Project 1 - FYS-STK4155

Markus Borud Pettersen

October 10, 2020

Abstract

We implement ordinary least squares-, Ridge-, and Lasso regression algorithms in order to explore some common machine learning concepts, including overfitting, and the bias variance tradeoff. We do this by first applying our algorithms to a dataset created using the Franke function, with and without added normally distributed noise. We show that the amount of noise added was too great, as models trained on the noisy dataset had an R^2-score of almost zero. We do however find that models trained on this dataset prove useful, as a demonstration of model overfitting. By running parameter searches, we determined that a Ridge regression model performed the best on the noise-free Franke function test dataset, achieving a bootstrapped mean squared error of approximately $3 \cdot 10^{-5}$. We also find that models with both low bias and low variance perform better. In addition, we trained models and performed parameter searches on a downsampled, real world terrain elevation dataset. For this dataset, a Ridge regression model also performed best, achieving a 10-fold crossvalidation mean squared error of approximately 0.007. For terrain data, we also found that low-bias models tended to performed better, as the error was dominated by the bias.

Introduction

While they are among the simplest methods we can imagine, linear models can be used to describe a wide variety of interesting systems. As a result, their use is widespread both in physics and other sciences. For example, linear models can be applied to describing problems as diverse as navigation using least squares trilateration [1], to finding predictors of prostate cancer [2]. In order to gain an understanding of these methods, we will explore three common forms of linear regression: ordinary least squares, as well as two regularized methods, Ridge and Lasso regression. For the latter two, which are examples of so-called shrinkage methods, a penalty is applied to the weights of the model, promoting a sparser model. As we will begin to explore, this hopefully leads to a model which generalizes better to unseen data.

We will first explore the theoretical underpinnings of least squares regression, and also consider formulations of Ridge and Lasso regression. Along the way, we also introduce and derive the important bias-variance tradeoff, which we also use to assess our later models.

In addition to looking at theoretical expressions, we implement the regression methods numerically, and apply them to data generated using a test function, the Franke function in order to benchmark our algorithms. To emulate real-world applications, we limit the size of our test dataset, and investigate the use of resampling methods, using both bootstrap and k-fold crossvalidation to compute statistical measures of model performance. After testing and verifying our models, we apply our algorithms to modelling real-world terrain data, by estimating the terrain elevation in a region around Mågerøy, Tjøme, Norway.

Background

Least Squares Regression

In order to start a discussion on linear regression methods, we can consider the following situation: We are performing an experiment, and would like to create a model that describes our findings. In other words, the outcome of our experiment is a dataset which contains pairs of observations. For example, we might have n experimental variables $\mathbf{x}_j$, resulting in equally many measurements y_j, for $j = 1, 2, ..., n$. Assuming there

is some connection between the two, and also that there is some noise in our system, a simple model of the situation would be

$$\mathbf{y} = f(\mathbf{x}) + \boldsymbol{\varepsilon},$$

where $\boldsymbol{\varepsilon}$ is a variable representing noise. For simplicity, we assume that $\boldsymbol{\varepsilon}$ is normally distributed. Note that the noise might be different for different observations, and hence we use a vector to denote it. Note also that this is of course general, we do not have to perform experiments of any kind, we just want to develop a model relating input variables $\mathbf{x}$, to output variables $\mathbf{y}$.

Regardless of the problem at hand, the simplest relationship we can imagine between $\mathbf{x}$, and $\mathbf{y}$ is a linear one, such that

$$f(\mathbf{x}) = \mathbf{X}\beta,$$

where $\mathbf{X}$ is a so-called feature matrix, and $\boldsymbol{\beta}$ a vector of coefficients. The feature matrix is in general a function of the input variables $\mathbf{x}$, but we are more or less free to choose this dependency, depending on the problem we wish to solve. As an example, we could take the columns of $\mathbf{X}$ to be sinusoidal functions of $\mathbf{x}$, or more commonly, as we will also be doing, to be powers of $\mathbf{x}$. In other words, the rows of our design matrix may be given by

$$X_i = [1 \; x_i \; x_i^2 \; ... \; x_i^d],$$

where d is the degree of polynomial we wish to use to represent $\mathbf{y}$. All of this simply means that we wish to approximate our observations $\mathbf{y}$, using a polynomial of some degree p in the inputs $\mathbf{x}$. In other words, we wish to find $\tilde{\beta}$, such that

$$\tilde{\mathbf{y}} = \mathbf{X}\tilde{\beta} \approx \mathbf{y}$$

A natural next question might then be how we determine the coefficients $\tilde{\beta}$ such that our approximation is as good as possible. Now there may be many ways in which an approximation can be good, but we will concern ourselves with the coefficients that minimize the mean squared error

$$C(\mathbf{X}, \tilde{\beta}) = \frac{1}{n}\sum_{j=0}^{n-1}(y_j - \tilde{y}_j)^2, \tag{1}$$

where $C(\mathbf{X}, \beta)$ denotes the so-called loss or objective function, i.e. the quantity we want to minimize, and $\tilde{y}_i = \sum_{i=0}^{p} X_{ji}\tilde{\beta}_i$ is our approximation to y_i. Thankfully, it is quite straight forward to find such a $\tilde{\beta}$, as the mean squared error is minimized under the *least squares* solution, given by

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\beta},$$

and it can be shown that

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \tag{2}$$

[2] where ^{T} denotes the transpose of a matrix. The least squares estimator $\hat{\beta}$ has a number of useful features. For one thing, we can find its variance-covariance matrix *exactly*, using

$$Var(\hat{\beta}) = \mathbb{E}_D[(\hat{\beta} - \mathbb{E}_D[\hat{\beta}])^2],$$

where E_D denotes the expected value over all possible *datasets*. While this might seem a somewhat strange concept, it is actually rather appealing, as we want to quantify how much our estimator varies on average, when we take into account that a given estimator $\hat{\beta}$ depends on the dataset on which it was trained [3]. Note that in order to do so, we have to fix the inputs we are "testing" our model on. For simplicity, we will suppress the notation going forward, and let $\mathbb{E}$ denote the expected value over all datasets, unless otherwise specified. Writing out the previous expression, we get that

$$Var(\hat{\beta}) = \mathbb{E}[\hat{\beta}\hat{\beta}^T - \mathbb{E}[\hat{\beta}](\hat{\beta} + \hat{\beta}^T) + \mathbb{E}[\hat{\beta}]^2],$$

but we know that $\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^TX\beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{\varepsilon}$, and so we can write $\hat{\beta} = \beta + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{\varepsilon}$, by our assumption that our output is a linear, noisy function. This means that $\mathbb{E}[\hat{\beta}] = \mathbb{E}[\beta] + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\varepsilon]$, as $\mathbf{X}$ is a non-random matrix (and we consider the expected value for fixed $\mathbf{x}$), but

we have also assumed that $\boldsymbol{\varepsilon}$ is zero-centered, gaussian noise, and so $\mathbb{E}[\hat{\beta}] = \mathbb{E}[\beta] = \beta$. This is reassuring, as the expected value of our parameters $\hat{\beta}$ is the true vector, β. This means that

$$Var(\hat{\beta}) = \mathbb{E}[\hat{\beta}^2] - \mathbb{E}[\hat{\beta}]^2 = \mathbb{E}[\hat{\beta}^2] - \beta^2,$$

