UiO : Department of Physics
University of Oslo

FYS-STK4155: Applied Data Analysis and Machine Learning

Project 1

Regression Analysis and Resampling Methods

Nicolai Haug Tobias Netskar Kristian Wold

September 30, 2019

Abstract

In this project the two-dimensional Franke's function and geographical terrain data are used to compare the regression analysis methods OLS, Ridge and Lasso. The methods are combined with the cross-validation resampling technique and the minimization of testing MSE was used as a metric for model selection.

When fitted to down-sampled terrain data, the best OLS model occurred for model complexity 5, with MSE 33300. The best Ridge model occurred for model complexity 7 and $\lambda = 10^{-6.7}$, with MSE 31110. The best Lasso model occurred for model complexity 9 and $\lambda = 0.053$, with MSE 32652.

Visually, the predicted models looked like highly smoothed models of the original data, with Ridge and Lasso displaying more features from the origianl dataset. The Ridge model was considered best, both from a visual and MSE standpoint.

None of the models were deemed very useful for prediction, for neither interpolation nor extrapolation. The models serve as a mean of terrain data compression at best. However, this toy example proved fruitful in gaining a deeper understanding of OLS, Lasso and Ridge.

Contents

1 Introduction

Linear regression is a useful tool for predicting a quantitative response. In this project, we study three regression methods, namely, Ordinary Least Squares (OLS), Ridge, and Least Absolute Shrinkage and Selection Operator (Lasso). The methods are combined with resampling methods, in particular the k-fold cross-validation technique.

In order to test the methods and compare them with each other, we will first study how to fit polynomials to a specific two-dimensional function called Franke's function. This function, which is the sum of bivariate Gaussian functions, is a widely used test function in interpolation and fitting problems. In the analysis with Franke's function, we also add stochastic noise to the function and study its effects on the model.

Training a model means setting its parameters so that the model best fit the training set. In order to find the best fit, we need to measure how well the model performs on the training data. Throughout this project, the mean squared error (MSE) will be used as the main performance metric. The R^2 score function, which is closely related to the MSE, will also be used.

Ridge and Lasso regression are so-called regularization methods, which adds information in order to solve ill-posed problems or prevent overfitting. A study of the regularization parameter will also be carried out for both Ridge and Lasso.

For all the regression methods, a central analysis is the bias-variance decomposition, which is a way of analyzing a learning algorithm's expected generalization error. Another study which will be applied to all regression methods, is the the k-fold cross-validation resampling technique for best model selection.

The project culminates in applying the regression methods on real terrain data, were we will study which method is best suited.

This project is structured by first presenting a theoretical overview of the aforementioned methods in Section 2. This is followed by a presentation on the approach to study the various computations of interest in Section 3. Next, the results of the implementation are presented and discussed in Section 4, before subsequently they are concluded upon in Section 5. Lastly, an outline of possible continuations of the models, with respect to the implementation, are presented in Section 6.

2 Theory

2.1 Regression Analysis

Let $X = (X_1, \ldots, X_p) \in \mathbb{R}^p$ and $Y \in \mathbb{R}$ be random variables. A regression model assumes that there is a function $f\colon \mathbb{R}^p \to \mathbb{R}$ such that

$$\mathrm{E}(Y|X) = f(X).$$

The components of X are referred to as the *inputs*, *predictors* or *independent variables*, while Y is referred to as the *output*, *response* or *dependent variable* [1, p. 9].

We will use the notation $\hat{f}$ for a function that is an estimator of f. A value $\hat{Y} = \hat{f}(X)$ is then referred to as the *fitted* or *predicted value* at input X. The estimator $\hat{f}$ is usually calculated from a collection $\{(X_i, Y_i)\, i = 1, \ldots, N\}$ of observations. For this reason this collection is often called a training set.

The *bias* of an estimator $\hat{f}$ in an input point X is the difference

$$\mathrm{Bias}(\hat{f}(X)) = \mathrm{E}(\hat{f}(X)|X) - f(X)$$

between the expected estimate and the true value in X. If the bias is 0, we say that the estimator is *unbiased*. Otherwise the estimator is *biased*.

A linear regression model assumes that f is such that

$$f(X) = \beta_0 + \sum_{i=1}^{p} X_i \beta_i$$

for some $\beta = (\beta_0, \ldots, \beta_p)^T \in \mathbb{R}^{p+1}$. It is customary to include 1 as the first component in X, so that $X = (1, X_1, \ldots, X_p)^T \in \mathbb{R}^{p+1}$ and one can write

$$\mathrm{E}(Y|X) = X^T\beta.$$

2.2 Ordinary Least Squares (OLS)

Assume a linear regression model $\mathrm{E}(Y|X) = X^T\beta$, $\beta \in \mathbb{R}^{p+1}$, for the random variables $X = (1, X_1, \ldots, X_p)^T \in \mathbb{R}^{p+1}$ and $Y \in \mathbb{R}$. The ordinary least squares is a method for estimating β. Namely, given a training set $\{(x_i, y_i) \in \mathbb{R}^{p+1} \times \mathbb{R} : i = 1, \ldots, N\}$, we choose the coefficients β that minimize the residual sum of squares

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} \left(y_i - x_i^T\beta\right)^2.$$

Since this is a quadratic function in the parameters β, it always has a minimum, but it need not be unique. By writing $\mathbf{X}$ for the $N \times p$-matrix with $x_1, \ldots, x_n$ as rows and $\mathbf{y}$ for the column vector with components $y_1, \ldots, y_n$, we get the residual sum of squares in matrix form as

$$\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta).$$

By the chain rule and the fact that the Jacobian matrix of $a \mapsto a^T a$ is $2a^T$, we get that the gradient of RSS is

$$\nabla\mathrm{RSS}(\beta) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta).$$

The Hesse matrix $H\text{RSS}(\beta)$ is equal to the Jacobian matrix of $\beta \mapsto \nabla\text{RSS}(\beta)$, thus

$$H\text{RSS}(\beta) = \mathbf{X}^T\mathbf{X}.$$

Assume now that $\mathbf{X}$ has full column rank. Then $\mathbf{X}^T\mathbf{X}$ is positive definite and in particular invertible. It follows that the equation

$$\nabla\text{RSS}(\beta) = 0 \quad \Leftrightarrow \quad \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

has the unique solution

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \tag{2.1}$$

which gives the unique minimum of RSS. The predicted value at an input vector x_0 (with 1 as the first component) is $\hat{f}(x_0) = x_0^T\hat{\beta}$. The fitted values at the training inputs are

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

Note that

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) = \|\mathbf{y} - \mathbf{X}\beta\|^2,$$

so a vector $\hat{\beta}$ minimizing RSS is such that $\mathbf{X}\hat{\beta}$ is the point in the column space of $\mathbf{X}$ that is closest to $\mathbf{y}$. This is valid also in the case when RSS does not have a unique minimum. In other words, $\mathbf{X}\hat{\beta}$ is the orthogonal projection of $\mathbf{y}$ onto the column space of $\mathbf{X}$. The matrix

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$$

is the projection matrix onto the column space of $\mathbf{X}$ (with the property $\mathbf{H}^T = \mathbf{H} = \mathbf{H}^2$). It is often called the "hat" matrix, since it puts a hat on $\mathbf{y}$.

Using linearity of the expected value we find that

$$\text{E}(\hat{\beta}|\mathbf{X}) = \text{E}((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}|\mathbf{X}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\text{E}(\mathbf{y}|\mathbf{X}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\beta = \beta.$$

Moreover, since in general $\text{Var}(\mathbf{A}Z) = \mathbf{A}\text{Var}(Z)\mathbf{A}^T$ for constant matrices $\mathbf{A}$ and random vectors Z (by the linearity of the expectation), it follows that

$$\text{Var}(\hat{\beta}|\mathbf{X}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\text{Var}(\mathbf{y})\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\sigma^2 I\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}. \tag{2.2}$$

A linear transformation of a multivariate normal random vector is again multivariate normal. Thus $\hat{\beta} \sim N(\beta, \sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$.

We usually estimate the variance σ^2 by

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

Plugging this into Equation (2.2), we get an estimate of $\text{Var}(\hat{\beta})$ by

$$\hat{\sigma}^2(\mathbf{X}^T\mathbf{X})^{-1} = \frac{1}{N-p-1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2(\mathbf{X}^T\mathbf{X})^{-1}. \tag{2.3}$$

It can be shown that $\hat{\sigma}$ has a distribution such that

$$(N-p-1)\hat{\sigma}^2 \sim \sigma^2\chi^2_{N-p-1},$$

the chi-squared distribution with $N-p-1$ degrees of freedom.[1, p. 47] Moreover, the random variables $\hat{\beta}$ and $\hat{\sigma}^2$ are statistically independent.

By writing the parameters $\hat{\beta}_j$ as linear transformations of $\hat{\beta}$, we can show that $\hat{\beta}_j \sim N(\beta_i, \sigma^2 v_j)$, where v_j is the j-th diagonal element of $(\mathbf{X}^T\mathbf{X})^{-1}$, or, in other words $\sigma^2 v_j$ is the j-th diagonal element of $\text{Var}(\hat{\beta})$.

Under the null hypothesis that $\beta_j = 0$, the random variable

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}}$$

is distributed as t_{N-p-1}, a t distribution with $N-p-1$ degrees of freedom.

