Project 1 FYS-STK4155

Trond Skaret Johansen, Eirik Salberg Pedersen, Jonatan Winsvold

October 15, 2023

Abstract

In this project we implement the methods ordinary least squares, ridge and lasso for regression. The methods are used to fit polynomials to Franke's function and real world topographical data from the coast of Norway from USGS [1]. Both are datasets with a a two-dimensional input and a height as output.

We propose an algorithm for computing a granular feature matrix with features represented as $x_i^a y_i^b$ where we run through more combinations than what will be defined as *complete polynomials*.

We mainly experiment with $N = 101$ datapoints. We also experiment with stochastic noise to mimic the terrain data. The importance of standard scaling to counteract numerical errors is emphasized and Performance is assessed using Mean Squared Error (MSE) and the R^2 score.

We begin by implementing the OLS method, before motivating the introduction of ridge and lasso by studying the bias-variance tradeoff using bootstrap techniques. Ridge and lasso are introduced as alternatives less prone to overfitting. Lasso is implemented with the help of the Scikit-learn library [5]. We conclude our paper with a comparison of the three techniques using cross validation as a computationally efficient alternative to bootstrap.

We find that the optimal parameters for Ridge and Lasso are:

$$\lambda_{\text{Ridge}} = 0.01, \quad \lambda_{\text{Lasso}} = 0.001.$$

The best model for fitting polynomials to the TSGS terrain data is using OLS or Ridge with 4 features.

1 Introduction

In this project we study polynomial fitting. We will use two sources of data. The Franke function will be our test case. This two-dimensional function, with its intricate peaks and valleys as shown in Figure 1, serves as the landscape to test various regression techniques. The Franke function has two peaks of different heights, and a small dip. It is relatively well behaved, but has a non simple definition, namely:

$$f(x,y) = \frac{3}{4}e^{-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}} + \frac{3}{4}e^{-\frac{(9x+1)^2}{49}-\frac{(9y+1)}{10}} + \frac{1}{2}e^{-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}} - \frac{1}{5}e^{-(9x-4)^2-(9y-7)^2}. \tag{1.1}$$

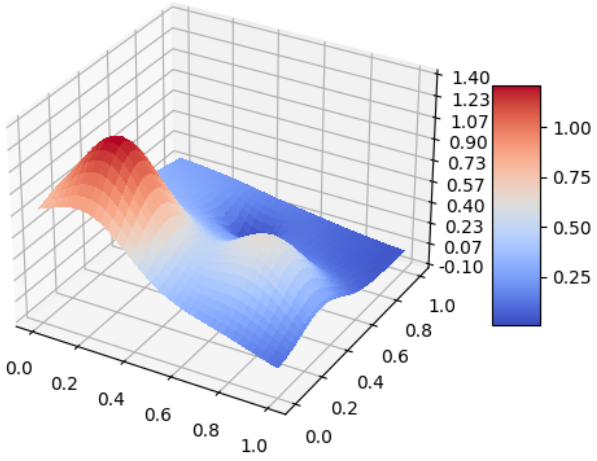


Figure 1: The Franke function.

In addition to studying how our models are able to fit to the Franke function we will also apply the methods a real dataset consisting of topographical data from the USGS [1]. Using this dataset we can assess how the different methods perform in a more realistic use case, where the data collected has an unknown underlying structure and natural noise infused into the data. To mimic this data with the Franke function, we also introduce noise.

We begin our paper with a discussion of the methods used in the project. We first discuss how the data is handled and scaled, and how we construct the feature matrix. As we advance, we examine three regression methods - Ordinary Least Squares (OLS), Ridge, and Lasso - clarifying their mechanisms, advantages, and potential pitfalls. After the results on OLS, we do an analytical detour through the bias-variance tradeoff to aid in our understanding model accuracy and the consequences of overfitting.

In addition to looking at the performance of the different methods we will also do a thorough error analysis of the methods. At first we will look at the bias-variance tradeoff using the bootstrap method to measure the distribution of our model error. Beyond this we will also investigate the more commonly used cross-validation method as a more robust measure of model performance.

Following our methodological discussion, we present results that highlight the capabilities of each regression technique. We will present the results of our investigation into the regression techniques on the Franke function alongside the results from the study of the terrain data. This will allow us to get a more in-depth view of how the techniques' performance differ between an idealized synthetic dataset and a more realistic noise-filled dataset.

In the same section discuss the findings and their implications. We will attempt to interpret the performance of the different techniques used, and find explanations of the patterns observed in the experiments. For instance an explanation of how the parameters obtained from Lasso and Ridge is given, where we go back to how their cost function is originally defined. We will also look into the potential causes of discrepancies in results between the study of the Franke function and of the terrain data. Furthermore, we will discuss the limitations of the methods used in our analysis as well as interesting and potentially insightful discoveries we encountered in the process.

The paper concludes with a synthesis of our findings and reflections on potential future work.

The programs deveoped during this project can be found at https://github.com/Trond01/Project1_FYS-STK4155 as detailed in Section 5.1 of the appendix.

2 Methods

2.1 Data Handling

2.1.1 The Feature Matrix

In our project we fit polynomials of the form:

$$p(x) = \beta_{0,0} + \beta_{1,0}x + \beta_{0,1}y + \cdots + \beta_{p,0}x^p + \beta_{p-1,1}x^{p-1}y, \tag{2.1}$$

to points sampled from the surface of the Franke function defined on the grid $[0,1]^2$ or from the terrain data as discussed in the introduction. The number of terms in this polynomials will be referred to as the *number of features*, and the number p is the *maximal degree*. One approach would have been to only consider polynomials where all terms of the maximal degree, $x^{p-j}y^j$ are included. We will refer to such polynomials as *complete polynomials*. When only using these, an increase of p from 3 to 4 corresponds to an increase in number of features by 5. (In general we get $p+1$ new features when increasing to degree p). This is why we have chosen an approach where it is possible to get *incomplete polynomials* like $1 + x + y + x^2$. We do not consider every possible incomplete polynomial. When we increase the feature number by one, the next polynomial is chosen by taking one step along the line in the plane of $\mathbb{N}^2$ shown in Figure 2.

2.1.2 Sampling, Noise, Scaling and Centering of Data

We scale our data in the design matrix using standard scaling.

Before implementing our regression model, the data was preprocessed to improve model performance and evaluation. Most notably we scaled the data using standard scaling, i.e. For each column j in the feature matrix, subtracting the columns mean from all elements i, j and dividing them by the columns standard deviation $x_{i,j,\text{scaled}} = \frac{x_{i,j}-\mu_j}{\sigma_j}$

For our data we have generated x and y values from the interval $[0, 1]$. This means that without scaling the rightmost terms in the feature matrix, i.e. the highest order polynomial terms, can become small numbers close to zero. The matrix X^TX can become close to singular, resulting in a large inverse which makes the coefficients $\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$ overly sensitive to small changes in $\boldsymbol{y}$.

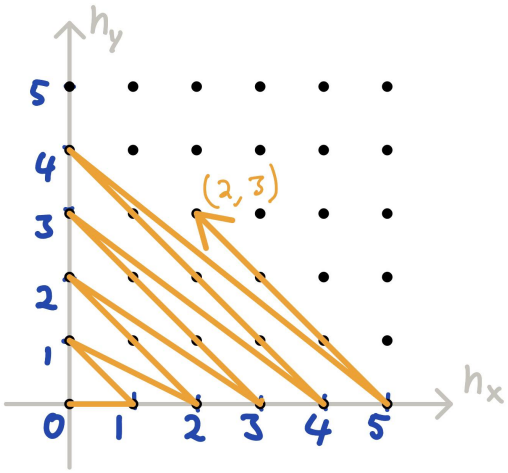


Figure 2: The path corresponding to the incomplete polynomial $p(x) = \beta_{0,0} + \beta_{1,0}x + \cdots + \beta_{5,0}x^5 + \beta_{4,1}x^4y + \beta_{3,2}x^3y^2 + \beta_{2,3}x^2y^3$.

Using standard scaling brings all our features to the same scale. Evenly scaled features makes the matrix $\boldsymbol{X}^T\boldsymbol{X}$ more well-conditioned which reduces our chances of numerical errors.

To evaluate our models ability to generalize we split the dataset into training and test sets. We used 20% for the test set and the remaining 80% for training. Which we assume will give us a sufficient amount of data to test on without sacrificing too much of the training data.

2.2 Ordinary Least Squares

For this project we created a dataset by sampling mainly using $N = 101$ points. We will also investigate the effect of decreasing and increasing this number. The sampled points take values (x, y) in $[0, 1]^2$. Additionally, to account for real-world noise, we introduced stochastic noise using a normal distribution with mean 0 and variance $\sigma^2 = 0.01$. We approximate the Franke function using ordinary least squares to compute the polynomial in two variables as discussed above. When performing OLS, we minimize

$$C_{\text{OLS}}(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}). \tag{2.2}$$

This minimisation problem has the analytical solution:

$$\boldsymbol{\beta}_{\text{OLS}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.3}$$

When this is found, the polynomial fit it given by $\boldsymbol{X}\boldsymbol{\beta}_{\text{OLS}}$.

To evaluate the model's performance, we used the Mean Squared Error (MSE) and the R^2 score given by:

$$1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}. \tag{2.4}$$

where $\hat{y}_i$ is the predicted value, y_i is the true value, and $\bar{y}$ is the mean value of $\boldsymbol{y}$. The sums are over the test dataset. The R^2 score offers a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.

2.2.1 Statistical Properties of OLS

Here we note some important results regarding the statistical properties of the approximators used in OLS. We now assume that the data does in fact come from a model of the form we attempt to fit to it with some additional noise, i.e. $y_i = \boldsymbol{X}_{i,*}\beta + \varepsilon_i$. If the noise is normally distributed with mean 0 and variance σ^2, then their expectation and variance is given by:

$$\mathbb{E}[y_i] = \mathbf{X}_{i,*}\boldsymbol{\beta} \tag{2.5}$$

$$\text{Var}[y_i] = \sigma^2. \tag{2.6}$$

From these we can derive the expected value and variance of the model parameters $\hat{\boldsymbol{\beta}}$, which show that the parameters we get using OLS is an unbiased estimator of the parameters of the underlying data given the assumptions made above.

$$\mathbb{E}[\hat{\boldsymbol{\beta}}] = \boldsymbol{\beta} \tag{2.7}$$

$$\text{Var}[\hat{\boldsymbol{\beta}}] = \sigma^2(X^T X)^{-1}. \tag{2.8}$$

The derivation of these equations can be found in appendix Section 5.3.

