Regression analysis and re-sampling methods

Johan Nereng

Department of Physics, University of Oslo, Norway

Oct 8, 2019

Abstract

This project aims to examine various regression methods; OLS, Ridge and Lasso, in predicting terrain data, when combined with the Cross-Validation (CV) technique k-Fold CV. The prediction accuracy of these methods are mainly measured through mean square error (MSE) and R^2 scores, with an empathizes on examination of model complexity in terms of choice of polynomial degrees. The project concludes that model complexities corresponding to polynomial degrees of order four to eight yield the best predictions when measured on data generated by the Franke's Function (for a 30×30 with noise $\sim N(0, 0.3)$, a MSE of ~ 0.89 was measured). Similar predictions on real data (terrain on a scale of kilometers with 3601×1801 resolution), indicated that polynomials of degree two, and between six and ten, performed best. Finally, prediction errors on 500×500 squares were estimated roughly to 16% percent deviation in terrain height. The project was unable to draw decisive conclusions on choice of regression models due to indications of implementation errors in the core code used in the project.

Important note on terminology: *(Oct 17.) After a conversation with Morten today, I learned that I have been using the term "predictor" inaccurately, also in this project. A reader will most likely find that this project implies that a model such as* $\beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 y^2 + \beta_5 xy$ *has two predictors (x and y), while it in reality has six:* $1, x, y, x^2, y^2, xy$. *Please keep this misconception in mind when reading the project*

Authors' comment: *While doing this project, I have spent a great amount of time reading various literature and trying to understand the methods and techniques the project describes. Before doing so, I constructed what turned out to be a faulty core program. In vastly underestimating how well my core code functioned prior to deadline, I planned with too little time to properly debug the code, and had to abandon hopes of doing so a day before hand in to ensure that I produced a report which contained all the necessary ingredients. I've come to experience that I have forgotten a lot of Python functionality after having spent a year using only C and C++. Luckily my many hours of attempted debugging has made me recollect some of them. For future projects, I will need to start by writing the methods sections in order to get a clearer picture of the algorithms involved before attempting to program them.*

1 Introduction

Linear regression models are popular machine learning tools. Especially for educational purposes. Methods employing such models use a set of data consisting of measurements of one or more independent variables, or predictors, and a corresponding dependent variable, or response. Estimating the relationship between the response and the predictors is the aim of regression analysis. This is achieved by minimizing what is known as a cost function - a function which describes the error between the model and the real data. For example, the most well known cost function is probably taking the sum of the square of the error. By representing this minimization problem on a matrix-vector form, methods from linear algebra may be applied, which greatly increases how easily one can find suitable parameters for the regression model.The aim of this project is to examine some of the properties of a specific type of linear regression models, namely bivariate polynomial regression. In addition, this projects seeks to examine the suitability of such models in predicting digital geographical terrain data.

Project flow: The project starts by doing a cursory mathematical description of multiple linear regression. How the problem may be represented on a matrix-vector form, as well as introducing key terminology and concepts. Then, the regression method Ordinary Least Squares (OLS) is introduced, with a brief description of computational speed and stability. Subsequently, the shrinkage methods Ridge Regression and Lasseo Regression are introduced through altering the cost function by a penalty term containing the shrinkage parameter λ. A brief general comparison of the two methods is lain, before moving on to describe the Re-Sampling technique k-Fold Cross Validation. Having described all of the methods applied in the project, measurements used in model evaluation is presented; MSE, R^2, and the confidence intervals, CI, of the regression coefficients β. Then the Bias-Variance trade-off is outlined, by examining the expected square deviation of the predictors. Lastly, Franke's Function is introduced - a function which this project utilizes in order to test the regression methods. Prior to presenting the results, a cursory explanation of the code is presented, followed by descriptions of how the different method implementations are evaluated. Based on these evaluations, a picture of the impact of parameters such as model complexity and λ on evaluation measurements is built up. Thereafter, real terrain data from two different locations in Norway is used to test the prediction ability of the regression methods, before the project rounds up by presenting it's main conclusions.

