FYS-STK4155 - Applied data analysis and machine learning

Project 1

Tommy Myrvik and Kristian Tuv

October, 2018

Github repository for the project

Contents

1 Abstract

In this project we have studied three different regression models: The Ordinary Least Squares, Ridge and Lasso methods, where the latter two introduce regularization of the regression parameters. We have introduced some theory on have these work, and applied them on two different types of data. It can be mathematically shown that a polynomial of high enough degree is able to fit any continuous function arbitrarily well. What one would want however, is that this fit also performs well on new data belonging to the same distribution. The models must hence be able to find a good balance between performance on both *seen* and *unseen* data to be considered as good models.

First we applied the regression methods on a self-generated data set that was sampled from a small region of Franke's Function, an exponential function consisting of two gaussian tops within the sampled region. We also applied our methods on real data of the terrain of a small part of the mainland of Norway. The goal was to find what models fit the data the best, trying out several models from each method in each case. We found that in addition to the model parameters, the results are also heavily dependent on the noise and sample size of the data, where a denser sample size seems to favor the otherwise poor-performing OLS method. For sparser data sets, it's more important to find the underlying structures of the data to be able to predict unseen points from the same distribution. This is where the ridge and lasso methods outperform the OLS method, as was found in both the cases of Franke's Function, and the real terrain data. Amongst these two the ridge seemed to be the preferred method on the terrain data, although this isn't fully conclusive, as the results were generated from relatively small subsets of the full data set.

2 Introduction

Linear regression is the process of fitting a polynomial of unknown degree to a data set. There is a limit to this process however, as the precision of our model scales poorly with increased amounts of dimensions, features and overall complexity of the data.
We will study uses and shortcomings of linear regression in this project.

2.1 Goals

The aim of this project is to study three different linear regression models, The Ordinary Least Squares (OLS), Ridge regression and Lasso regression, when they are applied to Franke's function [9]. We will compare the differences and the benchmarks of each model by evaluating the mean squared error, the R2 score function and the variance of the model parameters. For every method we will also use and compare the k-fold cross validation and bootstrap resampling techniques.
Finally, we will use the same three methods of linear regression, the same type or higher

order polynomial approximation and the same resampling techniques on real digital terrain data from earthexplorer.usgs.gov to see which model fits the best.

2.2 Structure

- We will first define the mathematical and computational theory of the linear regression models, necessary statistics and the resampling techniques
- We then present the datasets we will work with, before giving the results of our analysis on each data set
- Finally we will conclude the project and discuss future implementations

I Theory

3 Linear regression methods

In general, regression revolves around curve fitting, i.e. finding the best fit for a data set to a model of choice. This model can be a linear function, a polynomial, a sum of sinusoids, or anything in between. For our regression model to be *linear* doesn't mean that we fit the data to a first order polynomial, but rather that the models output is a linear product of the predictor variables $\hat{x} = [x_0, x_1, ...x_{p-1}]^T$ and the regression parameters $\hat{\beta} = [\beta_0, \beta_1, ...\beta_{p-1}]$ so that:

$$\tilde{y} = \sum_{i=0}^{p-1} \beta_i x_i \tag{1}$$

It's important to note here that the vector $\hat{x}$ is not a vector of different x points in the data set, but the representation of a single point in the model of choice, i.e. $\hat{x} = [1, x, x^2]^T$ for a second order polynomial.

In turn, an actual data point from the data set y can be written as:

$$y = \sum_{i=0}^{p-1} \beta_i x_i + \epsilon \tag{2}$$

where ϵ is the error/difference between the data and the model estimations:

$$y - \tilde{y} = \epsilon \tag{3}$$

When fitting a model to a data set, we also get a set of equations with the form described above, where the goal is to find a vector $\hat{\beta}$ that translates the input predictors into the data points in the best way possible. While the $\hat{\beta}$ vector is the same for all data point pair (x_i, y_i), each pair has it's own predictor vector $\hat{x}$ (i.e. each x value is represented in terms of the model), which can be gathered in a matrix $\hat{X}$ called the *design matrix*. Writing the set of equations for n (x, y) pairs using the elements in the design matrix we get:

$$
\begin{aligned}
y_0 &= \beta_0 x_{00} + \beta_1 x_{01} + ... + \beta_{p-1} x_{0p-1} + \epsilon_0 \\
y_1 &= \beta_0 x_{10} + \beta_1 x_{11} + ... + \beta_{p-1} x_{1p-1} + \epsilon_1 \\
&\vdots \qquad\qquad\qquad \vdots \\
y_i &= \beta_0 x_{i0} + \beta_1 x_{i1} + ... + \beta_{p-1} x_{ip-1} + \epsilon_i \\
&\vdots \qquad\qquad\qquad \vdots \\
y_{n-1} &= \beta_0 x_{n-10} + \beta_1 x_{n-11} + ... + \beta_{p-1} x_{n-1p-1} + \epsilon_{n-1}
\end{aligned}
$$

A convenient realization here is that all these equations can be gathered and written on matrix form like this:

$$\hat{y} = \hat{X}\hat{\beta} + \hat{\epsilon} \tag{4}$$

Thus far we've described the given data set as a pair-wise set of x and y, but there is nothing in the way of expanding the regressions into higher dimensions, say the data points z as a function of both x and y. Note that the only component that will change in this case is the design matrix $\hat{X}$, where it now will take both variables x and y into consideration. In the second degree polynomial case, a row in $\hat{X}$ will now consist of the elements $[1, x, y, x^2, xy, y^2]$, where each term in the polynomial is represented and consequently has it's own β parameter in the $\hat{\beta}$ vector. In this project it's these bivariate types of data sets we'll be working with.

The Weierstrass theorem [3] (or more general the Stone-Weierstrass theorem) says that any continuous function has a polynomial representation as accurate as desired, given enough terms (degrees). This means that we can find a polynomial model that gives a (nearly) perfect fit, no matter what the underlying function in the data looks like, as long as it's continuous. The problem comes when this function is disturbed by noise, hence altering the data set we have to work with. In most cases we only have the data set, not knowing what parts of the data is the underlying function, and what is just noise. The above theorem still stays true, raising questions on whether our model also has fitted the noise, and not the underlying function. This phenomena of *overfitting* will be discussed further in the next sections.

There are many ways to find an estimate for the optimal $\hat{\beta}$ for a given data set (with or without noise) and a model of choice, each with their pros and cons. Three of the most popular methods are the Ordinary Least Squares, Ridge and Lasso methods, and these will be outlined in the subsections of this chapter.

3.1 Ordinary Least Squares

In Ordinary Least Squares we are finding the β values that minimizes the residual sum of squares (RSS), which here is defined as the cost function Q:

$$Q(\beta) = \sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \sum_{i=0}^{n-1}(y_i - \sum_{j=0}^{p-1} x_{ij}\beta_j)^2 \tag{5}$$

or on matrix form:

$$Q(\beta) = (\hat{y} - \hat{X}\hat{\beta})^2 = (\hat{y} - \hat{X}\hat{\beta})^T(\hat{y} - \hat{X}\hat{\beta}) \tag{6}$$

We can interpret the RSS as the sum of all the squared errors, or distances, from the data points y to the corresponding points $\tilde{y}$ in the regressed line. This is illustrated in figure 1. We want as small an error as possible and hence find the minimum of the cost function (here the RSS) with respect to β.

$$\begin{gathered}\frac{\partial Q(\beta)}{\partial \beta_j} = 0 \\ \frac{\partial}{\partial \beta_j}\left(\sum_{i=0}^{n-1}(y_i - \sum_{j=0}^{p-1} x_{ij}\beta_j)^2\right) = 0 \\ -2\sum_{i=0}^{n-1} x_{ij}(y_i - \sum_{j=0}^{p-1} x_{ij}\beta_j) = 0\end{gathered}$$

which we on matrix form recognize as:

$$\begin{gathered}-2\hat{X}^T(\hat{y} - \hat{X}\beta) = 0 \\ \hat{X}^T\hat{y} = \hat{X}^T\hat{X}\beta\end{gathered}$$

$$\hat{\beta} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\hat{y} \tag{7}$$

Further, by differentiating again with respect to β_j one can show that:

$$\frac{\partial^2 Q}{\partial \beta^2} = 2\hat{X}^T\hat{X} \tag{8}$$

where the resulting matrix is positive definite (if we assume none of the columns in $\hat{X}$ are linearly dependent), which in turn proves that $Q(\beta)$ is at a *minimum* in equation (7).

Another quantity of interest is the *variance* of the parameters $\hat{\beta}$, which tells us how sensitive the β values are to changes in the sampling, i.e. if the model was fitted on different points from the same distribution. Starting from (7), we can show that the variance can be expressed as:

$$\begin{aligned}
Var(\hat{\beta}) &= Var((\hat{X}^T\hat{X})^{-1}\hat{X}^T\hat{y}) \\
&= (\hat{X}^T\hat{X})^{-1}\hat{X}^T)Var(\hat{y})((\hat{X}^T\hat{X})^{-1}\hat{X}^T)^T \\
&= (\hat{X}^T\hat{X})^{-1}(\hat{X}^T\hat{X})(\hat{X}^T\hat{X})^{-1}Var(\hat{y}) \\
&= (\hat{X}^T\hat{X})^{-1}Var(\hat{y})
\end{aligned}$$

ending up at

$$Var(\hat{\beta}) = (\hat{X}^T\hat{X})^{-1}\sigma^2 \tag{9}$$

where we have used the general result $Var(AB) = AVar(B)A^T$ with matrices A and B, where A is a fixed matrix, and also that

$$Var(\hat{y}) = \sigma^2 = \frac{1}{N-p-1}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2. \tag{10}$$

Equation (7) finds the best fit to data it's given, and in general the more data points, the better fit. This sounds good, but can give really catastrophic results when applied in practice. There is a difference between data that the model has seen, and data it hasn't seen. If the data the model is trained on is full of noise, the OLS regression will usually fit to this noise, given enough model complexity. It doesn't look for underlying features in the data, it just fits to what's given to it. It's prediction on any point it hasn't seen before is thus likely to be completely off, making the once promising model a quite bad one. This makes the variance (explained in section 4.2) of the model quite high, where the prediction of new data will vary greatly, depending on the data it has been fitted to. We'll now look into a few regression methods that might be able to look past the noise in the data, giving better predictions of unseen data, namely the Ridge and Lasso methods.