2.2.2 The bias-variance tradeoff

In any regression analysis, three critical properties can be used evaluate the model's performance. These are bias, variance, and total error. Understanding these properties aids in ascertaining whether the model is underfitting, overfitting, or is well-balanced.

To define these terms, we begin by supposing that any datapoint y, in our case the height of points on the Franke function or in terrain, is given by $y = f(x)+\boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon} \sim N(0, \sigma^2)$. We suppose further that we approximate the function f by fitting with a training dataset $\mathcal{D} = \{(x_i, y_i)|i = 0, 1, \ldots, n-1\}$. We show in the Section 5.4 of the appendix, that when drawing such a training set, the expected MSE error for another point not seen during training is given by:

$$\mathbb{E}\left[(y - \tilde{y})^2\right] = \text{Bias}[\tilde{y}] + \text{Var}[\tilde{y}] + \sigma^2, \tag{2.9}$$

This equation defines what is referred to as the *bias-variance tradeoff*. The bias term is defined as:

$$\text{Bias}[y] = \mathbb{E}\left[(f(x) - \mathbb{E}[\tilde{y}])^2\right] \approx \mathbb{E}\left[(y - \mathbb{E}[\tilde{y}])^2\right]. \tag{2.10}$$

We see that this is approximated by an average of squared differences between the expected model prediction, $\mathbb{E}[\tilde{y}]$ and the true value of the datapoint y. If the computed bias is high, it means that the average prediction is far from the true value. Models with a high bias make assumptions that might be too stringent or too simplistic. Consequently, such models may not capture crucial patterns in the data, leading to systematic errors regardless of the dataset size. Such behavior is often described as underfitting.

The variance term of equation 2.9 is :

$$\text{Var}[\tilde{y}] = \mathbb{E}\left[(\tilde{y} - \mathbb{E}[\tilde{y}])^2\right]. \tag{2.11}$$

It captures the dispersion of the model's predictions. Specifically, this term measures the extent of variability around their expected value. A model with a high variance reacts excessively to minor fluctuations in the training data. Thus, it might capture the random noise inherent in the training data and might not perform well on new, unseen data. Such behavior is often referred to as overfitting.

2.2.3 Bootstrap - estimating bias and variance

Before turning our attention to alternative models, we investigate how OLS performs for large feature numbers. We do this with the bootstrap technique to get more accurate representation of typical test and train errors. This is a re-sampling technique that involves drawing multiple samples from the

training dataset with replacement. This gives us a substitute for the population, meaning we have more samples to learn from. It is worth noting that bootstrapping assumes the data sample of 101 gives a good representation of the population.

Using bootstrap we will also approximate the bias and variance statistics defined above. For each of the M bootstraps we compute a prediction $\tilde{y}_i$ on each point y in the test set. Afterwards we can approximate the expected model prediction $\tilde{y}$ of the point y as:

$$\widehat{\mathbb{E}[\tilde{y}]} = \frac{1}{M}\sum_{i=0}^{M-1}\tilde{y}_i. \tag{2.12}$$

That is, we compute the average over the M bootstraps. With this, we can also approximate the variance by

$$\widehat{\mathrm{Var}[\tilde{y}_i]} = \frac{1}{M}\sum_{i=0}^{M-1}(y_i - \widehat{\mathbb{E}[\tilde{y}]})^2. \tag{2.13}$$

The bias is approximated by:

$$\widehat{\mathrm{Bias}[\tilde{y}]} = \frac{1}{M}\sum_{i=0}^{M-1}(y - \tilde{y}_i)^2. \tag{2.14}$$

Finally we take the mean over the results of all the test points to get an impression of how the model performs globally. We also compute the MSE error of the test data as the estimator for $\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right]$. The results and a discussion of these approximations can be found in Section 3.2.2.

2.3 Ridge Regression

When performing Ordinary Least Squares, there are two common problems you can run into. One of them is that the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is at risk of being singular matrix and thus cannot be inverted. The other is a more general problem of the model overfitting when using a high number of input variables. When this happens we often see the model finding parameters with a large magnitude, but cancelling each other out to properly fit the training data. This causes instability when interpolating between data points in the training set resulting in bad performance on the test set.

A common method to counteract overfitting of a model is to increase the size of the training data, however in many applications the amount of data we have to train on is limited. We therefore introduce a technique called Ridge regression to prevent the magnitude of the parameters from becoming high enough to make our model unstable.

Instead of optimizing the parameters of our model to minimize the standard MSE cost function we minimize the sum of the MSE function and the L^2 norm of our parameter vector $\boldsymbol{\beta}$. In matrix form our modified cost function becomes:

$$C_{\text{Ridge}}(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|_2. \tag{2.15}$$

We refer to the term $\lambda\|\boldsymbol{\beta}\|_2$ as a penalisation term, as it penalises large β-values. In our discussion we see this in action.

Similarly to OLS we minimize this function by finding the parameter vector which makes the derivative of our cost function zero. After differentiating our cost function and solving for our parameters, $\boldsymbol{\beta}$ we get [2]:

$$\boldsymbol{\beta}_{\text{Ridge}} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.16}$$

From this solution, we see that we may resolve the inversion problem by introducing the penalisation term.

2.4 Lasso Regression

When performing Lasso Regression we minimise the following cost function:

$$C_{\text{Lasso}}(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda\|\boldsymbol{\beta}\|_1. \tag{2.17}$$

Again, the term $\lambda\|\boldsymbol{\beta}\|_1$ is a penalisation term which prevents large β-values.

It has been demonstrated [3] that by taking derivatives and reordering the terms, one can show that the optimal parameter is the solution of:

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta}_{\text{Lasso}} + \lambda\text{sgn}(\boldsymbol{\beta}_{\text{Lasso}}) = 2\boldsymbol{X}^T\boldsymbol{y}. \tag{2.18}$$

Since this cannot be solved analytically, we have used the function `linear_model.Lasso` provided by scikit-learn. [5] With n denoting the number of samples, this function minimizes:

$$C(\boldsymbol{\beta}) = \frac{1}{2n}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}) + \alpha\|\boldsymbol{\beta}\|_1. \tag{2.19}$$

To make this equivalent to our problem, we pass the parameter $\alpha = \frac{\lambda}{2n}$ to the library function.

2.5 Cross-Validation

While bootstrapping is an effective method of measuring the distribution of our model error, it is often too computationally expensive to retrain the model numerous times like the bootstrap method requires. An alternative to bootstrapping is to use a technique called cross-validation. Cross-validation partitions our data into k-segments and trains the model k-times, each run using a different segment as the test data, with all the others being used as training data. This gives us a good idea of how stable our model is when trained on different parts of the dataset, while also limiting the number of training runs. For details of this method, see for example [2].

We will use cross-validation to compare the sensitivity of OLS, Ridge and Lasso to the choice of training data similar to how we do for OLS using bootstrap.

3 Results and analysis

In this section, we present the findings from our study, showing the performance of the Ordinary Least Squares (OLS), Ridge Regression, and Lasso Regression models on the Franke function dataset and on the terrain dataset. The results are organized based on the methods applied, each being accompanied by appropriate performance metrics, visual representations, and a brief interpretation.

3.1 OLS

For our OLS regression model, the Mean Squared Error (MSE) and the R^2 score were computed for an increasing number of features to evaluate its performance on both the training and test datasets. Throughout this paper, dots are used to mark feature numbers corresponding to complete polynomials as defined in Section 2.1.1. The results from the OLS experiment are shown in Figure 3. We see that the MSE error, both for test and training decreases towards 0. Similarly the R2 error increases towards 1. This is what we expect, as increasing the feature number should allow the model to capture more of the tendencies in the data. We will later see that the train error will at some point deviate from the test error much more than what is seen in this plot.

We perfom the experiment also with noise and with terrain data. While the noisy results are quite similar and can be found in the appendix, for the terrain data we see some concerning results. These are shown in Figure 4, where we see the phenomena of overfitting, which will be the object of further discussion. Before further investigating this, we take a look at the β-values found when performing OLS.

We plot the computed coefficients, β of our polynomials as a function of the number of features, in Figure 5a. We see that nothing keeps the β-values from spiking in both directions. This instability might be a sign that we are not finding the best fit. We shall later see how these parameters change in the case of Ridge and Lasso. Before discussing these methods, we investigate the over-fitting trends in more detail.

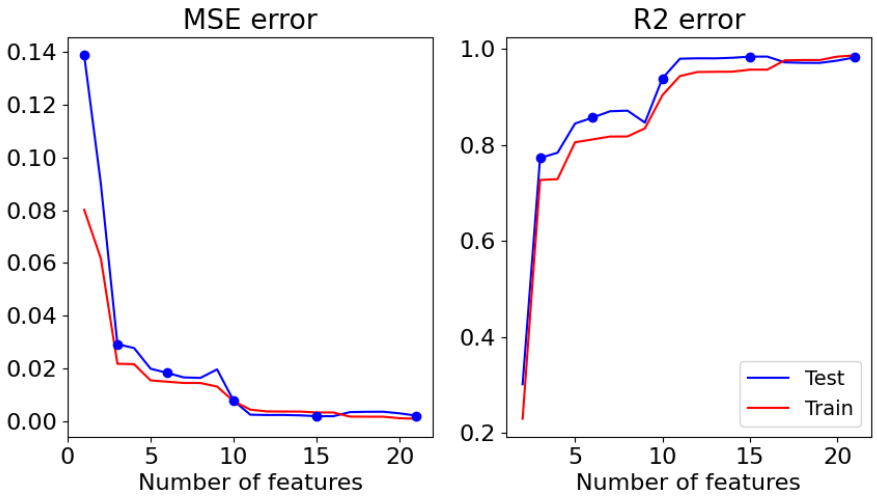


Figure 3: MSE and R2 errors when sampling 101 points from the Franke function and fitting with OLS for different complexities.