Main findings: The main findings of the project is that model complexities corresponding to polynomial degrees of order four to eight yield the best predictions when measured on data generated by the Franke's Function (for a 30×30 with noise $\sim N(0, 0.3)$, a MSE of ~ 0.89 was measured). Similar predictions on real data (terrain on a scale of kilometers with 3601×1801 resolution), indicated that polynomials of degree two, and between six and ten, performed best. Finally, prediction errors on 500×500 squares were estimated roughly to 16% percent deviation in terrain height. It is also worth mentioning that several of the produced results indicated faulty code implementation, as large deviations from expected measurements, as well out-of-domain measurements were recorded.

Tools used: For program implementation, this project used Python 3.7, with NumPy, as well as selected functionality from Scikit-learn. A number of books, web-pages and articles - most of which are listed under references - were drawn upon to build the theoretical basis for the project, found under the methods section. All code and results may be found in the author's github page .

2 Methods

2.1 Multiple Linear Regression

Usually, insights into the underlying mechanisms of the origins of a data set, will guide the the choices one makes when trying to model the relationship between the dependent and independent variables. If such insights suggest a linear relationship between response and target, the model describing that relationship should reflect this. When modeling for one explanatory variable, this means using multiple linear regression, while for more than explanatory variable, this would mean using multivariate linear regression. A response y, with approximation $\tilde{y}$, on predictor x with a $m-1$ degree linear approximation may be expressed as

$$z = \sum_{j=0}^{m-1} \beta_j f_j(x) + \epsilon = \tilde{z} + \epsilon \tag{1}$$

, where ϵ is the residual error between the model and the true response [7][p.19]. When the target-predictor relationship can be modeled by power functions, (1) leads to a polynomial fit, ie $f_j(x) = x^{j-1}$. For a data set $\{z_i, x_i\}_{i=0}^{n-1}$, where y_i is the response on predictor x_i, (1) results in n equations. In order to effectively solve for the regression coefficients (β_j), the set of equations may be represented by a the matrix-vector multiplication. For a polynomial of order $m-1$, $\boldsymbol{\beta} = [\beta_0, \beta_1, ..\beta_{m-1}]^T$, and for $n-1$ data points, $\boldsymbol{z} = [z_0, z_1, ..z_{n-1}]^T$, while $\epsilon = [\epsilon, \epsilon, ..\epsilon n-1]^T$. Lastly, the $n \times m$ matrix

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{m-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{m-1} \\ & & \dots\dots\dots\dots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{m-1} \end{bmatrix} \tag{2}$$

, known as a Vandermonde Matrix [1][p. 147-148], yield the vector matrix product:

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{3}$$

An outtake of the methods which may be applied to (3) in order to solve for the regression coefficients are discussed later. However, the Vandermonde Matrix - which is a special case of what is generally known as a design matrix - is limited to a univariate polynomials. If instead the response z_i is on two predictors (x_i and y_i), such as will later be used in this project, a bivariate polynomial approximation of z will result in a design matrix $\boldsymbol{X}$ where row i is on the form $[1, x_i, y_i, x_i^2, y_i^2, x_i y_i ...]$, with a total of $d = \binom{m+2}{m}$ coefficients.

2.1.1 Ordinary Least Squares (OLS)

OLS fits a function by minimizing the sum of the squares of the errors between a model and a data set. This results in the cost function

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 = \frac{1}{n}(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}^T\boldsymbol{\beta}) \tag{4}$$

which, if A has full column rank, has a unique solution [1][p.144] which satisfies the normal equations:

$$(\boldsymbol{X}^T\boldsymbol{X})\boldsymbol{b} = \boldsymbol{X}^T\boldsymbol{y} \tag{5}$$

Solving this corresponds to finding the minima of the derivative of the cost function with respect to $\boldsymbol{\beta}$:

$$\boldsymbol{\beta}^{OLS} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} = \boldsymbol{A}^{\dagger}\boldsymbol{y}. \tag{6}$$

This matrix inversion may then be carried out using for example SVD or LU decomposion. Alternatively, minimizing (4) may be accomplished using orthogonal transformation with QR decomposition, the details of which may be found in chapter one of Uri M. Ascher and Chen Grief's book [1]. The choice between the two approaches hinges on computational stability versus computational speed.