which is just another definition of the variance. We can rewrite $\hat{\beta}^2$ using the least squares solution as $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\mathbf{y}^T\mathbf{X}((\mathbf{X}^T\mathbf{X})^{-1})^T)$, and by using the fact that $\mathbf{X}$ is non-random (for fixed inputs),

$$Var(\hat{\beta}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{y}\mathbf{y}^T]\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} - \beta^2,$$

where we used that $\mathbf{X}^T\mathbf{X}$ is symmetric. Fortunately $\mathbb{E}[\mathbf{y}\mathbf{y}^T] = \mathbb{E}[(\mathbf{X}\beta + \boldsymbol{\varepsilon})(\mathbf{X}\beta + \boldsymbol{\varepsilon})^T]$ is just $\mathbb{E}[\mathbf{X}\beta\beta^T\mathbf{X}^T] + \mathbb{E}[\boldsymbol{\varepsilon}^2]$, (the cross terms vanish as the noise is zero-centered). Inserting this, we see that we obtain a spectacular cancellation, and that

$$Var(\hat{\beta}) = \beta^2 - \beta^2 + \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1},$$

as the variance of $\boldsymbol{\varepsilon}$ is just σ^2 (see the Bias-Variance Tradeoff section for more). Since this is the variance-covariance-matrix of $\hat{\beta}$, the variance of the coefficients are just the diagonal elements

$$Var(\hat{\beta}_i) = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}_{ii}, \tag{3}$$

which we will use to set up confidence intervals for the coefficients.

The Bias-Variance Tradeoff

As we mentioned briefly, there are many ways in which an approximation may be "good". However, it turns out that we can decompose the mean squared error into what is called a bias, and a variance term for our estimator, $\tilde{f}$. This decomposition is important, as it allows us to quantify to what degree a given model (on average) accurately predicts the correct output, and to what degree its outputs vary. We can imagine that for different applications we might require that our model performs better in one of these regards than the other. For example when determining disease risk, we would probably want our estimate for a given person's risk of falling ill to be slightly off, rather than to vary wildly between patients, but on average predict correctly! However, we can tell that an optimal model would have both zero bias, and zero variance, but as the term tradeoff suggests, we will struggle to achieve both. Another useful feature of the bias variance-tradeoff, is that it can help explain the benefits of shrinkage methods, like Ridge and Lasso regression.

To begin deriving it, we note that we can write our loss function in terms of the expected value of the deviation of our model from the actual observations:

$$C(\mathbf{X}, \boldsymbol{\beta}) = \mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] \tag{4}$$

which we can write as

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

or

$$C = \frac{1}{n}\sum_{i}(f_i + \varepsilon_i - \mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}] - \tilde{y}_i)^2,$$

where we have simply added and subtracted the expected value of $\tilde{\mathbf{y}}$, and used that $y_i = f_i + \varepsilon_i$. We can write this even more suggestively as

$$C = \frac{1}{n}\sum_{i}((f_i - \mathbb{E}[\tilde{\mathbf{y}}]) + \varepsilon_i - (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}]))^2,$$

which gives

$$C = \frac{1}{n}\sum_{i}(f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \varepsilon_i^2 + (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + 2\varepsilon_i(f_i - \mathbb{E}[\tilde{\mathbf{y}}]) - 2(f_i - \mathbb{E}[\tilde{\mathbf{y}}])(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}]) - 2\varepsilon_i(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}]),$$

or

$$C = \frac{1}{n}\sum_i \varepsilon_i^2 + \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + 2\varepsilon_i(f_i - \tilde{y}_i) - 2(f_i - \mathbb{E}[\tilde{\mathbf{y}}])(\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}]).$$

Before kneading this expression further, we remember that $\boldsymbol{\varepsilon}$ constitutes normally distributed noise, with expected value 0, and variance σ^2. In other words

$$Var(\boldsymbol{\varepsilon}) = \mathbb{E}[(\boldsymbol{\varepsilon} - \mathbb{E}[\boldsymbol{\varepsilon}])^2] = \mathbb{E}[\boldsymbol{\varepsilon}^2] = \sigma^2,$$

which we can recognize as the first term in C. Therefore,

$$C = \sigma^2 + \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + 2\varepsilon_i(f_i - \tilde{y}_i) - 2f_i\tilde{y}_i + 2f_i\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}]\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}]^2,$$

which appears somewhat messy. However, we still have that $\boldsymbol{\varepsilon}$ is random noise, and should as such be independent of our model f, and our approximation $\tilde{y}$. For independent random variables, X and Y, we have that

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y],$$

and so

$$\frac{1}{n}\sum_i 2\varepsilon_i(f_i - \tilde{y}_i) = 0,$$

as the expectation value of $\boldsymbol{\varepsilon}$ is zero. We are inching closer to a solution, and now we know that

$$C = \frac{1}{n}\sum_i \sigma^2 + (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 - 2f_i\tilde{y}_i + 2f_i\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}]\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}]^2,$$

and so we want to rewrite the final terms. To do so, we note that the expectation value of $\tilde{\mathbf{y}}$ is $\mathbb{E}[\tilde{\mathbf{y}}]$, of course, and f_i and $\tilde{y}_i$ are independent, as changing our model will not influence the actual function we wish to approximate! Finally, we get

$$\begin{aligned} C &= \sigma^2 + \left(\frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2\right) - 2\mathbb{E}[f]\mathbb{E}[\tilde{\mathbf{y}}] + 2\mathbb{E}[f]\mathbb{E}[\tilde{\mathbf{y}}] + \mathbb{E}[\tilde{\mathbf{y}}]^2 - \mathbb{E}[\tilde{\mathbf{y}}]^2, \\ &= \sigma^2 + \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2, \end{aligned}$$

where σ^2 is noise that we cannot hope to remove, while we recognize

$$Var(\tilde{\mathbf{y}}) = \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2,$$

and we also denote

$$b^2 = \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2,$$

as the (squared) bias of our estimator. Both of these quantities are somewhat understandable; the variance measures the average deviation of our estimator from its mean value. The bias, on the other hand, measures how far our estimator deviates from the true function f we hope to approximate. However, for these quantities to really make sense, we should once again consider that we are taking the expectation value over all datasets.

Thus, when we take the expectation value of $\mathbf{y}$, we fix the measurement $\mathbf{x}$ which leads to a certain f, and then we see what the "average" model would predict, which would then be $\mathbb{E}[\tilde{\mathbf{y}}]$. If we had an infinite amount of datasets to train with, we would expect that the average prediction of our model should be as close to f_i as is possible, given the constraints of the model we have selected. Therefore, the bias reflects the lowest error we can hope to achieve using our model[3].

Resampling Methods

As we have stressed for the bias-variance analysis, when we estimate the performance of our models, we do so by averaging their performance over all datasets. In real life, however, we are usually stuck with just one (often too small) dataset, and so we need some clever way of effectively generating more datasets. This is where resampling methods come in; by sampling the dataset over and over, so that each sample has a different composition of train, and possibly test samples, we can approximate having a larger set of datasets available. In this project, we only consider two resampling methods, bootstrapping and k-fold crossvalidation.

