol{X}^T \boldsymbol{Y} + 2\boldsymbol{X}^T \boldsymbol{X} \hat{\boldsymbol{\beta}} = -2\boldsymbol{X}^T \boldsymbol{Y} + 2\boldsymbol{X}^T \boldsymbol{X} \hat{\boldsymbol{\beta}} \stackrel{!}{=} 0, \tag{2}$$

which results in

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^T \boldsymbol{X} \right)^{-1} \boldsymbol{X}^T \boldsymbol{Y}.$$

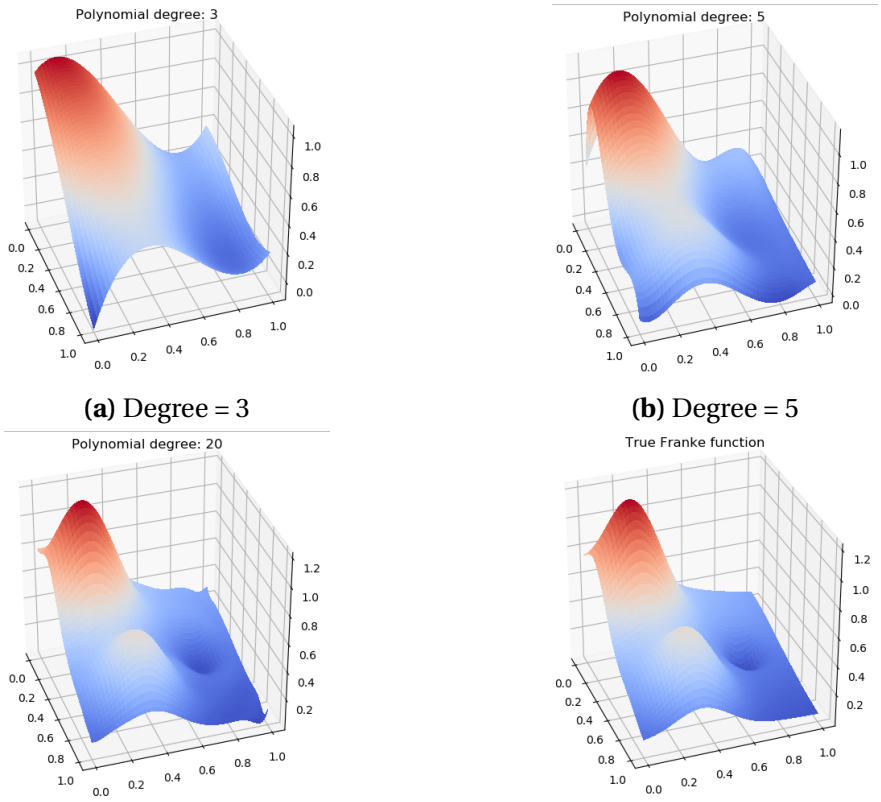
This is the optimal OLS estimator for $\boldsymbol{\beta}$ and is an unbiased, maximum likelihood estimator [DB12, p. 627, 642].

We can also calculate confidence intervals for $\hat{\boldsymbol{\beta}}$. To do this, we first introduce a quan-

tity, the *hat-matrix* $\boldsymbol{H} = \left(X^T X\right)^{-1}$. Then, $\sigma^2 \boldsymbol{H}$ is a matrix that contains the covariances between the β's. Specifically, the diagonal elements are the variances of each β. We can then estimate a 95% confidence interval for β_j as follows.

$$\hat{\beta}_j \pm 1.96 s \sqrt{\boldsymbol{H}_{j,j}}$$

Remark. *Let $\boldsymbol{\beta_1}, \boldsymbol{\beta_2}$ be coefficient vectors as defined above, associated with polynomials of degree m and n respectively, and let $m \leq n$. Then, the $MSE(\hat{\boldsymbol{\beta}}_{\boldsymbol{1}}) \geq MSE(\hat{\boldsymbol{\beta}}_{\boldsymbol{2}})$. Furthermore, $MSE(\hat{\boldsymbol{\beta}})$ can be made arbitrarily small by choosing the associated polynomial degree sufficiently large.*



(a) Degree = 3 **(b)** Degree = 5

(c) Degree = 20 **(d)** True Franke surface

Figure 3: OLS approximations of the Franke function with various polynomial degrees.

We will not include a full proof for this remark, but we note that the first statement follows trivially from the fact that the set of polynomials of higher degrees contains the sets of polynomials of lower degree. The second part of the statement follows from the

Stone Weierstrass theorem [Lin17, p. 127] which states that the polynomials are dense in the set of continuous functions, i.e. continuous functions can be expressed as limits of polynomials.

In other words, we can, at least in theory, approximate the Franke function as well as we like on $[0,1] \times [0,1]$ by a model with a sufficiently large polynomial degree. Figure 3 illustrates this principle.

2.2 Ridge Regression

The cost function in the Ridge model is defined as

$$C(\boldsymbol{X}, \boldsymbol{\beta}; \lambda) := \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_2^2, \quad \lambda > 0.$$

We want to find the parameters that minimizes the Ridge cost function and a calculation similar to Equation (2) yields

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^T \boldsymbol{X} + \lambda I\right)^{-1} \boldsymbol{X}^T \boldsymbol{Y}. \tag{3}$$

As we can see, the cost function associated with the Ridge regression is similar to the cost function of OLS regression. However, as we are to minimize this function, the extra term assigns an extra cost, which will penalize larger components of $\boldsymbol{\beta}$. Indeed, Ridge regression will tend to produce estimators for $\boldsymbol{\beta}$ with smaller numerical values than OLS estimators, and hence it can produce smoother, more regular curves. It can be shown that the variance of the Ridge estimator is less than the variance of the OLS estimator, i.e. $\mathrm{Var}\left[\hat{\boldsymbol{\beta}}_{Ridge}\right] \leq \mathrm{Var}\left[\hat{\boldsymbol{\beta}}_{OLS}\right]$ with equality if and only if the Ridge parameter, λ, is zero (hence OLS). However, this comes at a cost, namely that $\hat{\boldsymbol{\beta}}_{Ridge}$ is not a maximum likelihood nor an unbiased estimator for $\boldsymbol{\beta}$.

When we evaluate how good a model fits our data, it is crucial to consider how well the predictions extend to input data that has not been used to find the estimator, i.e. how well the model can predict unseen data. If a model performs very well on a set of data that has been used to calculate the model parameters, but performs significantly worse outside the set of data, we will say that the model is *overfitted*. As we have come to know that Ridge regression reduces variance in the $\hat{\boldsymbol{\beta}}$ parameters, one could expect that Ridge regression is less prone to overfitting. We will see in the following if this notion can be supported by the empirical results of the experiments in this paper.

2.3 Lasso Regression

The Lasso cost function, which is subject to minimization, is defined as

$$C(\boldsymbol{X}, \boldsymbol{\beta}; \lambda) := \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1.$$

As in the case of Ridge regression, we do not get an unbiased estimator, nor a MLE for $\boldsymbol{\beta}$, but we gain more regularity and less variance compared to OLS regression. Unlike Ridge regression, we can not generally derive an analytical expression for the optimal values of $\hat{\boldsymbol{\beta}}$ - they need to be estimated numerically.

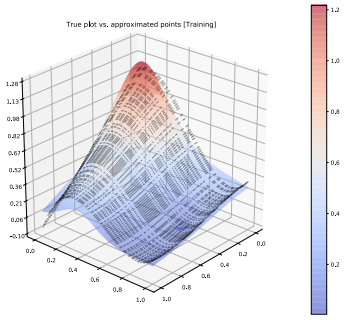
2.4 Train-Test-Split

The train-test-split algorithm is a model validation tool. The approach is to split the data into one training part and one testing part. This is so that we can train the model with one set of the data and then test the model on the rest of the data. For the full algorithm see Code.