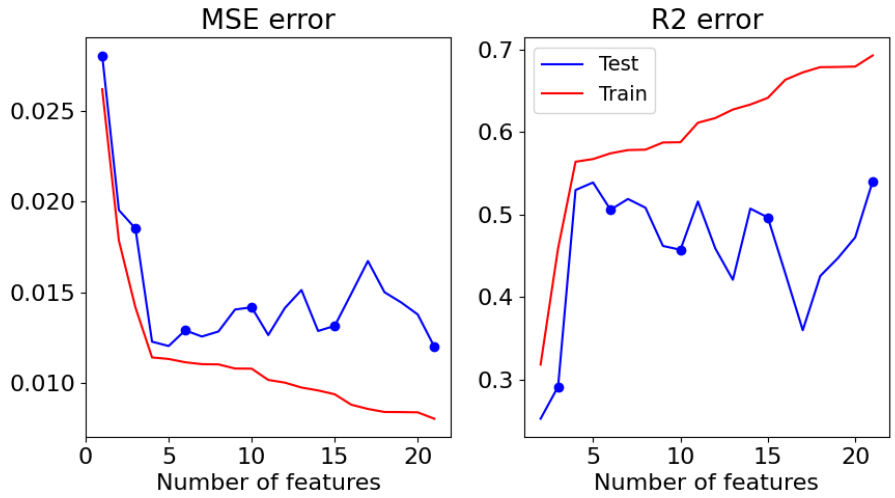
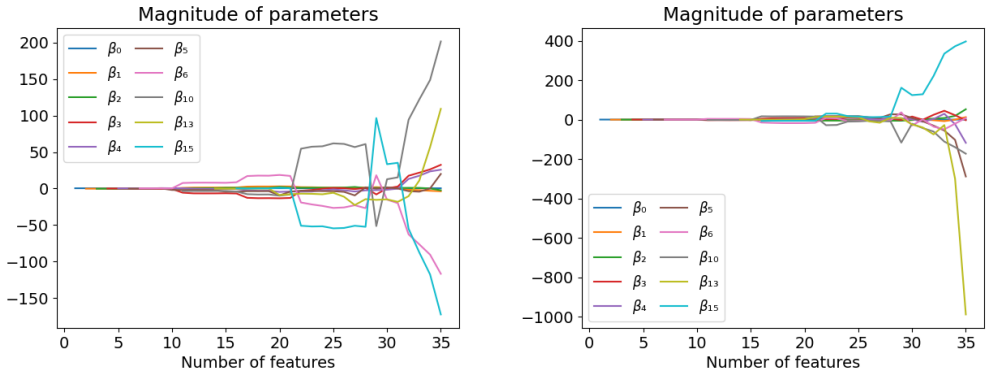


Figure 4: Errors from OLS regression performed on terrain data with $N = 101$ points.



(a) Franke function (b) Terrain data

Figure 5: Parameter values of OLS with varying number of features. Done with 101 points in the dataset.

3.2 The Bias-Variance Tradeoff

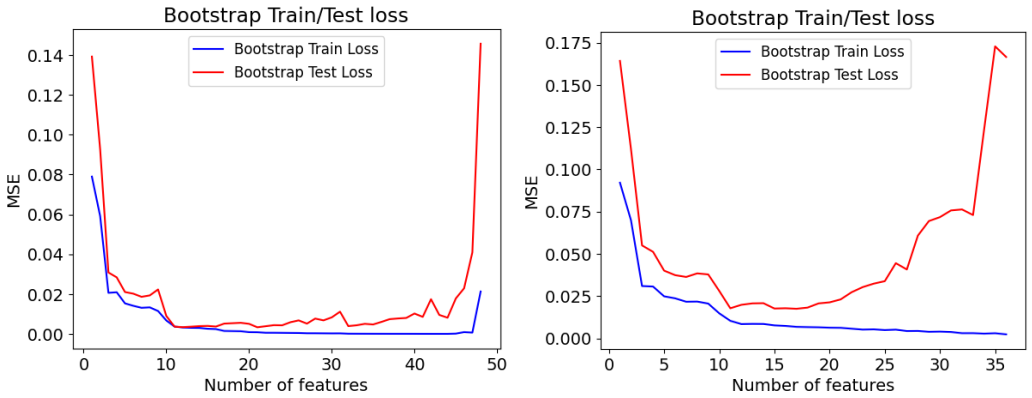
In this section we show results from our bootstrap method. Our first results are estimations of the average test and train errors as a function of feature-number. We then discuss their relation to the Bias-Variance Tradeoff making estimates of these statistics as well.

3.2.1 Average train and test error

Before performing analysis of the Bias-Variance Tradeoff, we used bootstrap techniques to investigate how the test and train loss is affected by the number of features. The results both with and without noise are shown in Figure 6.

We note that the introduction of noise gives a much clearer tendency to overfit to the training set sampled in each bootstrap. This is as we expect, since the Franke function is continuous [1] it can be approximated very well by a polynomial, while the noisy function is discontinuous and as a result can't be perfectly approximated by a polynomial. With this in mind, the fact that train and test loss follow each other in the noiseless case is not surprising, as we simply get closer to the actual polynomial.

[1] Since the function f is continuous on the compact space (closed subset of $\mathbb{R}^2$), there is a sequence of polynomials $\{p_n\}_{n\in\mathbb{N}}$ which converges uniformly to f by the generalised Stone-Weierstrass Theorem. See for example [4] for a proof of a special case.



(a) Without noise

(b) With $\sigma^2 = 0.01$ noise

Figure 6: Average train and test loss computed for different model complexity in bootstrap experiment done with 101 points sampled from the Franke function with and without the addition of noise. The estimates are computed using 50 bootstraps.

With the introduction of noise however, neighbouring points might suddenly be further apart, and the increasing of feature numbers causes the model to simply learn the noise and not the underlying surface as in the former case.

As mentioned previously, a goal of introducing noise is to mimic real world data better when studying possible models for it. In Figure 7 we see clearly that the results for real world data are indeed much more similar to our noisy Franke function in Figure 6. One key difference however is that the test loss starts to increase much earlier. One possible explanation is that the terrain data is even further from a polynomial than the noisy Franke function.

3.2.2 Numerical Results of the Bias-Variance Tradeoff

The test error in the plots above give an estimate for $\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2]$ by taking the average MSE test error in each bootstrap. This is the expected test error when drawing a set of training points and using them to make a model. By Equation 2.9, proved in the appendix, this can also be decomposed into bias and variance. Using the estimators introduced in Section 2.2.3 we estimate the bias and variance. Once again we perform our analysis with and without noise, to illustrate the importance of mimicking the actual data as much as possible. In Figure 8 we see these results.

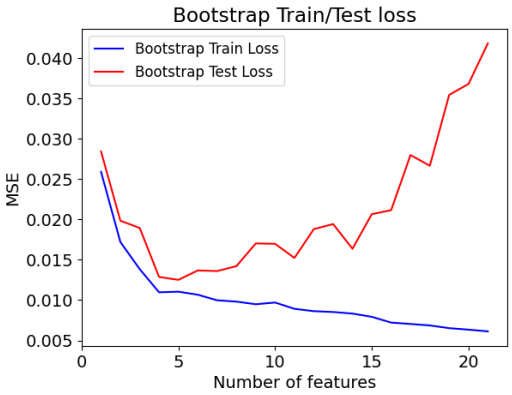
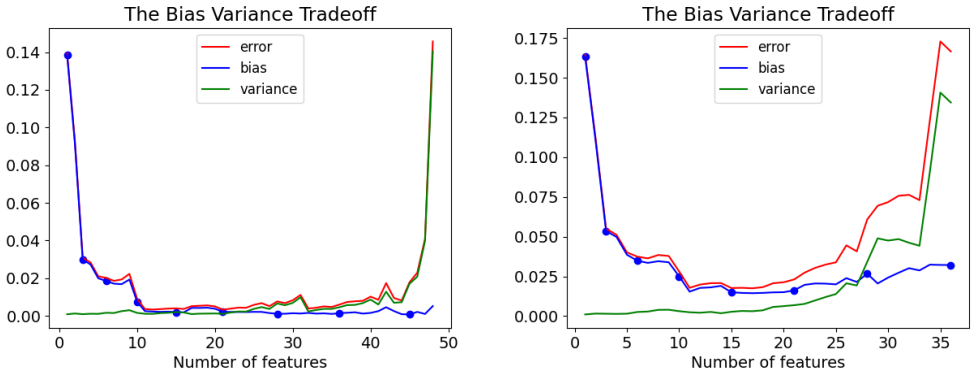


Figure 7: Average train and test loss computed for different model complexity in bootstrap experiment done with 101 points sampled from terrain data without noise.



(a) Without noise

(b) With $\sigma^2 = 0.01$ noise

Figure 8: Estimates of error, bias and variance computed from Bootstrap experiment done with 101 points sampled from the Franke function with and without noise.

In Figure 8a we see that around 50 features the error explodes. We believe a possible reason for the spike is a numerical error caused by the extremely low values of the higher order terms. We also see evidence of this in Figure 6a, where both the training error and test error spikes around 50 features, even though the OLS training error should always become lower when more features are added.

For the case of noise, we see that the error reaches its minimum around 11 features. It stays approximatelly constant, before increasing as the model learns an increasing amount of noise. We recall that the variance term represents the sensitivity of the model to the training data $\mathcal{D}$. A high variance is indicative of overfitting, as changing the training data drastically alters the model obtained. The results in this plot match well with our expectations; the model learns the noise.

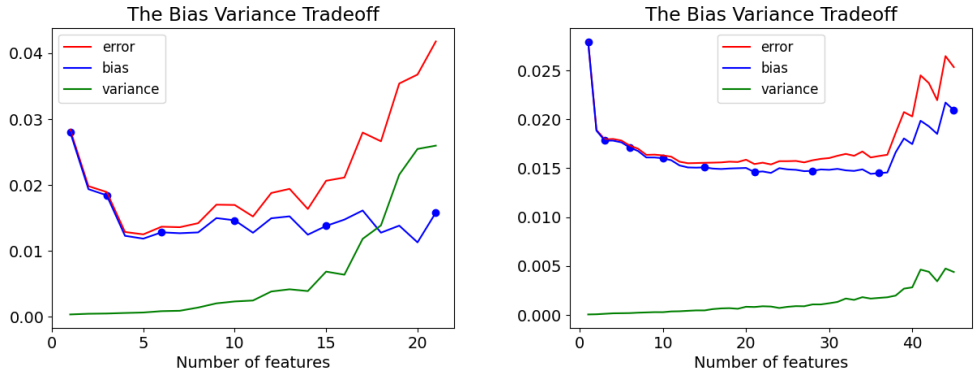
For the bias term, we note that high bias when using few features indicates that the expected fit does not match the test value. In the case of 1 feature, we attempt to fit a plane to a function that is not flat; hence we cannot expect to hit every test point. Here the variance is very low as the expected plane is roughly the average height of the points, which does not change much with the choice of training data.

Finally we note that the bias and variance seem to roughly sum to the error, which aligns with what we saw in Equation 2.9.