Computational Speed: The two approaches are as mentioned dissimilar in computational cost, measured in floating point operations (flops) - especially in overdetermined cases, where the matrix $\boldsymbol{X} \in \mathbb{R}^{n,m}$, has $m < n$. The normal equation solution uses $mn^2 + (1/3)n^3$ flops [1][p.146], while the QR decomposition requires $2mn^2 - (2/3)n^3$ flops [1][p.155]. Which means that the normal equations method is significantly faster than the QR approach.

Computational Stability: The conditioning of the two approaches, or stability with respect to input perturbation, are measured in conditioning number of the matrix representing the problem, $K(\boldsymbol{X})$. Higher $K(\boldsymbol{X})$ means less stable.

$$K(\boldsymbol{X}) = ||\boldsymbol{X}^{-1}|| \times ||\boldsymbol{X}|| \tag{7}$$

In order to acquire the normal equations, the product of $\boldsymbol{X}$ and it's transpose is used. Hence, it's conditioning number goes as $K(\boldsymbol{X}^T\boldsymbol{X}) = K(\boldsymbol{X})^2$. This squaring of the conditioning number is not required for the QR approach, which means that the QR method has a lower conditioning number, and is thus the more stable method.

2.2 Shrinkage methods

Alterations to the OLS cost function (4) may be done in order to assert some control of the bias variance trade off. Specifically, in order to reduce the variability of prediction errors associated with with fitting complex models. Shrinkage methods introduce a penalty parameter λ, which effectively shrinks the regression coefficients, thus decreasing bias towards towards data used in training the model. In this project, two such shrinkage methods are used, Ridge Regression and Lasso Regression.

2.2.1 Ridge Regression

The Ridge Regression cost function [6][p.22] minimizes the square of the residual sum with a quadratic penalty term:

$$C(\boldsymbol{\beta}) = \frac{1}{n}(\boldsymbol{z} - \boldsymbol{X}^T\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}^T\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{8}$$

, where $\lambda \geq 0$ is the penalty constant. As is evident from the equation, $\lambda = 0$ results in a solution similar to OLS. Higher values of λ results in increased shrinkage, and in fact each value λ will produce a different set of coefficient estimations [5][p.215]. This parameter may decrease variance when compared to OLS, especially when the dependency of the response is close to linear on the predictors and the number of coefficients is high. In such scenarios, minor perturbations in training data may cause large changes in the OLS estimated coefficients [5][p.219]. Originally, the shrinkage parameter was introduced to address rank deficiency [3][p.64] - which it does, as the method adds a non-zero constant to the diagonal elements.
The cost function (8) results in the following expression for the regression coefficients:

$$\boldsymbol{\beta}^{Ridge} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{z} \tag{9}$$

2.2.2 Lasso Regression

Lassso Regression is similar, but not identical, to Ridge Regression. Instead of a quadratic penalty term, Lasso Regression uses the absolute value of the coefficients [4]

$$C(\boldsymbol{\beta}) = \frac{1}{n}(\boldsymbol{z} - \boldsymbol{X}^T\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}^T\boldsymbol{\beta}) + \lambda||\boldsymbol{\beta}|| \tag{10}$$

The minimization problem based on this cost function has solutions which are non-linear to $\boldsymbol{y}$, and there is, in contrast to Ridge Regression, no closed form solution [3][p.68]. Thus, to determine model coefficients by Lasso Regression, a computing algorithm must be used. In addition, where Ridge Regression does not set any coefficients to exactly zero [3][p.68], Lasso Regression may do exactly that, which means that Lasso Regression may eliminate certain input parameters. This means that Lasso Regression produce *sparse* models - models which may only include a subset of the variables in the data set [5][p.219]. A consequence of this is that Lasso Regression generally produce models that are easier to interpret than Ridge Regression and OLS.