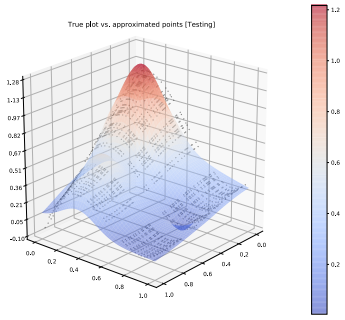
It is essential to split the data in a correct way to obtain useful results. There are two quite intuitive ways to split the data, namely to either split the input vectors and then set up a meshgrid for calculations and plotting, or to set up the meshgrid first and then split the data. The latter option is by far the best approach since the first option splits complete rows and columns and the resulting mesh is much less randomly split. This is nicely illustrated in Figure 4.

2.5 Bias-Variance Tradeoff

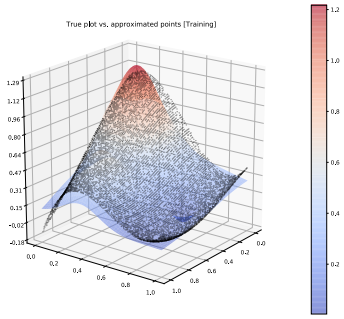
Ideally we would like to choose a model that has a low variance and a low bias. In a practical situation this is often not feasible, and we will need to find a good trade off between the two. As the model fits each data point more accurately, i.e. having a low bias, it is typical to see the variance increase. This is common as the complexity of the model increases. We are often left with two choices, namely either to choose a simpler model with an inherent larger bias, or to choose a larger degree of complexity, with more variance but less bias. Figure 2 illustrates the optimal complexity we want to obtain, and more complexity yields overfitting while less complexity yields underfitting. In the OLS model we end up having low bias and high variance, while in the Ridge regression model,



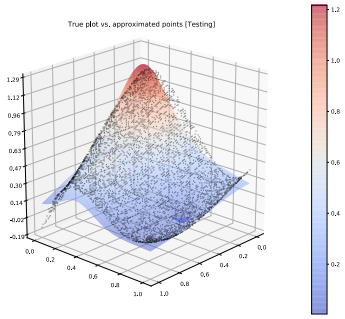
(a) Training data split before meshing.



(b) Testing data split before meshing.



(c) Training data meshed before splitting.



(d) Testing data meshed before splitting.

Figure 4: Training- and testing- data compared to each other in two cases. **a)** and **b)** illustrates the mesh when we split our data before meshing it, while **c)** and **d)** illustrates the data meshed before splitting.

the optimal $\boldsymbol{\beta}$ turns out to be the case where we have low bias and low variance. [Mar09, p. 35- 36]

3 Analysis

We will, in the following text, let U denote the unit square $[0,1] \times [0,1]$. In our examples we have considered either 20×20 or 100×100 points of data.

3.1 Ordinary Least Square on the Franke Function

Let us first consider a set of 100×100 points selected uniformly on U. For each point, we calculate the value of Franke's function, thus we get $10,000$ points of $\boldsymbol{x}$, $\boldsymbol{y}$ and $f(\boldsymbol{x}, \boldsymbol{y})$ in $\mathbb{R}^3$. Moreover, we want to use OLS to fit a polynomial to this set of points. We will let the polynomial complexity vary, and observe how the R^2-score, MSE and $\boldsymbol{\beta}$-values change.

3.1.1 MSE and R²-score

Degree	**MSE**	$\boldsymbol{R^2}$
1	0.0233	0.6042
2	0.0170	0.7393
3	0.0079	0.8954
4	0.0041	0.9479
5	0.0021	0.9740
50	$3.66e-5$	0.9995

Table 1: A view of the mean square error and R²-score for different polynomial degrees, using the OLS model. Created without train-test-splitting and no noise.

Let us look at the MSE and R^2-scores for OLS regression for a set of polynomial degrees. In Table 1 we see that the MSE tends quickly to values very close to 0. Even for a polynomial of degree 3, we have an MSE of less than 0.01 and an R2-score just below 0.9. In Figure 5 we illustrate the R^2-score and MSE as a function of model complexity, calculated on a 100×100 meshgrid on U.

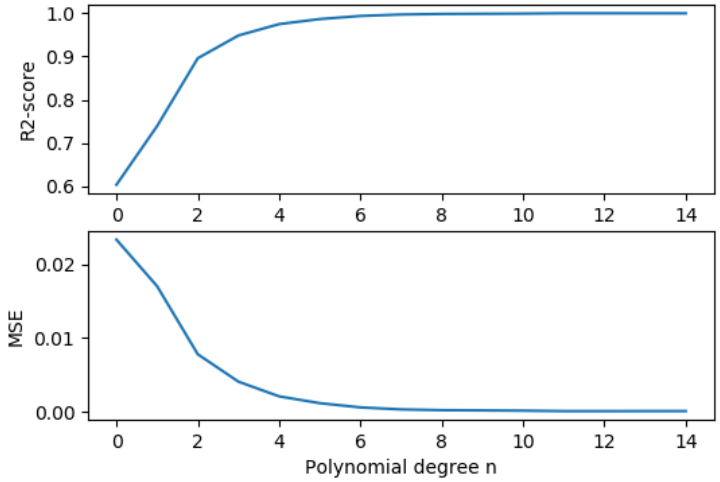


Figure 5: MSE & R²-score on training data as functions of complexity, using the OLS model.

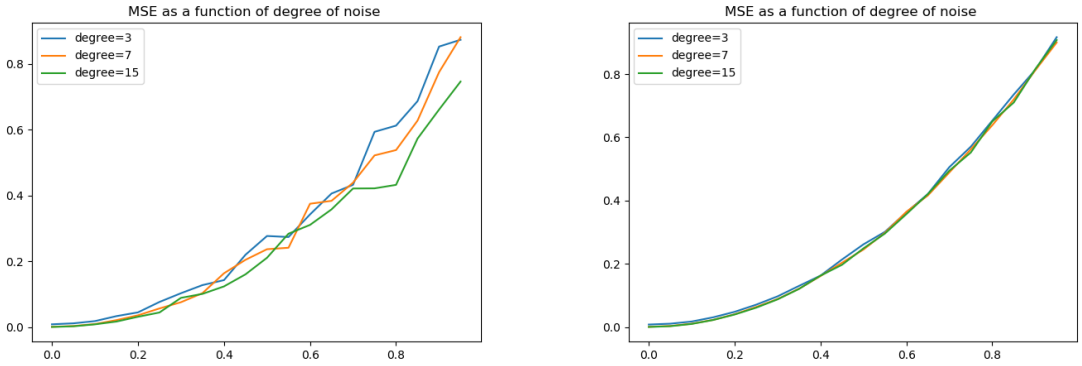
3.1.2 Confidence Intervals (95%)

We might also be interested in estimating our level of certainty of the component of $\hat{\boldsymbol{\beta}}$. We do this by calculating confidence intervals for each β_i. With a confidence degree of 95% (calculated from the normal distribution, not the t-distribution, as we have a large sample size) Tables 4 and 5 shows that that we have narrow confidence intervals for very simple models ($n=1$, $n=2$) - the widest spread is ± 0.161. As the degree of complexity increases, we see in Tables 6 to 8 that the confidence intervals quickly becomes much wider. For a model of polynomial degree 5, the spread in CI ranges from 0.089 to 11.355. It seems like a trend that there is more uncertainty associated with the β's associated with the higher degree parts of the polynomial (recall the construction of the design matrix. β_i where $i = 15, \ldots, 20$ correspond to the parts of the polynomial $x^j y^k$ such that $j+k=5$).

3.1.3 Adding Noise

We have so far considered a very simple scenario. We have only wanted to find an interpolation to a surface, and our observations of the true surface have not been subject to any noise. In a practical scenario we seldom have that luxury - it might for example be the case that the equipment used to measure an experiment is uncalibrated. To mimic such a scenario, we will add noise to the values we calculate from the Franke function. We will vary the degree of niose, and discuss how well the model performs for different degrees of complexity. We will try to answer the following two questions:

- How does a simple model compare to a complicated model when there is noise involved?
- Do complicated models handle noisy data well in a interpolation setting?