Bootstrap resampling consists of repeatedly drawing samples from the training dataset, with replacement, such that a given bootstrapped training set can contain the same sample multiple times. Then, statistics such as the test mean squared error are computed for each model (trained on a bootstrapped training set), and the final statistic is taken to be the mean over all bootstrapped samples, just as if we were taking the expected value over many datasets.

For k-fold crossvalidation, we take the original dataset, and divide it into k so-called folds, or partitions. Then, training is performed on $k-1$ of the folds, and the last fold is reserved for testing. In the next iteration, the next fold is taken to be the test fold, and the rest training folds, and so on, until all the data has been used up. For each iteration, statistics are computed, and the final statistic is taken to be the mean value over all folds. For more on these methods, see for example [2].

Ridge & Lasso Regression

As we learned from the bias-variance tradeoff, a model can have a low bias, or a low variance, but generally not both at the same time (as this would require the mean squared error to go to zero). As we saw for the ordinary least squares regression, the OLS solution was that which minimized the mean squared error. However, this comes with an important caveat, as it assumes that $\mathbf{X}^T\mathbf{X}$ is invertible/nonsingular. If it is not, or is numerically very close to being singular, it turns out that

$$\hat{\beta}_{Ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1})\mathbf{X}^T\mathbf{y}, \tag{5}$$

can actually give a model with a lower mean squared error, for a suitable λ, even though the added term $\lambda\mathbf{I}$ makes the ridge regression estimator biased [4]. Note that $\mathbf{I}$ is the identity matrix, and that for ridge regression, the objective being minimized is actually

$$C_{Ridge} = \mathbb{E}[\mathbf{X}\hat{\beta}_{Ridge} - \mathbf{y})^2] + \lambda\sum_j^p (\hat{\beta}_{Ridge})_j{}^2,$$

i.e. the sum of the mean squared error, and a term dependent on the magnitude of the coefficients. This is why Ridge (and Lasso) regression is often referred to as shrinkage methods, as they tend to shrink the coefficients of the estimator. It can also be shown [4] that the variance-covariance matrix for $\hat{\beta}_{Ridge}$ has an explicit expression, and is given by

$$Var(\hat{\beta}_{Ridge}) = \sigma^2[\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}]^{-1}\mathbf{X}^T\mathbf{X}[(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}]^T, \tag{6}$$

i.e. quite similar to the expression for the OLS case.

For Lasso regression, the objective function is almost the same as for Ridge:

$$C_{Lasso} = \mathbb{E}[\mathbf{X}\hat{\beta}_{Lasso} - \mathbf{y})^2] + \lambda\sum_j^p |(\hat{\beta}_{Lasso})_j|,$$

but we penalize the absolute value of the weights, not the square. Unfortunately, there is no closed form solution of the Lasso regression coefficients, but they can be determined numerically, as we will do in this project. While not immediately obvious, one important difference between Ridge and Lasso regression, is the way in which the coefficients are shrunk: for Ridge regression, the squared weight penalty discourages outliers, in the sense that coefficients that are much larger than others, are penalized much more harshly

(due to the square). Because of this, high values of λ tends to shrink *all* coefficients to zero. For Lasso regression, shrinking a small parameter to zero will cause the same benefit as decreasing the size of large values by the same amount. Therefore, Lasso regression allows for sparser models, in the sense that it can shrink only some coefficients to zero, while leaving others relatively large without incurring a great loss.

The Franke Function

As a useful test case, we will consider the so-called Franke function, illustrated in Fig. 1, and given by

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

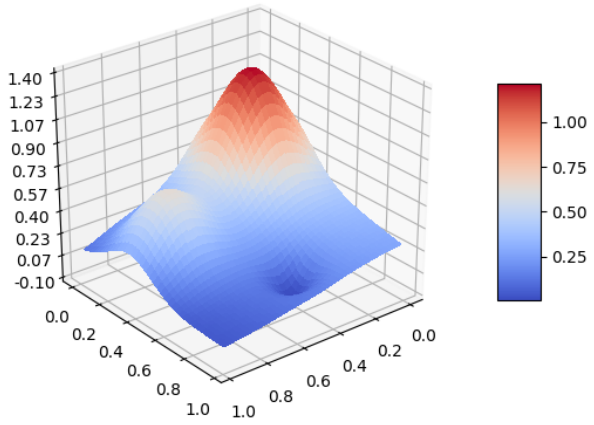


Figure 1: 3D plot of the Franke function, for evenly spaced values of x and y in the interval $[0, 1]$. Note that the code for producing this code is originally due to [5].

The reason this particular equation is useful, is that, as we can see, it is a sum of gaussians, i.e. quite far from a polynomial. Therefore, it poses a somewhat realistic problem when we try to fit a polynomial to it, and we can use it as a simple benchmark of our regression models. In addition, the "bumpiness" of a sum of Gaussians actually mimics quite well the hills and valleys of actual terrain, albeit very smoothly. Therefore, we hope that it is also a decent analogue for terrain data.

Methods

In order to explore the application of regression methods, we first created a simulated dataset for the Franke function. The input data to the Franke function was taken to be $n = 1000$ uniformly distributed points in the interval $x, y \in [0, 1]$. The addition of noise was explored, by adding pseudo-random noise to the input coordinates, drawn from a normal distribution $\mathcal{N}(0, 1)$. The dataset was split into a training and test set, with 80% used for training, and the remainder for testing. All input data to the model was scaled, by subtracting the training sample mean, and dividing by its variance. Note that the same scaling procedure was applied to every subsequent task and dataset in this project.

An ordinary least squares regression model implemented, and then trained on the input coordinates according to (2), with the rows of the design matrix taken to be given by polynomials in x and y, up to degree five. In other words, the rows were given by

$$X_i = [x_i^0 \quad y_i^0 \quad x_i^1 \quad y_i^1 \quad x_i^1 y_i^1 \quad x_i^2 \quad ... \quad x_i^5 y_i^5],$$

and in all later models, the design matrix followed the same structure, consisting of unique combinations of powers of x and y, up to the maximum polynomial degree p.

For the OLS-model trained on the Franke function dataset, the variance of its coefficients was computed according to (3), along with corresponding 95% confidence intervals. As an estimate of the variance of the model [2], the approximation

$$\hat{\sigma}^2 = \frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - \hat{y})$$

was used, where n is the number of samples, and p the degree of the polynomial used, as before.

To determine the degree to which the model predictions correlated with test data, the R^2-score

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

was computed, with $\bar{y}$ being the test sample mean, $\mathbf{y}$ the test data, and $\tilde{\mathbf{y}}$ being the model predictions.

Note that all project codes and results are freely available at `https://github.com/markusbp/fys_stk4155/tree/master/project1`.

In order to gain an appreciation for why datasets are split into separate training and test sets, several OLS regression models, was trained on a Franke function dataset, consisting of $n = 1000$ samples, with added noise, drawn from a normal distribution $\mathcal{N}(0, 1)$. In order to study the effect of increasing model complexity, one model was created and trained for each polynomial degree from 0 up to 20, such that the maximum polynomial degree of model number p was p. For this trial, as before, 80 % of the dataset was used for training and the rest reserved for testing. For all models, the mean squared error was determined for both the training- and test datasets.

In order to perform a bias variance analysis of a trained OLS regression model, we once again trained several sets of models, with polynomial degrees up to $p = 20$. In this case, the dataset used was Franke function data as before with $n = 1000$ samples, but for this trial, no noise was added. Once again, a train-test split of the data was performed, with 80 % used for training data, the rest for testing.