A general comparison between Ridge and Lasso may be studied further in literature, such a [5][p.223-224], on which the following is based. Simply put, Ridge regression leads to better prediction when performed on data where there are few predictors of roughly the same importance. However, for a large set of predictors, where a smaller subset of predictors are significantly stronger linked to the response than the rest, Lasso performs better. Especially so when the response has low dependency on the remaining subset.In this regard, it is however important to note that when analyzing a real data set, the number of significant predictors is unknown a priori. Therefor, in order to determine the method of regression, the use of re-sampling techniques such as cross validation or bootstrap is advisable.

2.3 Re-sampling procedure: k-Fold Cross-Validation

As other re-sampling procedures, the k-Fold Cross-Validation (KfCV)uses a limited training set to perform multiple model fittings in order to obtain additional information about model parameters [5][p.175]. In the case of this specific procedure, this is achieved by splitting the data into k, approximately equaled sized groups, normally after shuffling the data sets. Then, the first of those k sets is treated as a test set, or validation set, while the remaining $k-1$ sets are used to fit the model. This is then repeated k times, using a new subset for validation in each iteration [5][p.181]. In each iteration$i$, the mean square error, MSE_i (12), is calculated using the validation group as $\boldsymbol{z}$. After all iterations are complete, the Cross-Validation estimate is computed by:

$$CV_(k) = \frac{1}{k}\sum MSE_i \tag{11}$$

One of the applications of this procedure is finding a suitable value for the shrinkage parameter λ. Rather than For each value of a variety of λ-values, a KfCV is carried out, and the $CV_(k)(\lambda)$ computed. Thus, the λ which produced the lowest $CV_(k)(\lambda)$ may be applied in the finished regression based on the original data set.

2.4 Model evaluation

There are, as outlined above, several regression models to chose from. These models all have parameters that impacts their predictions, such as the number of polynomial coefficients and the regularization parameter. As there are many choices, a measurement of how well a regression model predicts the response corresponding to a real data point is needed. One such measurement is the mean square error:

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1} \tag{12}$$

This is a natural measurement to use, both because it's simple and because the cost function is based on this measurement when solving the normal equations. However, the MSE is data specific. An alternative is using the R^2, which is a normalized measurement:

$$R^2(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z}_i)^2} \tag{13}$$

, where $\bar{y}$ is the mean value of $\boldsymbol{y}$. As it is independent of scale, R^2 always returns a value between zero and one, making it easy to interpret.
Another way to measure how well the model works is by finding the confidence intervals for the regression coefficients. Following the train of thought from [3][p.47-49]: When using the normal equations for OLS, these may be found by using an analytic expression for the variances of the coefficients

$$Var(\boldsymbol{\beta}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\hat{\sigma}^2 \tag{14}$$

Where if σ is unknown, an unbiased estimate $\hat{\sigma}$ should be used. Having obtained the variance of the regression coefficients, the 95% confidence interval (CI) of β_i may be found by

$$CI_{0.95}(\beta_i) = \beta_i - 19.6v_j^{1/2}\sigma, \beta_i + 19.6v_j^{1/2}\sigma \tag{15}$$

Where v_j is the diagonal element of row j of the matrix $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$.