(a) 400 points **(b)** 10^4 points

Figure 6: MSE as a function of noise, using the OLS model, with polynomial degrees 3, 7 and 15.

We start by considering three OLS models, namely models of polynomial degrees 3, 7 and 15. We calculate values from the Franke function on the same set of points as before, with added noise, which yields

$$\boldsymbol{Y} = f_{Franke}(\boldsymbol{X}) + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \lambda \mathcal{N}(0,1), \quad \lambda \in (0,1). \tag{4}$$

We let the noise scaling factor λ vary from 0 to 1. The regression is preformed on a uniform grid of 20×20 points in U. We perform the same analysis, but this time we increase the number of data points to a 100×100 grid in U, see Figure 6a. From Figures 6a and 6b, two things stand out. When data is somewhat scarse, there is more variation between the MSE for different complexities. Moreover, it appears that the high complexity model tends to outperform the lower complexity model.

To answer the second question, we generate noisy data, and observe how the MSE varies as a function of model complexity. We let $\lambda = 0.3, 0.6, 0.9$, and generate noisy data as in Equation (4). This is illustrated in Figure 7. We see that as the complexity of the model increases, the MSE initially decreases rapidly. As we approach a polynomial complexity of about 4, we don't see systematic improvements. This is to be expected! As we are working with noisy data, we have introduced an irreducible error to our data. For more on this, see Bias-Variance Tradeoff for a discussion on the decomposition of the MSE.

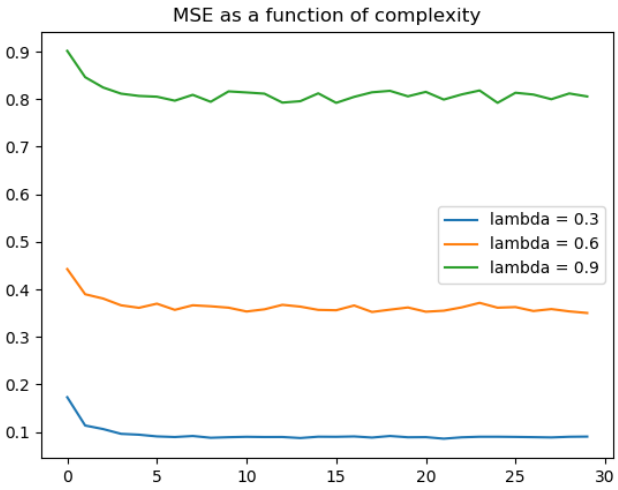


Figure 7: MSE as a function of complexity, using the OLS model, with noise scaling parameters $\lambda = 0.3$, 0.6, 0.9 on 10^4 points.

3.2 Resampling Techniques

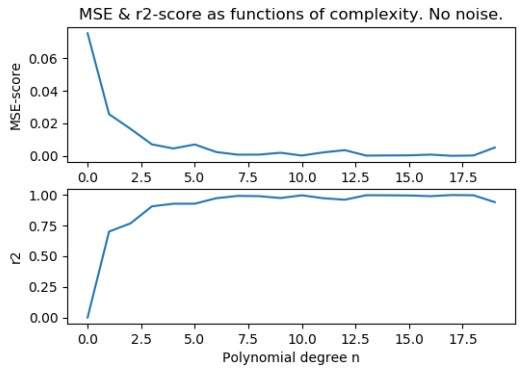
We have seen that in a situation where we do not have to worry about how our model performs outside the set of data it has been trained on, a higher degree of complexity has yielded better results. We now consider the case where we measure the performance on previously unseen data. We will be particularly interested in testing if the results from Ordinary Least Square on the Franke Function translates to the case with unseen data, and we will se whether or not a complex model usually outperform a simple one.

3.2.1 Train-Test-Split

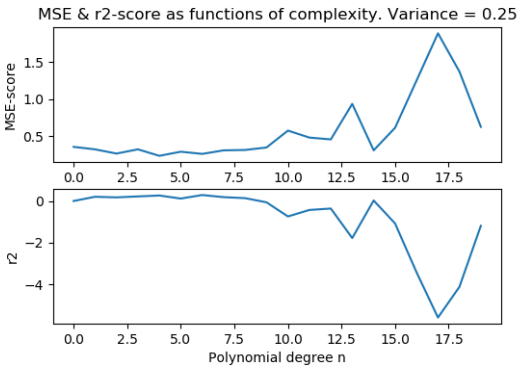
We begin with a grid of points on U, constructed and split as described in Section 2.4. We allocate 64% of the data to training, 36% to testing. We then train our OLS-model on the training set, but measure the performance on the test set. This will give us an understanding on how well the model is able to generalize.

Case 1: A small data set. Performance as a function of complexity

We begin with 20×20 points, split into training and testing sets. We generate values from the Franke function accordingly, with different degrees of noise added. Figure 8 shows that the MSE and R^2-scores do no longer monotonically improve as complexity increases. In Figure 8b we have added Gaussian noise with $\sigma^2 = 0.25$. When we do not have noisy data, it seems like we can sometimes handle a very complex model. However, we notice that we don't systematically get a better performance when we pass a complexity of



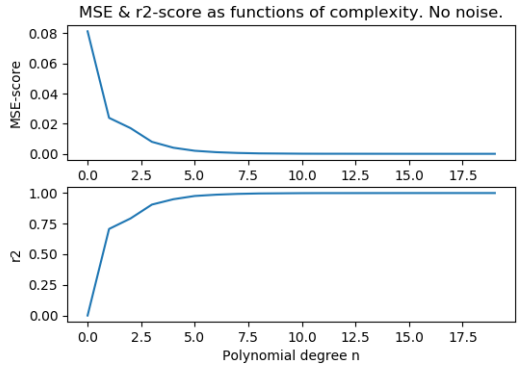
(a) 400 points, no noise added.



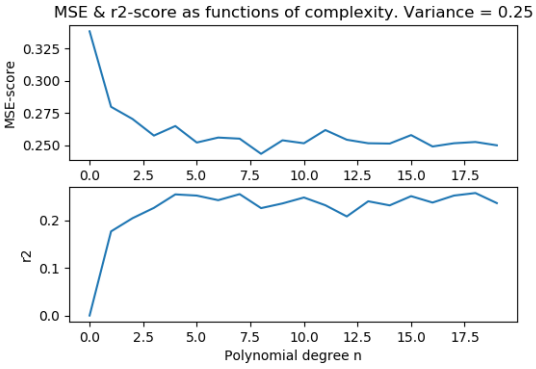
(b) 400 points, $\sigma^2 = 0.25$.

Figure 8: MSE & R²-scores on test set as a functions of complexity, using the OLS model. Dataset of 400 points.

polynomial degree 7. It is also clear that choosing a complex model when the data is scarce and subject to noise is very risky. With a polynomial degree of 16 we get an MSE of almost 2. This is not what we are after!



(a) 10^4 points, no noise added.



(b) 10^4 points, $\sigma^2 = 0.25$.

Figure 9: MSE & R²-scores on test set as functions of complexity, using the OLS model. Dataset of 10^4 points.

Case 2: A large dataset. Performance as a function of complexity

We repeat the same experiment as in case 1, but this time we use a 100×100 grid, giving us 25 times more data. The results are similar to the ones of case 1, but not as dramatic, see Figure 9. Without noise we achieve a stable performance of complexities from polynomial degree 5 to 20. When we add Gaussian noise, there is more variation in the performance as the complexity increases, all though much less so than in the first case. The optimal performance is in the case achieved when we choose a model of polynomial degree 4. We

cannot claim there is systematic benefits in increasing the complexity.

To summarize the above experiments we are left with three main points.