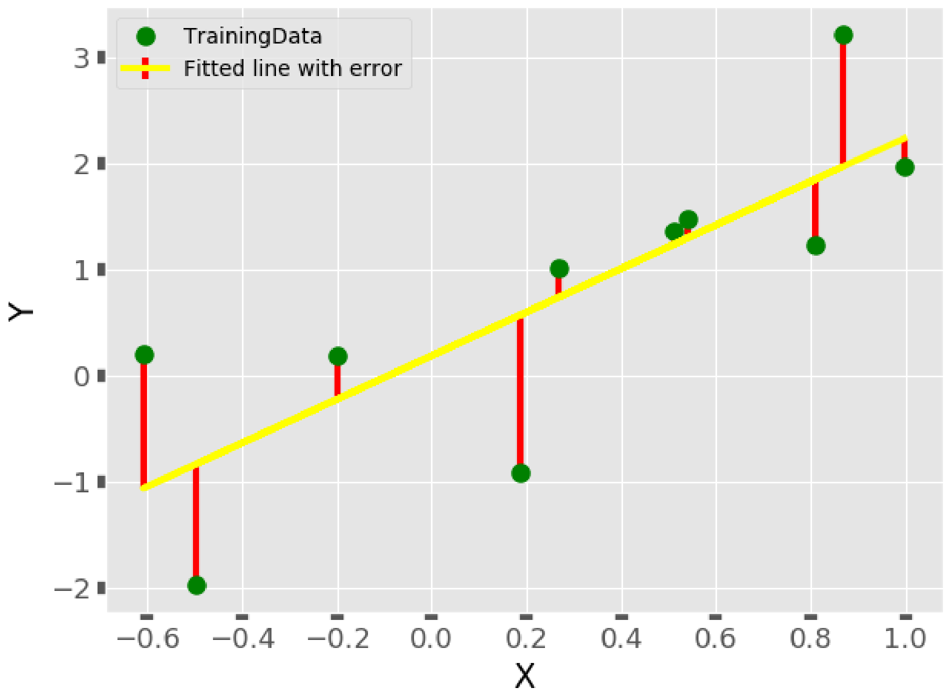


Figure 1: *Training data in green and a fitted line in yellow with the errors in red. We find the RSS by squaring these errors and summing them.*

3.2 Ridge

In OLS our startingpoint for minimizing our cost function was the RSS

$$Q(\beta) = RSS(\beta) = \sum_{i=0}^{n-1}(y_i - \sum_{j=0}^{p-1} x_{ij}\beta_j)^2 \tag{11}$$

The ridge regression method is a modification to OLS, where the cost function now has an extra term

$$Q(\beta) = \sum_{i=0}^{n-1}(y_i - \sum_{j=0}^{p-1} x_{ij}\beta_j)^2 + \lambda \sum_{j=0}^{p-1} \beta_j^2 \tag{12}$$

where the last term often is referred to as the *penalty term*. In this term we see that we have the factor β^2 (times a constant λ) that's effectively being added to the RSS. From this we can understand that large β values are bringing larger contributions to the cost function, and this is where the term *penalty* comes in. As the cost function is the quantity we want to minimize, it's beneficial for all the β values to be small, effectively penalizing the large ones and driving them down. Punishing the β values too much however sacrifices the fit given in the RSS-term, causing a low penalty, but high RSS. Minimizing the cost function then seeks to find a balance between a good general fit (RSS) and the amplitude of the feature estimates (β values). This balancing makes complex models more viable, as the large β values that would cause overfitting in OLS are "shrunk" down so that more of the features are contributing at the same levels.

This process of shrinking the coefficients is called regularization, and the strength of this regulating is determined by the regularization constant, λ. The value of this constant varies from problem to problem, meaning there's no general "best value". For each new model we have to try out several values for λ, and see which one performs best. One measure for this is the *Aikaike's information criterion* (AIC) [2], which calculates the balance of the model fit and model complexity via the log-likelihood of the fit and the number of degrees of freedom of the model. The λ value that minimizes the AIC is the best suited for the model.

Another way to compute the optimal regularization constant is to run a cross-validation algorithm, which calculates the performance (usually by the mean squared error) for different λ values directly on subsets of the data set, where the λ with the lowest average error is deemed the best one. This is the method we're using in this project, and will be discussed further in section 4.4.2.

Looking back on equation (12) we will derive an expression for the optimal $\hat{\beta}$ in the same way as we did for the OLS. Finding the minimum for the cost function gives us:

$$\frac{\partial Q(\beta)}{\partial \beta_j} = 0$$
$$\frac{\partial}{\partial \beta_j}\left(\sum_{i=0}^{n-1}(y_i - \sum_{j=0}^{p-1} x_{ij}\beta_j)^2 + \lambda \sum_{j=0}^{p-1}\beta_j^2\right) = 0$$
$$-2\sum_{i=0}^{n-1} x_{ij}(y_i - \sum_{j=0}^{p-1} x_{ij}\beta_j) + 2\lambda \sum_{j=0}^{p-1}\beta_j = 0$$

Writing this on matrix form gives:

$$-2\hat{X}^T(\hat{y} - \hat{\beta}\hat{X}) + 2\lambda\hat{\beta} = 0$$
$$(\hat{X}^T\hat{X} + \lambda)\hat{\beta} = \hat{X}^T\hat{y}$$

where we finally end up with the expression:

$$\hat{\beta}^{ridge} = (\hat{X}^T\hat{X} + \lambda I)^{-1}\hat{X}^T\hat{y} \tag{13}$$

where λ is actually $\lambda \cdot I_p$, so that the λ value is effectively added to the diagonal of the $\hat{X}^T\hat{X}$ matrix.

The calculation of the variance of the β parameters is a little more complicated, and we'll refer to Wessel N. van Wieringen (2018)[4] for the derivation. Here it's shown that the variance can be written as:

$$Var(\hat{\beta}^{ridge}) = [\hat{X}^T\hat{X} + \lambda I_{pp}]^{-1}\hat{X}^T\hat{X}([\hat{X}^T\hat{X} + \lambda I_{pp}]^{-1})^T\sigma^2 \tag{14}$$

where

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

as before.

One thing to note about the ridge method (and also the lasso method discussed in the next section) is that it's more a method of *predicting* rather than *infering.* The OLS on one hand always finds the best fit to the points it's being *fitted on*, but it doesn't necessarily transfer well to points it *hasn't* seen before (it often actually gives catastrophic results as we'll see later). As the cost of the ridge method always will be higher than the one for OLS, it will give a sub-optimal fit for the points it's being fitted on. In contrary to OLS however, the ridge method is less likely to overfit to the training data, by using all features a little rather than using few features a lot. This makes the model less sensitive to changes in the data set, where it in practice lowers the variance as λ increases, at the cost of some bias. A model with a good choice of λ is likely to generalize better on noisy data, making it better at predicting unseen points.

For more information on the ridge regression method, Wessel N. van Wieringen (2018)[4] is a recommended read.

3.3 Lasso

The Lasso regression method is another method that is very similar to the Ridge. The cost function looks like this:

$$Q(\beta) = \sum_{i=0}^{n-1}(y_i - \sum_{j=0}^{p-1}x_{ij}\beta_j)^2 + \lambda\sum_{j=0}^{p-1}|\beta_j| \tag{15}$$

where the only difference lies in the penalty term, where β^2 factor is replaced by the absolute value $|\beta|$. While the difference is quite subtle, it has a few consequences in terms of both further mathematical derivations, and the resulting regressions.

First of all, it's not possible to find a analytical expression for the $\hat{\beta}$ at which the cost function is minimized, as we've done for both the OLS and ridge. This means we have to use other minimization methods like gradient descent or Newton's method to try to find an approximation to the minimum. One will usually have to choose a random $\hat{\beta}$ vector to initialize the minimization, and depending on how close this guess is to the minimum, we might experience significant differences in number of iterations for the algorithm to converge. In some cases it won't converge at all. Such a method is needless to say a little more complicated to implement than the OLS and Ridge cases, so in this project we'll use Scikit Learn's Lasso regression function to make sure that the results we are getting are correct.

The Lasso methods effect on the resulting β values are much of the same as for Ridge, but there is a significant difference: While the Ridge drives the β values down towards, but not exactly 0, Lasso does just this. This is called feature selection, where some parameters can be set to 0, leaving only the more significant β values in the model. This makes the model simpler, and like the ridge, makes it less prone to over-fitting. As with Ridge, the parameter λ controls the strength of the penalization, and a stronger penalization lowers the variance (at the cost of some bias) also for this method.

3.4 Singular-value decomposition

Singular-value decomposition (SVD) is a way to decompose any matrix $\hat{A}$ into a product of three new matrices, often called $\hat{U}$, $\hat{D}$ and $\hat{V}$, so that

$$\hat{A} = \hat{U}\hat{D}\hat{V}^T \tag{16}$$

If the matrix $\hat{A}$ has dimensions $m \times n$, the dimensions for $\hat{U}$, $\hat{D}$ and $\hat{V}$ will be $m \times m$, $n \times n$, and $n \times n$ respectively. The matrix $\hat{D}$ is a diagonal matrix containing the *singular values* [1] of the matrix $\hat{A}$, while $\hat{U}$ and $\hat{V}$ are orthonormal matrices consisting of left-singular and right-singular vectors respectively.

A problem that may occur during linear regression is that our design matrix $\hat{X}$ might be nearly or fully singular, i.e. it's determinant being close to or exactly zero. This can especially be a problem when our data is high-dimensional, i.e. when the number of regressors β is close to, or even larger than the number of data points. A singular matrix (determinant exactly zero) has no inverse per definition, and trying to calculate the inverse of a nearly singular matrix directly is quite unstable numerically. In our analytic expressions (equation (7) for instance) we see that in order to find the regression parameters $\hat{\beta}$, we need to find the inverse of the matrix $(\hat{X}^T\hat{X})$, and we are thus in danger of getting the wrong results if this matrix happens to be close to singular. This is where SVD comes to our rescue, where we can circumvent the unstable inversion by decomposing our design matrix as described above. What's really convenient with SVD is that *every* matrix has such a decomposition, meaning it's a safe choice regardless of what our design matrix looks like.

For the OLS method, we can rewrite equation (7) as:

[1]The singular values of a matrix $\hat{A}$ can be described as the square roots of the eigenvalues of the matrix $\hat{A}\hat{A}^*$ (or equivalently $\hat{A}^*\hat{A}$).

$$\begin{aligned}
\hat{\beta} &= (\hat{X}^T\hat{X})^{-1}\hat{X}^T\hat{y} \\
&= ((\hat{U}\hat{D}\hat{V}^T)^T(\hat{U}\hat{D}\hat{V}^T))^{-1}(\hat{U}\hat{D}\hat{V}^T)^T\hat{y} \\
&= ((\hat{V}\hat{D}\hat{U}^T)(\hat{U}\hat{D}\hat{V}^T))^{-1}(\hat{V}\hat{D}\hat{U}^T)\hat{y} \\
&= (\hat{V}\hat{D}^2\hat{V}^T)^{-1}(\hat{V}\hat{D}\hat{U}^T)\hat{y} \\
&= \hat{V}\hat{D}^{-1}\hat{U}^T\hat{y}
\end{aligned}$$

where we have used the orthonormality of $\hat{U}$ and $\hat{V}$, so that $\hat{U}^T\hat{U} = I$ and $\hat{V}^T\hat{V} = I$. This result is called the pseudo-inverse of $\hat{X}$, where the only "inverting" that is done is calculating the inverse of the diagonal matrix $\hat{D}$, which just consists of the reciprocals of the diagonal elements d_{ii} (i.e. $diag(\hat{D}^{-1}) = 1/d_{11}, 1/d_{22}, ...$).

The same can be done to find a new expression for equation (9), and reusing some of the results above we can see that:

$$\begin{aligned}
Var(\hat{\beta}) &= (\hat{X}^T\hat{X})^{-1}\sigma^2 \\
&= (\hat{V}\hat{D}^2\hat{V}^T)^{-1}\sigma^2 \\
&= \hat{V}\hat{D}^{-2}\hat{V}^T\sigma^2
\end{aligned}$$

It's also possible to derive expressions for $\hat{\beta}^{ridge}$ and $Var(\hat{\beta}^{ridge})$ from equation (13) and (14), but it's not strictly necessary to decompose $\hat{X}$ when using this regression method. The reason is that the addition of λ in the diagonal of $\hat{X}^T\hat{X}$ circumvents the problem with singular matrices. As the derivation and implementation of the SVD equivalents are somewhat messier (for little to no gain), we choose to stick with the already derived equations for the ridge regression method.

4 Statistics/Model assessment

Now that we have derived different methods for fitting data to a model of choice, we need to introduce means to actually evaluate the models we are trying out. There are lots of properties of the models we can calculate, each telling us something about quality of the model, e.g. the overall error of the estimates (mean squared error), the models tendency to over-/underestimate (bias), how prone it is to change in the data set (variance), and so on. All these are among the quantities that will be discussed in the following sections.

One thing that's essential for evaluation of any model is to split your data set, usually into two parts; a *training set* and a *test set*. The training set is the data that you fit your model to, hence you would expect a good fit on this set as long as the model is complex enough. The test set however consist of data points that the trained model has never seen before, and is the vital part for the assessment of the model. Recall from section 3, that according to The Weierstrass theorem [3], the fit of a data set can be "infinitely" accurate given enough model complexity. This means that the fit we get on the training set doesn't really tell us that much other than giving an estimate of the complexity needed to get a reasonable fit. The meat and potatoes of the model assessment really boils down to how well the model perform on a set of points it hasn't seen before. It's important to note that both the training and test sets are sampled from the same population, meaning that a good model should perform well on both sets.