As discussed in the background section, a bias-variance tradeoff analysis requires taking the average or expected value over all datasets, so in order to approximate this, bootstrap resampling was performed of the training set with $b = 100$ bootstraps for each model. For each bootstrap dataset, the model was trained, and the mean squared error, squared bias, and variance of a given model applied to the test set, was calculated in accordance with expressions derived in the bias-variance tradeoff section. The final approximation of these quantities was taken to be the mean value over all bootstrapped samples. Since the true function values $f(\mathbf{X})$ was not known for our problem, the test set data values, $\mathbf{y}$ was used in their place as an approximation, for determining the bias.

To explore another common resampling technique, k-fold cross-validation was implemented, and for a dataset equal to that used for the bias variance-tradeoff analysis described in the previous section, and also the same polynomial degrees, the mean squared error was calculated using $k = 5$ and 10 folds.

In addition to ordinary least squares regression models, Ridge and Lasso models were also implemented and analyzed. For the Ridge regression, models were trained according to (5), while for Lasso models, pre-existing algorithms from the scikit-learn software library were used. For both the Ridge and Lasso models, a bias-variance analysis was performed with bootstrap resampling, for the same dataset as the OLS case. In this case, the mean squared error, bias, and variance was computed not only as a function of the maximum polynomial degree, but also of the shrinkage parameter λ. As an additional test of sanity, the mean squared error was also computed using 10-fold crossvalidation for both methods.

To explore the effects of the shrinkage parameter λ on the coefficients of a Ridge regression model, models with maximum polynomial degree $p = 3$ were trained for $\lambda \in [10^{-4}, 10^3]$. To also demonstrate the effects of noise, this analysis was performed for both noisy and noise-free Franke function datasets, with $n = 1000$ samples. The confidence intervals of the Ridge coefficients were also determined, using (6).

As a final application of all three methods, models were trained on a real-world dataset, consisting of terrain elevation data from a region around Mågerøy, Tjøme. While the original dataset included a sizeable swathe of eastern Norway, this was narrowed down to a 100×100 pixel region around Mågerøy. These elevation data were then scaled by dividing by their maximum value, so that all samples values were in the interval $[0, 1]$. In order to reduce the computational load of training, this region was downsampled to a 33×33-region, and it was this smaller subset which was used for model evaluation and training.

In order to determine the best model, all three regression methods were applied to the $33 \times 33 = 1089$ sample dataset, but as before, 80 % was used for training, and the rest withheld for testing. For the OLS regression, models were trained for all polynomials up to degree 20. For the Lasso and Ridge regression models, the same polynomials were used, but a grid search was also performed for shrinkage parameters in the interval $[10^{-5}, 100]$ for eight evenly spaced (on a log scale) values of λ.

Results

Fig. 2 shows the result of training an OLS model on a Franke function dataset, with $n = 1000$ samples, with and without added noise. As we can tell, the coefficients of models trained on noisy datasets have the highest variance, reflected by the 95 % confidence intervals. However, we see that there is agreement between the noisy and noiseless case, as the 95 % confidence interval contains the corresponding noiseless coefficient in all cases. For the indicated models, the R^2 score was 0.98 in the noise-free case, and 0.07 in the noisy case. The test MSE was 0.002 for the noise-free case, and 1.00 (after rounding) for the noisy model.

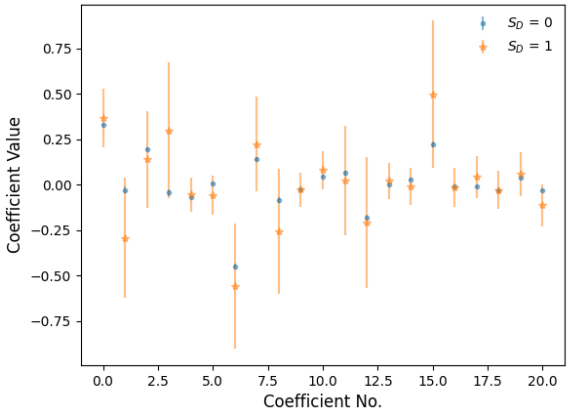
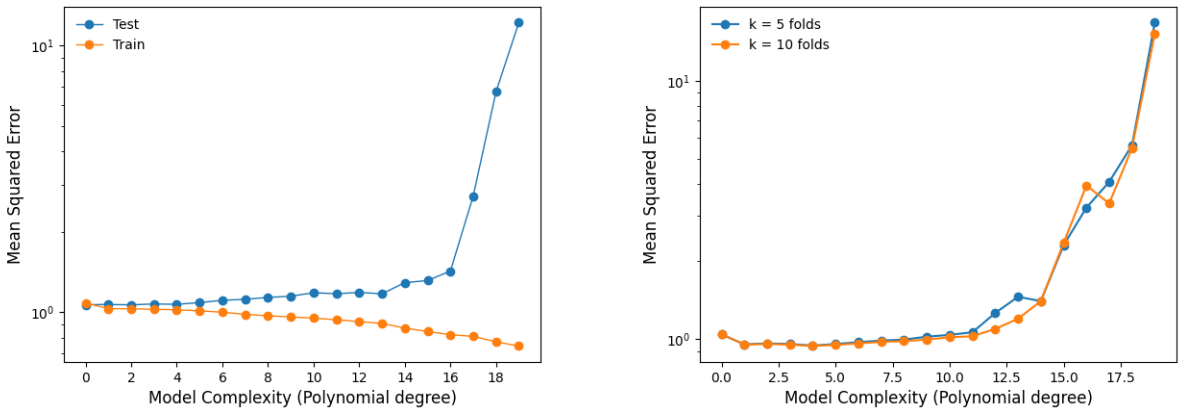


Figure 2: Coefficients of ordinary least squares regression model trained on a Franke function dataset. Indicated as error bars, are the 95 % confidence intervals of each parameter. Shown are results for datasets with added normally distributed noise with standard deviation $S_D = 0$, and $S_D = 1$.

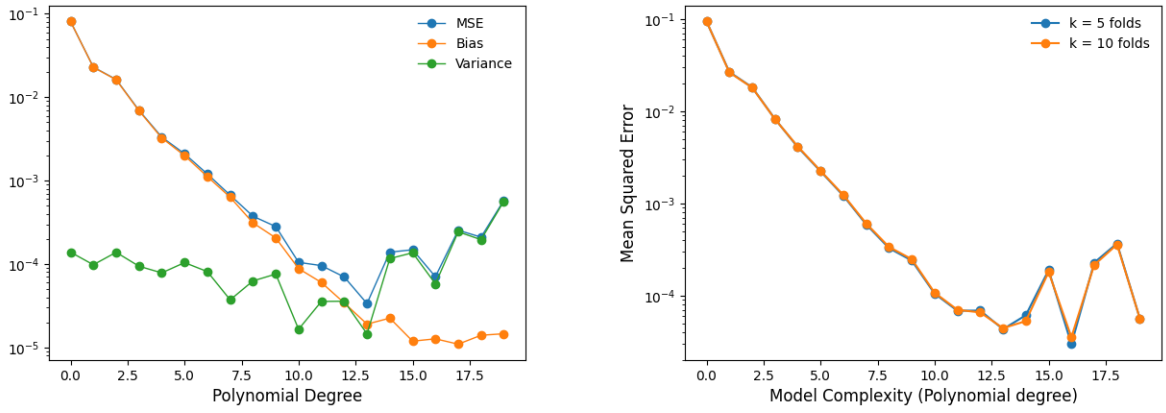
As for the test- and train loss of the OLS model trained on an $n = 1000$ sample Franke function dataset with added noise, Fig. 3 tells us that the training error continues to decreases as we increase the model complexity (polynomial degree). For the test loss, however, we can tell that it actually increases with increasing model complexity, and that a sharp increase is seen for polynomial degrees greater than 16. Fig. 3 also shows the test mean squared error computed for a model trained on the same dataset, using crossvalidation, which apparently reveals the same trend, both for $k = 5$ and 10 folds.