2.3 Ridge Regression

Ridge regression is an example of a method that gives a biased estimator. It is similar to least squares, but instead of minimizing the sum $\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2$, we minimize

$$\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{i=1}^{p} \beta_i^2, \tag{2.4}$$

where λ is some positive real number. The consequence, compared to least squares, is that the coefficients β are shrunken. The amount of shrinkage is controlled by the value of λ. Larger λ means more shrinkage. The solution is denoted $\hat{\beta}^{\text{ridge}}$.

It can be shown that an equivalent way of formulating the ridge problem is as

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2$$

subject to the constraint

$$\sum_{i=1}^{p} \beta_i^2 \leq t.$$

for some $t > 0$. There is a one-to-one correspondence between the parameters λ and t. It can be shown [1, Exercise 3.5] that if we replace each x_{ij} by $x_{ij} - \overline{x}_j = x_{ij} - \frac{1}{N}\sum_{i=1}^{N} x_{ij}$, the ridge problem has the following two step solution: First estimate β_0 by $\overline{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$, then estimate the remaining coefficients by a ridge regression without the intercept. If we remove the intercept from β and remove the column of ones from the design matrix $\mathbf{X}$, we can write the expression to minimize in the ridge regression step as

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

The ridge regression solution can then be shown to be

$$\hat{\beta}^{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.5}$$

Adding a strictly positive constant λI to $\mathbf{X}^T\mathbf{X}$ makes the problem nonsingular also in the case where $\mathbf{X}^T\mathbf{X}$ is not invertible.

Similarly to for OLS, we can show that

$$\mathrm{E}(\hat{\beta}^{\mathrm{ridge}}|\mathbf{X}) = (\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{X}\beta$$

and

$$\mathrm{Var}(\hat{\beta}^{\mathrm{ridge}}|\mathbf{X}) = \sigma^2(\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda I)^{-1}. \tag{2.6}$$

This shows that the ridge estimator is unbiased if and only if $\lambda = 0$.

2.4 Lasso Regression

Lasso (least absolute shrinkage and selection operator) is a regression method where we choose the coefficients β by minimizing the sum

$$\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{i=1}^{p}|\beta_i|, \tag{2.7}$$

for some $\lambda > 0$. The solution is denoted $\hat{\beta}^{\mathrm{lasso}}$. Equivalently,

$$\hat{\beta}^{\mathrm{lasso}} = \arg\min_{\beta}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2$$

subject to the constraint

$$\sum_{j=1}^{p}|\beta_j| \leq t$$

for some $t > 0$. The parameters t are in a one-to-one correspondence with the parameters λ.

2.5 Summary Statistics

Training a model means setting its parameters so that the model best fits the training set. In order to find the best fit, we need to measure how well the model performs on the training data. A common performance measure is the Mean Square Error (MSE)[2, p. 109]:

$$\mathrm{MSE}\left(\hat{y}, \tilde{\hat{y}}\right) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2, \tag{2.8}$$

where $\tilde{y}_i$ is the predicted value of the ith sample and y_i is the corresponding true value. In order to train a regression model, we need to find the value of β that minimizes the MSE.

The R^2 score function is also a useful statistic in the analysis of how well a model performs. It is a measure of how close the data are to the fitted regression line, and is given by

$$R^2\left(\hat{y}, \tilde{\hat{y}}\right) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y})^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} = 1 - \frac{\mathrm{MSE}}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2}, \tag{2.9}$$

where the mean value of y_i is defined as

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$$

2.6 Effective Parameters

In the least squares regression, the number of the parameters β tells something about the complexity of the model. With more parameters, the model is able to adapt to more fine structures in the data. This doesn't make the same sense for e.g. ridge regression, where we need to take into account also the coefficient of the penalty term.

Suppose that we have the outputs $\mathbf{y} = (y_1, \dots, y_N)^T$ with predictions $\hat{\mathbf{y}}$. We say that the fitting method is linear if there exists an $N \times N$ matrix $\mathbf{S}$ depending only on the inputs, not the outputs, such that

$$\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}.$$

In this case we define the *effective number of parameters* of the fit as

$$\mathrm{df}(\mathbf{S}) = \mathrm{tr}(\mathbf{S}), \tag{2.10}$$

where tr is the trace on $\mathbb{R}^{N\times N}$[1, pp. 232-233]. In the case of OLS, we have $\mathbf{S} = \mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$. Since $\mathbf{H}$ is idempotent, the trace is equal to the rank of the matrix. Hence $\mathrm{tr}(\mathbf{H}) = p + 1$, the number of parameters.

2.7 Bias-variance Tradeoff

Suppose that we have a regression function f such that

$$Y = f(X) + \varepsilon$$

for random variables $X, Y, \varepsilon \in \mathbb{R}$, where $\varepsilon \sim N(0, \sigma^2)$. Let $\hat{f}$ be an estimator for f and let $(X, Y) = (X, f(X) + \varepsilon)$ be an observation independent from $\hat{f}$. For $X = x_0$ we get that the expected prediction error is

$$\begin{aligned}
\mathrm{E}((Y - \hat{f}(x_o))^2) &= \mathrm{E}[((Y - f(x_0)) + (f(x_0) - \mathrm{E}(\hat{f}(x_0))) + (\mathrm{E}(\hat{f}(x_0)) - \hat{f}(x_0)))^2] \\
&= \mathrm{E}[(Y - f(x_0))^2 + (f(x_0) - \mathrm{E}(\hat{f}(x_0)))^2 + (\mathrm{E}(\hat{f}(x_0)) - \hat{f}(x_0))^2 \\
&+ 2(Y - f(x_0))(f(x_0) - \mathrm{E}(\hat{f}(x_0))) + 2(Y - f(x_0))(\mathrm{E}(\hat{f}(x_0)) - \hat{f}(x_0)) \\
&+ 2(f(x_0) - \mathrm{E}(\hat{f}(x_0)))(\mathrm{E}(\hat{f}(x_0)) - \hat{f}(x_0))].
\end{aligned}$$

Since Y is independent from $\hat{f}(x_0)$ and $\mathrm{E}(Y) = f(x_0)$, it follows that

$$\begin{aligned}
\mathrm{E}((Y - \hat{f}(x_o))^2) &= \mathrm{E}((Y - f(x_0))^2) + (f(x_0) - \mathrm{E}(\hat{f}(x_0)))^2 + \mathrm{E}((\mathrm{E}(\hat{f}(x_0)) - \hat{f}(x_0))^2) \\
&= \sigma^2 + \mathrm{Bias}^2(\hat{f}(x_0)) + \mathrm{Var}(\hat{f}(x_0)).
\end{aligned}$$

This is referred to as the bias-variance decomposition of the expected prediction error. The first term is the variance of $f(x_0)$. The second is the squared bias of $\hat{f}(x_0)$, and measures how far $\hat{f}(x_0)$ is from the true value $f(x_0)$. The last term is the variance of the estimator in the point x_0.

The bias-variance decomposition gives a motivation to consider biased estimators. To get the lowest expected test error, we must find the right balance between the bias and the variance of the estimator. If the variance is sufficiently low, it is worthwhile to have a little bias.

2.8 Resampling Techniques

When the complexity of the model – measured as the number of effective parameters, as defined in Section 2.6 – increases, the bias typically decreases, while the variance increases. This is because the model adapts more to the underlying structure. The result could be a model that is well fitted to the training data, but performs badly on new data. This overfitting to the training data makes the training error a bad estimate for the test error.

Given a model with parameter α, we want to find the α that minimizes the test error. In a situation with a "large" amount of data, we would estimate this α as follows: Divide the data into three parts; a training set, a validation set and a test set. The training set is used to train the model. For (a subset of) all parameters α, the validation set is used to estimate the test error. We chose the α that gives the minimum of these estimates. Finally, we use the test set to estimate the test error for the chosen α. A typical split is 50 % for training and 25 % each for validation and testing.

In less data-rich settings, a widely used method to estimate the test error is cross-validation. Cross-validation typically estimates well the expected test error, and not, as one would have hoped, the conditional test error.