We will now delve into the theory behind the statistical quantities used to evaluate the models, what they actually measure, and what we expect from them when evaluated on both the test sets and the training sets.

4.1 Error measurements

A very direct way to measure the performance of your model is to measure how different your models estimates are from the data, point for point. Perhaps the most direct of these is the mean squared error (MSE), defined as:

$$MSE(\hat{y}, \hat{\tilde{y}}) = \frac{1}{N}\sum_{i=0}^{N-1}(y_i - \tilde{y}_i)^2. \tag{17}$$

Note that this is almost the same as the definition of the RSS used in the cost function of the OLS regression method (equation (5)), but in contrary to this quantity, the MSE calculates the mean value of the errors. The RSS will naturally increase it's value with increasing number of data points, while the MSE disregards this dependence, giving a nice comparable value for data sets of different sizes.

As stated in the introduction of this section, a data set can be fit as well as we want, given enough complexity. When the data is split into training/test, it means that the difference between estimations and data (hence the MSE) on the *training set* goes towards 0 as complexity increases. On the test set however, we are likely to see a high MSE for both too low and too high complexities, with a sweet spot somewhere in the middle where it's at it's lowest. It's not guaranteed that the MSE at this point is 0, due to noise in the fitted data, and sparsity of data points. .

The R2-score is another measurement of the error, and is defined as:

$$R^2(\hat{y}, \hat{\tilde{y}}) = 1 - \frac{\sum_{i=0}^{N-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{N-1}(y_i - \bar{y})^2}, \tag{18}$$

where $\bar{y}$ is the mean value of $\hat{y}$:

$$\bar{y} = \frac{1}{N}\sum_{i=0}^{N-1} y_i. \tag{19}$$

The second term in (18) can be written as $\frac{RSS}{TSS}$, where RSS is the residual sum of squares that has been discussed earlier, and TSS is the *total sum of squares* which is proportional to the variance of $\hat{y}$.

The R2-score is also known as the coefficient of determination, and can be said to describe how much of the variance in the data is accounted for by the model. This could be seen as giving a measure on the likelihood of how well a model predicts future points. Another way to interpret it is that the RSS of the predictions is compared to model of 0th order (constant) set at the average of the data points, $\bar{y}$. A regression of a 0th order polynomial would then give an R2-score of 0.0, as the second term would give exactly 1. An R2-score of 1.0 is the best one possible, and essentially says that the predictions fit the data perfectly. A negative R2-score is also possible, as a fit (especially on a test set) can be arbitrarily worse than the 0th order fit described above.

Another reason for the addition of the R2-score is that it's invariant to the scale of the data, in which the MSE is not. If the data set contains rather large values (which is the case for the terrain data we'll use later in the project), and not normalized/down-scaled before use, the MSE-values will naturally be large since the squared differences may become enormous. This may cause confusion in whether the measured MSE is actually good or not, without having to go in and check the exact size of the data being used. The R2-score will always be near 1 if the model is good, and arbitrarily negative if the fit is bad.

There are many more ways to calculate the errors of the predictions, e.g. the Mean Absolute Error (MAE), the Squared Logarithmic Error (MSLE) and the Adjusted R2-score to mention a few, but in this project we're going to stick with using the two measures discussed in this section.

4.2 Bias-Variance trade off

Assume we have a set of data which goes like $y = f(x) + \epsilon$ where f(x) is the underlying function the data comes from, and ϵ is the random noise in the datapoints, which we assume are coming from the normal distribution with $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma^2$. Hence $f(x)$ is deterministic and all the randomness in y comes from ϵ
Because f is deterministic and $\epsilon \in N(0, \sigma)$ we have the following implications:

$$E[\epsilon] = 0 \tag{20}$$

$$E[f] = f \tag{21}$$

$$E[y] = E[f + \epsilon] = f + 0 = f \tag{22}$$

$$Var[\epsilon] = \sigma^2 \tag{23}$$

$$Var[f] = 0 \tag{24}$$

$$Var[y] = Var[f + \epsilon] = Var[\epsilon] = \sigma^2 \tag{25}$$

$$\tag{26}$$

And in general we have the expected value of a random variable squared:

$$E[X^2] = Var[X] + (E[X])^2 \tag{27}$$

We define another variable $\hat{f}$ which is our models predicted value of y.

Now, the Mean Squared Error of the datapoints y and the predicted values $\hat{f}$ are defined as[1]:

$$\begin{aligned}
E[(y-\hat{f})^2] &= E[y^2 - 2y\hat{f} + \hat{f}^2] && (28)\\
&= E[y^2] + E[\hat{f}^2] - 2fE[\hat{f}] && (29)\\
&= Var[y] + E[y]^2 + Var[\hat{f}] + E[\hat{f}]^2 - 2fE[\hat{f}] && (30)\\
&= \sigma^2 + Var[\hat{f}] + (f^2 - 2fE[\hat{f}] + E[\hat{f}]^2) && (31)\\
&= \sigma^2 + Var[\hat{f}] + (f - E[\hat{f}])^2 && (32)\\
&= \sigma^2 + Var[\hat{f}] + Bias[\hat{f}]^2 && (33)
\end{aligned}$$

The bias of a model can in component form be written as

$$Bias = \frac{1}{N}\sum_{i=1}^{N}(\hat{f}_i - y_i) = \frac{1}{N}\sum_{i=1}^{N}(\hat{f}_i - (f_i + \epsilon)) \tag{34}$$

and gives a measure on our models tendency to either over- or underestimate it's estimates $\hat{f}$, corresponding to a positive or negative bias respectively. In most cases we have no idea what the underlying function $f(x)$ is, all we have is the data points y, which will contain some unknown amount of noise. One thing to note is that the bias is (theoretically) not effected by whether we know $f(x)$ or not, as seen from rewriting the above equation:

$$Bias = E[\hat{f}_i] - E[f_i] - E[\epsilon] \tag{35}$$

where the last term is 0, as was priorly assumed. As we're dealing with a finite-sized data set however, this quantity won't be *exactly* 0. The noise will thus have some impact on the bias, although this effect is usually quite small.

As seen above, the MSE can be rewritten into one bias-term and *two* variance-terms. In practice, we can only compute these two distinct variances if we know $f(x)$, as we can then extract the noise, and it's variance σ^2. If we only have the data points, we can only calculate the variance of our estimates $\hat{f}$ with respect to y, which will the sum of the two variance-terms. This makes it difficult to find a model that has both low bias and low variance, as we can't be sure what fluctuations of the data are noise, and what are actual features of the underlying distributions.

We'll then have to come to terms with there being a trade-off between the bias and the variance of the model, where the goal is to find the optimal model where both quantities

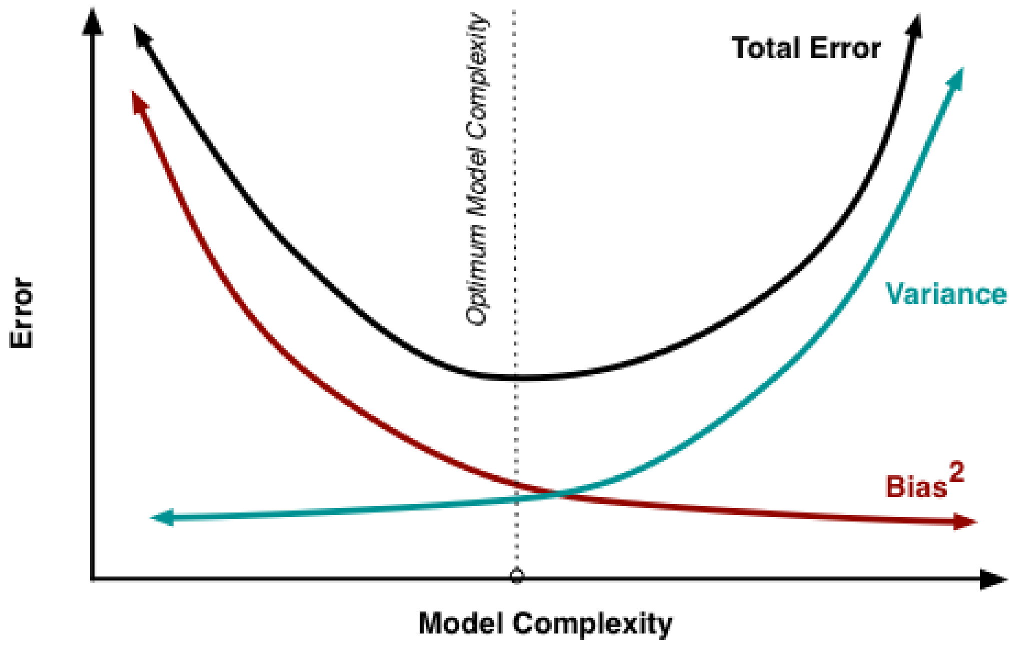


Figure 2: *Schematic figure of the trade-off between variance and bias, in relation to the MSE. Source: http://scott.fortmann-roe.com/docs/BiasVariance.html[7]*

are kept as low as possible. One will usually see that as one increases the complexity of the model, the bias will drop but the variance increase. Low-complexity models are likely to not being able to pick up on the features in the data-set, where they instead tend to give estimates that are quite similar, but "equally wrong" for changes in the data set. This is called *underfitting*, and is characterized by a high bias, but low variance. Highly complex models on the other hand tend to fit more of the fluctuations in the data, also those caused by noise. This gives a good fit on the data it's being fitted on, but is prone to giving huge errors when exposed to data it has never seen before. This is called *overfitting*, and is characterized by low bias, but high variance.

Figure 2 shows a schematic plot of how the bias and variance of the models behave as model complexity increases, where we see that both the bias and variance are somewhat low at the point where the model error is at it's lowest. We'll generate similar plots for the models we create later in this project.

4.3 Confidence intervals

A point estimate, like the sample mean

$$\bar{x} = \sum_i x_i \tag{36}$$

provides no information about the precision of our estimation, and in general due to sampling variability the sample statistics measured does not equal the population parameters. What we might do instead is give an *interval estimate* or a *confidence interval* which might contain the true parameter.
We calculate a confidence interval by first selecting a *confidence level.* The standard confidence level is 95 %. Higher and lower confidence levels might be used, but unless otherwise stated we are always talking about a 95 % confidence level in this project.
A confidence level of 95% would imply that 95% of all samples would yield an interval which includes the true parameter.

Before we can find any confidence interval, we need to manipulate our data.
It is easy to see that as long as the the population is normally distributed, the expectation value of the sample distribution of the sample mean $\hat{X}$ will be equal the the population mean.

$$E(\hat{X}))\mu_{\hat{X}} = \mu \tag{37}$$

This holds for any sample size n.
When the population is not normally distributed however, we need *The central limit theorem (CLT*[10]. The CLT states that even if the population is very much not normally distributed, the sample mean will be, as long as n is large.
And as the size n of the sample increases towards infinity the mean of this normal distribution tends towards the population mean $\mu_{\bar{x}} = \mu$, and the variance towards the population variance divided by the sample size $\sigma_{\bar{x}}^2 = \sigma^2/n$.
When a random variable X is normally distributed, we compute probabilities of X by first *standardizing.* The standardized random variable is

$$Z = (X - \mu)/\sigma \tag{38}$$

Where μ is the population mean and σ is the population standard deviation. We normally do not know the true value of σ, and use the sample variance $\sigma/\sqrt{n}$ as an approximation. Subtracting μ shift the mean to zero, and dividing by σ scales the variable to get a standard deviation of 1. Now we have a relationship between the random variable X and the standard normal distribution. We want to do this in order to use a table over standard normal curve areas like in Appendix 1, and from this calculate the confidence interval. From the table we find that the area between -1.96 and 1.96 of the standard normal distribution is 0.95 and so when first standardizing the sample mean we have the relationship

$$\begin{gathered} P(-z < Z < z) = 0.95 \\ P(-1.96 < Z < 1.96) = 0.95 \\ P(-1.96 < \frac{\bar{X} - \mu}{\sigma/\sqrt{n}} < 1.96) = 0.95 \\ P\left(\bar{X} - 1.96\frac{\sigma}{\sqrt{n}} < \mu < \bar{X} + 1.96\frac{\sigma}{\sqrt{n}}\right) = 0.95 \end{gathered} \tag{39}$$

This equation can be used directly to calculate confidence intervals .
As we can see from this probability, it is not a probability of 95 % of the population mean

to be in a given interval, but rather a probability of calculating a confidence interval from a random sample which will contain the mean. I.e there is not a 95 % probability of finding μ in the interval, but we are 95% confident the interval will contain μ. The reason for this difference is that as soon as you have calculated an interval, the parameter is either contained in the interval or it is not. There is not a probability for it being contained. If however we repeated the experiment an infinite number of times, we would calculate an interval which contains the parameter 95% of the time. This is a small, but to a frequentist statistician important difference. However in practice, for the laymen consumer of statistics the difference is negligible.