(a) (b)

Figure 3: a) MSE for OLS regression models trained on a $n = 1000$ sample Franke function dataset with added noise drawn from a normal distribution $\mathcal{N}(0, 1)$, for varying polynomial degrees. Indicated is both the test and training error. b) Test MSE for the same dataset and models, calculated using k-fold cross-validation for $k = 5$ and 10 folds.

Fig. 4 shows the results of the bias- variance analysis for the model trained on the same Franke function dataset with no added noise, using bootstrap resampling. Also shown is the test MSE calculated using k-fold crossvalidation. There is an apparent trend that as the model complexity (polynomial degree) increases, the bias of the trained models decreases, while the variance increases, in keeping with the bias-variance trade-off. We also see that as the model complexity becomes large, the test MSE begins to increase, consistent with the results of Fig. 3. In addition, we see that the crossvalidation results closely follow those found using bootstrap resampling, for both $k = 5$ and $k = 10$ folds. However, we can read off the minimum MSE for $p = 13$ at around $5 \cdot 10^{-5}$ for the bootstrapped results, but for the crossvalidation error, the lowest error is found for $p = 16$ (the error is approximately the same, however).



(a) (b)

Figure 4: a) Bias-variance analysis of an ordinary least squares model, for polynomial degrees up to $p = 20$. Models were trained on a Franke function dataset, with no added noise. All models were trained on 100 bootstrapped samples, and the shown quantities were calculated using the holdout test dataset. b) Test MSE for the models trained on the same dataset using the same polynomial degrees, calculated using k-fold crossvalidation for $k = 5$ and 10 folds.

For the Ridge regression methods, Fig. 5 illustrates the effect of adding a weight regularization to the objective function. For the indicated coefficients, all models were trained on Franke function datasets, with and without added noise. As we can tell, large values of the shrinkage parameter λ, tend to cause the weights to shrink to zero, both for noisy and noiseless models. Interestingly, however, we see that a few coefficients actually increase for very high values of λ. Fig. 5 also indicates 95 % confidence intervals for the coefficients, and as we might expect, these are close to zero in the noiseless case, and large for the noisy case, but it once again seems that the confidence intervals of the noisy coefficients contain their noiseless counterparts.

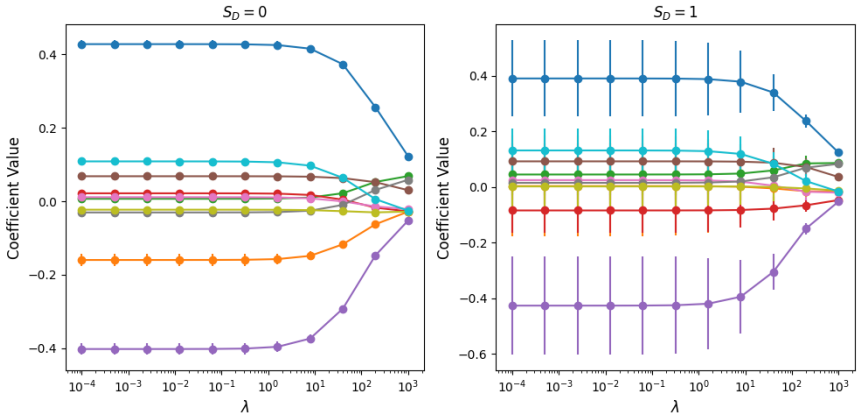


Figure 5: Coefficient values for a Ridge regression model, as a function of the shrinkage parameter λ. Shown are models trained on a Franke function dataset with $n = 1000$ samples for polynomials up to degree 3 in x and y. For the left pane, no noise was added to the dataset, while for the right hand figure, noise drawn from a normal distribution $\mathcal{N}(0, 1)$ was added.

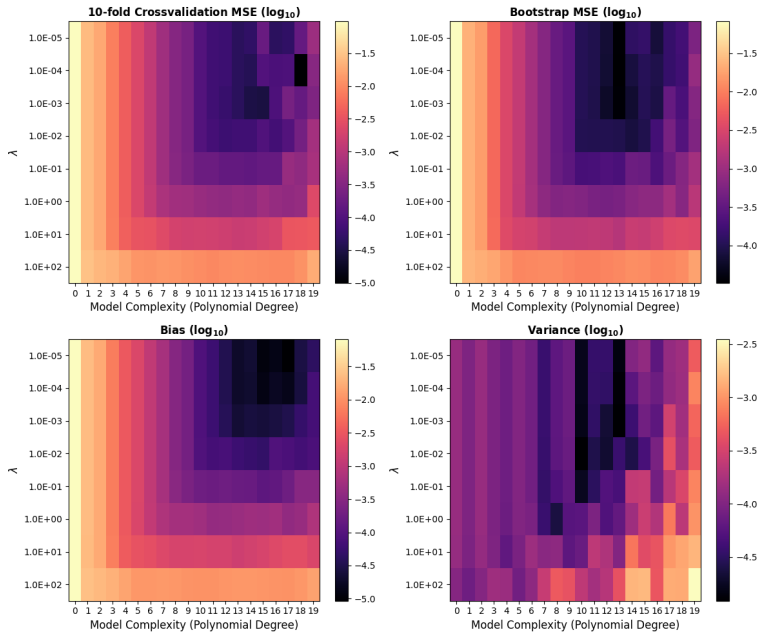


Figure 6: Results of a 2D grid search for maximum polynomial degree p and shrinkage parameter λ for Ridge regression models trained on a $n = 1000$ sample Franke function dataset with no added noise. Shown are MSE values calculated using 10-fold crossvalidation, as well as using bootstrap resampling. Also shown is the model bias and variance. Note that all bootstrapped quantities were calculated using 100 bootstrap samples.