With K-fold cross-validation, we split the data into K roughly equal-sized parts. For the kth part, with $k = 1, \ldots, K$, we train the data on the other $K - 1$ parts. For a tuning parameter α for the model, denote the fitted function by $\hat{f}_\alpha^{-k}$. Let $\kappa\colon \{1, \ldots, N\} \to \{1, \ldots, K\}$ be the function such that $\kappa(i)$ is the number of the partition containing the ith observation. We estimate the test error by

$$\mathrm{CV}(\hat{f}_\alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}_\alpha^{-\kappa(i)}(x_i)).$$

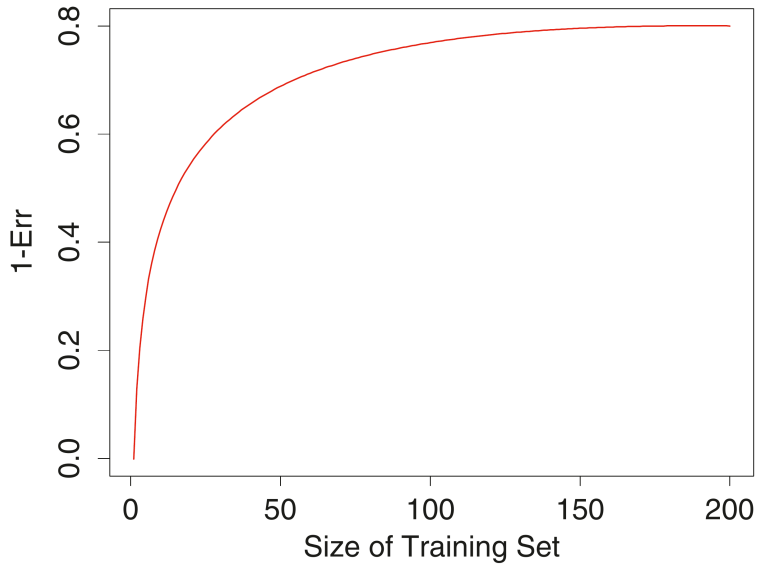


Figure 2.1: *Hypothetical learning curve for a classifier on a given task: a plot of* $1 - \text{Err}$ *versus the size* N *of the training set. The figure is retrieved from Figure 7.8 in [1]. If we use 5-fold cross-validation on a dataset of size 200, the training sets will consist of 160 elements. The expected test error for 160 is similar to the one for 200, so we could expect a good estimate. If we, however, have a dataset of size 50 observations, the training sets consists of 40 observations, and since the expected test error for 40 is quite a bit higher than for 50, we can expect the estimate to be too high for the whole dataset.*

2.9 Franke's Bivariate Test Function

Franke's function is a widely used test function in interpolation and fitting problems. It is the following weighted sum of four exponentials

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left\{\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right)\right\} + \frac{3}{4}\exp\left\{\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right)\right\} \\ &+ \frac{1}{2}\exp\left\{\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right)\right\} - \frac{1}{5}\exp\left\{\left(-(9x-4)^2 - (9y-7)^2\right)\right\}, \end{aligned} \tag{2.11}$$

where $x, y \in [0,1]$. As seen in Figure 2.2, Franke's function has two Gaussian peaks and a Gaussian dip.

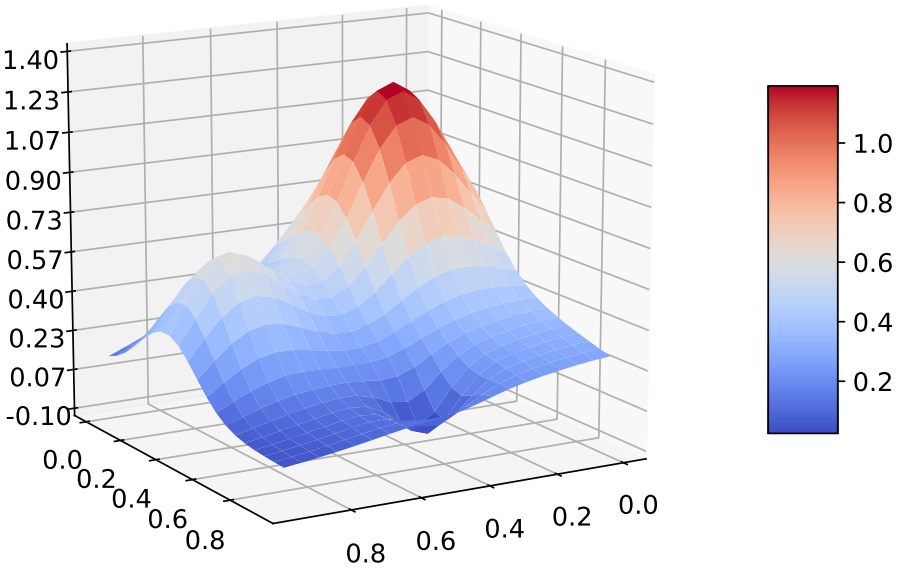


Figure 2.2: *The two-dimensional Franke's function, which is the sum of bivariate Gaussian functions given by Equation 2.11, is a test function used for verifying various interpolation and fitting algorithms. The surface consists of two Gaussian peaks of different heights and a smaller Gaussian dip.*

3 Method

3.1 Fitting with Multivariate Polynomials

Our first step in the analysis of regression and sampling methods, is to perform regression analysis on the Franke function, given by Equation (2.11). That is, we will try out a polynomial fit. A one-dimensional polynomial of second degree is on the form

$$f(x) = \beta_1 + \beta_2 x + \beta_3 x^2,$$

whereas a two-dimensional polynomial of second degree is on the form

$$f(x, y) = \beta_1 + \beta_2 x + \beta_3 y + \beta_4 xy + \beta_5 x^2 + \beta_6 y^2$$

We will try a polynomial fit with an x and y dependence and with different degrees p. This means that the design matrix will have rows of the form

$$\left[x^0 \cdot y^0, x^1 \cdot y^0, x^0 \cdot y^1, x^2 \cdot y^0, x^1 \cdot y^1, x^0 \cdot y^2, ...\right]$$

3.2 Standardizing data

For all predictive models in all of the analysis, a standardization has been done on the resulting design matrix for several reasons. The standardization of the design matrix involves subtracting the mean and dividing by the standard deviation column-wise. Firstly, this results in easier comparison between the confidence intervals of the parameters, since as each feature is standardized, the parameters are not sensitive to the unit or the magnitude of the features. Further, regularizing models such as Ridge and Lasso are especially sensitive to the magnitude of the features, as the parameters are being penalized according to Equation 2.4 and Equation 2.7. Larger parameters are thus susceptible of being penalized more artificially by their magnitude, which is counteracted by standardizing. Lastly, iterative optimizers used for models such as Lasso can have trouble establishing convergence among differently sized features. Standardization therefore speeds up the building of the model from a computational point of view.

3.3 Performance Metrics and Analysis Approach

As mentioned in the introduction, the central performance metrics in this projects are the MSE, given by Equation (2.8), and the R^2 score, given by Equation (2.9). However, since the latter is closely related to the former, we will emphasize and mainly study the MSE for the various regression methods.

In analysing the regression methods on Franke's function, we will try to fit models of different complexities, that is, different polynomial degrees in this case. We first explore the effect of added noise (irreducible error) to the data samples from Franke's function. The next item of the study is resampling techniques. We start by simply dividing the data samples into a training and test set, and find the MSE for both the training and test data. This will give some insight into how well the model fits the data, specifically, if the model over- or

underfits. For these first items of the study, we will use the OLS regression, described in Section 2.2.

Next, we will add the k-fold cross-validation resampling technique, described in Section 2.8, and study the training and test MSE when the data is resampled.

Ridge regression, described in Section 2.3, and Lasso regression, described in Section 2.4, are regularization methods that adds a penalty in order to solve ill-posed problems or prevent overfitting. We will analyse the regularization parameter (penalty) by finding confidence intervals that shows how the predictors are affected by the penalizing.

A bias-variance decomposition for all regression methods will also be carried out, as this will indicate which method and model, in terms of complexity and also penalty for the regularization methods, is the best.

3.4 Cross-Validation for Best Model Selection

In order to find the best model, we use k-fold cross-validation, as described in Section 2.8. Algorithm 1 below shows the general procedure for k-fold cross-validation for best model selection.

Algorithm 1 k-fold cross-validation for best model selection

Input
training data $S = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)$
integer k
Shuffle S randomly
Partition S into $S_1, ..., S_k$
for each model **do**
    for fold= 1 **to** k **do**
        Fit model on every fold but one
        Evaluate model on the held out fold
        Store the training and validation error, $\text{err}_{T;\text{fold}}$ and $\text{err}_{V;\text{fold}}$
    end for
    Compute and store the average error across all folds;
    $1/k \sum_{\text{fold}=1}^{k} \text{err}_{T;\text{fold}}$ and $1/k \sum_{\text{fold}=1}^{k} \text{err}_{V;\text{fold}}$
end for
Pick the model with the lowest average validation
Evaluate its performance on a final, held out test set

3.5 Implementing OLS and Ridge

OLS and Ridge has been implemented as python subclasses of a custom class `LinearModel`, which contains methods for general statistical analysis; such as calculating the MSE, as given by Equation (2.8), and the R^2 score, as given by Equation (2.9). The subclasses implement each a method for fitting and for prediction. `Numpy`'s `linalg` package has been used for numerically calculating the linear projection for OLS and Ridge, given by Equation 2.1 and Equation 2.5, respectively. Especially, the psudo-inverse `numpy.pinv` has been used for the matrix inversion, as this is a much more numerically stable method than numpy's

standard `numpy.inv`. For small datasets and a huge number of features, the design matrix approaches being singular, and ordinary methods of inversion becomes ill-conditioned. The pseudo-inverse tends to perform better here.

The prediction methods of the OLS and Ridge class also calculates the confidence interval for each parameter. To calculate the confidence interval, one must first establish the variance of the parameters, which we have analytical results for OLS and Ridge, Equation 2.2 and Equation 2.6. We estimate the irreducible error with the MSE on test-data with a correcting factor to make the estimate unbiased, such as

$$\tilde{\sigma}^2 = \frac{N}{N-P}\text{MSE}_{\text{train}} \tag{3.1}$$

where N is the number of data points and P is the number of parameters being fitted. In the case of Ridge, the effective number of parameters given by Equation (2.10) are being used instead of P.