4.4 Resampling techniques

Resampling methods are ways of repeatedly drawing samples from a training set and re-fitting a model on each sample. This is done to obtain new information about the model, which would not be available when only fitted once to the original data.

We will in this chapter look at two popular resampling methods, the Bootstrap and the k-fold cross validation.

4.4.1 Bootstrap

There are several types of Bootstraping. The nonparametric, the semiparametric and the parametric are just a few examples. We will only discuss the nonparametric version in this section because this is what is mostly used, and indeed what we have used for generating statistics from our data.

When discussing confidence intervals we used that either the population distribution was normal, or we used the central limit theorem to argue that the sample distribution of the mean would be normally distributed as n goes to infinity.
What can we do if the population is not normal, or at least we do not know if it is, and the sample size is not very big? Or the statistic we want to compute can not be represented by the mean, and hence CLT can not be used?

Suppose we are interested in a confidence interval for the population mean. We draw a sample $S = s_1, s_2, s_3, ..., s_n$ from a population $P = p_1, p_2, p_3, ...p_N$. The size of S is small, say 30, and P is not normally distributed. Because the size of S is small, we can not use CLT to infer that the sampling distribution of the mean will be normally distributed, and because of this we can not in turn create a confidence interval of the sample average $\bar{S}$.
Because we can not assume the sampling distribution is normal, we will have to compute it. If we could redraw from the population this would be easy, but in the real world this is rarely a possibility (say if your datapoints are extremely expensive to create). What we instead do is assume the sample is a good representation of the population. If this assumption is correct and we draw a new sample from our original sample with

replacement, this will also be a good representation of the population. By doing this a large number of times, say 1000, we can now create a sample distribution of the mean, illustrated in figure 3.
Now, if the size of you original sample is sufficiently large(a reasonable number might be n=30) the bootstrap distribution is likely to look close to normal. It might however be slightly skewed, depending on the skewness in your original sample.
If the bootstrap distribution looks reasonably normal, we can use the techniques from the last section to find a confidence interval. In the cases where the bootstrap is clearly not normal however, we need a different technique.
The *bootstrap percentile interval* uses the 2.5 percentile and the 97.5 percentile of the bootstrap as the confidence limits for a confidence level of 95%.
In practice, what we do is sort the bootstrap values of the statistic we are calculating, remove the 2.5% smallest and largest values, and the new endpoints will be to confidence limits of our percentile interval.

Bootstrap algorithm[5]:

- Draw with replacement n values from your sample S of size n
- Define a vector S^* which contain your redrawn values from S
- From S^* compute whatever quantity θ^* you are interested in. E.g the regression coefficients of your model
- Repeat the process k times. Where k must be sufficiently large.
- Calculate statistics on the k bootstrap result. E.g the confidence intervals of the regression coefficients

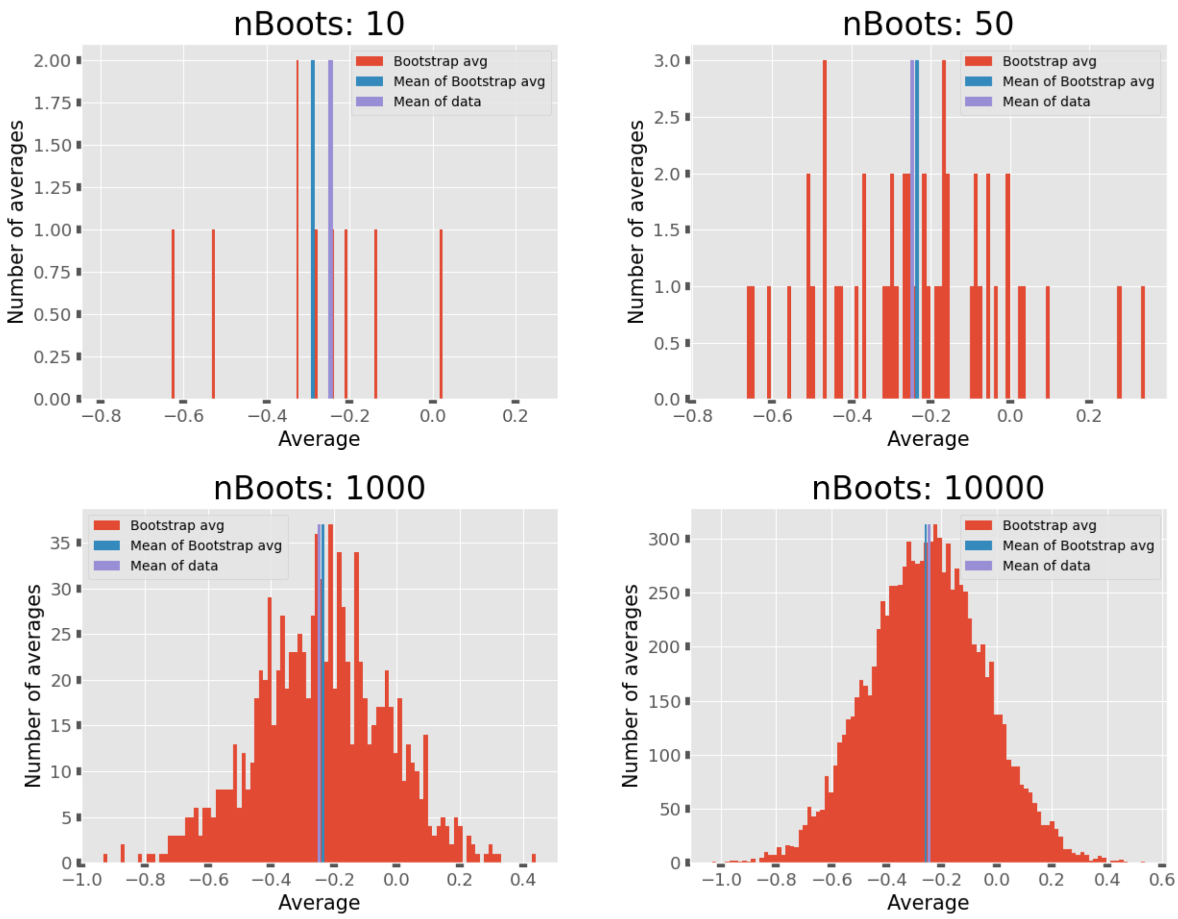


Figure 3: Example of using the bootstrap algorithm on a sample. As we can see, the distribution of the mean looks closer to normal as the number of bootstraps increases. The mean of the bootstrap also converges towards to sample mean. From this bootstrap data we can calculate a confidence interval

4.4.2 k-fold cross validation

In machine learning and regression algorithms there are several parameters which have to be set before the learning or regression process can begin. We call these hyperparameters and examples are the degree of our polynomial fit or the λ parameter in Ridge and Lasso regression. We have to make an educated guess for what are the values for these hyperparameters which will generate a model that generalize well to new data.
A first approach might be to split our data into training data and validation data and train a range of different models with different hyperparameters and choose the model that fits best to the validation data. This is a good way of determining the hyperparameters, however a prerequisite is that you have enough data to split it into batches and still be confident that your batches is a good representation of your data. In many cases the amount of data is not nearly enough to do this confidently, and we try a similar approach but with some added finesse, the *k-fold cross-validation.*

We split the training data into k equally sized parts (folds). A normal number for k is 5 or 10, but it can in principle be as large as your full dataset (leave-one-out cross-validation). We set aside one of the folds to be the validation set, and train our model for a given set of hyperparamters on the remaining k-1 folds, then we test our model on the validation set. We repeat this procedure until all the folds have been validated. We then average the k validation-errors and this is our total estimate for the validation error. This can be repeated for several sets of hyperparamters and we choose the model which gives the best average of the validation-errors.
It is important to remember that we are in practice overfitting our model to the validation set, and the error from the validation will most likely be an overestimate of the true test error.

K-fold cross-validation algorithm:[6]

- Decide on a range of hyperparameters you want to test
- Split your sample S into k mutually exclusive subsets S_i, whose union is S
- Apply model for a given set of hyperparameters on k-1 of the subsets S_i
- Find the error by testing on the remaining fold
- Repeat for every k
- Calculate average error
- repeat for several sets of hyperparameters
- Choose the model with the lowest average error

II Implementation and results

5 Data sets

5.1 Franke's function

The first data set we're going to apply our implementations on is some self-generated data from *Franke's function*, where we pick out random points in the xy-plane, and calculate the corresponding function values $f(x, y)$. Franke's function is a quite complicated function;

$$\begin{aligned} f(x,y) &= \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right). \end{aligned} \tag{40}$$

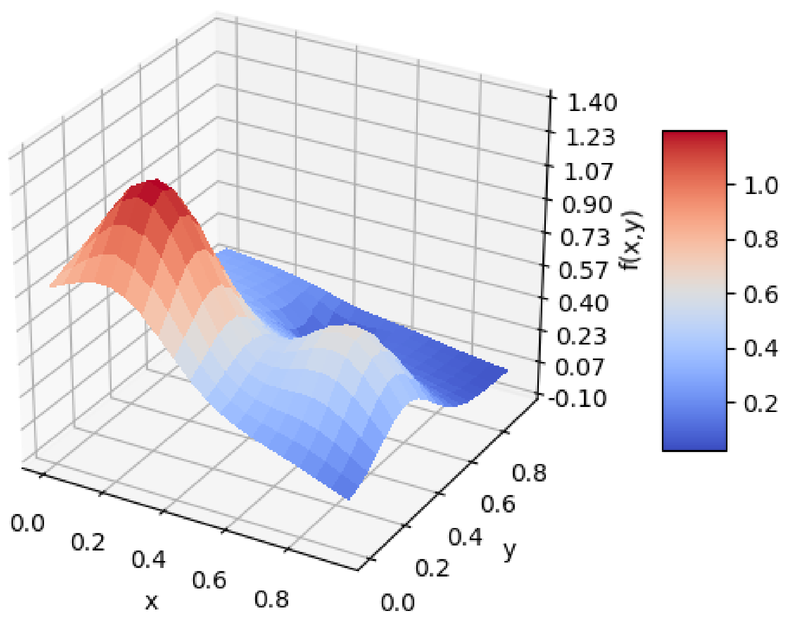


Figure 4: *Surface plot of Franke's function in the region* $x \in [0, 1]$, $y \in [0, 1]$.

consisting of several exponential functions, creating a two gaussian peaks with different heights. A plot of the function in the region we're interested in can be seen in Figure 4. Although the surface of the function is somewhat irregular, it's quite "curvy", implying that it should be possible to find a reasonably good fit by approximating it with polynomials.

5.2 Terrain data

In nature, it is not feasible that fluctuations in the terrain inside a single valley would be a good representation of the fluctuations in the next valley over. These fluctuations are of course quite random due to erosion, local variations in weather, man made structures etc.
The height of mountains and depth of valleys are of course in large random as well, but standing in one valley you might assume that the next valley over are of similar depth, at least if the closest 10 valleys in the other direction are of similar depth. Your prediction might still be extremely wrong though, and in general making prediction of large, irregular areas of terrain are virtually impossible with simple linear regression models. We might not be able to make better predictions than "This terrain has a lot of valleys, then the the terrain next to this one should also have a lot of valleys...possibly".