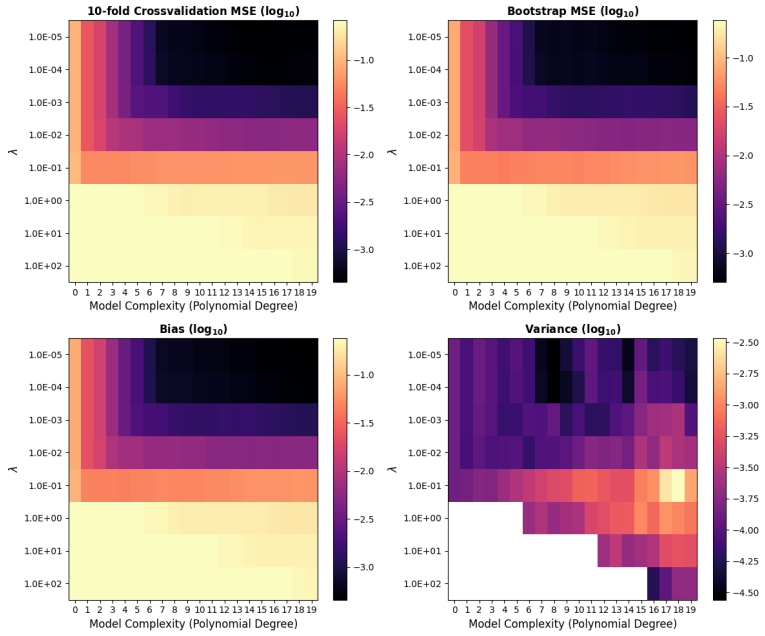


Figure 7: Results of a 2D grid search for maximum polynomial degree p and shrinkage parameter λ for Lasso regression models trained on a $n = 1000$ sample Franke function dataset with no added noise. Shown are MSE values calculated using 10-fold crossvalidation, as well as using bootstrap resampling. Also shown is the model bias and variance. Note that all bootstrapped quantities were calculated using 100 bootstrap samples.

Figures 6 and 7 show the results of 2D grid searches for shrinkage parameters λ and polynomial degrees p for Ridge and Lasso regression methods, respectively. All models were trained $n = 1000$ sample Franke function datasets, with no added noise. For Ridge regression, we can see that there is a clear region of low test error (MSE) for polynomial degrees in the region 9-18, and λ values approximately in the interval $[10^{-5}, 10^{-2}]$, and this is true both for the crossvalidation, and bootstrap MSE. However, we see that the 10-fold crossvalidation MSE is slightly lower than the corresponding bootstrapped MSE. Interestingly, we can also recognize a bias variance tradeoff, and that the model MSE appears lowest in the region where both the variance, and bias are both low. As with the OLS case, however, the bias appears to continue do decrease, but there is a small uptick for large p.

For Lasso regression, the results appear somewhat similar, but in this case, the model failed to converge for many values of λ and p, as suggested by the relatively large error compared to the Ridge results. We can see that the best test set error is almost two orders of magnitude worse than for the Ridge regression case.

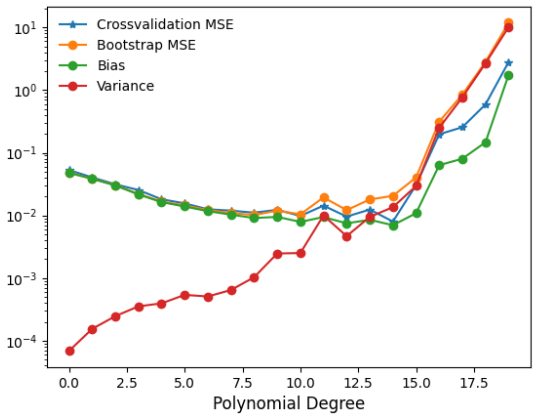


Figure 8: Results of grid search for the maximum polynomial degree of OLS regression models. In all cases, models were trained on the downsampled terrain data.

Figures 8, 9 and 10 shows the results of parameter searches for OLS, Ridge, and Lasso regression methods, respectively, each of which was trained on the downsampled terrain dataset. Starting with the ordinary least squares search, we can first note that the bias-variance tradeoff is slightly different from before; while it is true that the bias decreases with increasing complexity this is only up to a point, at around $p = 13$ at which the bias also increases strongly. The test set variance and MSE behaves as one might expect, however, with the variance steadily increasing, and the crossvalidation MSE reaching a minimum of $8.0 \cdot 10^{-3}$ at $p = 11$, before increasing.

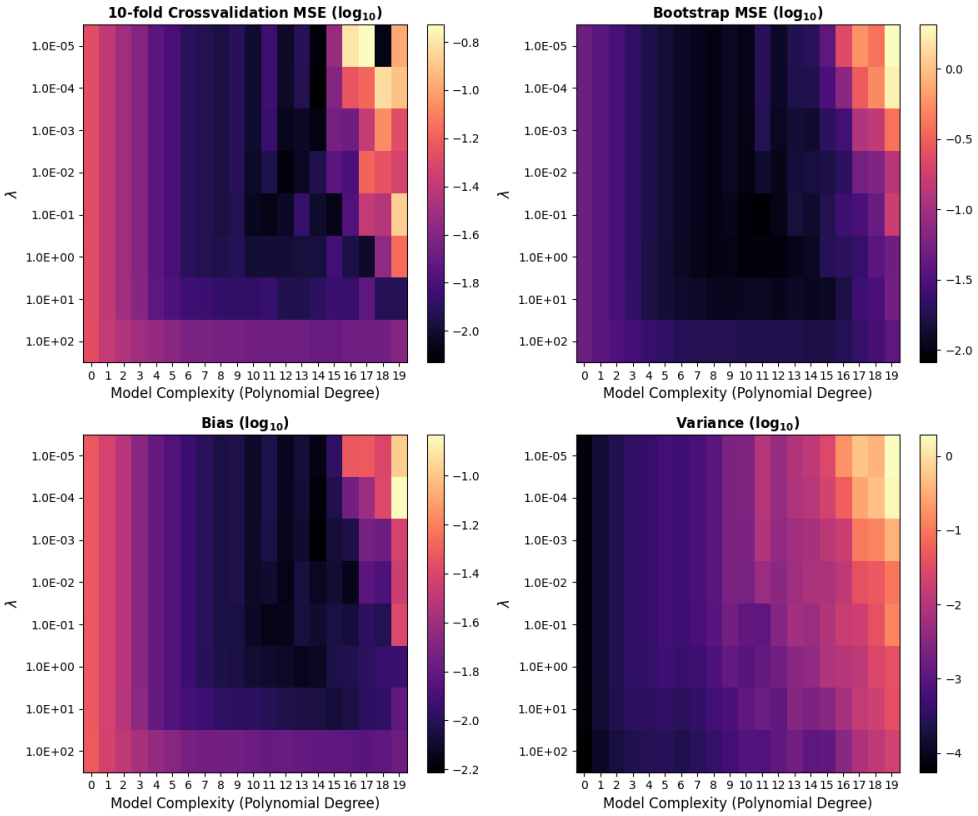


Figure 9: Results of a 2D grid search for maximum polynomial degree p and shrinkage parameter λ for Ridge regression models trained on the downsampled terrain data. Shown are MSE values calculated using 10-fold crossvalidation, as well as using bootstrap resampling. Also shown is the model bias and variance. Note that all bootstrapped quantities were calculated using 100 bootstrap samples.

For the Ridge regression search, we see that the results are similar to that of the noiseless Franke function case in Fig. 6, except that the loss is several orders of magnitude greater. We do however see that the same parameters as before tend to yield a good model, with λ approximately in the interval $[10^{-5}, 1]$ and p around $[7,\ 14]$ gives an error of around 10^{-2}. The lowest achieved crossvalidation MSE of 0.0074 was found for $p = 14$, $\lambda = 10^{-4}$. Notice that there is a larger difference between the bootstrap and crossvalidation estimate of the mean squared error. According to the bootstrapped MSE, a much wider range of models give low errors. We can however observe that the minimum error of the bias, and the crossvalidation MSE seem to coincide, suggesting that a low bias model could be useful for this application.