The confidence interval for β_j is then established by

$$x = \beta_j \pm t_{0.05}^{N-P}\sqrt{\tilde{\sigma}_j^2} \tag{3.2}$$

where $\tilde{\sigma}_j^2$ is the j-th diagonal element of the estimated covariance matrix (Equation 2.3) and $t_{0.05}^{N-P}$ is the 5% percentile of the t-distribution.

3.6 Implementing Lasso

Lasso has been implemented as a subclass of `LinearModel` as well, though the functionality is implemented using `sklearn`'s Lasso regression. The number of iterations have been increased to 1 million and `warm_start` is set to true in an effort to speed up the computation.

3.7 Examining the Bias-Variance Tradeoff

The Bias-Variance tradeoff is evaluated in a Monte-Carlo fashion by sampling a $N = 300$ reference data set from Franke's function added stochastic term $\varepsilon \sim N(0, 0.25)$. 30 new data sets are then resampled from the same distribution, building a predictive model on each new resample. This ensemble of models are then used to predict on the reference sample. A point wise estimate for the variance of the model for each x_i in the reference sample can be calculated by

$$\text{Var}(\tilde{f}(x_i)) = \frac{1}{30}\sum_j((\tilde{f}_j(x_i) - \bar{\tilde{f}}_j(x_i))^2) \tag{3.3}$$

where $\tilde{f}_j$ is the model made from the j-th resample. Similarly, the squared bias is calculated as

$$\text{Bias}^2(\tilde{f}(x_i)) = (\frac{1}{30}\sum_j(\tilde{f}_j(x_i)) - f(x_i))^2 \tag{3.4}$$

where $f(x_i)$ is the real, noiseless Franke's function.

The final bias/variance is calculated by averaging over the pointwise estimates

$$\text{Bias}^2(\tilde{f}) = \frac{1}{N}\sum_i \text{Bias}^2(\tilde{f}(x_i)) \tag{3.5}$$

$$\text{Var}(\tilde{f}) = \frac{1}{N}\sum_i \text{Var}(\tilde{f}(x_i)) \tag{3.6}$$

As this is a very data-intensive procedure, it is not suited for analysis on real-world data, where bootstrap-methods and similar are favored. However, as means of benchmarking the OLS, Ridge and Lasso, it is sufficient.

3.8 Model selection for Terrain data

Before any fitting was done on the terrain data, a very aggressive 80-fold down-sampling was done in order to reduce the huge amount of data and make the fitting procedure feasible, especially for the Lasso model which tend to be rather time consuming for a small λ. The down-sampling was performed by averaging the terrain data over 80×80 domains, decreasing the data by a factor $\frac{1}{80^2}$. The down-sampled data in then split using 5-fold cross validation. OLS, Ridge and Lasso are then fit to each training set and MSE calculated on testing set.

OLS was fitted to the data using model complexities between 1 and 10. Ridge and Lasso where fitted using combinations of selected model complexities and penalties. The best model was selected from the combination of complexity and penalty that yielded the lowest MSE. The R^2 score was not regarded, as we felt it was redundant to include both metrics.

4 Results and Discussion

The programs containing the implementation of the methods discussed in the previous section, as well as the results presented in this section, can be found at the GitHub repository

https://github.com/nicolossus/FYS-STK4155/tree/master/Project1

4.1 MSE, Bias and Variance of OLS

Table 4.1 tabulates the training MSE and training R^2 score when performing a standard least squares regression analysis using polynomials in x and y up to fifth order to fit Franke's function with N data samples and added normal ($\sim N(0, \sigma^2)$) stochastic noise. As seen in the table, OLS performs better when σ^2 is smaller, that is, less added noise. For $N = 100$ data samples and $\sigma^2 = 0.001$, the model has a low MSE and high R^2 score, which is favourable as this indicates that the model explains the variability of the response data well. The added noise, $\sigma^2 = 1$, makes a huge impact on the performance, and is even worse when the number of data samples is larger.

Table 4.1: *Training MSE and training R^2 score for OLS using a two-dimensional polynomial fit of up to fifth degree on Franke's function with N data samples and added normal stochastic noise, $\sim N(0, \sigma^2)$.*

N	σ^2	MSE	R^2
100	0.001	0.002	0.980
100	1	0.106	0.490
1000	0.001	0.002	0.976
1000	1	0.213	0.206

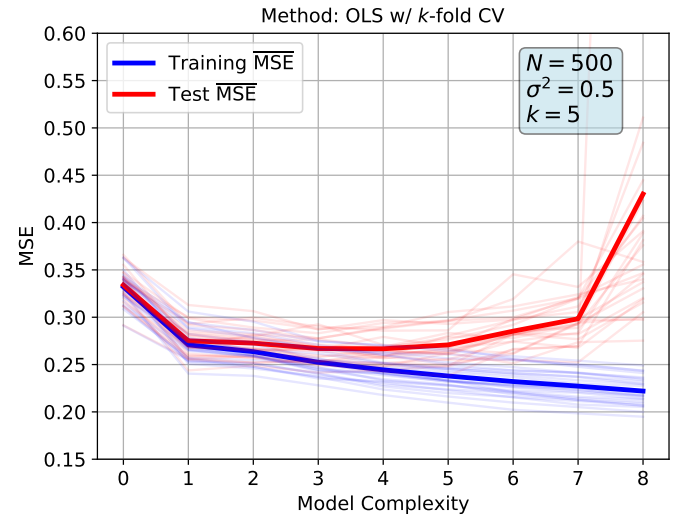
Table 4.2 tabulates the training and test MSE is tabulated for a test case with $N = 300$ data samples from Franke's function with an added noise term $\varepsilon \sim N(0, \sigma^2)$. The variance $\sigma^2 = 1.0$ and the model complexity (polynomial degree) is 7. The data is split into 75% train and 25% test sets. This table serves as a simple demonstration of the splitting of data, and the result of training on one part and testing on the other. As expected, the training MSE is lower than the test MSE. Note that the training MSE is under less than the irreducible error of value 1 in this case. This is a sign of severe overfitting, since this error is indeed irreducible and we can't hope to make a model which performs better. This is a result of too high model complexity(7'th degree with interacting terms), so the model tends to fit the noise as well as the general structure of the data. The test MSE is as expected higher than σ^2, since this is MSE on unseen data, and thus the random noise of this sample will not coincide with the overfitting.

Table 4.2: *Training and test MSE for OLS on Franke's function with an added noise term, $\varepsilon \sim N(0, \sigma^2)$, and $N = 300$ data samples split into 75% training and 25% test data. The fit is done with a model complexity (polynomial degree) of 7.*

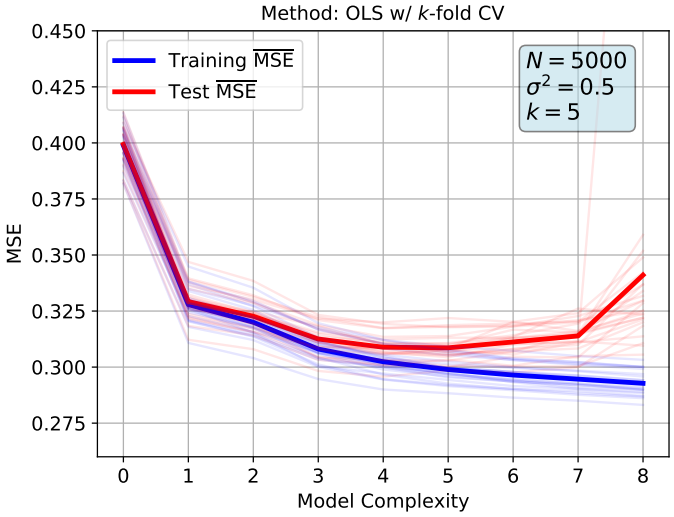
N	σ^2	Model Complexity	Training MSE	Test MSE
300	1.0	7	0.839	1.372

In Figure 4.1, the MSE is shown as a function of varying model complexity for the OLS method using 5-fold cross-validation. Data sets has been samples from the Franke function several times, yielding an ensemble of OLS for each choice of complexity. The test and training MSE of the different models are plotted as transparent red and blue lines, respectively. The opaque line are the average test and training MSE between the models. These figures highlights the result already discussed in the previous paragraph. The training MSE drops off monotonically as the model tends towards interpolating the data with noise and everything. The test MSE however attains a minimum at model complexity 3 and 5 for $N = 500$ and $N = 5000$, respectively. We see that a larger amount of data allows us to go to a higher complexity before overfitting. This is because with more data, individual outliers will have less impact on the overall fit. One can intuitively think that the random noise average out when the data set becomes large enough.

Moreover, another interesting detail emerges as we look at the individual models fitted on independent samples. As the model complexity increases, the test MSE tends to be more different between the models. This is a result of the bias variance tradeoff, established in Section 2.7. As the complexity increases, each models becomes more sensitive to each individual sample, as the specific noise pattern determines how the model behaves to a large degree.