2.4.1 Bias Variance trade-off

For a response $\boldsymbol{y} = \boldsymbol{f} + \boldsymbol{\epsilon}$, with $\mathbb{E}[\boldsymbol{\epsilon}] = 0$ and $Var[\boldsymbol{\epsilon}] = \sigma^2$, the expected mean square error (MSE) of a regression model fit $\tilde{\boldsymbol{y}}$ from an input may be calculated by (see appendix 1):

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2 \tag{16}$$

$$= Bias^2(\tilde{\boldsymbol{y}}) + Var(\tilde{\boldsymbol{y}}) + Error \tag{17}$$

As indicates above, the first term is the squared bias. This corresponds to the average of how much the predicted response, $\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]$, deviates from the true mean of the real response, f_i. The second term is the variance, or the squared expected difference of the predictions about their mean [5][p.223]. When fitting a model to a particular data set, increased complexity or flexibility will tend to make highly accurate predictions on the data set. This corresponds to having a low bias. Simultaneously, such a model fit would have very high variance - or to put it differently, if small variations were made in the data set, the resulting model would drastically change. On the other hand, if the model does not predict accurately on it's own training data, the bias would be high, while the variance would be low - perturbations in input would result in only minor changes of the regression.

2.5 Franke's function

A popular function for evaluation of polynomial approximations is the The Franke's function. Named after Richard Franke's, and published in his report "*A Critical Comparison of Some Methods for Interpolation of Scattered Data*" [2], the function is a weighted sum of four exponential;

$$\begin{aligned} f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\right)+\frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49}-\frac{(9y+1)}{10}\right) \\ +\frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\right)-\frac{1}{5}\exp\left(-(9x-4)^2-(9y-7)^2\right) \end{aligned} \tag{18}$$

, defined on the domain $x, y \in [0, 1]$. A visualization of (18) can be seen in figure 1

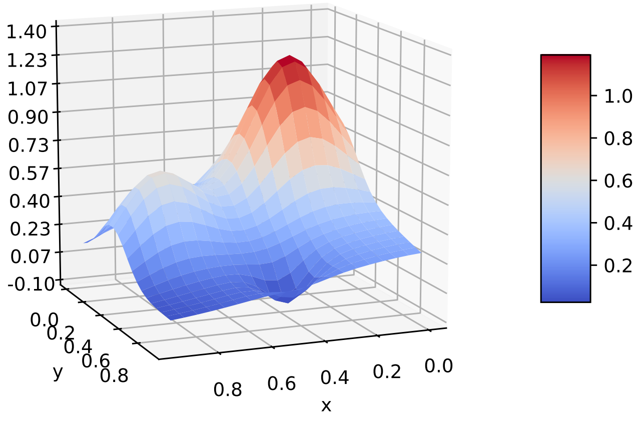

Figure 1: (18) $x, y \in [0, 1]$ with grid spacing 0.05

2.6 Code implementation

The regression methods are implemented into a program using Python 2.7, using numpy and Scikit-learn. The code is structured around a program which contains a data class, with methods for different regression procedures, as well as data generation and data reading. The results presented in the following section are produced using their own smaller programs, which draw on the classes from the main program. The class method for Lasso uses the Scikit-learn, while OLS and Ridge do not. Additionally, the k-Fold CV draws on some utility from Scikit-learn, such as dividing folds.

2.6.1 Initial evaluation of OLS implementation

In order to make an initial evaluation implementation of the OLS regression, terrain data is generated using Franke's Function (18), with 30×30 grid points. One data set with no noise, one with noise $\sim N(0,1)$ and one with $\sim N(0,0.3)$. Polynomials of up to fifth order are directly fitted using the entire data set, and the accuracy of the model fittings are recorded in terms of MSE and R^2, as well as 95% confidence interval for the regression coefficients.

2.6.2 Test of k-Fold Cross Validation implementation

Following the test of the OLS method, the k-Fold Cross Validation re-sampling method is tested. The aim of this test is to try to establish a suitable number of folds, k. A data set using the same parameters as previously, (with noise $\sim N(0,0.3)$) is used. Under increasing model complexity, the data is split up in folds as described in the methods sections. For each value of k, the average training data MSE is calculated, so that it may be used as a quality measurement.