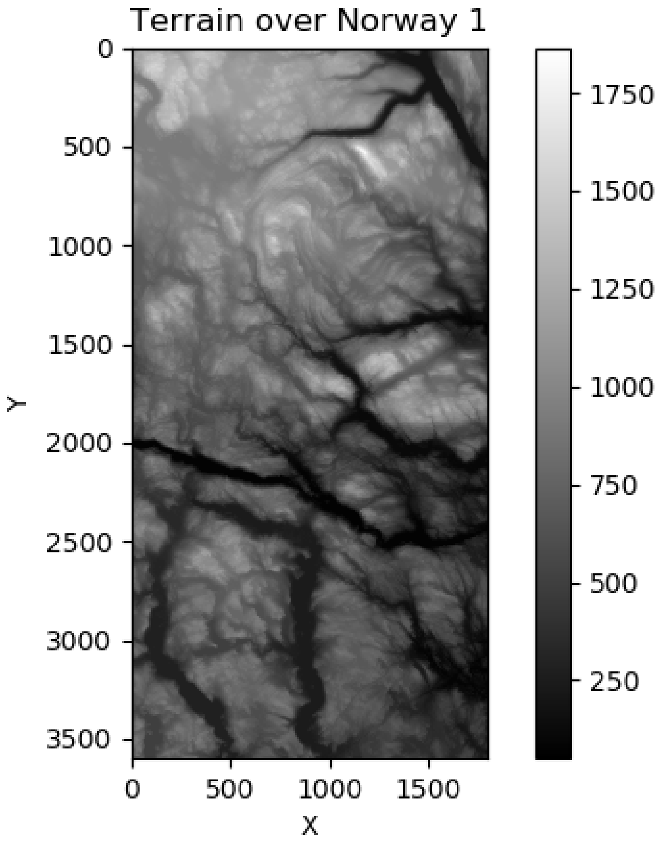


Figure 5: *Plot over the terrain data we are trying to fit. We are randomly selecting patches of 100x100 data points*

In figure 5 we have a look at the data we are trying to fit a polynomial to. It is not possible to see from the image, but in reality the data is extremely irregular and random. The black valleys in the picture might look uniform at zero value, but in reality there are lots of variations in height which are not visible in the image. If we want to have any chance of fitting a polynomial to this data we need to view these irregularities as noise and only try to fit the larger trends of the image. Naturally an algorithm with regularization would be preferred for doing exactly this.
If we look at say the lower 20 % of the image and imagine only fitting a polynomial in the x-direction, we see that by only fitting the largest trends (the three deep valleys), a third/fourth degree polynomial could possibly not be too bad of a fit.
A higher degree polynomial will naturally start fitting more and more irregularities. Because we still want to fit the bigger trends in the data, we need to regularize and punish the higher polynomials, and we would hence expect Ridge and Lasso to make the best fits.

Because the nature of our data is so random we would expect the best model to be one which finds the best compromise between fitting small and large fluctuation in the dataset.

6 Results

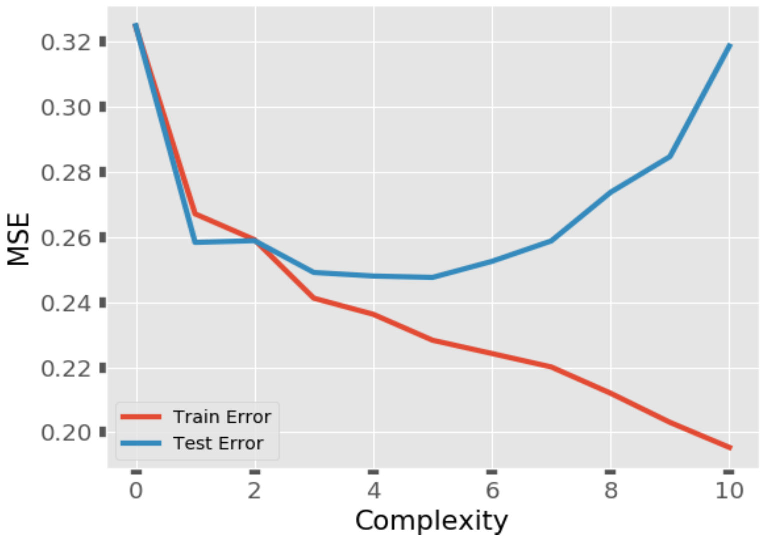
6.1 Estimates on Franke's Function

6.1.1 OLS

From the theory discussed about the OLS we know that this method is prone to overfitting when the model complexity increases, and it was also shown that the variance should decrease and bias increase in this case. We ran a simulation on Franke's function to test this, and the results can be seen in Figure 6. From the top plot we see that the MSE for the training data keeps falling as we increase the polynomial degree, while the test MSE falls at first, but starts to increase when a certain complexity was reached. This was an expected result, which shows that the OLS model starts to overfit to the noise in the data, leading to a worse prediction accuracy on the test set.

In the bottom plot of the same figure we see the same test error decomposed into bias and variance terms. Here we see that the bias and variance add up to the test error quite well, confirming the theory. There are some thing to note about this result however. In section 4.2 we found that the MSE of a test set can be written in terms of one bias term, and two variance terms, where one of them is the irreducible error created by the noise. The variance of the error in this case is $\sigma^2 = 0.25$, and if we subtract this from the bias, we see that it has it's minimum close to 0. The reason the bias is shifted up with σ^2 is that we calculate the bias through the noisy data y, and not the underlying function f. It's also worth noting that, although not very prominent in this plot, the bias starts to rise again for higher complexities. Had we plotted for even higher degrees, we would have seen that it would explode together with the MSE and variance, especially if we lowered the sample size. This effect is most likely due to two things, a finite sample size, and a finite number of bootstrap estimates. The fluctuations of the estimates for each $\tilde{y}$ will start to fluctuate enormously as the complexity grows high (reflected in a rising variance), and as the bias is defined as the average of all estimates (in our case all bootstrap estimate), the chance for it to hit exactly the value it's supposed to diminishes with complexity. Increasing the sample size and number of resamplings fixes this up to a certain polynomial degree, but the bias will eventually start to rise again for some complexity.

Further we can calculate the confidence intervals of the parameters $\hat{\beta}$. As a small test the theory we have derived for confidence intervals and bias-variance tradeoff, we have in Figure 7 plotted the computed confidence intervals (95%) (using equation (9)) for each β in the polynomial approximations, for different polynomial degrees and strengths of noise. In this experiment we have used $N = 1000$ random (x, y) points the interval $x, y \in [0, 1]$,



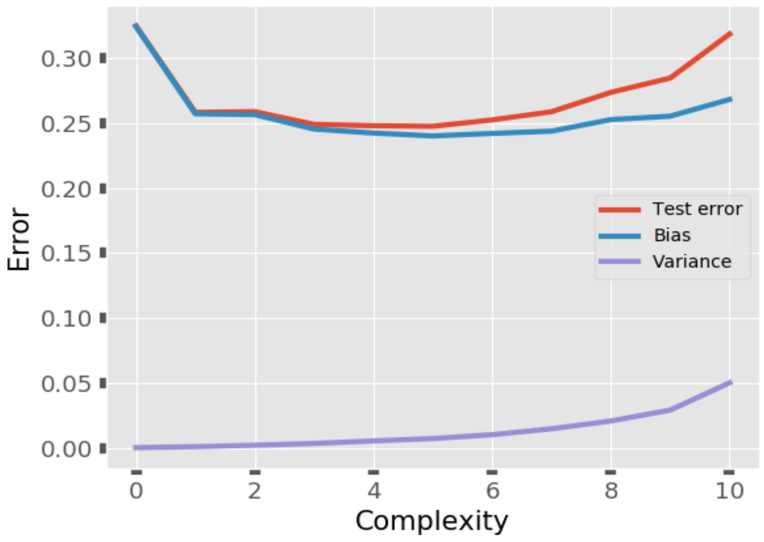


Figure 6: *OLS estimations of the MSE as function of model complexity run on Franke's function with noise $\sigma = 0.5$. Top: The MSE for both the training set and test set. Bottom: Bias-Variance decomposition of the test error.*

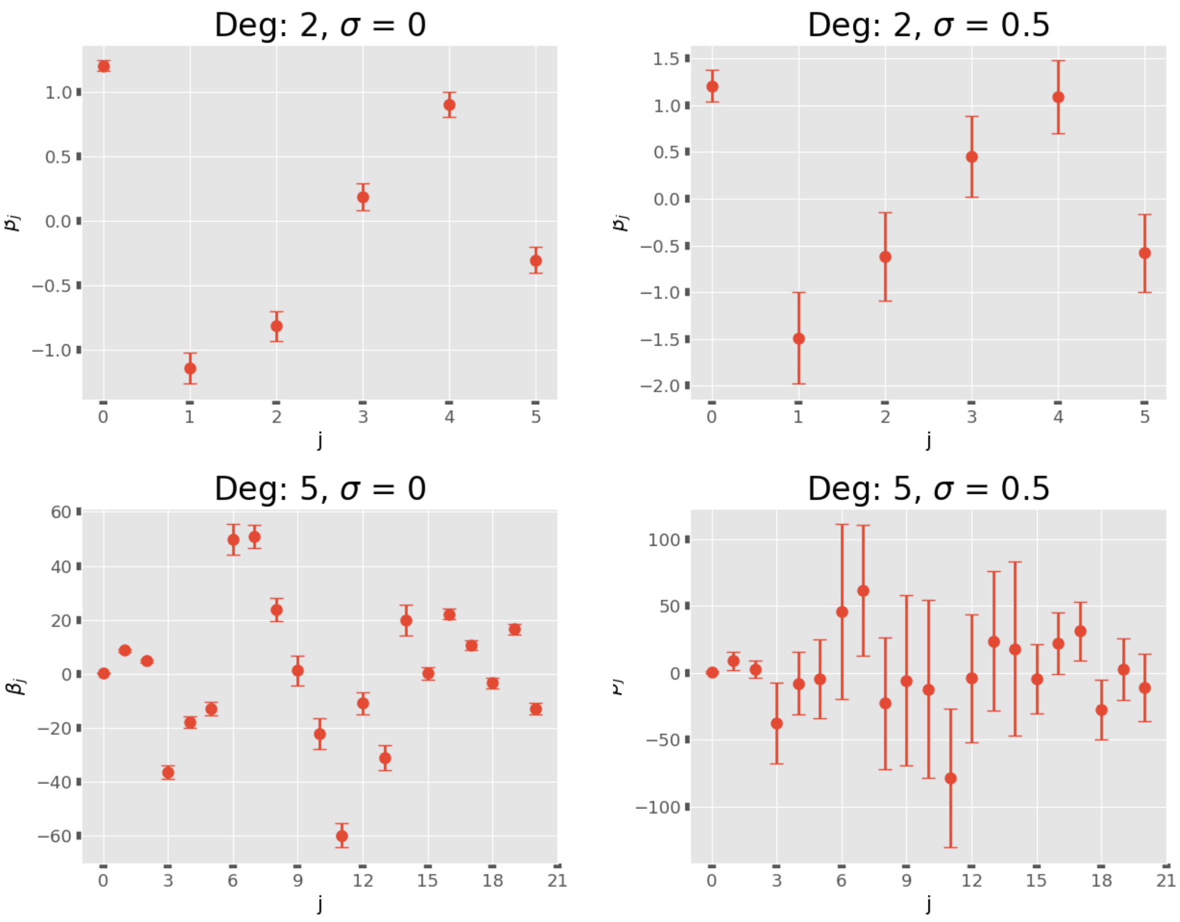


Figure 7: *The confidence intervals for the βs in OLS regression for different degrees and strengths of noise. Top: Franke's Function with no noise (left) and $\sigma = 0.5$ (right) approximated with a 2nd order polynomial. Bottom: Same noise, but now approximated with a 5th order polynomial.*

so that the sample is able to catch at least *some* of the features in the shape of the Franke function, but still leaving room for some variance. The MSE on a separate test set for each case was found to be (corresponding to the models in the figure):

$$MSE = \begin{bmatrix} 0.0225 & 0.2939 \\ 0.0029 & 0.2572 \end{bmatrix}$$

From the figure, we see that the confidence intervals (hence also variances) are relatively small for the 2nd order polynomial, even with significant noise. For 5th degree, we see that we relatively small variances for no noise, but quite large for the noisy case. Noe comparing with the computed MSEs, we see that the more complex model fits much better when there's no noise, but performs on more or less the same level (actually a bit *worse*) when noise is present. From what we know from the bias-variance discussion, a complex model should have high variance but low bias, with the opposite going for a less complex model. We see that this relationship is confirmed in this case, where the MSE is close to the same, but the variance in the complex model is much larger. If we increase the number of data points the variances/confidence intervals will indeed go down, but the relationship shown in Figure 7 will still hold.