After having seen the results on the Franke function, we turn to the real data. In Figure 9a we see the results from the terrain data using 101 points as we did for the Franke function. As expected the behaviour is most similar to the noisy Franke function. Also here, the optimal complexity seems to be the point just before completing a degree 2 polynomial, i.e. a polynomial of the form:

$$p(x) = \beta_{00} + \beta_{10}x + \beta_{01}y + \beta_{20}x^2 + \beta_{11}xy. \tag{3.1}$$

Figure 9b is included in our discussion to emphasize the dependence of the size of the dataset. Here we have used 500 points instead of merely 101, and see that we can add many more features before seeing the model worsens. However, the payoff is limited. There is barely decrease when increasing from 20 to 35 features. Here one should also consider the added computational cost



(a) 101 points (b) 500 points

Figure 9: Estimates of error, bias and variance computed from Bootstrap experiment done on terrain data for two numbers of data points.

from this increase. Since the improvement in performance is near negligible, choosing the simpler model might be the better choice.

3.3 Ridge Regression

In our Ridge regression model, the MSE values were computed for an increasing number of features as well as for different values of λ.

We see in Figure 10 that when training the model with 101 datapoints using the noiseless Franke function we get a good model with both low training error and test error for sufficiently low values of λ. However when repeating the same experiment with fewer datapoints we see another pattern emerge.

From Figure 11, we see that for very low values of λ, in the range between 10^{-8} and 10^{-5}, the model overfits to the training data when trained with a high number of features. This is likely due to the λ being too low so it does not penalize the parameters enough, creating a similar overfitting phenomenon to the one seen with OLS. However for values of λ between 10^{-3} and 10^{0} we see that you can use a far higher number of features while still retaining low MSE loss on both the training and test set. We also see that as λ increases the error increases and then plateaus. This is the region where the model prioritizes lowering the parameter values close to zero rather than trying to predict the data. We get this since the penalisation term, $\lambda\|\boldsymbol{\beta}\|_2$, in the loss

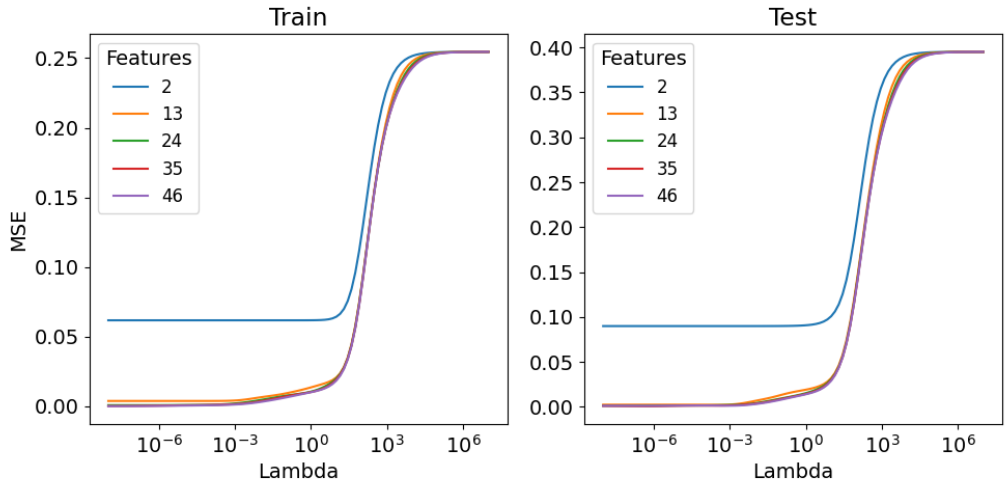


Figure 10: Train and test loss for Ridge regression with varying λ and number of features. Done with 101 points in the dataset.

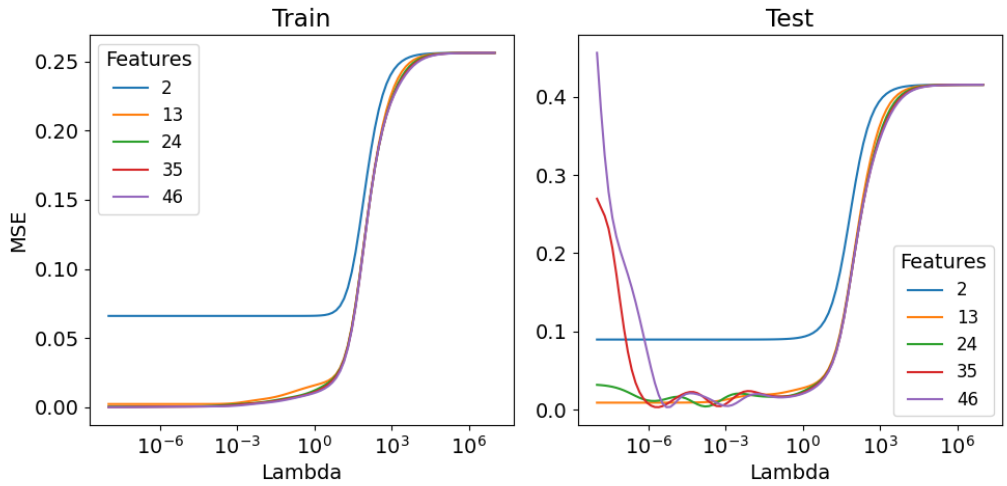


Figure 11: Train and test loss for Ridge regression with varying λ and number of features. Done with 51 points in the dataset.

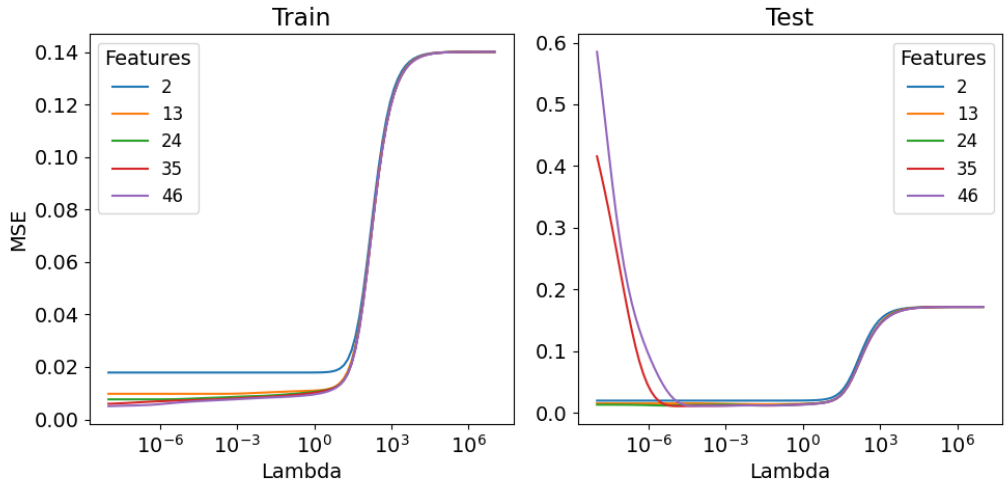


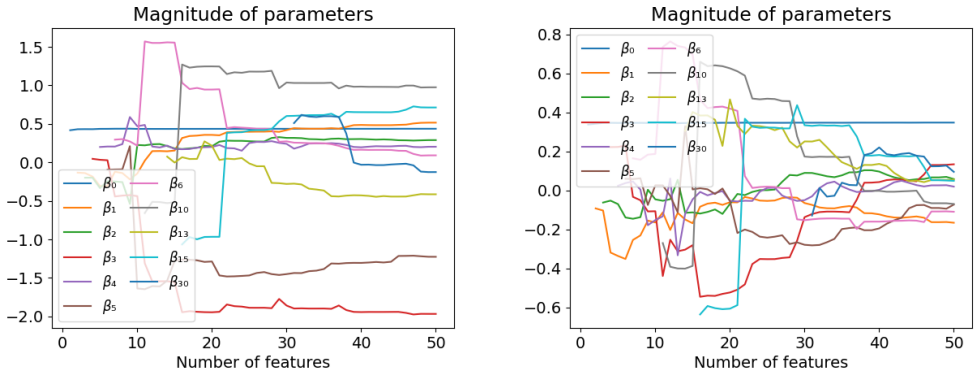
Figure 12: Train and test loss for Ridge regression with varying λ and number of features. Dataset is sampled from USGS terrain data. Done with 101 points in the dataset.

function we minimize becomes bigger than the mean squared error between the training data and the zero function. At the plateau, the β values are set to 0. For other seeds, we have seen that the test and train error sometimes have the same plateau.

When looking at the USGS dataset we see an even stronger overfitting phenomenon where the model completely fails to predict the test set. As seen in Figure 12, for low values of λ and high number of features the test errors are two orders of magnitude higher than the respective test errors for the dataset without noise.

In the middle region of Figure 12 where the Ridge regression prevents overfitting we also see how this changes the parameter values compared to the OLS model. The plots indicate that a lambda value of $\lambda_{\text{Ridge}} = 0.01$ works well, and we will use this for the remaining experiments.

Compared with Figure 5a and Figure 5b we see that the parameters stay close to 0 irrespective of the number of features the model uses. We also notice that the parameters are far more stable when increasing the number of features the model is trained with, likely encouraged by the penalisation term. This also likely indicates that the model has found some good approximation of



(a) Franke function (b) Terrain data

Figure 13: Parameter values of Ridge regression, with $\lambda = 0.01$, with varying number of features. Dataset is sampled from Franke and from USGS terrain data, both with 101 points in the dataset.

the training set with a fairly low number of parameters and that when new parameters are added they are merely used to tweak the model slightly to fit the training data. In contrast when looking at the parameter values in Figure 5a we see that the model finds radically different values for the same parameters when varying the number of features in the model. One possible explanation of why the Ridge regression model performs better is that the normal OLS model struggles to find the underlying pattern in the training data, thus creating radically different solutions depending on the feature number. On the other hand we might think that the Ridge regression model finds a good approximation for the underlying pattern early and improves on this approximation with the higher order terms while keeping the lower order terms relatively stable.

3.4 Lasso Regression

For the Lasso regression, we performed a similar analysis as we did for Ridge Regression. We computed the MSE with an increasing number of features along and with a range of λ values and study the development of the parameters β.