(a)



(b)

Figure 4.1: *The* MSE *for both training and test data as a function of varying model complexity. The transparent red and blue lines are respectively test and training MSE of models trained on individual resamples from the Franke function. The opaque red and blue lines shows the average* MSE, $\overline{\text{MSE}}$. *The method used here is OLS with 5-fold cross-validation where the sample set in* **(a)** *is* $N = 500$ *data points, and in* **(b)** $N = 5000$ *data points.*

Figure 4.2 shows the bias-variance tradeoff, as just explained, even better. As the complexity increases, the bias diminishes as the higher flexibility allows for a model that follows more closely to the real Franke function. However, the variance increases with complexity as the models starts to fit the random noise more and more. The sum, being the MSE, reveals the optimal tradeoff between the two. The resulting optimal complexity of 5 does coincide with the previous result of complexity 3 for a similarly sized dataset. This is most likely due to statistical variance, as only one reference datasample was used in the MC-simulation when producing Figure 4.2.

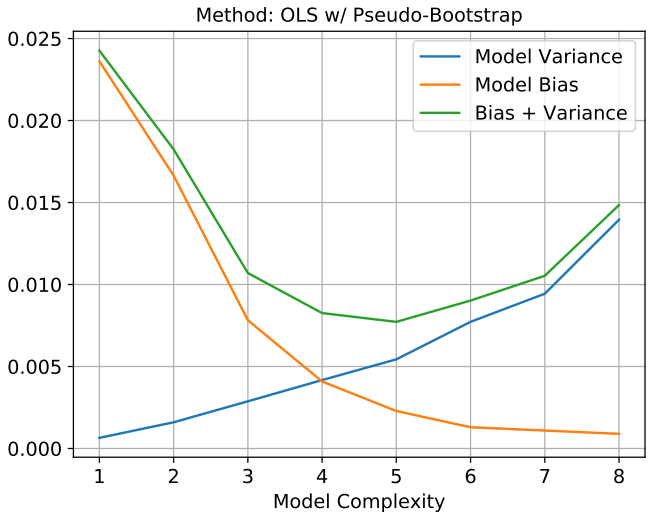


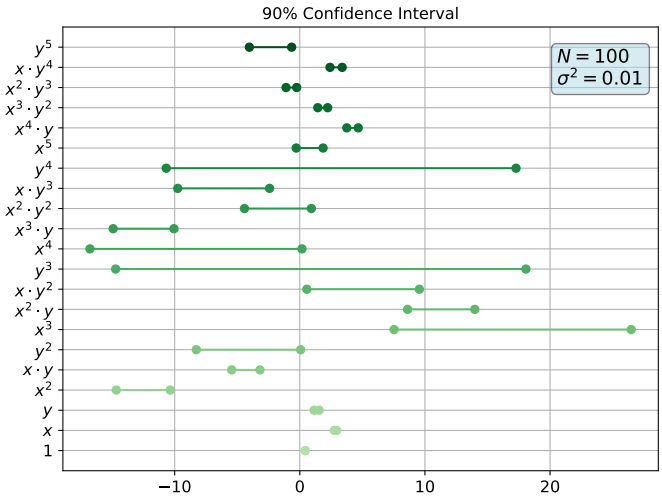
Figure 4.2: *Variance-Bias tradeoff as a function of model complexity for OLS calculated using a Monte-Carlo approach 3.7. The calculation is based on 30 MC samples of size $N = 300$ each, with noise $\sigma^2 = 0.5$*

4.2 Confidence Interval of OLS

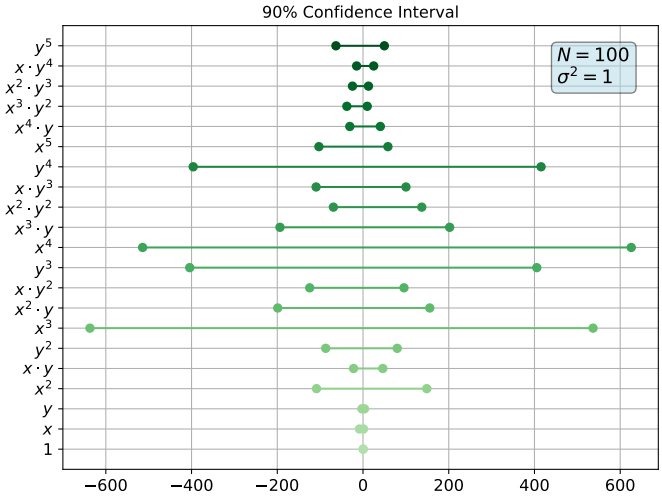
In Figure 4.3, the 90% confidence interval for each parameter for OLS fitted to data of size $N = 100$ from the Franke function is presented. For $\sigma^2 = 0.01$, several interesting features can be observed. Firstly, all the low degree parameters of the model is very significant, as the interval is very narrow. This is not too surprising, as they describe the simplest variation of the Franke function. The increasingly higher order terms are less and less significant, with the exception of some of the interacting terms. Especially the 5'th degree terms containing at least x and y are very significant. This is interesting, as it reveals that the structure of the Franke function, and by extrapolation the terrain, is very dependent on interacting terms and are therefore a necessity for a good model.

Moreover, high degree terms purely in x or y have a exceptionally low degree of significance. Note that y^3, x^4, x^2y^2, y^4 and x^5 all have 0 laying inside its confidence interval, raising the question whether these features are relevant for the model at all.

For $N = 100$, $\sigma^2 = 1$, all the intervals broaden as the increase in irreducible noise propagate to the variance of the parameters, evident of the estimate of σ^2 given by the MSE. Similarly, Figure 4.4 shows the same statistics, but for $N = 1000$. The increase of data has the opposite effect, and tends to shrink the intervals. This makes sense intuitively, as more data sets ground for a more confident estimate of each parameter. It can also be seen more mathematically from Equation 2.2, as the variance of each parameter is proportional to the diagonal elements of $(X^TX)^{-1}$. As the amount of data increases, these diagonal elements will decrease.

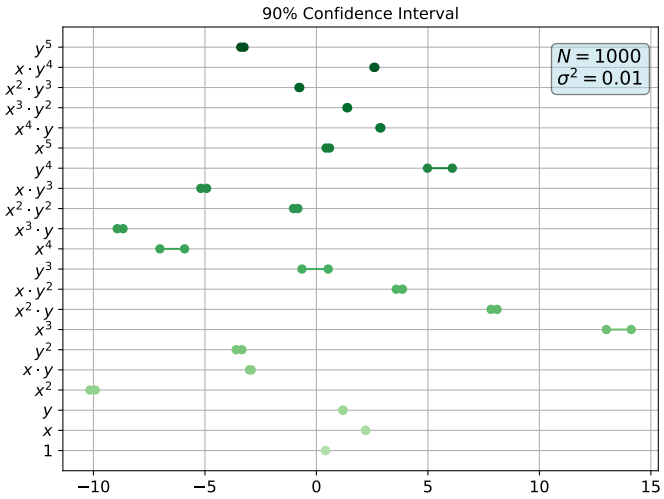


(a)

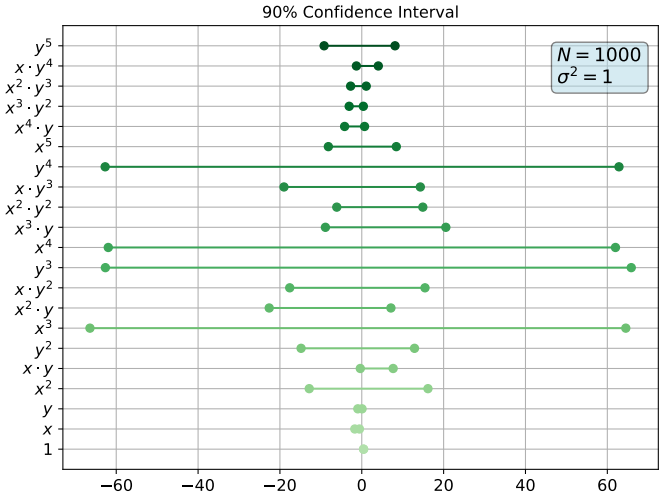


(b)

Figure 4.3: 90% *confidence interval for each parameter for OLS fitted to data from the Franke function.*



(a)

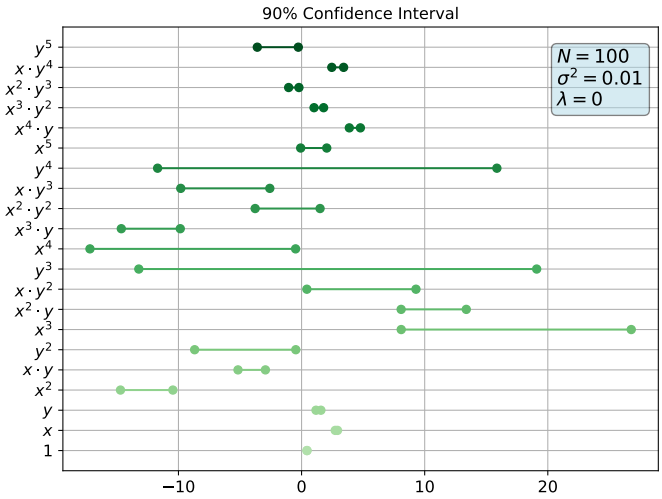


(b)