6.1.2 Ridge & Lasso

As the effects of ridge and lasso regularization is quite similar and easily comparable, we will list and compare some results for both method in this section.

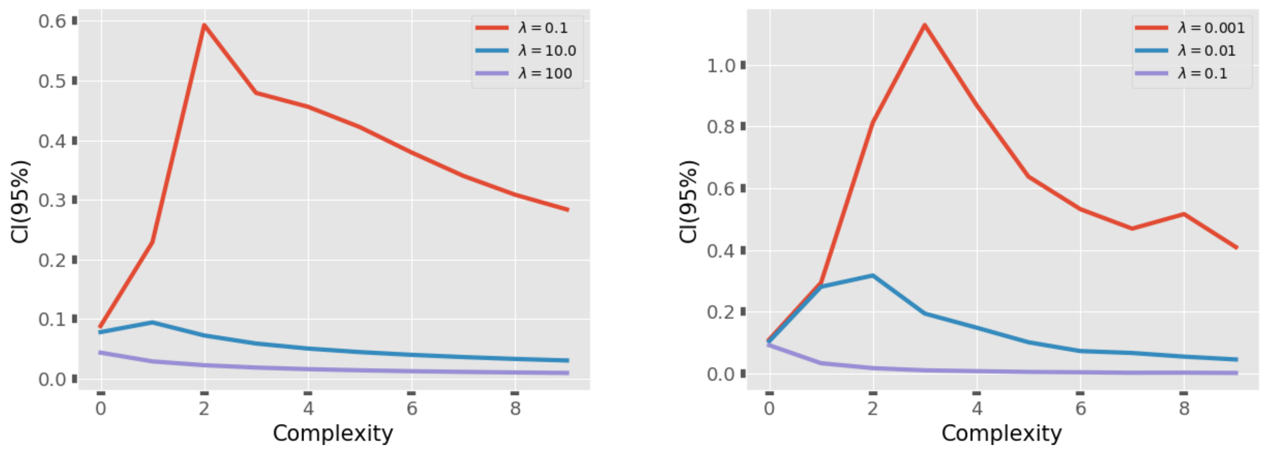


Figure 8: *Average of confidence intervals of βs as function of complexity for different regularization strengths λ.*

	β_0	β_1	β_2	β_3	β_4	β_5	β_6	β_7	β_8	β_9
OLS	1.124	-1.431	2.174	-0.295	0.761	-7.423	0.536	0.651	-0.540	4.557
Ridge	1.139	-1.042	-0.097	-0.343	0.163	-0.906	0.393	0.763	-0.239	0.060
Lasso	1.164	-1.228	0.	0.	0.	-0.988	0.209	0.749	-0.	0.

Table 1: *Table reflecting the results in Figure 8, ridge regularized with* $\lambda = 0.1$, *and lasso with* $\lambda = 0.001$.

What's perhaps most interesting about the ridge and lasso methods is their effects on the regression parameters $\hat{\beta}$. Figure 8 shows how the regularization affects the confidence intervals of the betas (here the average of them for visualization). First of all we see there is a difference in the sensitivity on the λ values for the methods, where a much smaller λ is needed for the lasso method to heavily impact the β values. We see that their effect in the confidence intervals (average) is the same, lower allows for larger βs. As the complexity of the models increase, there are introduced more betas which will have to "share" the magnitudes, hence lowering the average. In Table 1 we see the values for each β in detail, and how they change from the betas found in a OLS regression on the same data set as they are regularized by the ridge and lasso methods. We see that ridge penalizes the larger heavily, almost driving them down to 0, while the less impacting ones are left at approximately the same values. For the lasso we see that many of the βs are set to 0, leaving only the few it has found to describe the data best. These results fit well with the theory that has been discussed in previous sections.

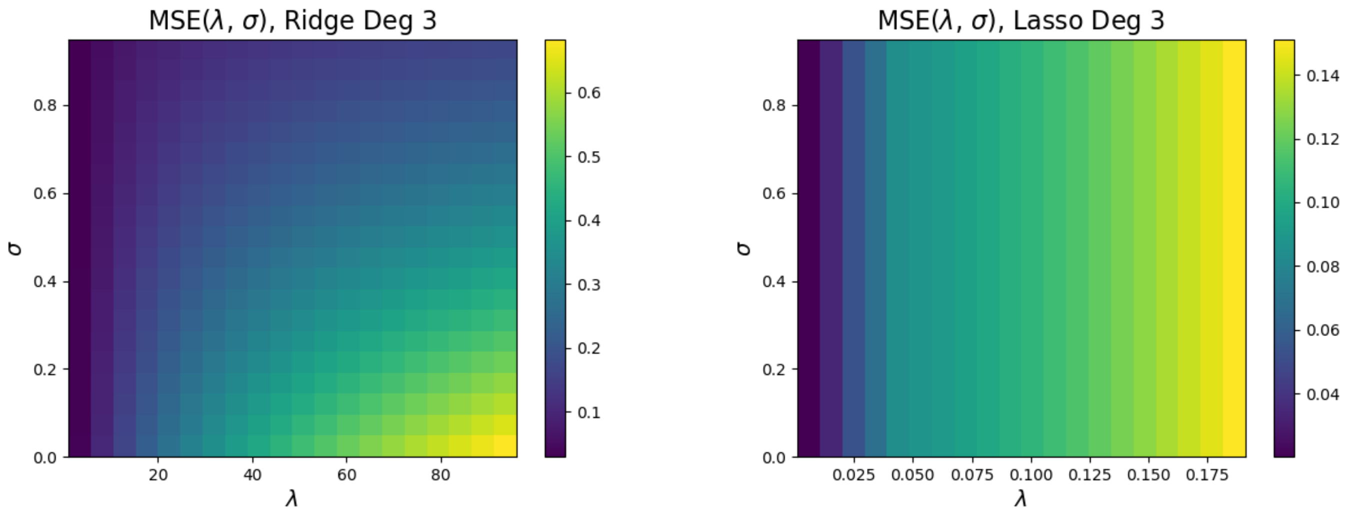


Figure 9: *Heat maps for how Ridge and Lasso treats noisy data with different regularization strengths. Degree 3 is a random choice of a polynomial.*

Further, it's interesting to explore how the regularization reacts to noise in the data. Figure 10 shows a heat map of the computed MSE for a grid of different amounts of noise

σ, and regularization strengths λ. We expect them to act somewhat similarly, and that's exactly what we see, but some interesting observations can be made. The left plot shows that the ridge is most wrong when there is a large penalty on less noisy data. This makes sense, as little noise makes the data more "fitable", and heavily penalizing these βs thus gives an awful fit. We see some of the same behaviour for lasso, but here we observe something interesting. The lasso gives approximately the same error for the same λ, no matter the strength of the noise. This implies that the lasso picks out the same βs as the most significant ones each time, setting the rest to 0. This means that it's able to pick out the important features of the data, although the overall MSE might suffer some. This also coincides with the theory developed.

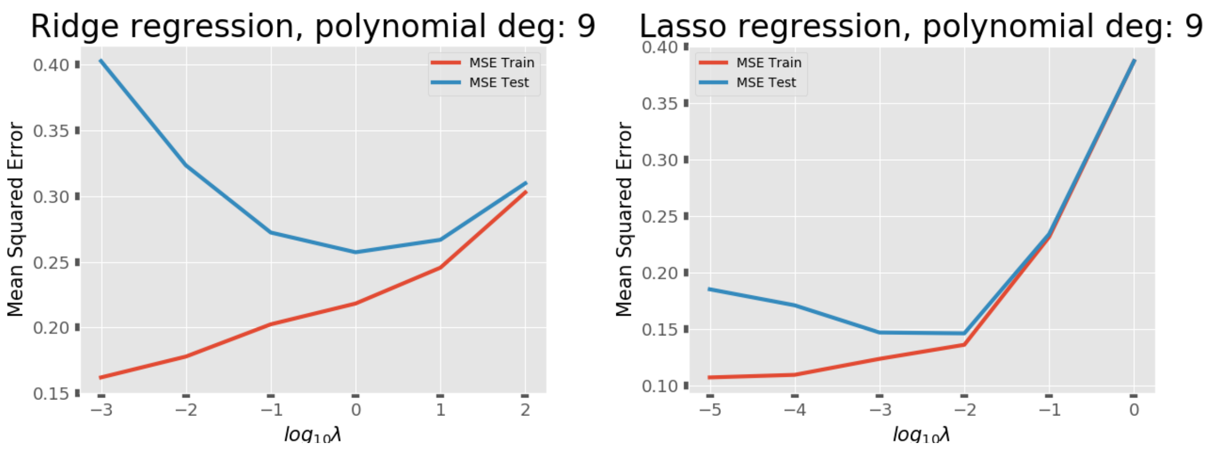


Figure 10: *Training and test error as function of regularization strength λ for a polynomial approximation of degree 9 for both the ridge and lasso methods. Approximated on quite noisy Franke data. As λ increases, variance decreases, making the test and training errors more and more equal.*

We can also investigate what difference regularization makes on the training and test sets. In Figure 10 we see an experiment where we have measured their performance in terms of the MSE for both test and training sets as a function of λ. We chose degree 9 simply to showcase their behaviours, as we found that the only difference of these plots for lower degrees were that the MSEs had smaller gaps between them. In the figure we see that the MSEs for the ridge method slowly coincide, but don't actually meet, not even for a strong regularization like $\lambda = 100$. The Lasso however seems to equalize the MSEs rather fast, where the difference between them are really close to 0. A regularized models goal is to find the underlying structure in the data, and we can see that the methods indeed to that, as the βs are generalized to fit equally on the training and test set. This makes the general fit of the noise data suffer a little as the MSEs go up, but it's the MSE on the test set that really counts. For this specific data set, we see that regualrization of about 1.0 is optimal for ridge, and 0.01 for lasso, as it's for these values that the test error is minimized.

6.1.3 Best model analysis

When it comes to trying to pinpoint *the best* model for a given data set, there are quite some considerations to make. One usually look for a model with a relatively low prediction error on the test sets, as well as having a good trade-off between bias and variance. As we'll see, we're likely to get many models that perform well, some better on certain aspects, and worse on others. It's therefore preferable to consider several models, and not disregard a model completely even if it performs worse in say R2-score than others.

In addition to evaluating different models of OLS, Ridge and Lasso separately, we have also written a program that runs estimations for all three methods on the same data set, compares them, and picks out the best ones. This is done for different polynomial degrees and λs, where each model is evaluated on the basis of average MSE on the test/validation sets through a 10-fold cross-validation. Then the 10 best models are picked out, with a bootstrap resampling being performed on each to determine a better estimate of the bias and variance of each model. At last the models are presented in a print-out table containing all relevant statistics. There are more parameters that could be adjusted to alter the results, like the sample size and strength of the noise, but here we will put en emphasis on the former. The general rule "more is better" in terms of the number of data points gives indeed a better fit, but the results might be a bit surprising. The results for two simulations with different sample sizes, one large and one small, are shown in Tables 2 and 3. Note that the same amount of noise is used in both cases.