2.6.3 Bias-Variance trade-off

Upon concluding that $k = 10$ is a suitable number of folds, the kFCV method using OLS regression is now applied to 50 generated data set, similar parameters as previously, with noise $\sim N(0,0.3)$. For each data set, a design matrix is set up, and $k = 10$-fold CV is carried out over increasingly complex models with polynomial degrees ranging from 1 to 10. After the 50 iterations, mean values of the mean MSE (of the kFCV) on test data and training data are calculated. This final mean over the 50 iterations, are meant to illustrate the bias-variance trade-off, as well as helping in further establishing a suitable complexity for $n \times n$ grids on Franke's Function.

2.6.4 Ridge and Lasso implementation testing

Lasso and Ridge regression is in this project tested and evaluated simultaneously. Using a selected complexities and a range of $\lambda = 0$ to 1, Ridge and Lasso regression is carried out with 10-fold Cross Validation on the same 30×30 grid as previously. The MSE values for training and test data for the two regression methods are measured and plotted as functions of λ, in order to discuss their aptness for terrain prediction.

2.7 Predicting real terrain data

Having done, in varying depth, parameter and method evaluations, the core program is now used to predict real terrain data. Using digital terrain data over Møsvatn Austfjell, in Norway, provided through the course github page . First, the 3601×1801 terrain is reduced to 121×61 by selecting every 30'th grid value, then a 10-Fold Cross Validation using OLS is carried out for a range of complexities. Based on R^2 scores for training data, a suitable polynomial degree is then chosen, and the corresponding model fit used to predict the full 3601×1801 terrain.

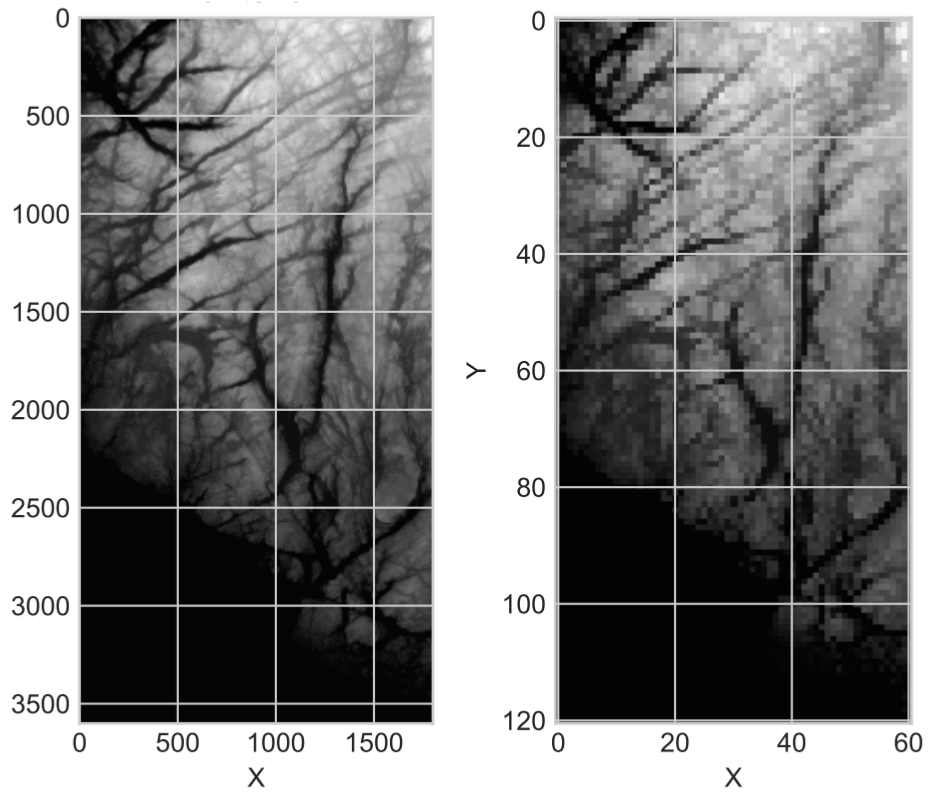

Figure 2: Terrain data from Møsvatn Austfjell, Norway. ((left) 3601×1801 - used for comparison with model prediction. (right) 121×61 - used for model fitting.