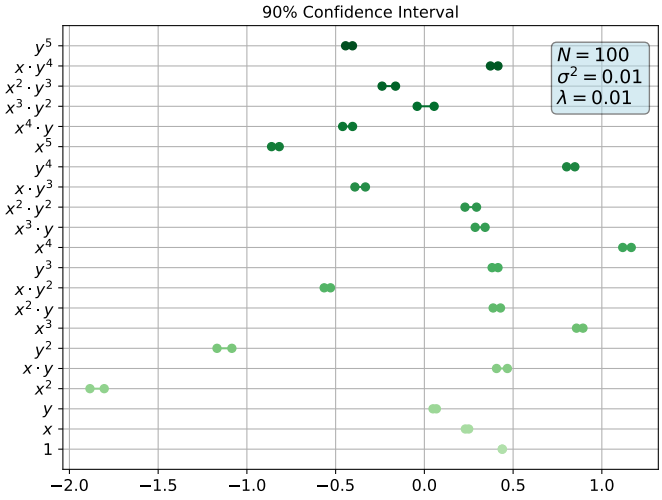
Figure 4.4: 90% *confidence interval for each parameter for OLS fitted to data from the Franke function.*

4.3 Shrinkage of the Parameters

In Figure 4.5, we see yet again the same statistics as in Figure 4.3, but now for an unpenalized and a penalized Ridge model on the noisy data from Franke's function. As can be seen, the confidence interval of the unpenalized Ridge is very similar to OLS, as it is in fact mathematically identical for $\lambda = 0$. However, an interesting effect can be seen for the penalized model in Figure 4.5(b). The penalty makes a small perturbation to the already significant terms. Simultaneously, the broad intervals corresponding to not significant parameters are made narrow, with the center of the interval generally moving towards zero. This highlights Ridge's effect of introducing a little bias in order to greatly decrease the variance.



(a)



(b)

Figure 4.5: 90% *confidence interval for each parameter for Ridge, both unpenalized and penalized. The penalty is* $\lambda = 0$ *and* $\lambda = 0.01$*, respectively.*

Figure 4.6 shows the coefficients β_j as a function of the common logarithm of the regularization parameter (penalty) $\log_{10}(\lambda)$ for Ridge regression. This gives another viewing angle to the same phenomenon discussed in the previous section. The figure shows that with increasing penalty term, the parameters are being smoothly shrunk towards zero. The higher order terms are shrunk more aggressively towards zero, which we from Figure 4.3 know corresponds to the least significant parameters. This means the parameters less important for the response are penalized more, which is the expected behavior of Ridge.

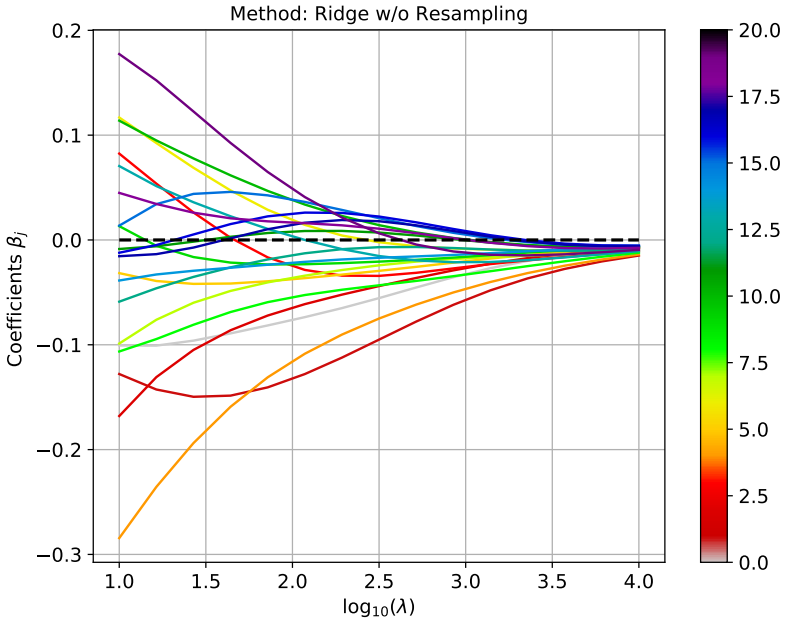


Figure 4.6: *The jth coefficient, β_j, as a function of the common logarithm of the penalty, $\log_{10}(\lambda)$, where the (integer) value of the j's are given by the colorbars to the left of the plots. The regularization method is Ridge.*

Figure 4.7 shows the coefficients β_j as a function of the common logarithm of the regularization parameter (penalty) $\log_{10}(\lambda)$ for Lasso regression. Being a regularized model similar to Ridge, Lasso penalized the L1 norm of the parameters rather than L2, which have some nontrivial consequences. As can be seen, instead of smoothly shrinking the parameters, the parameters are brought all the way to zero, effectively eliminating any given parameter for a high enough penalty. This has the effect of getting completely rid of features, which can be hypothesised to be favorable in the case that you have several features completely unimportant to the response variable. From Figure 4.3, the confidence intervals centered around zero might correspond to such features.

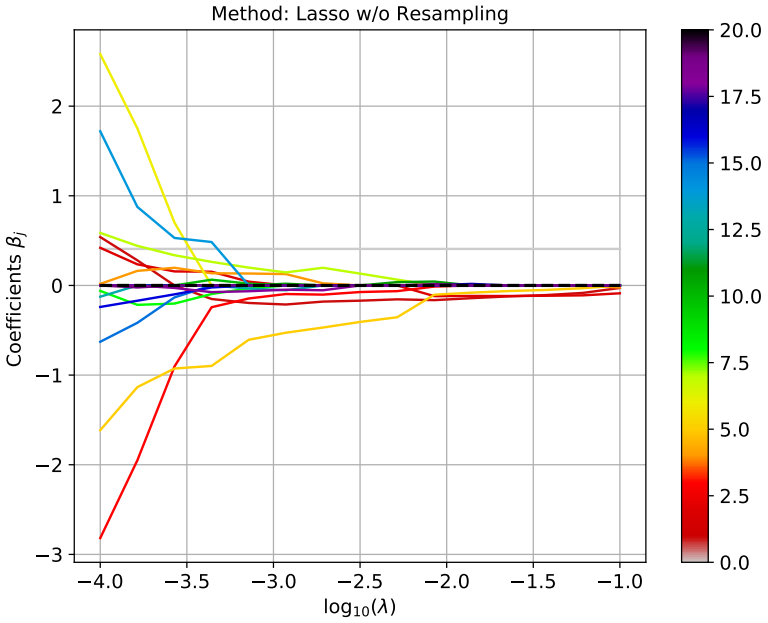


Figure 4.7: *The jth coefficient, β_j, as a function of the common logarithm of the penalty, $\log_{10}(\lambda)$, where the (integer) value of the j's are given by the colorbars to the left of the plots. The regularization method is Lasso.*

4.4 Model Selection for Ridge

In Figure 4.8, a similar bias-variance analysis as Figure 4.2 can be seen. The bias-variance is plotted as a function of the log of the penalty λ, for a fixed model complexity of 9. As already shown, the variance decreases and the bias increases as a function of penalty. Trading bias for variance is not interesting on its own. However, we see that it is actually worthwhile as the sum of the variance and bias, the MSE, has a minimum smaller than the unpenalized model. Thus, we obtain a better model using Ridge rather than OLS on the Franke function.

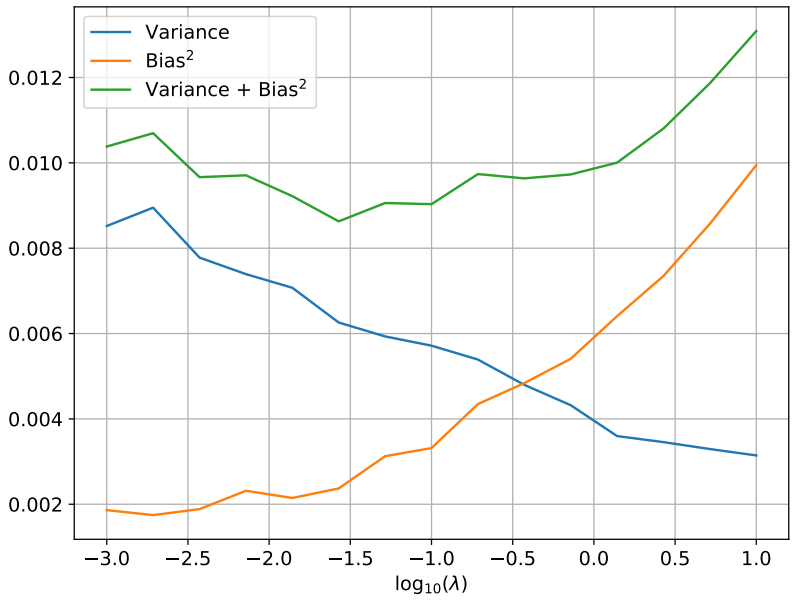


Figure 4.8: *Variance-Bias tradeoff as a function of penalty for Ridge calculated using a Monte-Carlo approach 3.7. The calculation is based on 30 MC samples of size $N = 300$ each, with noise $\sigma^2 = 0.25$ and model complexity 9*

With confidence that Ridge can produce good models, we do a models selection based on minimizing test MSE with respect to model complexity and penalty λ using 5-fold cross validation. Figure 4.9 demonstrates such an analysis, based on a data set of size $N = 500$ and $\sigma^2 = 0.25$. The best model was yield for the lowest complexity of 3, with a penalty term of about $\lambda = 10^{-0.3}$. The corresponding training MSE was about 0.278, better that the best model from Figure 4.1.