	Method	Degree	λ	MSE Test	R2 Test	Bias	Variance	MSE Boot
1.	Ols	9	0	0.0913847	0.467661	0.089971	0.000744	0.090716
2.	Ols	8	0	0.0915293	0.466813	0.089953	0.000614	0.090567
3.	Ols	7	0	0.0916811	0.465979	0.090026	0.000498	0.090524
4.	Ols	6	0	0.0920998	0.463517	0.090329	0.000372	0.090701
5.	Ridge	9	0.001	0.0923693	0.461970	0.090839	0.000318	0.091157
6.	Ridge	8	0.001	0.092479	0.461335	0.091069	0.000305	0.091374
7.	Ridge	7	0.001	0.0927493	0.459767	0.091480	0.000293	0.091772
8.	Ridge	6	0.001	0.0930685	0.457903	0.091859	0.000257	0.092116
9.	Ols	5	0	0.0930884	0.457819	0.091812	0.000277	0.092089
10.	Ridge	9	0.01	0.0931289	0.457506	0.091926	0.000275	0.092201

Table 2: *The 10 best models based on MSE on the test sets during cross-validation. Evaluated on a data set of size $N = 10000$, and noise $\sigma = 0.3$. The columns "MSE Test" and "R2 Test" are evaluated during CV, while "Bias", "Variance" and "MSE Boot" are found during bootstrapping. OLS is well represented with such a large sample size.*

In Table 2 we see something that might seem strange: the OLS outperforms both the ridge and lasso methods in terms of the MSE on the test sets (in both cross-validation and bootstrap), even if there is noise of strength $\sigma = 0.3$ present. This seems counter-intuitive, as the OLS models are expected to overfit and miss with their predictions more and more as the polynomial degree increases. What we see is the exact opposite - the OLS models with the highest degrees perform better. We also see some ridge models following close behind, but that can simply be explained by the fact that these ridge models are the models that are the *most similar* models to the mentioned OLS models, as $\lambda = 0.001$ is the lowest value we tested for.

This strange result can be explained by the sample size. With a large set of data points concentrated on one small area ($x, y \in [0, 1]$ in this case), the distance between the points in the test set and the training set is likely to be very small when sampled randomly. The samples within the training set will also be very close, meaning that the models we have used aren't complex enough to overfit to the densely packed data so that it impacts the test data in between, leading to a decently generalized fit *within* the current region. Extrapolating outside this region is though very likely to yield awful results.

Although the OLS methods performed the best in terms of MSE, it doesn't mean that they are the models we should choose as the best ones. We see that all models listed perform at more or less the same level, with small deviations in certain quantities. One thing to note is that the variance of the complex OLS models are larger than the complex ridge models, and one is usually happy to sacrifice *some* MSE and bias to lower the variance, thus increasing the stability of the model. This means that we probably are better off by picking a complex Ridge model in this case, although the differences are quite small.

	Method	Degree	λ	MSE Test	R2 Test	Bias	Variance	MSE Boot
1.	Lasso	3	0.0001	0.0921372	0.169103	0.077055	0.027374	0.104429
2.	Lasso	5	0.001	0.0930235	0.148263	0.068061	0.015548	0.083609
3.	Ridge	3	0.01	0.0930892	0.161851	0.071416	0.015847	0.087263
4.	Lasso	6	0.001	0.0934351	0.146241	0.074954	0.017307	0.092261
5.	Ridge	4	0.01	0.0936102	0.166783	0.080088	0.021135	0.101223
6.	Lasso	4	0.001	0.0936251	0.139539	0.065541	0.013456	0.078997
7.	Ridge	3	0.001	0.0936679	0.162499	0.078216	0.031508	0.109723
8.	Lasso	7	0.001	0.0938302	0.145882	0.079208	0.022036	0.101244
9.	Ridge	5	0.1	0.0939318	0.159822	0.069666	0.016860	0.086525
10.	Lasso	8	0.001	0.0940999	0.144658	0.081713	0.023269	0.104982

Table 3: *The 10 best models based on MSE on the test sets during cross-validation. Evaluated on a data set of size $N = 100$, and noise $\sigma = 0.3$. The columns "MSE Test" and "R2 Test" are evaluated during CV, while "Bias", "Variance" and "MSE Boot" are found during bootstrapping. With a sparser data set, the ridge and lasso models now perform better than the OLS.*

Table 3 however contains the results of the exact same simulations, but now with a much smaller sample size. Here we see that the OLS models are completely absent, leaving only ridge and lasso models as candidates. One thing to note here is that the ”top” model performs quite clearly better during the CV process than the others, it has rather high bias and variance (hence also MSE of the bootstrap) compared to some of the models. If we look at entry no. 6 instead, we observe that this model has the lowest bias, variance and bootstrap MSE of all, while only performing slightly worse during CV. Since we're dealing with a small data set, we're more trustful of the bootstrapped measurements, and would thus pick this model over entry no. 1. This shows the importance of considering several qualities of the models when deciding on the ultimate pick for *the best one.*

6.2 Estimates on terrain data

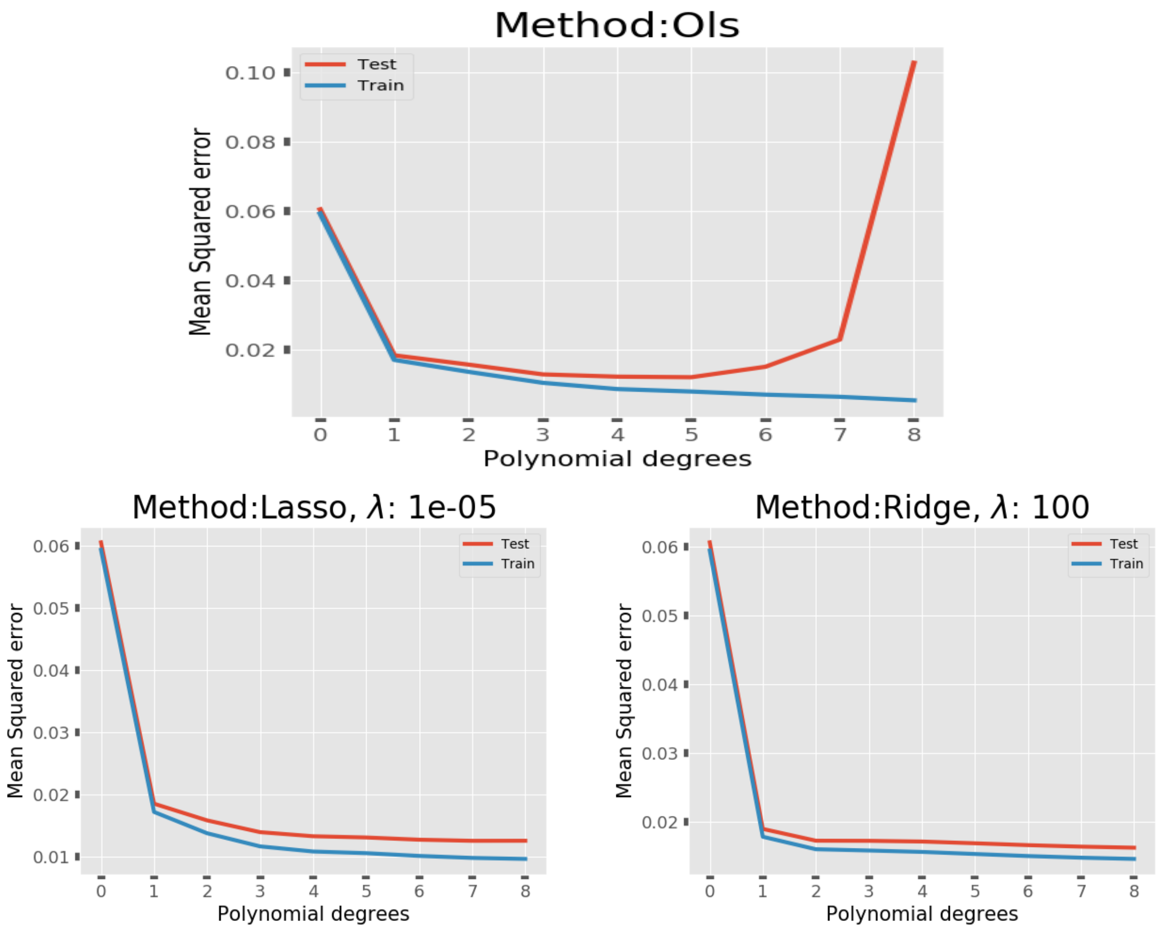


Figure 11: *Mean squared error as a function of polynomial degrees for Ordinary least squares, Lasso and Ridge regression. Simulated on 5 random patches of real terrain data of size 100x100 using 5-fold cross validation*

All the mean squared errors on the terrain have been scaled by the highest terrain value in the patches chosen.

Figure 11 shows the mean squared error for all three methods as function of complexity in the model. The same calculations were also run for different λ values, however they all showed the same trends and the two shown here are representative.
The distinct difference in the plots are between the Ordinary least square model and the models using regularization. From the discussion of the terrain data in section 5.2 we know that if we want to make the model remotely usable, we need to regard the smaller fluctuations in the data as noise. We know from the discussion on OLS, that this method quickly will adapt to noise and overfit the data. This is also the trend we see here.
For Ridge and Lasso we see that they both fit the data quite well, but that Ridge might

perform a bit better. More on this in section 6.2.1

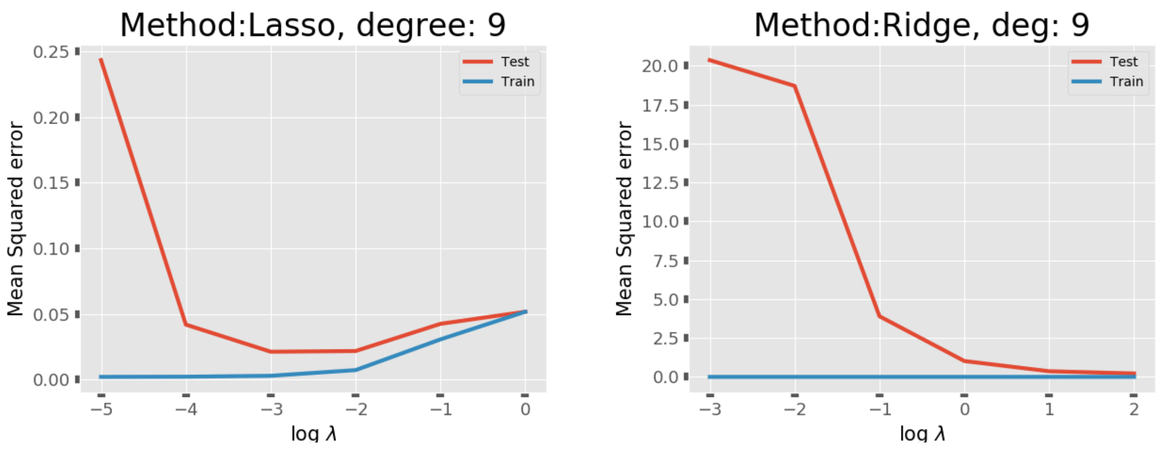


Figure 12: *Training and test error as function of regularization strength λ for a polynomial approximation of degree 9 for both the ridge and lasso methods. Approximated on 5 random patches of real terrain data of size 100x100 data points using 5-fold cross validation. As λ increases, variance decreases, making the test and training errors more and more equal.*

6.2.1 Best model analysis

As discussed in section 5.2, we can not expect the terrain to follow any underlying function or distribution. Because of this the best model will most likely be the one who finds the best compromise between fitting the big and small fluctuations in the data. We already know from figure 11 that OLS will not be a good fit because it adapts to the small fluctuations in data far to quickly.

From our discussion of Lasso regression we know that Lasso is unforgiving and pushes the insignificant beta values quickly to zero, hence Lasso is likely to only find the "big picture" in our data set and look past all the noise. This might be beneficial if we're trying to fit large patches of the terrain, where all the smaller fluctuations don't really have much impact on the overall shape that we're trying to fit. When looking at terrain data we might not want to remove all the "noise" as it is not really noise, but actual features of the data set, although on smaller scales.

Ridge punishes the small fluctuation as well, but it does not necessarily set any of the coefficients of higher polynomials to zero, hence not removing the smaller features completely. We may expect Ridge to fit data well on smaller patches of the data, finding finding the "Goldie locks zone" of fluctuations we are looking for.