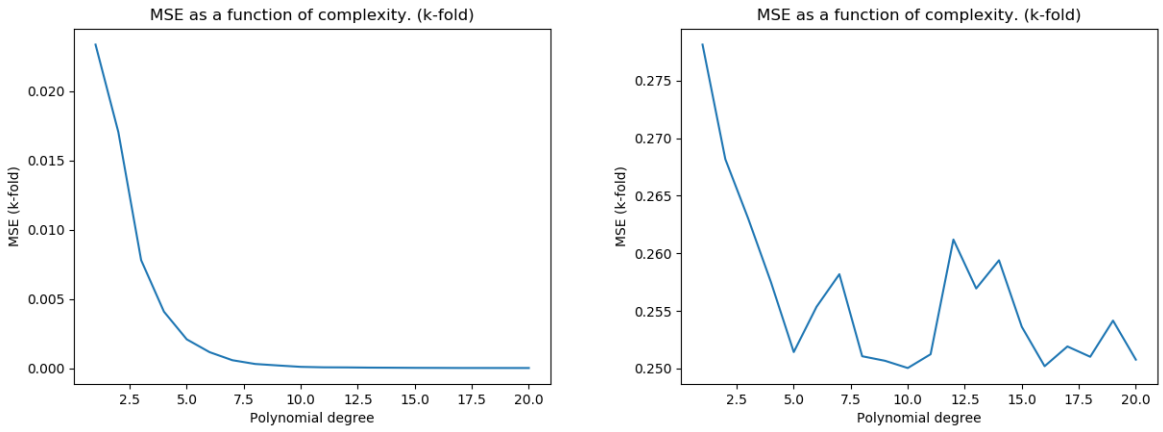
1. The overall performance of the model is sensitive to the amount of data. When we have more data, we have seen better and more stable performance as we vary the complexity compared to when we have little data.

2. We have seen optimal performance achieved for complexities around polynomial degrees 4 to 7. This has been the case regardless of if the data was noisy or not.

3. If we need to handle noisy data, we must be aware that choosing a large complexity easily can lead to overfitting. More data can help to remedy this, but high complexities should probably be avoided in such scenarios.

3.2.2 k-fold Cross-Validation

In the discussion above we have calculated the MSE and R^2-score as described in Mean Squared Error & R^2-score. We may also use k-fold cross-validation (see k-fold Cross-Validation) to get a more robust estimate of the MSE. We use 10^4 data points, and test for $\sigma^2 = 0$ and $\sigma^2 = 0.25$. The results are similar to what we found using *train_ test_ split*, but the estimates are slightly lower, see Figure 10.



(a) 10^4 points, no noise added.

(b) 10^4 points, $\sigma^2 = 0.25$.

Figure 10: MSE calculated by k-fold cross-validation, using OLS model. Dataset of 10^4 points.

3.3 Bias-Variance Tradeoff

We will provide a brief discussion of the bias-variance trade-off in this section. We begin to show how the MSE cost function may be decomposed:

$$\begin{aligned}
MSE(\boldsymbol{X},\hat{\boldsymbol{\beta}}) =& \mathbb{E}\left[\left(\boldsymbol{y}-\tilde{\boldsymbol{y}}\right)^2\right] \\
=& \mathbb{E}\left[\left(f+\boldsymbol{\epsilon}-\tilde{\boldsymbol{y}}\right)^2\right] \\
=& \mathbb{E}\left[\left(f+\boldsymbol{\epsilon}-\tilde{\boldsymbol{y}}+\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2\right] \\
=& \mathbb{E}\left[\left(f-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2\right]+\mathbb{E}\left[\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)^2\right]+2\mathbb{E}\left[\boldsymbol{\epsilon}\left(f-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)\right] \\
&+2\mathbb{E}\left[\boldsymbol{\epsilon}\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)\right]+2\mathbb{E}\left[\left(f-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)\right]+\mathbb{E}\left[\boldsymbol{\epsilon}^2\right] \\
=& \left(f-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2+\mathbb{E}\left[\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)^2\right]+2\left(f-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)\mathbb{E}\left[\boldsymbol{\epsilon}\right] \\
&+2\mathbb{E}\left[\boldsymbol{\epsilon}\right]\mathbb{E}\left[\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)\right]+2\left(f-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)\mathbb{E}\left[\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)\right]+\mathbb{E}\left[\boldsymbol{\epsilon}^2\right] \\
=& \left(f-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right)^2+\mathbb{E}\left[\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)^2\right]+\mathbb{E}\left[\boldsymbol{\epsilon}^2\right] \\
=& BIAS\left[\tilde{\boldsymbol{y}}\right]^2+VAR\left[\tilde{\boldsymbol{y}}\right]+\sigma^2.
\end{aligned}$$

The terms with coefficient 2 is zero since

$$\mathbb{E}\left[\boldsymbol{\epsilon}\right]=0 \quad \text{and} \quad \mathbb{E}\left[\left(\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\tilde{\boldsymbol{y}}\right)\right]=0.$$

This shows that the MSE can be decomposed into three parts, namely the variance of the predicted values, the square of the bias and σ^2. As we assume that there is an element of noise in our observations with variance σ^2, it follows that this term is an error we cannot control. We may therefore think of this term as an irreducible error in our model. In Ordinary Least Square on the Franke Function and Resampling Techniques we have provided a discussion on how the the bias variance trade-off can be viewed as a problem of optimizing MSE as a function of complexity in OLS models.

3.4 Ridge Regression on the Franke Function with Resampling

We will now explore how the Ridge regression model performs on the types of data we have been working with so far. Let us begin the discussion by considering the case with 100×100 points on U, without noise. We make the arbitrary choice $\lambda = 0.1$ for the ridge parameter, but we will later discuss how this hyperparameter can be tuned. We repeat the entries of Table 1, and report MSE and R^2-values for ridge polynomial complexities

1, 2, 3, 4, 5 and 50, see Table 2.

Degree	MSE	R^2
1	0.023	0.716
2	0.017	0.793
3	0.008	0.905
4	0.004	0.951
5	0.002	0.975
50	0.290	-2.520

Table 2: A view of the MSE & R²-score for different polynomial degrees, using Ridge regression, with Ridge parameter $\lambda = 0.1$. Calculated with k-fold cross-validation.

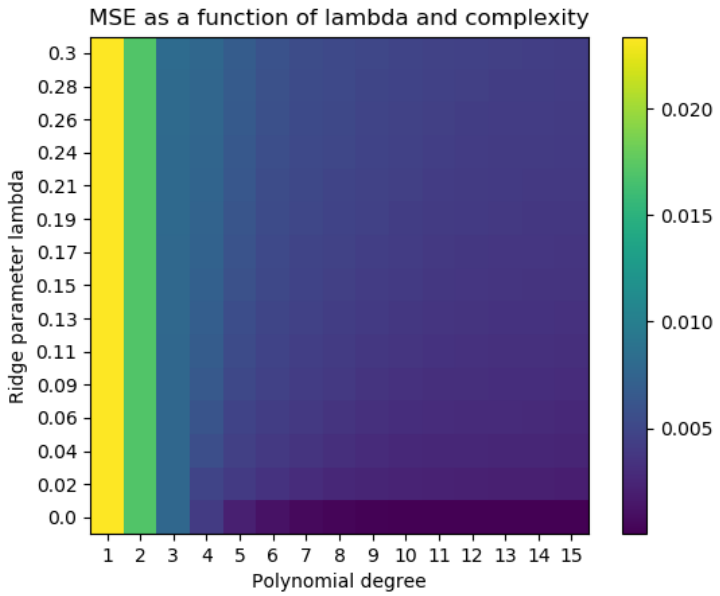
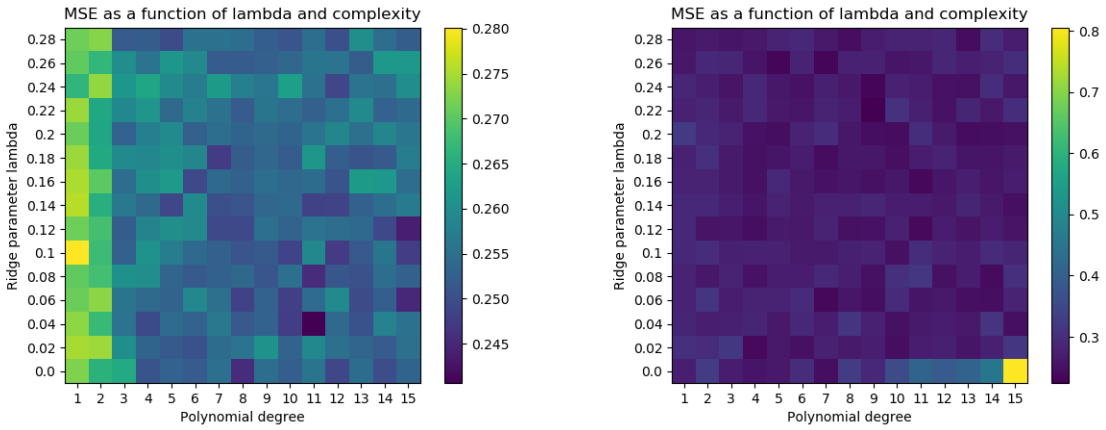


Figure 11: MSE as a function of complexity and Ridge parameter λ, using 10^4 points and $\sigma^2 = 0$.