In Figure 14 we see a similar pattern to the one we got for the Ridge regression model, however with noticaebly less overfitting at low λ values. We find that

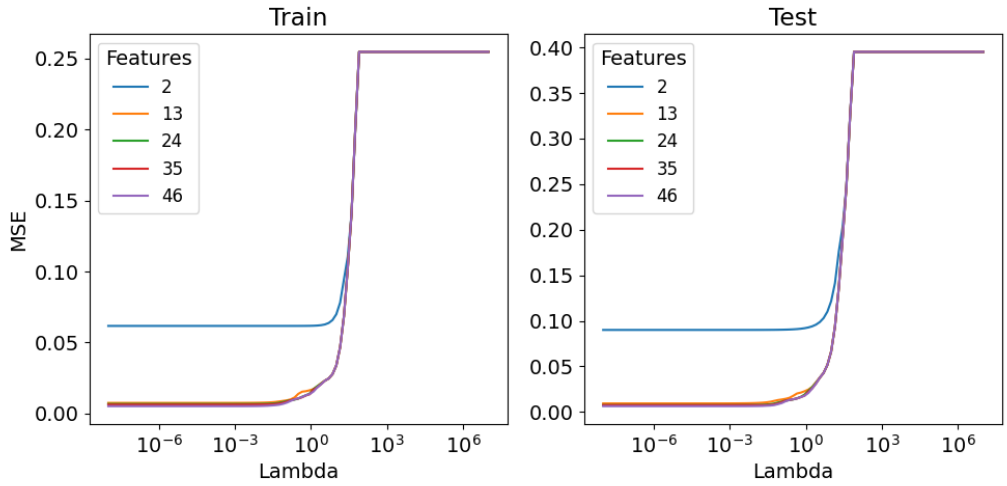


Figure 14: Train and test loss for Lasso regression with varying λ and number of features. Done with 101 points in the dataset.

the value $\lambda_{\text{Lasso}} = 0.001$ to be sufficient, and use this in the succeeding experiments.

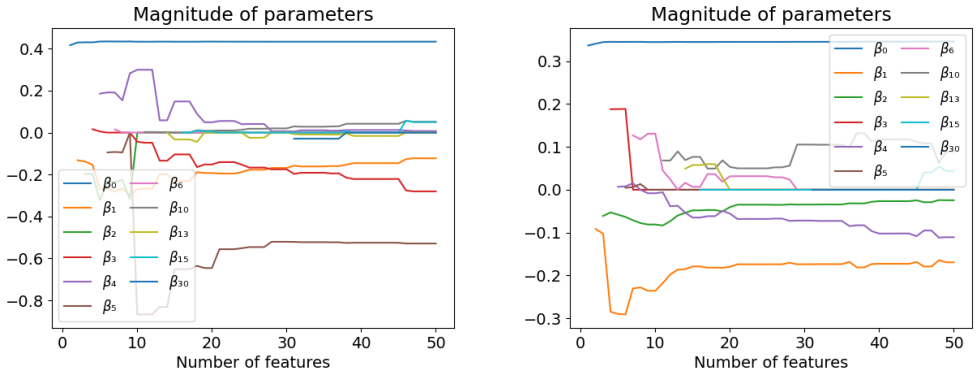
We see signs of the stronger regularization effect in the parameter values we get from the Lasso regression when using $\lambda_{\text{Lasso}} = 0.001$ in the range from Figure 14 where the MSE for both train and test is low. We see in Figure 15a and Figure 15b that far more of the parameters are zero or very close to zero, even though the λ used for the Lasso regression is only a tenth of that used for the Ridge regression in Figure 13a and 13b.

The explanation can be found by recalling a general inequality for the norms used in the two penalisation terms. For N features, we have $\beta \in \mathbb{R}^N$ and it follows that:

$$\|\beta\|_2 \leq \|\beta\|_1. \tag{3.2}$$

A proof of this inequality can be found in the appendix Section 5.2. The inequality shows that we get stronger penalisation from the L1 norm, which explains how the parameters from Lasso are closer to 0.

Another property of Lasso regression we observe is many of the parameters are almost constant as we increase the number of features, whereas in Ridge regression there were more small fluctuations among the parameters. This



(a) Franke function (b) Terrain data

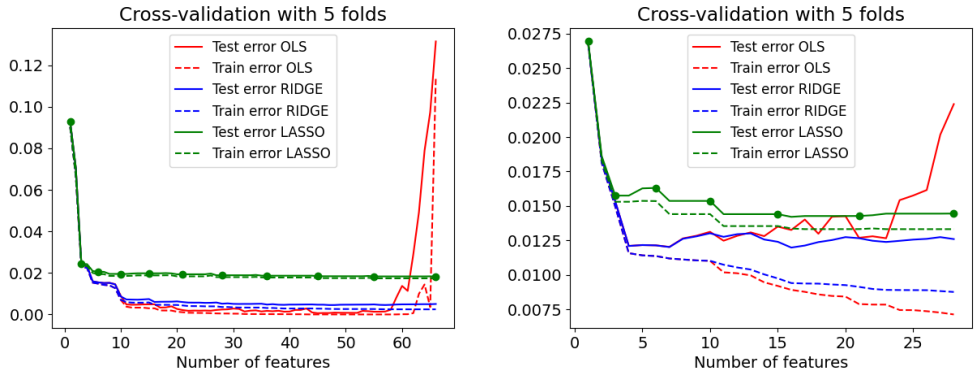
Figure 15: Parameter values of Lasso regression, with $\lambda = 0.001$, with varying number of features. Dataset is sampled from Franke and from USGS terrain data, both with 101 points in the dataset.

may indicate that using Lasso regression imposes some level of rigidity to the model making it use very few parameters to approximate the training data. A possible explanation for the rigid nature of the parameters we get from Lasso regression is that the L1 norm used in the penalisation term, $\lambda\|\boldsymbol{\beta}\|_1$, has a discontinuous derivative while the penalisation term in the Ridge regression is continuous. Since the Lasso regression models are found using coordinate descent, which uses the derivative to update the parameters, it is reasonable that the model could make more discontinuous jumps in parameter values when trained with an increasing number of features.

In the next section we will compare the relative performance of Lasso regression and the other methods, where we can assess whether this rigidity has enough of an impact on the model performance to the point where the Lasso model may be underfitting.

3.5 Cross-validation

Having implemented and experimented with Ridge and Lasso regression, we are ready to see how they compare with OLS when it comes to overfitting and numerical issues. To do this we apply cross-validation techniques as described in Section 2.5. Figure 16 shows the results of cross-validation experiments when using $k = 5$ folds. We recall from the discussion of bootstrap, that as



(a) Franke (b) Terrain data

Figure 16: Cross validation of regression on the noiseless Franke function and on terrain data with 101 points evenly distributed in $k = 5$ folds.

the feature number increases, OLS eventually show a blowup of the test error, partially due to overfitting but likelly also the result of issues with the matrix inversion. We see that Ridge and Lasso do not have this blowup around 60 features. This was part of the reason of introducing the penalisation term, which aids in inversion and combats overfitting. However these penalisation terms also cause the expected solution to be slightly different from the one obtained in OLS since we are not optimizing for the lowest possible mean squared error. This has effects on how well the model is able to approximate the both the training and test data, which may give a worse overall model. This is evident in Figure 16a, where OLS performs best until it explodes.

In Figure 16b we see how the test error in Lasso seems to remain constant after both OLS and Ridge start overfitting. Ridge shows some tendency to overfit, but it is nowhere near that of OLS for higher feature numbers. This plot demonstrates well the increased resilience to overfitting for Ridge and Lasso, and after about 15 features, Ridge is the best model. Still, the best model over all feature numbers seems to be a draw between Ridge and OLS with quite few features.

One disappointing observation in both figures is that Lasso is consistently outperformed by OLS and Ridge at lov degrees. This might indicate that it is not only resistant to overfitting, but also to resistant to finding the actual global minimum when computing the fit. One possible explanation is that

the Lasso algorithm is unable to find the global minimum.

4 Conclusion

In this paper we have explored three classic methods for linear regression. We began by introducing our way of constructing the feature matrix using incomplete polynomials. This allowed more careful study of how the model behaves as we incrementally increase its complexity. With this tool we explored bootstrapping techniques and cross-validation and used to reason.

Our findings indicate that with our number of points and our data, the best parameters for Ridge and Lasso are:

$$\lambda_{\text{Ridge}} = 0.01, \quad \lambda_{\text{Lasso}} = 0.001.$$

We find that the best model for fitting polynomials to the terrain data is found using Ridge regression with 4 features. This conclusion could only be reached by introducing the incomplete polynomials.

During the project we have reflected on multiple possible goals for further research. Our observations of the poorer performance of Lasso, motivates the implementation of our own optimisation algorithms, which will be part of the objective of the next project.

5 Appendix

5.1 GitHub

Our code as well as jupyter noteboooks used to generate the figures in this paper can be found at:

https://github.com/Trond01/Project1_FYS-STK4155

5.2 Norm inequality

Claim: For any $\boldsymbol{x} \in \mathbb{R}^n$ we have

$$\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1. \tag{5.1}$$

Proof. Since squaring is an increasing function, we can consider:

$$\|\boldsymbol{x}\|_2^2 = \left(\sqrt{\sum_{i=1}^{n} |x_i|^2}\right)^2 = \sum_{i=1}^{n} |x_i|^2$$
$$\|\boldsymbol{x}\|_1^2 = \left(\sum_{i=1}^{n} |x_i|\right)^2 = \sum_{i=1}^{n} |x_i|^2 + 2\sum_{i<j} |x_i||x_j|.$$

From which the result follows.

□

5.3 Deriving the expectation and variance of $\hat{\boldsymbol{\beta}}$ in OLS

We assume that a point $y_i = \boldsymbol{X}_{i,*}\boldsymbol{\beta} + \varepsilon_i$ where $\varepsilon_i \sim N(0, \sigma^2)$. We find the expecation of y_i:

$$\begin{aligned}\mathbb{E}(y_i) &= \mathbb{E}(\mathbf{X}_{i,*}\boldsymbol{\beta}) + \mathbb{E}(\varepsilon_i) \\ &= \mathbf{X}_{i,*}\boldsymbol{\beta}.\end{aligned}$$

For the variance of y_i we find

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

Hence, $y_i \sim N(\mathbf{X}_{i,*}\boldsymbol{\beta}, \sigma^2)$, that is $\boldsymbol{y}$ follows a normal distribution with mean value $\mathbf{X}\boldsymbol{\beta}$ and variance σ^2. We can now use this to derive the expectation and variance of the optimal OLS parameters $\hat{\boldsymbol{\beta}}$.

$$\begin{aligned}
\mathbb{E}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}[(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y}] \\
&= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbb{E}[\mathbf{Y}] \\
&= \boldsymbol{\beta}, \\
\mathrm{Var}(\hat{\boldsymbol{\beta}}) &= \mathbb{E}[\boldsymbol{\beta} - \mathbb{E}(\boldsymbol{\beta})][\boldsymbol{\beta} - \mathbb{E}(\boldsymbol{\beta})]^T \\
&= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbb{E}[\mathbf{Y}\mathbf{Y}^T]\mathbf{X}(\mathbf{X}^\top\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T \\
&= \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}.
\end{aligned}$$

5.4 Deriving the Bias-Variance Tradeoff

Our setup is as follows. We assume that we have a dataset

$$\mathcal{D} = \{(x_i, y_i) | i = 0, 1, \ldots, n-1\}. \tag{5.2}$$

We assume that the data is generated from a underlying function f with the addition of some normally distributed noise with mean 0 and variance σ^2, that is

$$y_i = f(x_i) + \boldsymbol{\varepsilon}_i. \tag{5.3}$$

Let $\tilde{y}$ denote the prediction of our model. We are interesting in studying the expected test error

$$\mathbb{E}[(y - \tilde{y})^2]. \tag{5.4}$$

We will be assuming that the function f is of the form $f(x) = x\boldsymbol{\beta}$. The parameters $\boldsymbol{\beta}$ are in turn found by optimizing the MSE error, our cost function. To evaluate the model we look at the same cost function evaluated at a set of test data. Let $\tilde{y}$ denote a prediction from our model.