In Table 4 we have the 10 best fits for degrees up to 15 of all the models, with Ridge and Lasso tested for several λ values. As we suspected in our discussion, Ridge completely dominates the table. As a general trend, we also see that lower complexity with a soft regularization is preferred over increasing complexity and using a tougher regularization.

As we can see, the MSE for the k-fold test and for bootstrap are very small, as well as reasonable biases and variances, so all the models could be considered reasonable choices of a final model.

As a side note, the fact that we choose random patches from the data for reducing the number of data points is not necessarily the best approach, as the patches we choose might look very similar to each other or be vastly different. In this case we have showcased we obviously got quite lucky, since R2-scores around 0.8 are signs of very good fits. A better approach might be to just choose a corner of the data set and use this as the full data set, and from this small patch we can choose randomly what should be regarded as training and test set. This approach will not help generalize back to the full data set, but we might be able to make predictions on a test set inside the patch we chose.

	Method	Degree	λ	MSE Test	R2 Test	Bias	Variance	MSE Boot
1.	Ridge	8	0.001	0.00899399	0.816390	0.009265	0.000017	0.009282
2.	Ridge	9	0.001	0.00901746	0.816436	0.009144	0.000020	0.009164
3.	Ridge	11	0.01	0.00901943	0.815164	0.009349	0.000018	0.009367
4.	Ridge	10	0.01	0.00908196	0.813440	0.009358	0.000018	0.009375
5.	Ridge	12	0.01	0.00909741	0.813921	0.009510	0.000019	0.009529
6.	Ridge	12	1	0.00923309	0.808968	0.010418	0.000012	0.010430
7.	Ridge	13	1	0.00924227	0.808800	0.010356	0.000012	0.010368
8.	Ridge	9	0.01	0.00924324	0.809612	0.009512	0.000014	0.009526
9.	Ridge	11	1	0.00926006	0.808454	0.010630	0.000010	0.010640
10.	Ridge	7	0.001	0.00926544	0.809752	0.009593	0.000016	0.009608

Table 4: *The 5 best models based on MSE on the test sets during 5-fold cross-validation. Evaluated on a data set of 5 patches randomly chosen from the terrain data, each of size* $N = 100x100$*. The columns "MSE Test" and "R2 Test" are evaluated during CV, while "Bias", "Variance" and "MSE Boot" are found during bootstrapping.*

III Conclusions and future work

7 Summary and conclusion

In this project we have studied the linear regression models Ordinary Least Squares, Lasso and Ridge when applied to the Franke function, and compared them by using the Mean Squared Error and the R2 score function. In addition we have used k-fold cross-validation and bootstraping to resample the data and generate statistics. Finally we applied all of this to real terrain data.

We saw from the our evaluation of the regression models on Franke's function that what model is the best depends on a lot of factors. If there is noise in the data, we need regularization to try to see through this noise. This is particularly true if the number of data points are few and spread out, which gives the ordinary least squares algorithm free room to fluctuate away from the true underlying function. How much regularization is needed and what complexity is the best is not something we can decide in advance, but rather something we have to have to give an educated guess through the means of e.g. k-fold cross-validation.

The best model chosen by the cross-validation might not be the best result however. We might want to have an additional look at the results from the Bootstraping bias/variance trade-off, as the best model from the cross-validation might have a bias or variance we are not happy with.

When evaluated on small patches of real data, we saw that Ridge regression was by far the superior choice as it seemed to generalize just the right amount to fit the randomness of data. The analysis process was done on relatively small areas of the total data set, as the regression on larger areas was quite time consuming, especially for the lasso method. We picked out a few patches of the data as training data, and one or two as test data. These were randomly picked, so there was no guarantee that the training and test sets looked alike. We ran a few experiments however, generating similar statistics to those in Table 4, all showing that ridge models with a few different complexities and regularization strengths were the preferred models. This conclusion is in no way set in stone, as we would need to run more experiments on larger parts of the data set, but this demands more computational power and patience to wait for the simulations to finish. The code we wrote could always be improved in terms of efficiency, and the introduction of e.g. parallelization would likely be helping a lot. We feel however that we got a small jist of how the regression methods perform on these types of data.

8 Prospects and future work

A possible solution for making better predictions using the same techniques could be to preprocess our data by using a smoothing filter like a uniform filter, locally averaging the values within patches of the data [8]. This would help remove the smallest fluctuations in the data set which we are not interested in anyways.

A big draw back of our analysis was that we were not able to use enough training data to confidently represent the entire terrain due to computational limitations. A way of reducing the amount of data could be to use max or average pooling, where we're choosing the biggest values/average values in local patches as a representation of that local patch. This would also be an artificial way of removing some noise in the data, effectively helping out our regression methods in that sense. We would naturally loose a lot of information about the sample data by doing this (depending on the sizes of the pooling patches), but if all data sets are preprocessed this way we might end up with a model which generalizes a lot better for larger regions of the terrain.

The obviously most efficient model for representing a pictures of terrain data is using a convolutional deep neural network, where every layer looks for distinct features like edges or corners. This is however beyond the scope of this project, but will perhaps be the topic of a future project.

IV Appendix

A.1 Z-table

z	0	0,01	0,02	0,03	0,04	0,05	0,06	0,07	0,08	0,09
0	0,5	0,50399	0,50798	0,51197	0,51595	0,51994	0,52392	0,5279	0,53188	0,53586
0,1	0,53983	0,5438	0,54776	0,55172	0,55567	0,55962	0,56356	0,56749	0,57142	0,57535
0,2	0,57926	0,58317	0,58706	0,59095	0,59483	0,59871	0,60257	0,60642	0,61026	0,61409
0,3	0,61791	0,62172	0,62552	0,6293	0,63307	0,63683	0,64058	0,64431	0,64803	0,65173
0,4	0,65542	0,6591	0,66276	0,6664	0,67003	0,67364	0,67724	0,68082	0,68439	0,68793
0,5	0,69146	0,69497	0,69847	0,70194	0,7054	0,70884	0,71226	0,71566	0,71904	0,7224
0,6	0,72575	0,72907	0,73237	0,73565	0,73891	0,74215	0,74537	0,74857	0,75175	0,7549
0,7	0,75804	0,76115	0,76424	0,7673	0,77035	0,77337	0,77637	0,77935	0,7823	0,78524
0,8	0,78814	0,79103	0,79389	0,79673	0,79955	0,80234	0,80511	0,80785	0,81057	0,81327
0,9	0,81594	0,81859	0,82121	0,82381	0,82639	0,82894	0,83147	0,83398	0,83646	0,83891
1	0,84134	0,84375	0,84614	0,84849	0,85083	0,85314	0,85543	0,85769	0,85993	0,86214
1,1	0,86433	0,8665	0,86864	0,87076	0,87286	0,87493	0,87698	0,879	0,881	0,88298
1,2	0,88493	0,88686	0,88877	0,89065	0,89251	0,89435	0,89617	0,89796	0,89973	0,90147
1,3	0,9032	0,9049	0,90658	0,90824	0,90988	0,91149	0,91309	0,91466	0,91621	0,91774
1,4	0,91924	0,92073	0,9222	0,92364	0,92507	0,92647	0,92785	0,92922	0,93056	0,93189
1,5	0,93319	0,93448	0,93574	0,93699	0,93822	0,93943	0,94062	0,94179	0,94295	0,94408
1,6	0,9452	0,9463	0,94738	0,94845	0,9495	0,95053	0,95154	0,95254	0,95352	0,95449
1,7	0,95543	0,95637	0,95728	0,95818	0,95907	0,95994	0,9608	0,96164	0,96246	0,96327
1,8	0,96407	0,96485	0,96562	0,96638	0,96712	0,96784	0,96856	0,96926	0,96995	0,97062
1,9	0,97128	0,97193	0,97257	0,9732	0,97381	0,97441	0,975	0,97558	0,97615	0,9767
2	0,97725	0,97778	0,97831	0,97882	0,97932	0,97982	0,9803	0,98077	0,98124	0,98169
2,1	0,98214	0,98257	0,983	0,98341	0,98382	0,98422	0,98461	0,985	0,98537	0,98574
2,2	0,9861	0,98645	0,98679	0,98713	0,98745	0,98778	0,98809	0,9884	0,9887	0,98899
2,3	0,98928	0,98956	0,98983	0,9901	0,99036	0,99061	0,99086	0,99111	0,99134	0,99158
2,4	0,9918	0,99202	0,99224	0,99245	0,99266	0,99286	0,99305	0,99324	0,99343	0,99361
2,5	0,99379	0,99396	0,99413	0,9943	0,99446	0,99461	0,99477	0,99492	0,99506	0,9952
2,6	0,99534	0,99547	0,9956	0,99573	0,99585	0,99598	0,99609	0,99621	0,99632	0,99643
2,7	0,99653	0,99664	0,99674	0,99683	0,99693	0,99702	0,99711	0,9972	0,99728	0,99736
2,8	0,99744	0,99752	0,9976	0,99767	0,99774	0,99781	0,99788	0,99795	0,99801	0,99807
2,9	0,99813	0,99819	0,99825	0,99831	0,99836	0,99841	0,99846	0,99851	0,99856	0,99861
3	0,99865	0,99869	0,99874	0,99878	0,99882	0,99886	0,99889	0,99893	0,99896	0,999
3,1	0,99903	0,99906	0,9991	0,99913	0,99916	0,99918	0,99921	0,99924	0,99926	0,99929
3,2	0,99931	0,99934	0,99936	0,99938	0,9994	0,99942	0,99944	0,99946	0,99948	0,9995
3,3	0,99952	0,99953	0,99955	0,99957	0,99958	0,9996	0,99961	0,99962	0,99964	0,99965
3,4	0,99966	0,99968	0,99969	0,9997	0,99971	0,99972	0,99973	0,99974	0,99975	0,99976
3,5	0,99977	0,99978	0,99978	0,99979	0,9998	0,99981	0,99981	0,99982	0,99983	0,99983
3,6	0,99984	0,99985	0,99985	0,99986	0,99986	0,99987	0,99987	0,99988	0,99988	0,99989
3,7	0,99989	0,9999	0,9999	0,9999	0,99991	0,99991	0,99992	0,99992	0,99992	0,99992
3,8	0,99993	0,99993	0,99993	0,99994	0,99994	0,99994	0,99994	0,99995	0,99995	0,99995
3,9	0,99995	0,99995	0,99996	0,99996	0,99996	0,99996	0,99996	0,99996	0,99997	0,99997
4	0,99997	0,99997	0,99997	0,99997	0,99997	0,99997	0,99998	0,99998	0,99998	0,99998

V References

References

[1] Stephen Marsland, *LaTeX: a document preparation system*, Machine Learning An Algorithmic Perspective, 2nd edition, 2015., chapter 1, p. 215.

[2] Akaike, H. (1974). A new look at the statistical model identification. IEEE Transactions on Automatic Control,19 (6), 716–723

[3] Cullen, H. F. ”The Stone-Weierstrass Theorem” and ”The Complex Stone-Weierstrass Theorem.” In Introduction to General Topology. Boston, MA: Heath, pp. 286-293, 1968.

[4] Wessel N. van Wieringen ”Lecture notes on ridge regression”, 2018. Source: https://arxiv.org/pdf/1509.09169.pdf

[5] Morten Hjort-Jenssen, Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis, Department of Physics, University of Oslo [2] Department of Physics and Astronomy and National Superconducting Cyclotron Laboratory, Michigan State, sept 25 2018. Source:Regression.html

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, Massechusetts Institute of Technology, 2016

[7] Scott Fortmann-Row, ”Understanding The Bias-Variance Tradeoff”, June 2012. Source:http://scott.fortmann-roe.com/docs/BiasVariance.html

[8] No visible author, http://www.mif.vu.lt/atpazinimas/dip/FIP/fip-Smoothin.html

[9] Franke, R. (1979). A critical comparison of some methods for interpolation of scattered data (No. NPS53-79-003). NAVAL POSTGRADUATE SCHOOL MONTEREY CA.

[10] Jay L. Devore, Kenneth N. Berk, ”Modern Mathematical Statistics with Applications, second edition, 2012, chapter 6.2, p. 298