3 Results

3.1 OLS implementation

Below, MSE and R^2 values for the initial evaluation of the OLS is presented in table 3 for polynomials up to degree five, using the 30×30 Franke's Function generated data set with noise $\sim N(0, 0.3)$. Similar tables for noise $\sim N(0, 1)$ and no noise can be foind in appendix 2. These tables clearly show that both R^2 values and MSE decrease with increased model complexity up to the fifth degree, and that both measurements increase with increased noise.

Polynomial degree	R^2	MSE
1	0.3521	0.1169
2	0.3947	0.1093
3	0.4519	0.0989
4	0.4904	0.0920
5	0.5032	0.0897

Table 1: MSE and R^2 scores for OLS on Franke's Function generated responses using a 30×30 grid with noise $\sim N(0, 0.3)$

Figure 3 shows the spread of confidence intervals for a 1. order up to 5. order fitting using OLS on the 30×30 Franke's Function generated grid with noise $\sim N(0, 0.3)$. Looking at the figure, it is evident that the magnitude of the largest coefficients increase with increased complexity. Furthermore, β_0 appears to stay stationary at a small positive magnitude. This also appears to be that case for β_1 and β_2, which corresponds to the response dependency on x and y. As the confidence intervals for the coefficients are mostly small relative to the spread of the coefficient's values, a second plot showing the confidence intervals of a fifth order fitting is shown in figure 4. From said figure, it is clear that coefficients with large values have a larger uncertainty.

3.2 Test of k-Fold Cross Validation implementation

Figure 5 shows the average MSE from $k = 5, 10, 20, 40$ folds on the same data set. Having already established a decrease in MSE for this type of data set following complexity increase up to fifth order polynomials, the mean MSE values are shown as functions of model complexity in range from 4 up to 9, in order to further investigate the complexity-accuracy relationship. It is evident that MSE initially decreases with complexity, before increasing at the sixth order for all values of k. In addition, the figure indicates that $k = 5$ fold CV yields decreased accuracy in comparison to higher k values, while $k = 10, 20$, and 40 have approximately the same MSE, with a minimum of ~ 0.89 at the fifth order of complexity.

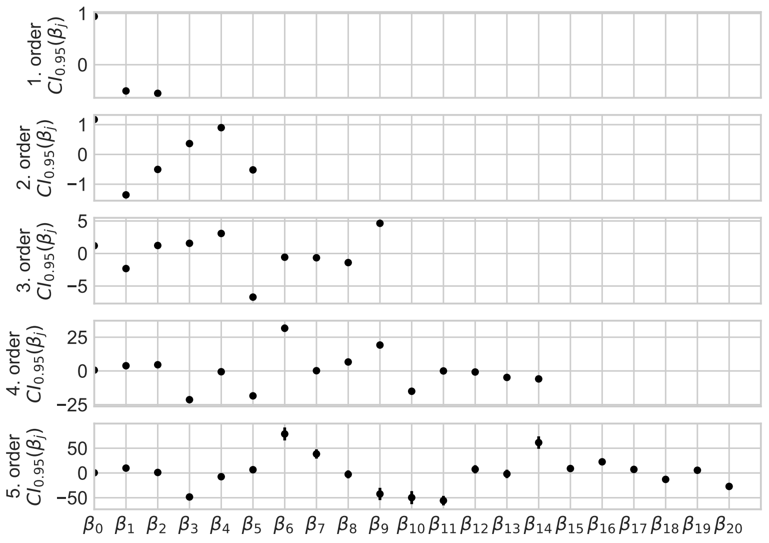

Figure 3: 95% confidence intervals for the OLS regression coefficients for polynomials of order one to five, using the Franke's Function generated response on a 30×30 grid