There are only slight deviations between these results and those of OLS regression. We do not see an increase in performance on the MSE for degrees $1-5$, but we do get a slightly higher R^2 score for a first and second order polynomial model. For a polynomial degree of 50 we get significantly worse MSE and R^2-scores. We will not speculate if this is due to numerical issues, but we conclude that there are no grounds to say that the Ridge model performed better than the OLS model in terms of MSE in this context. This is probably to be expected. As we have stated in the introduction, the Franke function is an infinitely differentiable function, which means that it is smooth and cannot exhibit

explosive growth in the unit square. Hence, the need for regularization should probably be small.



(a) 10^4 points. $\sigma^2 = 0.25$.

(b) 400 points. $\sigma^2 = 0.25$.

Figure 12: MSE as a function of complexity and ridge parameter λ calculated using k-fold cross-validation and 10^4 points.

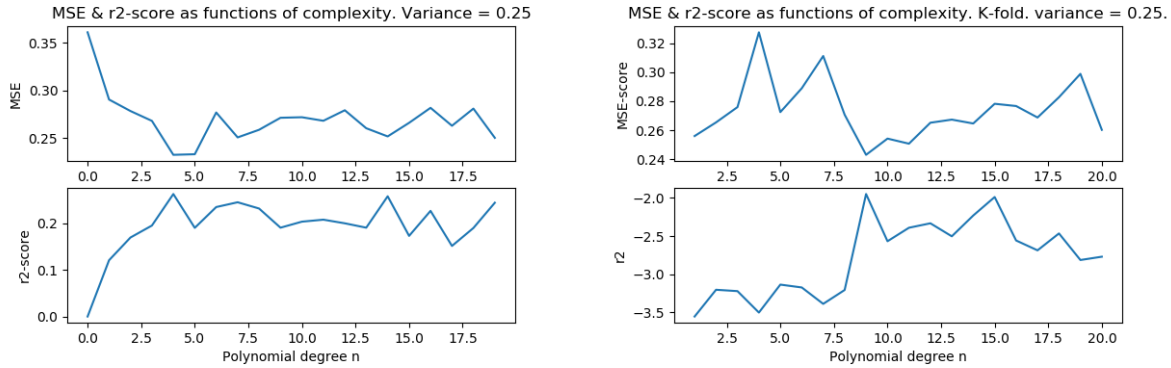
As we have introduced another parameter to be optimized, we will choose a different visualization of performance of the Ridge model, see Figures 11 and 12. We let complexity increase along the x-axis and Ridge parameter λ increase along the y-axis. We plot MSE as a function of complexity and λ, and use colors to represent the values of the MSE. To calculate the MSE, we use k-fold cross validation with 10 folds. The bottom row corresponds to Ridge regression with parameter $\lambda = 0$, i.e. OLS regression, row two to Ridge regression with $\lambda = 0.02$. This continues up to $\lambda = 0.3$.

Figure 11 shows the color-map where we use 100×100 points in U, without noise added. We see that the best performance is attained when $\lambda = 0$, for a complexity of 6 or greater. This is in consistent with what we observed in Figure 10a. When we have a lot of data without noise, the model handles high complexity well. This behaviour is in this case consistent for each value of lambda λ in the plot. We note that if we repeated the experiment with a 20×20 grid, the results would be consistent with this. We also observe that for each choice of λ the MSE seems relatively monotone as we increase the complexity beyond degree 5. This suggests that with a lot of data and no noise, Ridge regression is not prone to overfitting.

Let us see what happens when we add noise the observations. We use 10^4 points on U. We add Gaussian noise with $\sigma^2 = 0.25$, see Figure 12a. For 10^4 points, we see an optimal MSE $= 0.2406$ for polynomial degree 11, $\lambda = 0.04$. With 400 datapoints, we get

an optimal MSE $= 0.224$ with complexity 9 and $\lambda = 0.22$.

As each row in the plot corresponds to a fixed value of λ, we can study the bias-variance trade-off for each such λ by inspecting how the MSE changes along that row. When we have 10^4 data points we observe some variability in all the rows, but the differences are relatively small, approximately in the range $(0.24, 0.28)$. The big difference is in the case with only 400 data points. We see that the MSE in the first row tends to get much worse when the complexity approaches 15. This is not the case for the other rows. We therefore conclude that when have relatively little, noisy data, the Ridge model performs more stably and in general better than OLS regression.



(a) 400 points. **(b)** 10^4 points

Figure 13: MSE as a function of complexity with Ridge parameter $\lambda = 0.04$ and $\sigma^2 = 0$.

Let us examine Figure 13 closer. Indeed, the pairs of parameters that results in low MSE seem to be scattered somewhat randomly. This suggests that one should treat each regression case individually, and to optimize the hyperparameters λ and complexity to find the best model for each case. As we inspect the plot with $\sigma^2 = 0.25$ we see that $\lambda = 0.04$ produced good results for higher complexities. We therefore explore how the MSE and R^2-score vary as functions of model complexity with noise $\sigma^2 = 0.25$ added, on 10^4 points and 400 points in U . We use k-fold cross-validation to evaluate the MSE and R^2-score. The results are plotted in Figure 13. We do probably not have grounds to claim that the Ridge regression model performs better than OLS when we have 10^4 points of data. However, when we only have 400 datapoints, the effect of Ridge regularization becomes evident. Allthough the R^2 score remains bad (this is after all a very noisy set of data), the MSE remains much better than with OLS!

3.5 Lasso Regression on the Franke Function with Resampling

We will repeat the same analysis as in the previous section. We look first at 100×100 points in U, without noise. We let the Lasso parameter $\alpha = 0.001$, and perform the same calculations as in Table 1. The MSE and R^2-values for Lasso polynomial complexities 1, 2, 3, 4, 5 and 50 are as shown in Table 3.

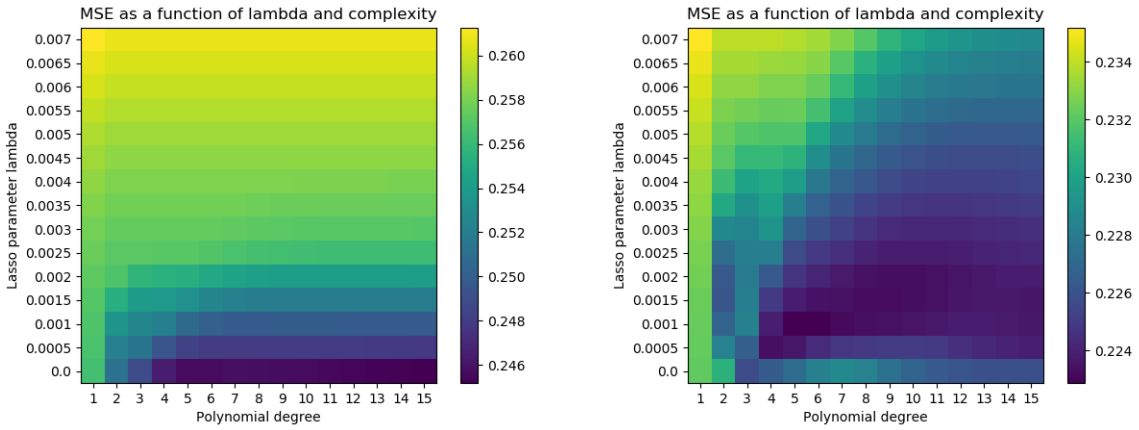
Degree	**MSE**	R^2
1	0.023	0.716
2	0.017	0.793
3	0.008	0.905
4	0.004	0.951
5	0.002	0.973
50	0.290	-2.516

Table 3: A view of the MSE & R²-score for different polynomial degrees, using Lasso regression, with Lasso parameter $\alpha = 0.001$. Calculated with k-fold cross-validation.