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \approx \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right]. \tag{5.5}$$

Defining the Bias and Variance:

$$\begin{aligned}\text{Bias}[\tilde{y}] &= \mathbb{E}[(f(x) - \mathbb{E}[\tilde{y}])^2] \approx \mathbb{E}[(y - \mathbb{E}[\tilde{y}])^2] \\ \text{Var}[\tilde{y}] &= \mathbb{E}[(\tilde{y} - \mathbb{E}[\tilde{y}])^2],\end{aligned}$$

we show the following result:

Claim: $\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \text{Bias}[\tilde{y}] + \text{Var}[\tilde{y}] + \sigma^2$,

Proof. First of we have by linearity that

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}[\boldsymbol{y}^2] - 2\mathbb{E}[\boldsymbol{y}\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}^2]. \tag{5.6}$$

Further, we always have $\text{Var}[\tilde{\boldsymbol{y}}] = \mathbb{E}[\tilde{\boldsymbol{y}}^2] - (\mathbb{E}[\tilde{\boldsymbol{y}}])^2$ so clearly

$$\mathbb{E}[\tilde{\boldsymbol{y}}^2] = \text{Var}(\tilde{\boldsymbol{y}}) + (\mathbb{E}[\tilde{\boldsymbol{y}}])^2. \tag{5.7}$$

We will use our assumption that the y's are generated from a function f evaluated at non-stochastic x and adding some noise. For simplicity denote $f(x)$ by f. We find:

$$\mathbb{E}[\boldsymbol{y}^2] = \mathbb{E}[(f + \boldsymbol{\varepsilon})^2] = f^2 + 2f\ \mathbb{E}(\boldsymbol{\varepsilon}) + \mathbb{E}[\boldsymbol{\varepsilon}^2]. \tag{5.8}$$

Since $\boldsymbol{\varepsilon} \sim N(0, \sigma^2)$ we know $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$ and $\mathbb{E}[\boldsymbol{\varepsilon}^2] = \sigma^2$. This gives $\mathbb{E}[\boldsymbol{y}^2] = f^2 + \sigma^2$. Finally for the middle term we have that

$$\mathbb{E}[\boldsymbol{y}\tilde{\boldsymbol{y}}] = \mathbb{E}[(f+\boldsymbol{\varepsilon})\tilde{\boldsymbol{y}}] = f\ \mathbb{E}[\tilde{\boldsymbol{y}}] + \mathbb{E}[\boldsymbol{\varepsilon}]\ \mathbb{E}[\tilde{\boldsymbol{y}}] = f\ \mathbb{E}[\tilde{\boldsymbol{y}}], \tag{5.9}$$

since again $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$. If we now collect all the terms and regroup to find:

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

□

5.5 MSE and R2 scores

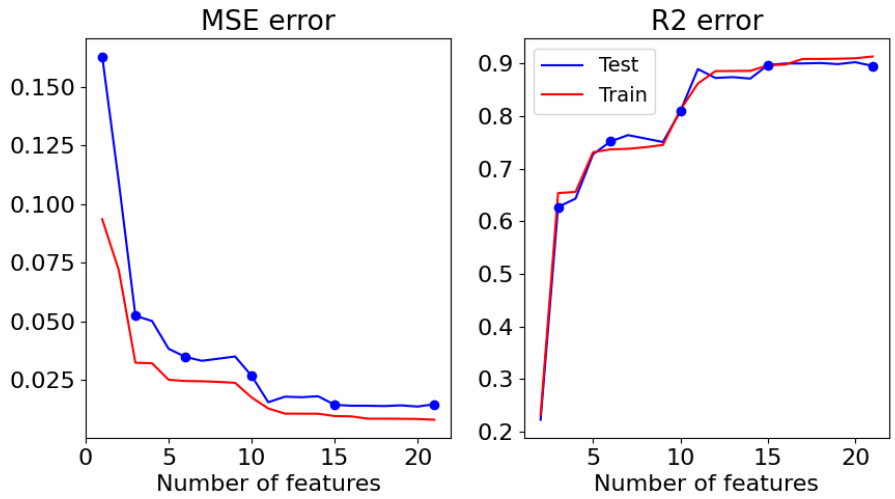


Figure 17: As in Figure 3, but with $N(0, \sigma^2)$ noise with $\sigma^2 = 0.01$.

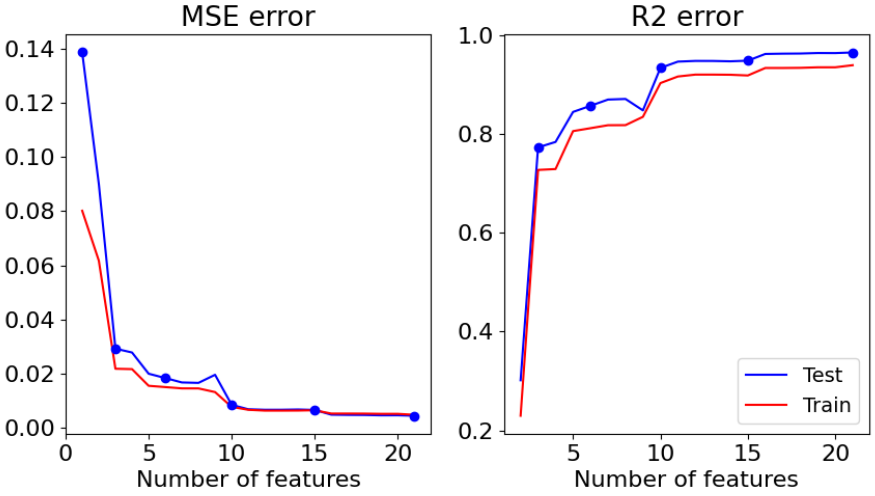


Figure 18: MSE and R2 errors for the Franke function using Ridge regression with $\lambda = 0.01$

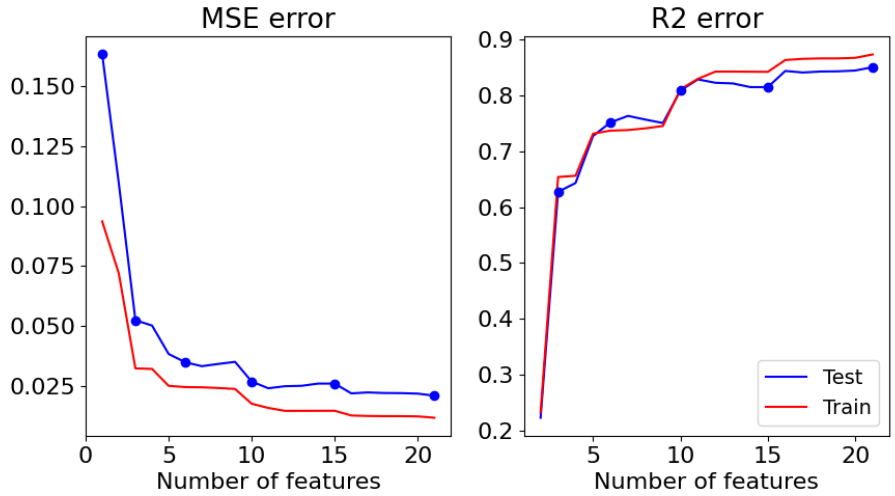


Figure 19: As in Figure 18, but with $N(0, \sigma^2)$ noise with $\sigma^2 = 0.01$.

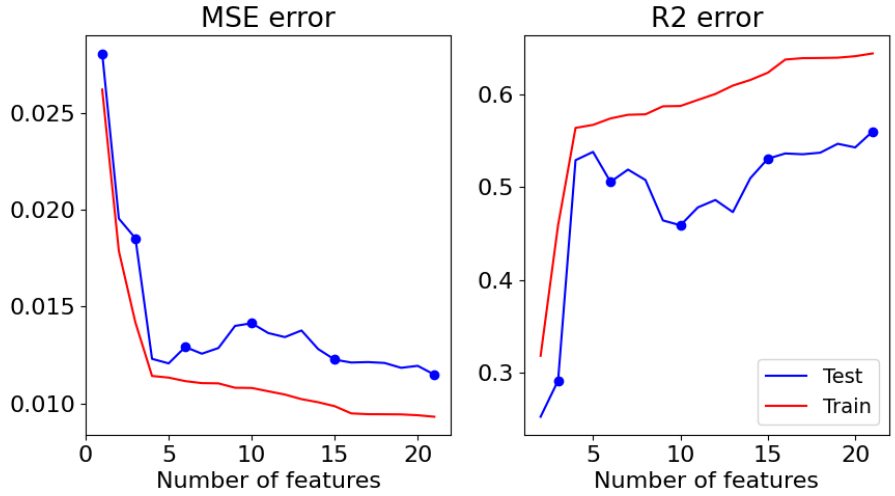


Figure 20: Errors from Ridge regression performed on terrain data with $N = 101$ points.