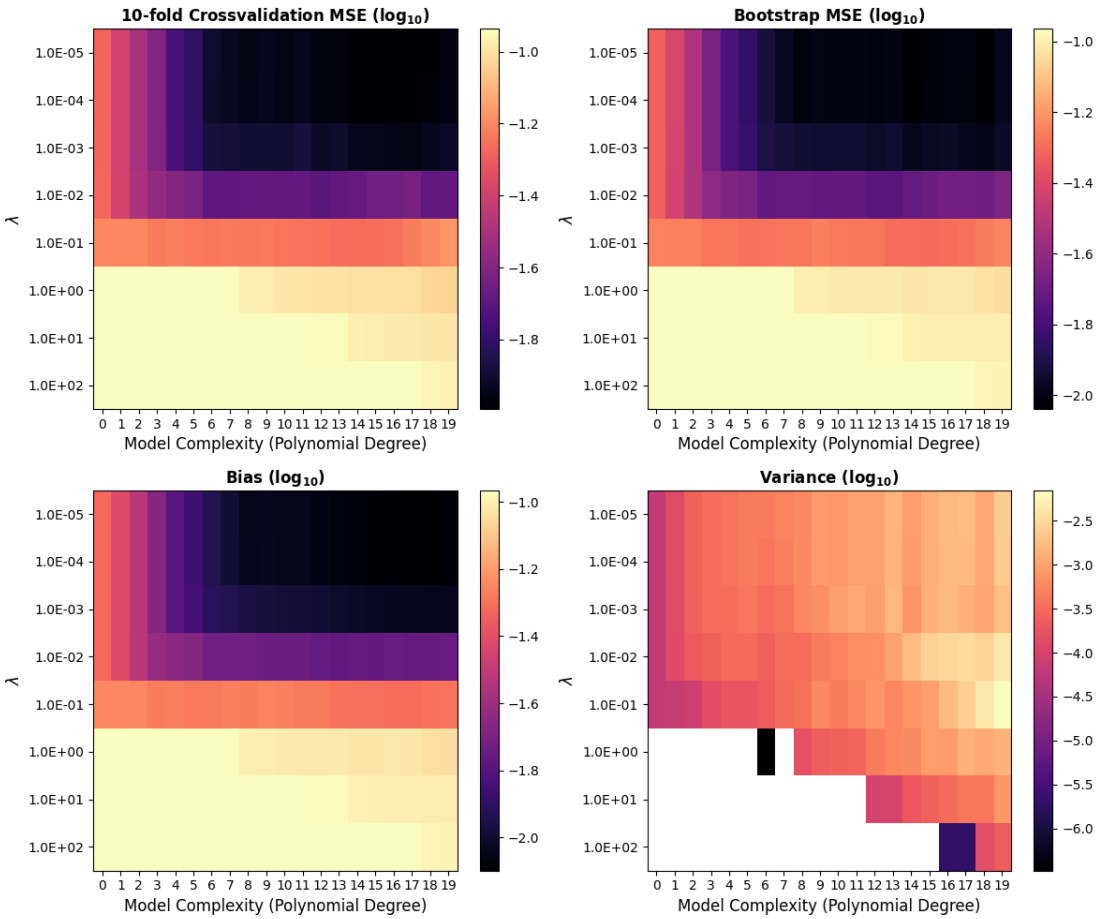


Figure 10: Results of a 2D grid search for maximum polynomial degree p and shrinkage parameter λ for Lasso regression models trained on the downsampled terrain data. Shown are MSE values calculated using 10-fold crossvalidation, as well as using bootstrap resampling. Also shown is the model bias and variance. Note that all bootstrapped quantities were calculated using 100 bootstrap samples.

For the Lasso regression results, we again find that the model failed to converge for many values of λ and p, and that the test mean squared error is somewhat greater than for its Ridge counterpart. We see however, that the error is dominated by the bias, suggesting once more that a low bias model could be beneficial.

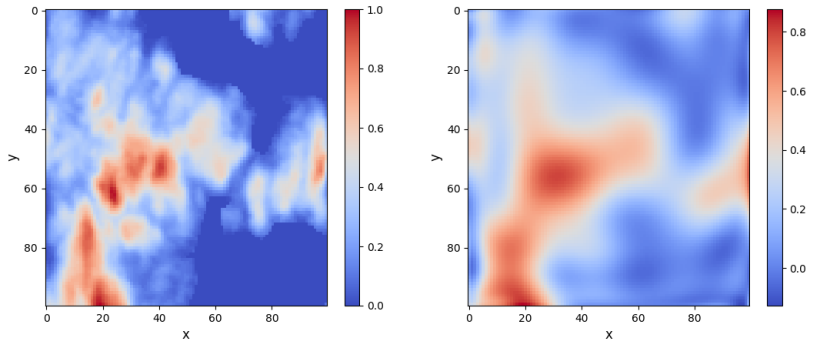


Figure 11: Results of applying the top performing model from the grid search, a Ridge regression model with $\lambda = 10^{-4}$ and $p = 14$, to the full 100×100 original image. The left pane shows the original image, while the right pane shows the model output for scaled inputs corresponding to the original image coordinates.

Finally, Fig. 11 shows the effect of applying the best trained model, a Ridge regression model with $p = 14$ and $\lambda = 10^{-4}$ to the original 100×100 terrain data. We can clearly see that the model captures the large features of the terrain, but fails to capture fine details, such as smaller hills and valleys in the landscape.

Discussion

For the OLS regression coefficients of Fig. 2, the fact that the noiseless case coefficients are contained in the confidence intervals of the noisy coefficients suggests that the models are working as intended. For the noiseless case, this is also supported by the fact that the R^2-score was close to 1, suggesting a high correlation between model predictions and the test data.

However, it seems that the addition of noise from a normal distribution $\mathcal{N}(0, 1)$ to the Franke dataset is excessive, as the R^2-score is close to zero, and the MSE almost 1, close to the maximum value of the noiseless Franke function. This suggests that scaling the noise, or possibly adding noise with a smaller variance, is necessary for a more insightful analysis. As such, doing so would be an interesting further development.

However, even though it is difficult for the model to learn anything useful from such a noisy dataset, we can see from Fig. 3 that we can still learn something, namely that overfitting can occur even in simple models. As we just discussed, the R^2 score of the model was close to zero in the noisy case, yet the training error continues to decrease with increasing p (while the test error soars). This is of course because a more complex model has the capacity to memorize data, and so when the number of parameters is large enough, the model can simply memorize the dataset, irrespective of the amount of noise. This is probably what we are witnessing in Fig. 3, and also explains why the test error is so large, as memorization is far from generalization.

It is also worth pointing out that since the crossvalidation MSE and the single-run test MSE seem to coincide, we have some evidence that the implementation of the k-fold crossvalidation algorithm is working as intended. The same can be said for the results of Fig. 4, where we see that crossvalidation and bootstrapping appear to yield similar results. On the other hand, there was some discrepancy in the minimum values found using bootstrapping and crossvalidation, so it would be interesting to run some more rigorous analyses to determine which method provided the better estimate of the mean squared error. It might also be somewhat strange that 5 and 10-folds yield such similar results, but a trial simulation with $k = 2$ folds revealed a more noticable difference in the MSE estimate (compared to 5 and 10 folds), suggesting that there is in fact a difference between the $k = 5$ and 10-fold estimates.