Again, these results are consistent with those found with Ridge and OLS regression. We have a slightly better R^2-score for low complexity models compared to OLS regression. When the model complexity increases, both MSE and R^2 are as previously seen. When we reach a complexity of 50, we get vastly worse results.

Let us now try to get an impression on how the choice of Lasso parameter α affects the MSE. We have seen already that when we have much data to work with, the model has been less prone to overfitting, but regularization had a positive effect when we had noisy, scarce data. We hence make two color-maps and let α and complexity vary. We look at the situation with 400 and 10^4 points in U, with Gaussian noise with variance $\sigma^2 = 0.25$ added to it. The results are shown in Figure 14. With 400 data points the MSE is consistently slightly higher than with 10^4 data points. We observe that there is more consistency in the distribution of low MSE values. If we look at the point where the polynomial degree is 5 and $\alpha = 0.001$ we can in fact find a contour with various complexities approximately in the range $[5, 12]$ and α in $[0.001, 0.002]$ containing similar MSE-values. From this we may infer that the Lasso regression has had a more regularizing effect than Ridge regression did on the same set of data. It also suggests that there is more stability and flexibility in the choice of complexity and parameter. A small perturbation in either α or complexity produces relatively similar MSE-values.

In the case where we have 10^4 data points, we again see that a model close to or equal to OLS is superior. The model is not prone to overfitting, and there does not seem to be



(a) 10^4 points. $\sigma^2 = 0.25$.

(b) 400 points. $\sigma^2 = 0.25$.

Figure 14: Ridge: MSE on test set as functions of complexity and ridge parameter λ by k-fold.10^4 points.

obvious benefits in MSE performance when using Lasso regression.

3.6 Introducing Real Data

In the following section we will study the models implemented above using real data of a part of the Norwegian terrain.

3.7 OLS, Ridge and Lasso Regression with Resampling

In this section we will use the methods and results we have derived previously, but on real terrain data. We will consider the data set SRTM_data_Norway_1.tif, shown in **??**. Terrain data is clearly subject to more noise than a theoretical, smooth function such as the Franke function. We therefore want to test the assumption that models that have regularization features will outperform OLS on such data, and we will explore how the density of our data points affects the model performance.

The terrain data covers a large geographical area with a large variety in topology. We see pointy mountains, steep valleys and rivers, spread out over several square kilometers, as illustrated in Figure 15. To hope to obtain a polynomial regression model that fits such a complicated set of data very well seems optimistic at best. A range of mountains may have quite similar topology to a different range of mountains, but has very little in common with the flat surface of a river. It would be interesting to look into models that first categorized different types of terrains, and then fitted models for each such terrain

type, but this is well beyond the scope of this paper. We will keep the above problem in

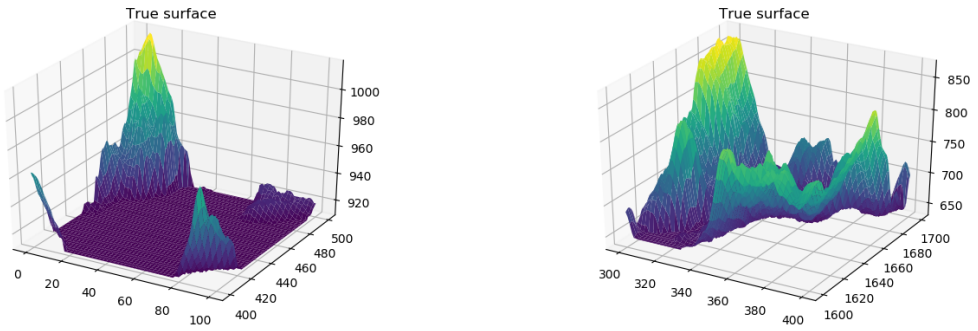
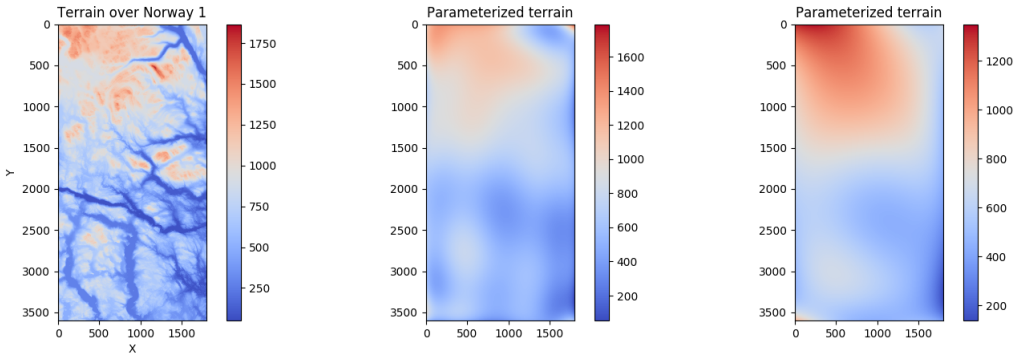


Figure 15: Selected terrains with great variety.

mind as we analyze the data sets and we will fit a model that considers the entire data set.

We use k-fold cross validation to estimate the performance in terms of R^2 and MSE. As the the set of data is relatively large, approximately 6.5 million data points, training a model with large complexity will be quite computationally expensive. We will therefore be limited to test the models that are computationally feasible. In Table 9 we have reported the MSE and R^2-score for several different choices of regularization parameter λ for Ridge, estimated by k-fold cross-validation with Ridge and OLS. We have found that Lasso if very computationally expensive, even for smaller degrees of complexity on large data sets. We are therefore not able to include Lasso in the comparison table, but we have noted that we obtained the best performance for the Lasso model in terms of estimated MSE with $\alpha = 0.002$ and a polynomial degree of 3 with MSE estimated to 44723.7616. For a polynomial complexity of 9 we estimate the R^2-scores and MSE as to 0.3274 and 37200.023 respectively with OLS. For a complexity of degree 10 the results were 0.3273 and 37196.943. We were not able to produce reliable results for Ridge with similar complexities. As we see from the results there is no clear tendency that Ridge performs better than OLS with these complexities. We do however get reasonably good R^2-scores, with $R^2 = 0.3274$ as our best observation.

In figure Figure 16 we compare the parameterized plots from an OLS and a Ridge regression model, of polynomial degree 9, and Ridge parameter $\lambda = 0.04$. As we can see, the effect of the smoothing is noticeable. The contours are more regular, but it does in fact appear to fit the terrain worse than what the OLS model does.



(a) True terrain. **(b)** Ordinary Least Square. **(c)** Ridge parameter $\lambda = 0.04$.

Figure 16: Data of Norwegian terrain computed with polynomial degree 9.

4 Summary and Conclusion

We have tested several different regression models on fundamentally different sets of data. The main differences are to what degree the data has been subject to noise, and if the data set is large, and indirectly how densely the data is distributed. The ideal case seems to be a large set of data without noise, drawn from a smooth surface. We have recorded the best performance overall for all choices of models with this type of data. In particular we have observed that it is particularly hard to overfit the models in this scenario. Having an abundance of high quality data is of course a luxury we seldom have.

When we subjected the data drawn from a smooth function to noise, and reduced the number of observations, the regularization models stand out as more stable than OLS. With 400 points and $\sigma^2 = 0.25$, Ridge and Lasso have better MSE scores across a wide set of complexities and regularization parameters. Especially Lasso with a regularization parameter of 0.001 and a polynomial complexity of 5 has proven to perform especially well. It is tempting to conclude that this is a condition where Lasso is preferable, but more testing and analysis is needed.