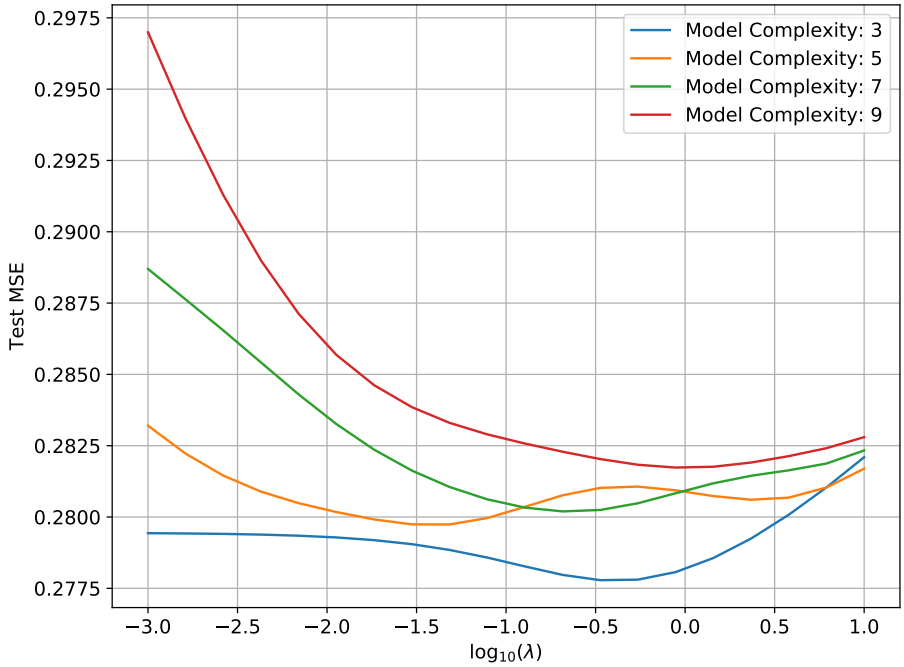


Figure 4.9: *Test MSE of Ridge models using different penalties and model complexity. The MSE has been calculated using 5-fold CV. The data is of size $N = 500$ and sampled from the Franke function with added noise $\sigma^2 = 0.5$*

4.5 Model Selection for Lasso

In Figure 4.10 the bias-variance tradeoff similar to Figure 4.2 is show for Lasso as a function of penalty. With the Lasso, the plot is sadly much more noisy than the previous bias-variance tradeoff for OLS and Ridge, though it is still possible to see the same general trend: Variance is traded for bias when penalty increases, with a net gain in accuracy due to the decreasing MSE.

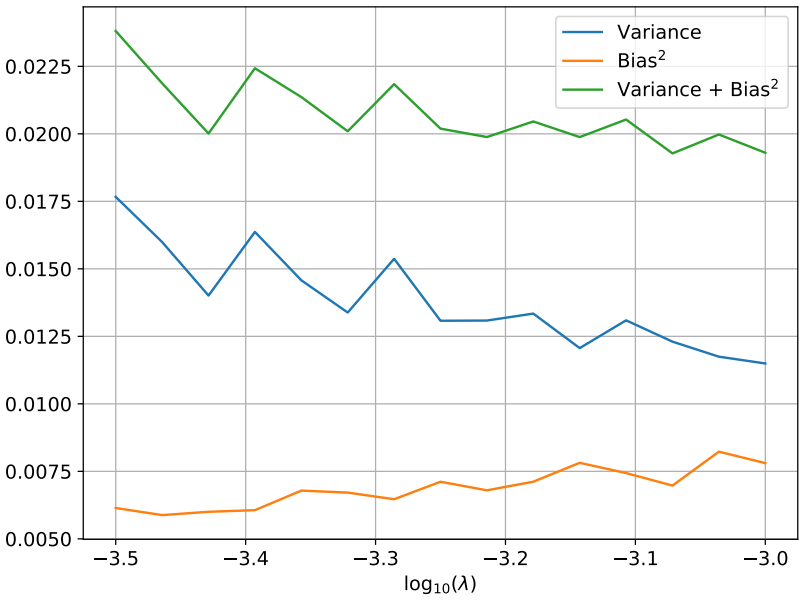


Figure 4.10: *Variance-Bias tradeoff as a function of penalty for Lasso calculated using a Monte-Carlo approach 3.7. The calculation is based on 30 MC samples of size $N = 300$ each, with noise $\sigma^2 = 0.25$ and model complexity 9*

Figure 4.11 shows the training MSE calculated using 5-fold CV on Lasso models of varying model complexity and penalty. Also this plot is plagued by noise, even though an attempt at averaging the MSE by running several independent runs and averaging was made. However, we still get some relevant information from the plot, with some uncertainty in how legitimate the result is given the noise. The best model surprisingly had the high complexity of 9, clocking in with the lowest MSE yet of about 0.259. Compared to Figure 4.1 and Figure 4.9, this is the best result yet. This could be a deviation, but at any rate the next best model of complexity 3 is still better than OLS and Ridge. This result strengthens the hypothesis from Figure 4.3 that some of the higher order parameters are not important for our response, and are better left out by Lasso more aggressive penalty with respect to Ridge.

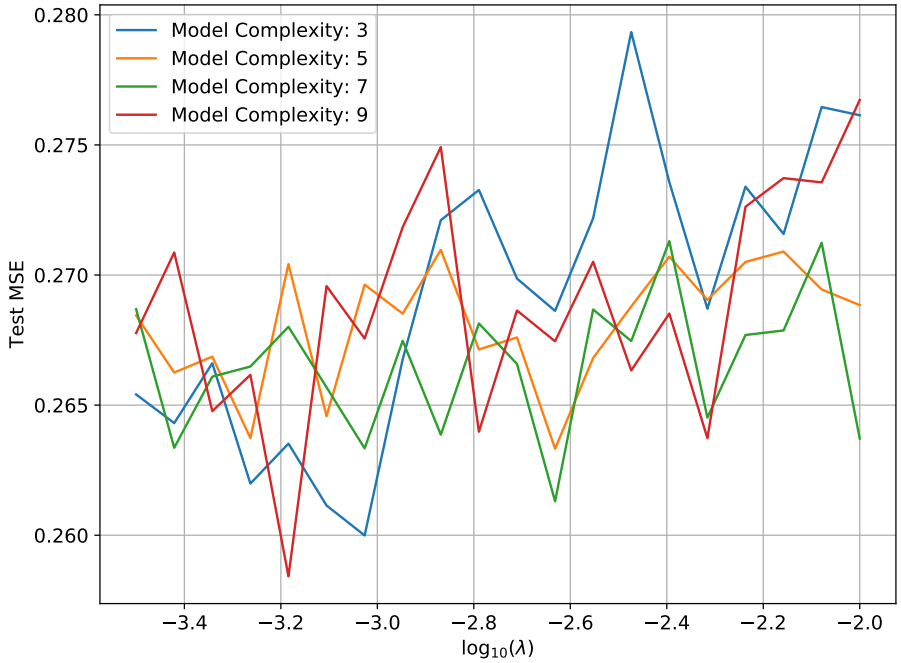
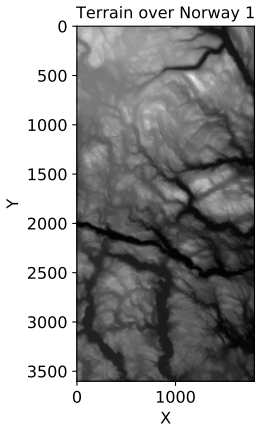


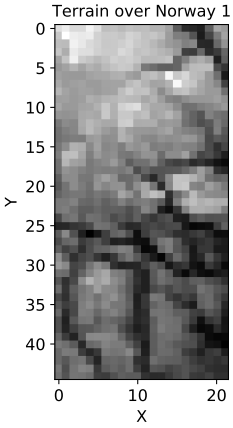
Figure 4.11: *Test MSE of Lasso models using different penalties and model complexities. The MSE has been calculated using 5-fold CV. The data is of size* $N = 500$ *and sampled from the Franke function with added noise* $\sigma^2 = 0.25$

4.6 OLS on terrain data

Figure 4.12 Shows the original and down-sampled terrain data that we will attempt fit various models to. The aggressive down-sampling was necessary, as the overwhelming amount of data cause the fitting procedure to become very slow, especially for Lasso.



(a)



(b)

Figure 4.12: *Original terrain data and down-sampled terrain data. The downsampling is done by taking the average of the terrain data for* 80×80 *pixel domains.*

Figure 4.13 shows the test MSE of OLS models for various model complexities on the final down-sampled terrain data, using 5-fold CV. The minimum MSE was found to be 33300 for OLS, and occurred for complexity 5.

Figure 4.14 visualizes the prediction of the best OLS model predicting on the whole set of data. Although the predicted terrain looks much smooth with respect to Figure 4.12(b), the most eye catching features are still visible, such the rivers in the top and bottom right of the picture.