For the shrinkage analysis in 5, the problem of a possibly too large noise is once again apparent as the confidence intervals are very large, but it is interesting to see how the variance of the model coefficients shrink with increasing λ, an effect which is only somewhat visible in all of the 2D grid searches. However, the 2D grid searches were performed for noiseless data, except for the terrain data, in which this trend *is* somewhat visible for large λ. As for the fact that some coefficients do not go to zero, this might be explained

by the fact that Ridge regression tends to shrink *all* the weights to zero together, and so as long as there are larger weights in the model, smaller weights could in principle become slightly larger, which is what we see in Fig. 5. This can of course be verified by computing the size of the coefficients for even larger λ.

When it comes to the 2D grid searches for the Franke function datasets it is interesting to see how the best model can be explained in terms of the bias-variance tradeoff, as the best model has both low variance and low bias. For the Ridge case, however, it would be nice to determine what causes the slight difference in MSE estimation between the bootstrap and crossvalidation estimates. As we can see, at $\lambda = 10^{-4}$, $p = 18$, the crossvalidation MSE has a seemingly out of place, very low error. If we disregard this particular value, the bootstrap and crossvalidation estimates seem more aligned. This might also suggest that the relatively small test fold used for 10-fold crossvalidation is more susceptible to sample variations, and so it could be interesting to repeat this run with for example fewer folds.

Regardless of the slight discrepancy between the crossvalidation and bootstrap MSE, it appears that the OLS and Ridge regression methods yield by far the lowest test MSE for the Franke function, and in addition train much faster than the Lasso models which failed to converge. Therefore, it would appear obvious that a Ridge or OLS model should be preferred over a Lasso model for this problem. As mentioned previously, it would be intriguing to add some more well-conditioned noise to the Franke function dataset, and see whether Ridge regression could outperform OLS for noisier Franke function data. The fact that Ridge slightly outperformed OLS on the terrain data, suggests that this could in fact be the case.

For the terrain data, the results are somewhat similar. While it is true that the Ridge regression model had the lowest crossvalidation MSE of 0.0074 (compared to 0.008 for OLS and 0.01 for Lasso), the difference between the three is quite small, and judging from Fig. 10, it might actually be that the Lasso model could perform better given more training time and/or a larger set of parameters to train with. However, this would have to be done in future works, and based on our current findings, it appears that the terrain data is best approximated using Ridge regression, if even just by a small margin. Furthermore, the fact that Lasso regression might take much longer to train, depending on the problem, is in itself a drawback of the method.

As a fun little ending note, we can consider the application of the best Ridge model to the full terrain dataset in Fig. 11. Purely by visual inspection, it appears that the model has learned a smoothed version of the terrain. This might not be that surprising, as the model has only been presented with a downsampled version of the original map. If this downsampling is too coarse, we cannot hope to capture the finer details of the landscape, as they might simply not be in the dataset at all. In this sense, the model generalizes quite well, in that it does not memorize the exact shapes of the dataset, it instead creates a smoothly varying approximation. In fact, the predicted terrain is somewhat reminiscent of the Franke function, which might help to explain why the results are quite similar for both problems.

Conclusion

By implementing Ordinary Least Squares-, Ridge- and Lasso regression algorithms, and applying them to a test dataset created using the Franke function, we have developed useful intuitions about a variety of machine learning concepts, such as overfitting and the bias variance tradeoff. For an OLS model of polynomial degree five, we also explored the addition of normally distributed noise. However, we found that the noise, drawn from a normal distribution $\mathcal{N}(0, 1)$, was too great, as the model R^2-score was close to zero, indicating that the model had been unable to learn the underlying structure of the data. Even though the model could not learn from the noisy dataset, we found that it could be a useful example of overfitting: by increasing the polynomial degree of the model, there was a clear trend that the training error decreased with increasing model complexity, while the test set error behaved oppositely.

For the noiseless dataset, we performed parameter searches for all models, and found that the Lasso regression models failed to converge properly for most polynomial degrees p and shrinkage parameters λ, resulting in a test mean squared error of almost two orders of magnitude more than the Ridge and OLS counterparts. As for the Ridge and OLS models, we found that performance in terms of bootstrapped mean squared error was similar, but the MSE of the OLS model was slightly higher, at around $5 \cdot 10^{-5}$ for $p = 13$ versus approximately $3 \cdot 10^{-5}$ for the Ridge model, for $\lambda = 10^{-4}, p = 13$.

In addition to determining the mean squared error, we also considered the bias-variance decomposition of said error, and found that there was indeed a clear tradeoff between bias and variance, as models with

low bias tended to have higher variance, and vice versa. We found that the best performing models tended to have both low variance and low bias. We also investigated the effects of the shrinkage parameter on the coeffiecents of a Ridge regression model, and found that large values of λ tended to shrink the weights to zero, collectively.

We applied all models to actual downsampled terrain elevation data taken from a region around Tjøme, Norway. As with the Franke function dataset, parameter searches were performed, and the resulting 10-fold crossvalidated mean squared error indicated that Lasso regression was again the poorest fit to the problem, with a best test MSE of 0.01, while the best OLS model achieved an error of 0.008. The lowest error was once again achieved using Ridge regression, at a test MSE of around 0.007, for $p = 14$, $\lambda = 10^{-4}$. While the Ridge regression model was best, both in terms of crossvalidated and bootstrapped MSE, it is interesting to note that the model parameters of the best performer was different depending on which MSE estimate was used. It would therefore be a natural extension of this project to examine these differences more closely, and perhaps determine which error estimate is most accurate.

Finally, we tested the top performing Ridge regression model on the full terrain data, and found that the resulting predicted terrain appeared visually similar to the actual terrain data. The predicted map of the terrain was however smoothed, and did not capture the finer structures of the landscape, which might be a consequence of the fact that the training data was downsampled from the original terrain. Interestingly, the predicted terrain data appeared somewhat similar to a Franke function dataset, suggesting that our test data was in fact a useful analogue to terrain data.

References

[1] C. Poellabauer, "Chapter 10: Localization" - lecture notes. University of Notre Dame, Indiana, 2010. Available at `https://www3.nd.edu/~cpoellab/teaching/cse40815/Chapter10.pdf`

[2] T. Hastie, J. Friedman and R. Tisbshirani, "The Elements of statistical learning". Cancer example: pp. 49-50. Least squares estimator and variance estimate: pp. 45-47. Resampling methods: pp. 241-254. New York: Springer, 2017.

[3] P. Mehta et al., "A high-bias, low-variance introduction to Machine Learning for physicists", Physics Reports, vol. 810, pp. 12-13, 2019. Available at: `https://arxiv.org/abs/1803.08823`

[4] M.J. Jensen, "Week 36: Resampling techniques and Ordinary Least Square" - lecture notes. Department of Physics, University of Oslo, Norway, 2020. Available at `https://compphysics.github.io/MachineLearning/doc/pub/week36/html/week36.html`.

[5] M.J. Jensen, "Project 1 on Machine Learning - Data Analysis and Machine Learning FYS-STK3155/FYS4155", Department of Physics, University of Oslo, Norway, 2020