When we analyzed real data we found, contrary to our initial assumption, that we achieved the best fit with OLS, all though differences in R^2-scores and MSE between OLS and Ridge regression models were in general relatively small. We found that Lasso did not outperform the other two models when it was commutable. However, due to computational limits it was not comparable for most complexities. We have also observed that the R^2-scores MSE were monotonically for increasing polynomial degrees from 1 to 10. It is quite possible that we would see even better results for a higher complexity model, all though we know that increasing the model complexity at some points will run

the risk of overfitting our model.

In light of the results and analysis in this paper, we suggest that simple regression models can be effective tools to explain complicated surfaces in R^3. We have shown that even when the data is subject to vast noise, or is of a topologically complicated nature, the simple regression models still prove to be effective tools to describe such surfaces. Lasso regression proved to be especially useful when we deal with a combination of little data and much uncertainty, but it is not equally useful when the dataset is large.

5 Appendix

5.1 Code

The following code is a short version of some of the algorithms implemented to be able to calculate the different parts of the experiments.

5.1.1 General

```
def CreateDesignMatrix(x, y, pol_deg = 5):
    if len(x.shape) > 1:
        x, y = np.ravel(x), np.ravel(y)
    N, l = len(x), int((pol_deg+1)*(pol_deg+2)/2)
    X = np.ones((N, l))
    for i in range(1,pol_deg+1):
        q = int((i)*(i+1)/2)
        for k in range(i+1):
            X[:,q+k] = x**(i-k) * y**k
    return X

def FrankeFunction(x, y):
    term1 = 0.75*np.exp(-(0.25*(9*x-2)**2) - 0.25*((9*y-2)**2))
    term2 = 0.75*np.exp(-((9*x+1)**2)/49.0 - 0.1*(9*y+1))
    term3 = 0.5*np.exp(-(9*x-7)**2/4.0 - 0.25*((9*y-3)**2))
    term4 = -0.2*np.exp(-(9*x-4)**2 - (9*y-7)**2)
    return term1 + term2 + term3 + term4

def MSE(z, z_hat):
    if (len(z.shape) > 1):
        z = np.ravel(z)
    if (len(z_hat.shape)) > 1:
        z_hat= np.ravel(z_hat)
    SYY = (z-z_hat)@(z-z_hat)
    return SYY/len(z)

def r2_score(z, z_hat):
    if (len(z.shape) > 1):
            z = np.ravel(z)
    if (len(z_hat.shape)) > 1:
        z_hat = np.ravel(z_hat)
    n = len(z)
    en = (z-z_hat)@(z-z_hat)
```

```
    mean = np.sum(z)/n
    sst = (z - mean)@(z - mean)
    return 1-(en/sst)
```

5.1.2 Ordinary Least Squares

```
def ols(X, z):
    H = np.linalg.pinv(X.T@X)
    beta = H@(X.T@z)
    return beta
```

5.1.3 Ridge

```
def ridge(X, z, l):
    I = np.identity(X.shape[1])
    H = np.linalg.pinv(X.T@X + l*I)
    beta = H@(X.T@z)
    return beta
```

5.1.4 Lasso

```
def lasso(X, z, l):
    clf = linear_model.Lasso(alpha=l, fit_intercept=False)
    clf.fit(X, z)
    beta = clf.coef_
    return beta
```

5.1.5 Train-Test-Split

```
def train_test(x, y, model, noise_deg, pol_deg, num_points, lamb=0):
    x, y = np.meshgrid(x, y)
    x, y = np.ravel(x), np.ravel(y)
    z = FrankeFunction(x,y) + noise_deg*np.random.randn(len(x))
    X = CreateDesignMatrix(x, y, pol_deg)
    X_train,X_test,z_train,z_test=train_test_split(X,z,test_size=0.36)
    beta = model(X_train, z_train, lamb)
    z_pred = X_test@beta
    return z, z_pred
```

5.1.6 k-fold Cross-Validation

```
def k_fold_CV(k, model, x, y, num, pol_deg, lamb, noise_deg = 0):
    ind =  np.array(range(num**2))
    np.random.shuffle(ind)
    fold_size = int((num**2)/k)
```

```
    x_train, y_train = [np.zeros((k-1)*fold_size) for i in range(2)]
    x_test, y_test = [np.zeros(fold_size) for i in range(2)]
    mse_arr, r2_arr, bias_arr, s_arr = [np.zeros(k) for i in range(4)]
    beta_arr = np.empty((k, int((pol_deg+1)*(pol_deg+2)/2)))
    train_noise = noise_deg*np.random.randn(len(x_train))
    test_noise = noise_deg*np.random.randn(len(x_test))
    for i in range(k):
        ind1, ind2 = ind[:fold_size*i], ind[fold_size*(i+1):]
        train_ind = np.array(list(ind1)+list(ind2))
        for j in range(len(train_ind)):
            x_train[j], y_train[j] = x[train_ind[j]], y[train_ind[j]]
        X_train = CreateDesignMatrix(x_train, y_train, pol_deg)
        z_train = FrankeFunction(x_train, y_train) + train_noise

        test_ind = ind[fold_size*i: fold_size*(i+1)]
        for l in range(len(test_ind)):
            x_test[l], y_test[l] = x[test_ind[l]], y[test_ind[l]]
        X_test = CreateDesignMatrix(x_test, y_test, pol_deg)
        z_test = FrankeFunction(x_test, y_test) + test_noise
        beta = model(X_train, z_train, lamb)
        z_pred = X_test@beta
        mse_arr[i] = MSE(z_pred, z_test)
        r2_arr[i] = r2_score(z_pred, z_test)
        s_arr[i] = np.mean((z_pred - np.mean(z_pred))**2)
        bias_arr[i] = np.mean(z_test-np.mean(z_pred))
    r2,mse,bias,s = [sum(i)/k for i in [r2_arr,mse_arr,bias_arr,s_arr]]
    return (r2, mse, s, bias)
```

5.2 Tables

The following tables is a view of all 95% confidence intervals for all $\boldsymbol{\beta}$-vectors for polynomials of degree 1 to 5. The choice of Ridge parameter and Lasso parameter is $\lambda = 0.1$ and $\alpha = 0.005$, respectively. All tables are generated with k-fold cross validation where $k = 5$ and the degree of noise is set to 0.5.

Remark. *There are several β's in the $\boldsymbol{\beta}$-tables that have equal CI-ranges. This is due to a symmetry property in the construction of the design matrix. Let $q = i + j$ be the sum of the powers of the elements of $\boldsymbol{x}$ and $\boldsymbol{y}$. Further, let β_k^q denote all β's belonging to q-th power, where $k = 1, \ldots, n$. Then, the range of β_k^q equals the range of β_{n-k}^q.*

Beta	OLS	Ridge	Lasso
β_0	0.976 ± 0.012	0.959 ± 0.012	0.909 ± 0.033
β_1	-0.490 ± 0.016	-0.471 ± 0.016	-0.412 ± 0.048
β_2	-0.663 ± 0.016	-0.651 ± 0.016	-0.602 ± 0.022

Table 4: $k = 5$, noise= 0.5, $\lambda = 0.1$, $\alpha = 0.005$, degree= 1

Beta	OLS	Ridge	Lasso
β_0	1.144 ± 0.024	1.217 ± 0.026	0.789 ± 0.016
β_1	-1.013 ± 0.072	-1.149 ± 0.077	-0.377 ± 0.079
β_2	-0.685 ± 0.072	-0.841 ± 0.077	-0.049 ± 0.093
β_3	0.081 ± 0.064	0.206 ± 0.069	-0.029 ± 0.063
β_4	0.881 ± 0.057	0.859 ± 0.061	0.000 ± 0.000
β_5	-0.390 ± 0.064	-0.296 ± 0.069	-0.518 ± 0.103