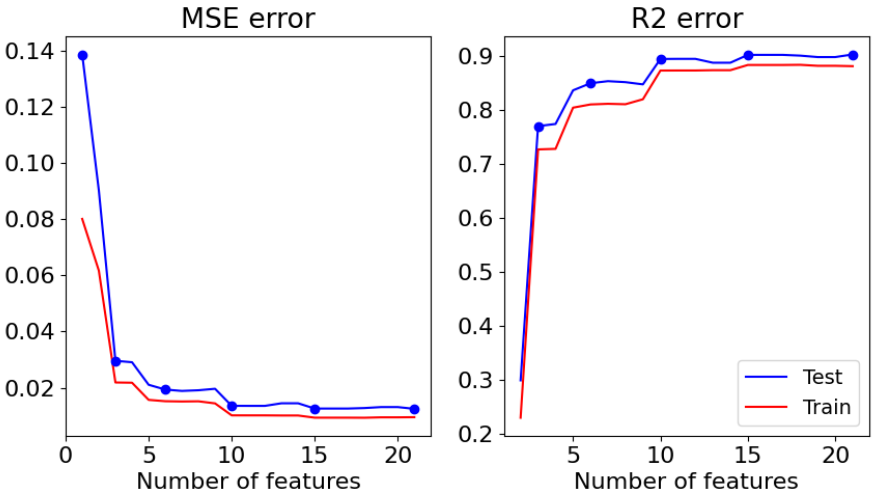


Figure 21: MSE and R2 errors for the Franke function using Lasso regression with with $\lambda = 0.001$

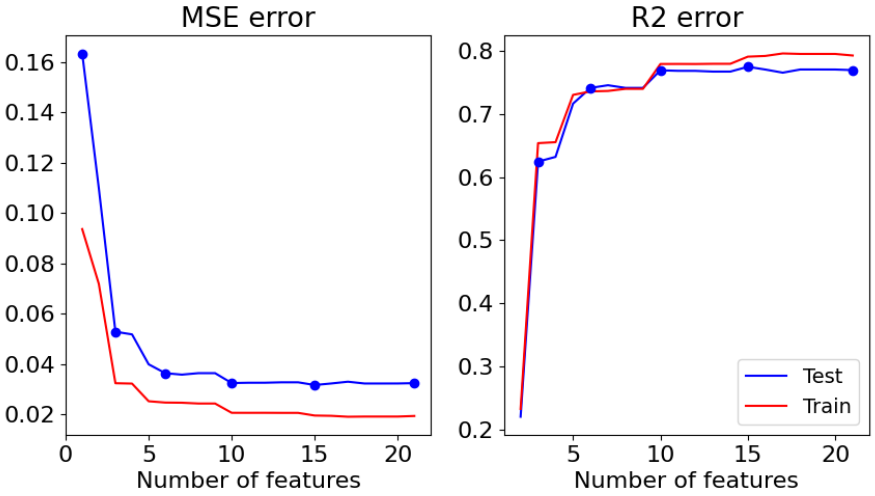


Figure 22: As in Figure 21, but with $N(0, \sigma^2)$ noise with $\sigma^2 = 0.01$.

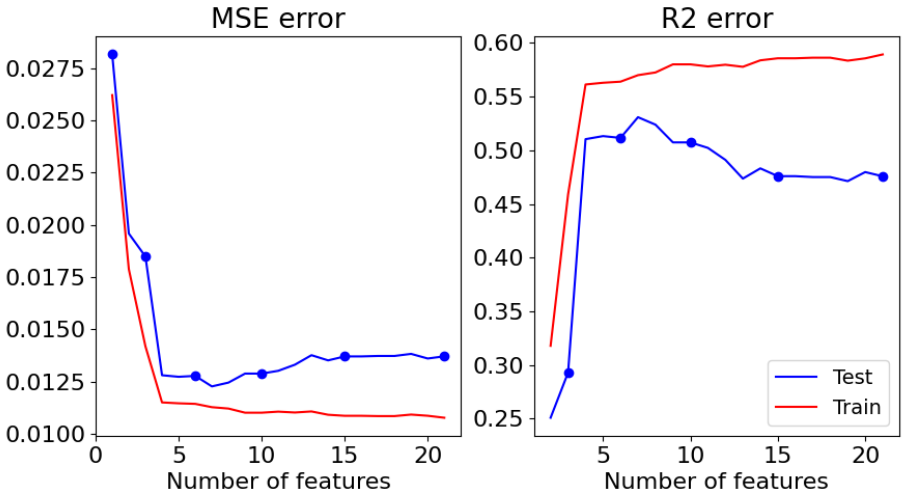


Figure 23: Errors from Lasso regression performed on terrain data with $N = 101$ points.

5.6 Analysis of lambda ranges

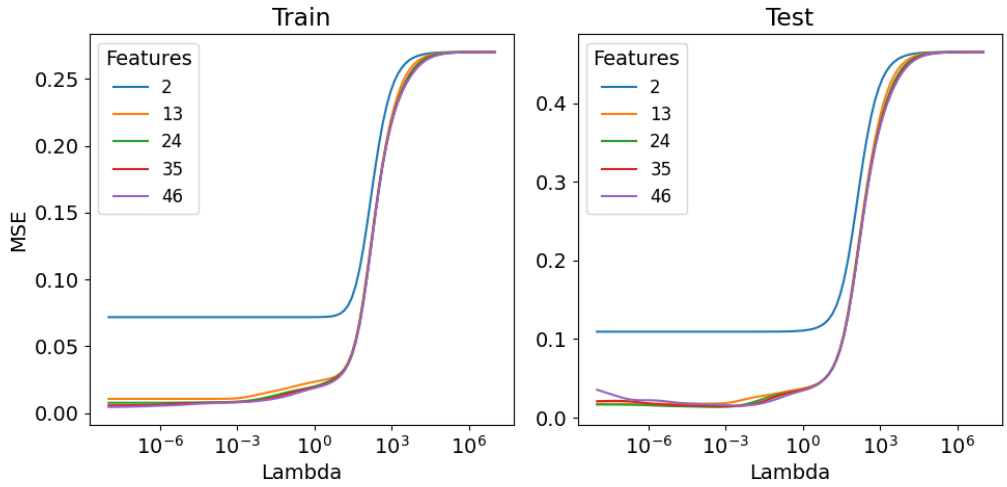


Figure 24: Train and test loss for Ridge regression with varying λ and number of features. Dataset is sampled from a noisy Franke function where the noise is normally distributed with variance 0.01. Done with 101 points in the dataset.

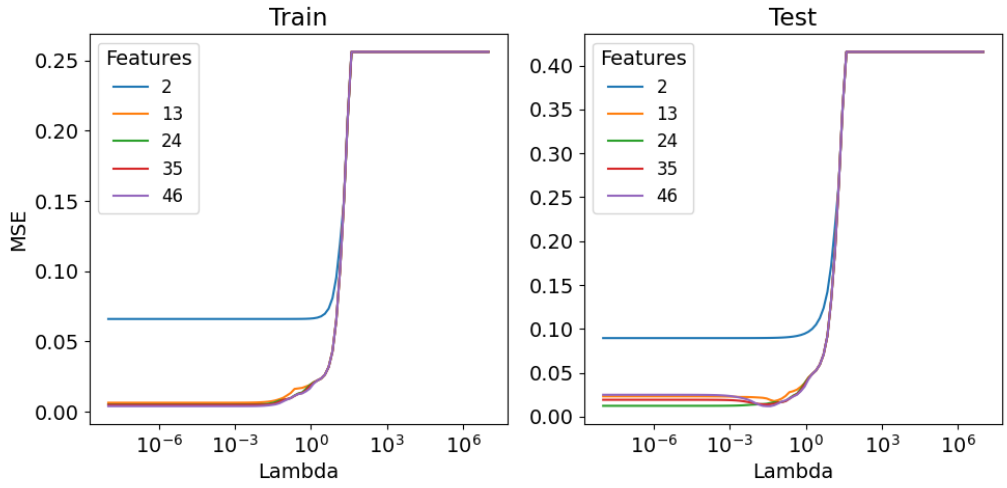


Figure 25: Train and test loss for Lasso regression with varying λ and number of features. Done with 51 points in the dataset.

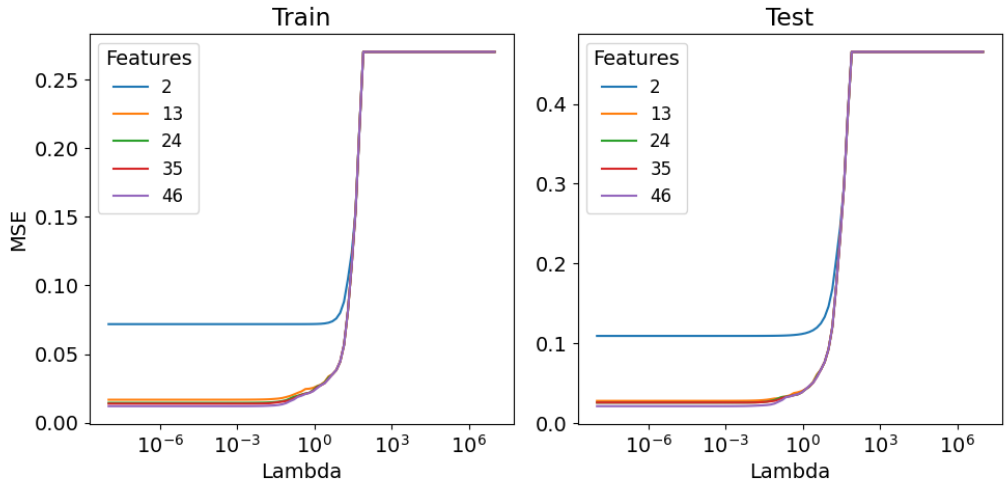


Figure 26: Train and test loss for Lasso regression with varying λ and number of features. Dataset is sampled from a noisy Franke function where the noise is normally distributed with variance 0.01. Done with 101 points in the dataset.

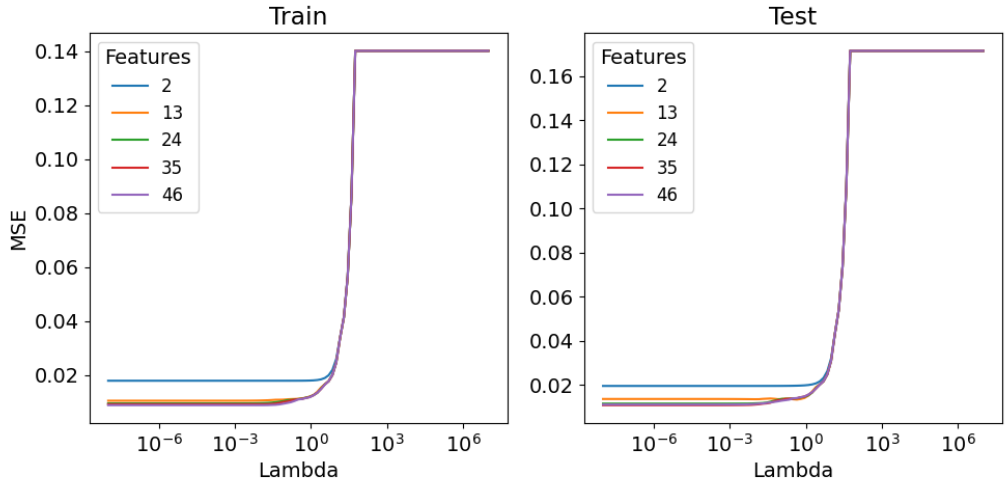
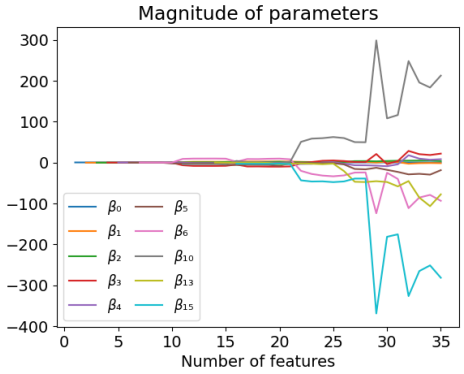
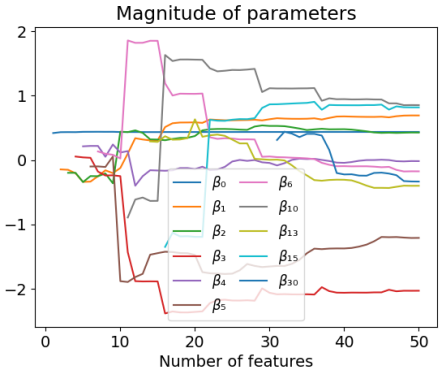


Figure 27: Train and test loss for Lasso regression with varying λ and number of features. Dataset is sampled from USGS terrain data. Done with 101 points in the dataset.