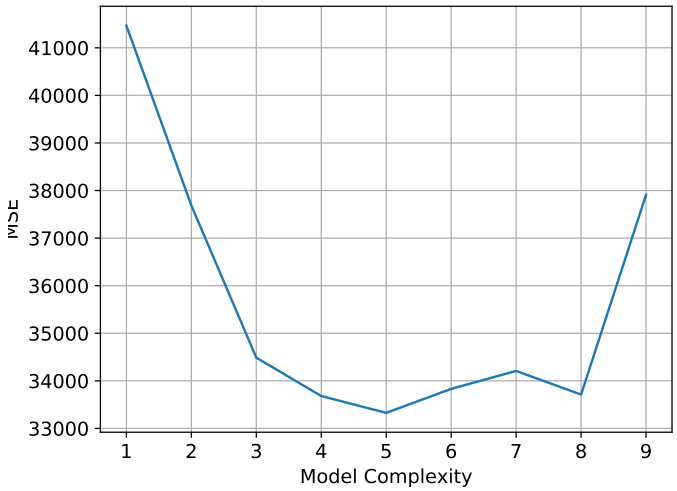


Figure 4.13: *Test MSE of OLS on the down-sampled terrain data using 5-fold CV and varying model complexities. The minimum MSE was 33300 and occurred for complexity 5*

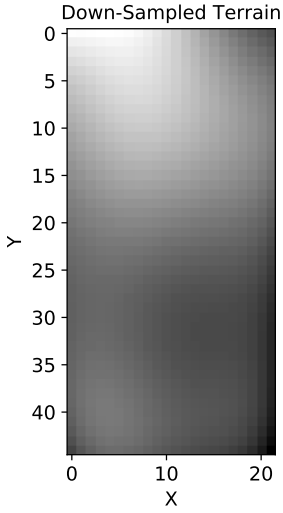


Figure 4.14: *Visualisation of the best OLS fit on the down-sampled terrain data, MSE = 33300*

4.7 Ridge on Terrain Data

Figure 4.15 shows the test MSE of Ridge models for various model complexities and penalties on the final down-sampled terrain data, using 5-fold CV. The minimum MSE was found to be 31110 for Ridge, and occurred for complexity 7 and $\lambda = 10^{-6.7}$. This is a lower MSE than OLS, which indicates that is a better model. The complexity is also higher, which means the model can adapt more closely to the terrain. The Ridge is able to go to higher complexities than OLS without overfitting, since the penalty limits the amount the model changes because of the noise.

Figure 4.16 visualizes the prediction of the best Ridge model predicting on the whole set of data. As with Figure 4.14, the predicted terrain is much smoother than the original data, although visually the Ridge-model shows more complex features in the terrain, and is not as smooth as OLS.

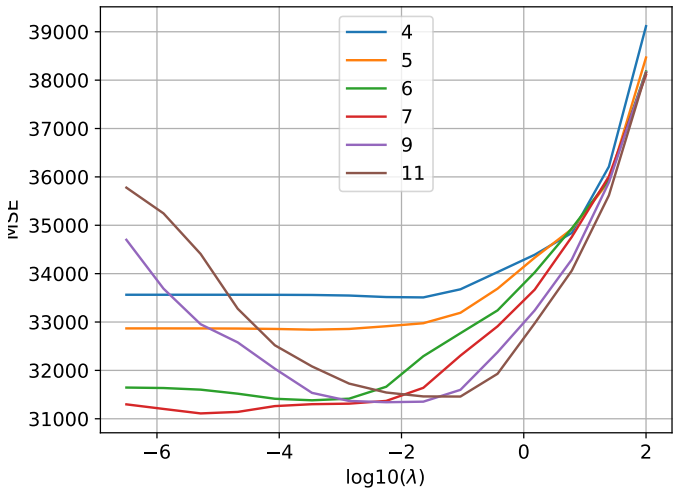


Figure 4.15: *Test MSE of Ridge on the down-sampled terrain data using 5-fold CV, varying model complexities and penalties. The minimum MSE was 31110 and occurred for complexity 7 and* $\lambda = 10^{-6.7}$

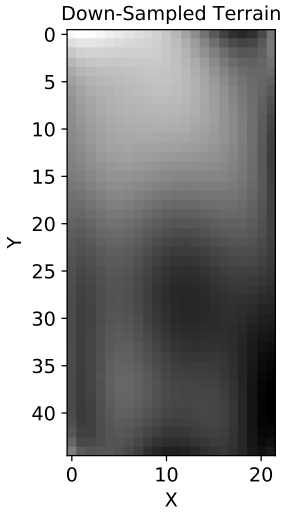


Figure 4.16: *Visualisation of the best Ridge fit on the down-sampled terrain data, MSE = 31110*

4.8 Lasso on Terrain Data

Figure 4.17 shows the test MSE of Lasso models for various model complexities and penalties on the final down-sampled terrain data, using 5-fold CV. The minimum MSE was found to be 32652 for Lasso, and occurred for complexity 9 and $\lambda = 0.053$. This is a better result then OLS, but not as good as Ridge.

Figure 4.18 visualizes the prediction of the best Lasso model predicting on the whole set of data. It terms of visually complexity and its similarity to the original data, it is somewhere in between Figure 4.14 and Figure 4.16, just as the MSE would imply.

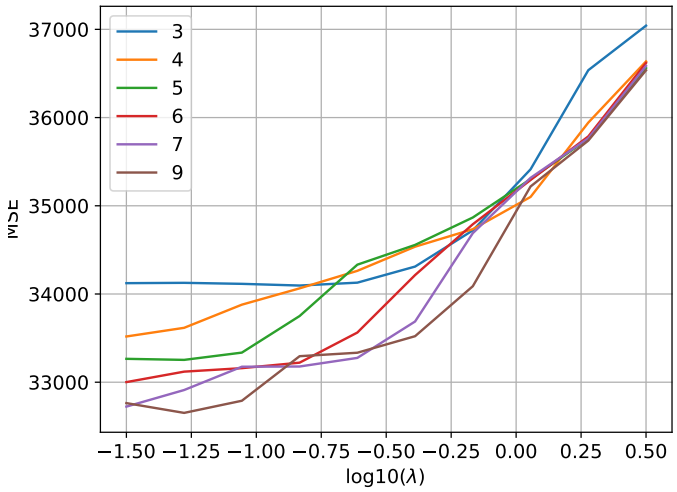


Figure 4.17: *Test MSE of Lasso on the down-sampled terrain data using 5-fold CV, varying model complexities and penalties. The minimum MSE was 32652 and occurred for complexity 9 and* $\lambda = 0.053$

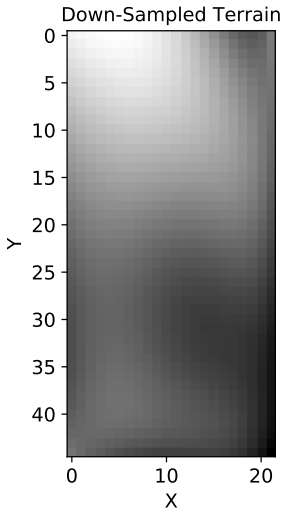


Figure 4.18: *Visualisation of the best Ridge fit on the down-sampled terrain data, MSE = 32652*

5 Conclusion

From the benchmarking of OLS, Ridge and Lasso on the Franke function, which served as a toy model for terrain data, it was shown from Figure 4.9 and Figure 4.11 that they both performed better than OLS. Figure 4.11 shows that, possibly outside of statistical significance because of noise, Lasso performed better than Ridge when fitting Franke's function. This result is backed by the hypothesis based on the confidence intervals in Figure 4.3; that some of the parameters are not significant at all and is better left out of the model. From this, Lasso was hypothesised to be the best choice to fit the Frank, since it also has the ability to penalize parameters all the way to zero, as shown in Figure 4.7.

In the end, Ridge was shown to work better for this specific terrain sample, with a MSE of 31110, followed closely by 32652 for Lasso. Both regularized methods were better than the regular OLS, with a MSE of 33300.

As some final thoughts, what is the achievement of our best model? As this was mostly an analysis of how the most common model of linear regression behaves and interact with data, the final model is not particularly interesting. It does not succeed at interpolating between the data points in the down-sampled terrain, as this information is readily available in the uncompressed terrain data. It would be foolish to consult the model, which was trained on the heavily compressed data. Without having done the analysis, we propose that the model also performs poorly on extrapolation. Leaving out a chunk of the terrain during training, and then predicting on this chunk will most likely not work, as there is no physical correspondence between the terrain and the coordinates: If the model was blind to a small valley in the terrain during training, it will not be able to predict the valley from the surrounding area.

However, the model might serve as, although poorly, a compressing of terrain data. Much like a JPEG, which uses sine and cosine function to predict the value of pixels, the model does the same with polynomials.

6 Future Work

A limitation of all the models explored in this project lies in the fact that they utilise polynomials that stretch over the whole domain. Fitting a single function expression over a large domain, especially if the response vary wildly (like terrain), is exceptionally hard. A large complexity is necessary to fit the terrain closely, but this can introduce oscillations in the model.

A much better, and more flexible alternative, is the use of splines. Splines are much more flexible in the sense they fit the terrain more locally, so one avoids the problem of fitting a single expression over the whole domain.

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. eng. Second Edition. Springer Series in Statistics. New York, NY: Springer New York, 2009. ISBN: 9780387848570 (cit. on pp. 2, 4, 6, 8).

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. eng. 1st ed. O'Reilly Media, Inc, 2017. ISBN: 9781491962299 (cit. on p. 5).