Table 5: $k = 5$,noise= 0.5,$\lambda = 0.1$,$\alpha = 0.005$,degree= 2

Beta	OLS	Ridge	Lasso
β_0	0.982 ± 0.043	1.011 ± 0.041	0.774 ± 0.010
β_1	-0.627 ± 0.220	-0.540 ± 0.211	-0.289 ± 0.038
β_2	1.275 ± 0.220	1.012 ± 0.211	-0.046 ± 0.039
β_3	-1.378 ± 0.430	-1.446 ± 0.413	-0.103 ± 0.035
β_4	2.193 ± 0.343	1.748 ± 0.334	0.000 ± 0.000
β_5	-6.410 ± 0.430	-5.606 ± 0.413	-0.518 ± 0.043
β_6	1.001 ± 0.272	0.918 ± 0.262	0.000 ± 0.000
β_7	-0.070 ± 0.239	0.538 ± 0.235	0.000 ± 0.000
β_8	-1.338 ± 0.239	-1.452 ± 0.235	0.000 ± 0.000
β_9	4.494 ± 0.272	3.972 ± 0.262	0.000 ± 0.000

Table 6: $k = 5$,noise= 0.5,$\lambda = 0.1$,$\alpha = 0.005$,degree= 3

Beta	OLS	Ridge	Lasso
β_0	0.663 ± 0.063	0.925 ± 0.043	0.783 ± 0.019
β_1	4.040 ± 0.500	0.632 ± 0.272	-0.277 ± 0.055
β_2	2.294 ± 0.500	0.880 ± 0.272	-0.082 ± 0.137
β_3	-18.366 ± 1.633	-5.521 ± 0.790	-0.130 ± 0.043
β_4	-2.421 ± 1.277	0.766 ± 0.697	0.000 ± 0.000
β_5	-8.929 ± 1.633	-4.511 ± 0.790	-0.470 ± 0.125
β_6	24.025 ± 2.244	6.226 ± 1.092	0.000 ± 0.000
β_7	7.168 ± 1.718	3.505 ± 0.999	0.000 ± 0.000
β_8	1.899 ± 1.718	-2.187 ± 0.999	0.000 ± 0.000
β_9	7.461 ± 2.244	2.428 ± 1.092	0.000 ± 0.000
β_{10}	-10.528 ± 1.088	-2.297 ± 0.572	0.000 ± 0.000
β_{11}	-3.636 ± 0.949	-2.223 ± 0.646	0.000 ± 0.000
β_{12}	-1.091 ± 0.931	0.219 ± 0.668	0.000 ± 0.000
β_{13}	-1.512 ± 0.949	0.539 ± 0.646	0.000 ± 0.000
β_{14}	-1.147 ± 1.088	0.703 ± 0.572	0.000 ± 0.000

Table 7: $k = 5$,noise= 0.5,$\lambda = 0.1$,$\alpha = 0.005$,degree= 4

Beta	OLS	Ridge	Lasso
β_0	0.293 ± 0.089	0.879 ± 0.047	0.800 ± 0.022
β_1	9.224 ± 1.002	0.983 ± 0.300	-0.313 ± 0.100
β_2	4.600 ± 1.002	1.207 ± 0.300	-0.098 ± 0.076
β_3	-42.238 ± 4.840	-6.441 ± 0.829	-0.093 ± 0.079
β_4	-17.306 ± 3.780	1.420 ± 0.719	0.000 ± 0.000
β_5	-11.189 ± 4.840	-5.324 ± 0.829	-0.479 ± 0.089
β_6	67.712 ± 10.873	4.281 ± 1.159	0.000 ± 0.000
β_7	50.398 ± 8.058	1.087 ± 1.097	0.000 ± 0.000
β_8	21.569 ± 8.058	-1.548 ± 1.097	0.000 ± 0.000
β_9	-3.931 ± 10.873	1.670 ± 1.159	0.000 ± 0.000
β_{10}	-44.364 ± 11.355	4.286 ± 1.253	0.000 ± 0.000
β_{11}	-61.843 ± 8.669	0.707 ± 1.293	0.000 ± 0.000
β_{12}	-8.511 ± 8.067	0.652 ± 1.292	0.000 ± 0.000
β_{13}	-28.242 ± 8.669	-0.981 ± 1.293	0.000 ± 0.000
β_{14}	25.661 ± 11.355	3.761 ± 1.253	0.000 ± 0.000
β_{15}	9.413 ± 4.453	-4.122 ± 0.701	0.000 ± 0.000
β_{16}	23.575 ± 3.871	-1.412 ± 0.890	0.000 ± 0.000
β_{17}	10.352 ± 3.775	0.400 ± 1.088	0.000 ± 0.000
β_{18}	-5.387 ± 3.775	-0.605 ± 1.088	0.000 ± 0.000
β_{19}	15.636 ± 3.871	0.943 ± 0.890	0.000 ± 0.000
β_{20}	-15.326 ± 4.453	-1.860 ± 0.701	0.000 ± 0.000

Table 8: $k = 5$,noise= 0.5,$\lambda = 0.1$,$\alpha = 0.005$,degree= 5

Ridge				Ridge			
Degree	λ	**MSE**	R^2	**Degree**	λ	**MSE**	R^2
1	0.000	40369.04	-0.312463	5	0.000	42617.63	0.1388
	0.010	52277.37	-0.312		0.010	42617.59	0.1388
	0.020	52277.35	-0.312		0.020	42617.70	0.1388
	0.030	52277.32	-0.312		0.030	42617.56	0.1388
	0.040	52277.34	-0.312		0.040	42617.55	0.1388
	0.050	52277.33	-0.312		0.050	42617.69	0.1388
	0.060	52277.34	-0.312		0.060	42617.63	0.1388
	0.600	52277.31	-0.3124		0.600	42617.66	0.1388
2	0.000	47789.06	-0.078	6	0.000	40369.04	0.2198
	0.010	47789.05	-0.078		0.010	40369.11	0.2197
	0.020	47789.05	-0.078		0.020	40369.08	0.2197
	0.030	47789.11	-0.078		0.030	40369.07	0.2196
	0.040	47789.07	-0.078		0.040	40369.12	0.2196
	0.050	47789.11	-0.078		0.050	40369.19	0.2195
	0.060	47789.04	-0.078		0.060	40369.20	0.2195
	0.600	47789.15	-0.078		0.600	40382.20	0.2181
3	0.000	44626.86	0.0601	7	0.000	39058.62	0.2637
	0.010	44626.84	0.0601		0.020	39059.67	0.2632
	0.020	44626.87	0.0601		0.030	39061.01	0.2630
	0.030	44626.81	0.0601		0.040	39061.83	0.2628
	0.040	44626.86	0.0601		0.010	39059.11	0.2634
	0.050	44626.79	0.0601		0.050	38691.03	0.2756
	0.060	44626.87	0.0601		0.060	38691.05	0.2755
	0.600	44626.87	0.0601		0.600	39128.70	0.2582
4	0.000	43403.52	0.1088	8	0.000	38691.20	0.2756
	0.010	43403.44	0.1088		0.010	38691.03	0.2756
	0.020	43403.54	0.1088		0.020	38691.05	0.2755
	0.030	43403.51	0.1088		0.030	38691.23	0.2755
	0.040	43403.46	0.1088		0.040	38691.19	0.2755
	0.050	43403.49	0.1088		0.050	38691.21	0.2755
	0.060	43403.49	0.1088		0.060	38691.32	0.2755
	0.600	43403.45	0.1088		0.600	38684.54	0.2748

Table 9: MSE and R^2-score of Ridge regression using different polynomial degrees and Ridge parameters λ. When $\lambda = 0$ the values are estimated with OLS.

Bibliography

[DB12] Devore, J. and Berk, K. *Modern Mathematical Statistics with Applications*. 2012.

[HTF01] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. 2001.

[Lin17] Lindstrøm, T. L. *Spaces: An Introduction to Real Analysis*. 2017.

[Mar09] Marsland, S. *Machine Learning - An Algorithmic Perspective*. 2009.