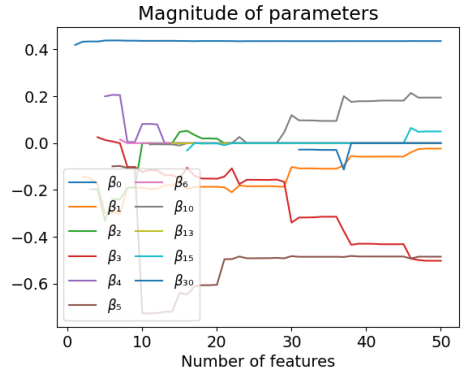
5.7 Plots of beta parameters with noise



(a) OLS



(b) Ridge



(c) Lasso

Figure 28: Parameter values of OLS, Ridge and Lasso regression, with varying number of feature. $\lambda = 0.01$ and $\lambda = 0.001$ for Ridge and Lasso respectivelly. Dataset is sampled from a noisy Franke function where the noise is normally distributed with variance 0.01. Done with 101 points in the dataset.

5.8 Cross Validation of the Franke function

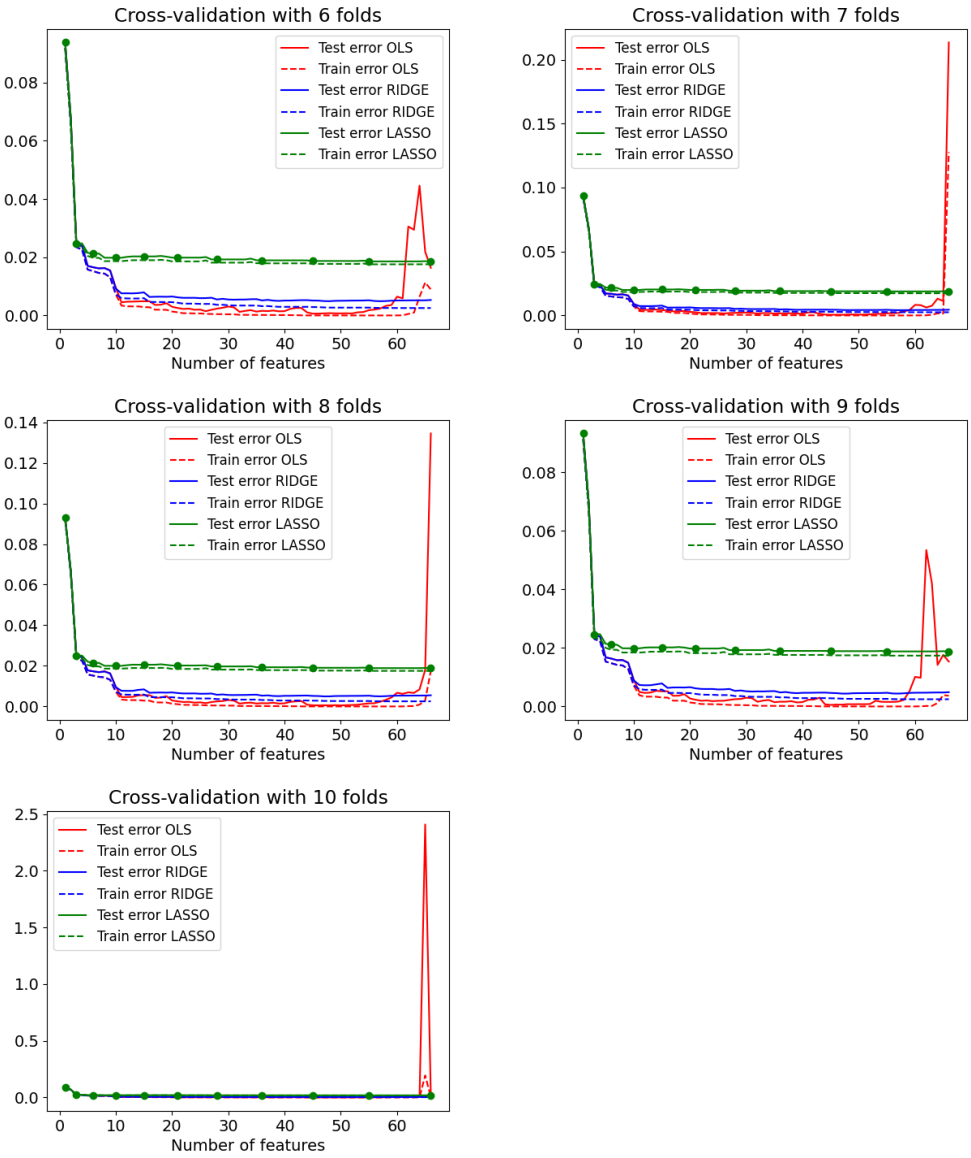


Figure 29: Cross validation of regression on the Franke function with 101 points evenly distributed in $k = 6, 7, 8, 9, 10$ folds.

5.9 Cross Validation of the terrain data

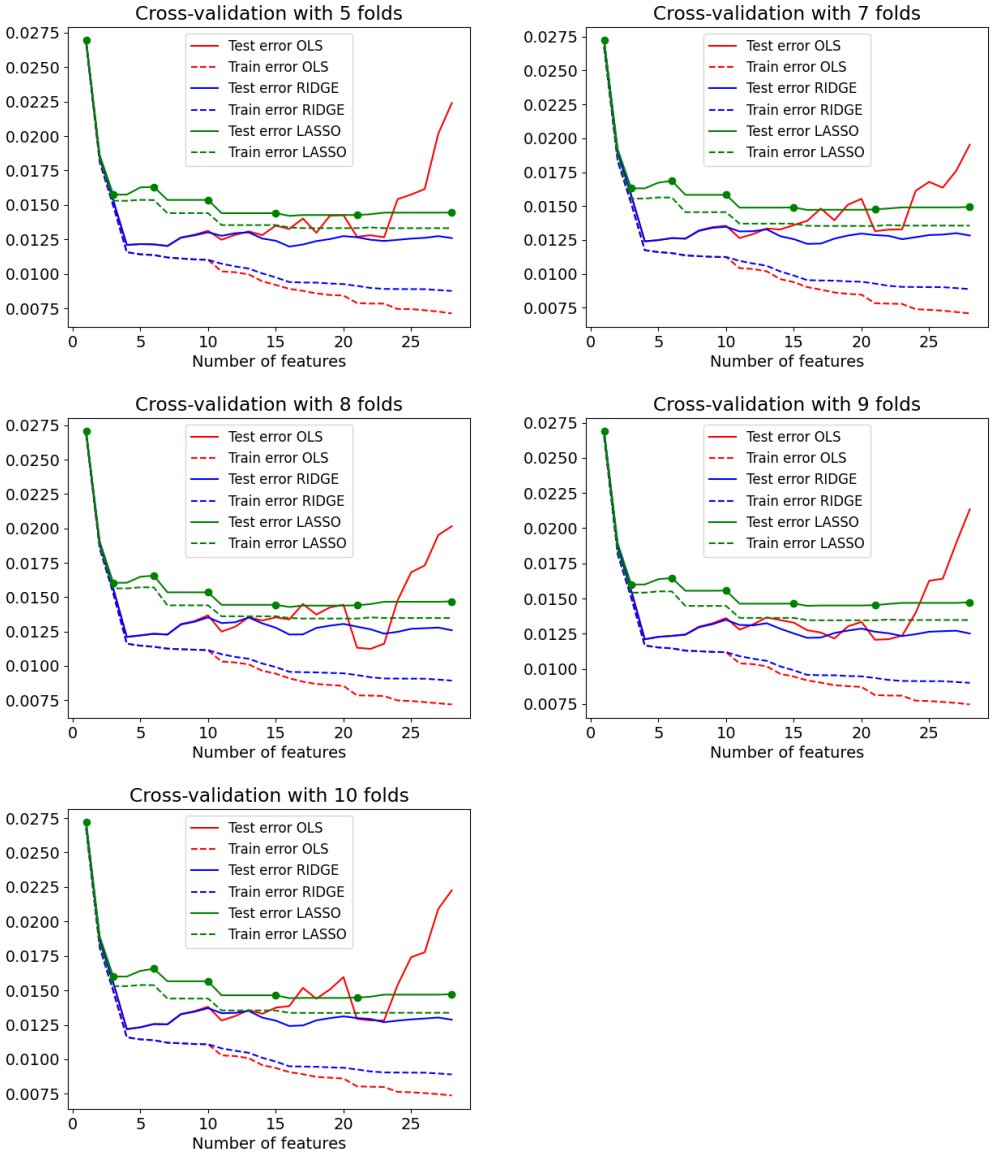


Figure 30: Cross validation of regression on the terrain data with 101 points evenly distributed in $k = 6, 7, 8, 9, 10$ folds.

References

[1] (USGS), U. G. S. *EarthExplorer*. Accessed: 2023-09-29. 2023. URL: https://earthexplorer.usgs.gov/.

[2] Hastie, T., Tibshirani, R., and Friedman, J. *Elements of Statistical Learning: Data Mining, Inference, and Prediction*. eng. Second Edition. Springer Series in Statistics. New York: Springer, 2009, pp. xxii–xxii.

[3] Hjorth-Jensen, M. *Applied data analysis and machine learning: Ridge and Lasso Regression*. Accessed: 2023-10-15. 2023. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter2.html#id1.

[4] Lindstrøm, T. L. *Spaces : an introduction to real analysis*. eng. Providence, R.I, 2017.

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