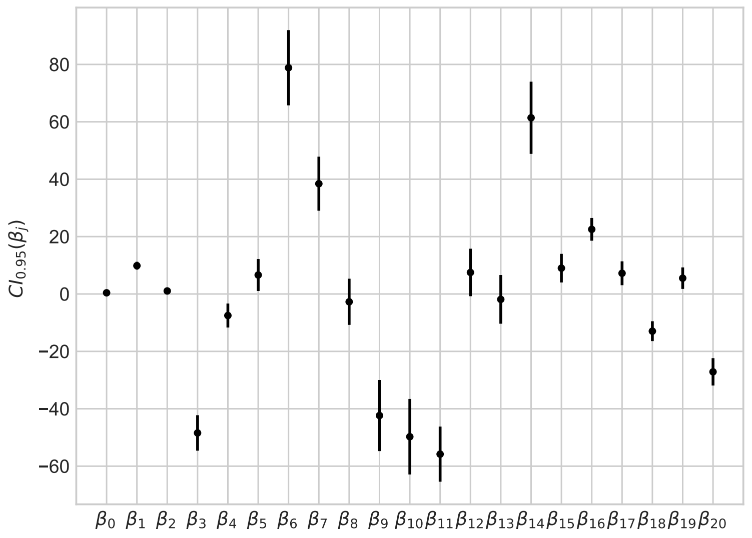

Figure 4: 95% confidence intervals for the OLS regression coefficients for a fifth order polynomial, using the Franke's Function generated response on a 30×30 grid

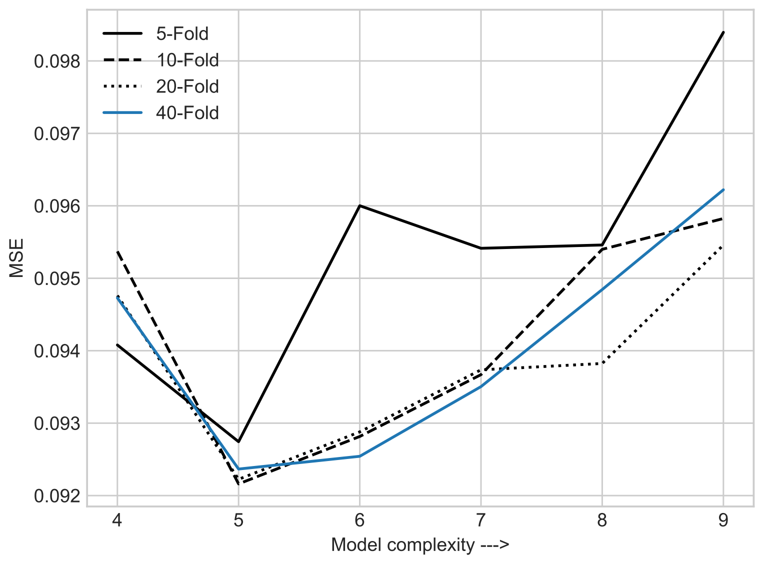

Figure 5: Comparison of produced average MSE from training data using OLS with k-Fold CV under increasing model complexity for $k = 10, 20, 20, 40$. Data originates from Franke's Function generated response on a 30×30 grid

3.3 Bias-Variance

Figure 6 shows averaged mean MSE from kFCV using OLS of varying complexity over 50 data sets. It is evident from the plots, that - when averaged over many data sets - the MSE from training and test data first decrease proportionally, then diverge, with increased complexity. For complexity corresponding 4th, 5th, 6th and 7th order polynomials, the averaged test MSE appears to be at a minimum ~ 0.095. For polynomials of higher order than 7, the test MSE increases above ~ 0.095 towards 1.

3.4 Ridge and Lasso

7 shows test and training MSE for Ridge and Lasso regression for a number of λ values for selected complexities, based on the same data set as in the last kFCV examination. Based on the plot $\lambda \to 0$ seems to produce the most accurate results for Ridge regression, while the choice of λ appears somewhat arbitrary in the for Lasso regression. Additionally, MSE (for both test and training) seems to increase drastically for any values of $\lambda > 0$.

3.5 Predicting real terrain data

8 shows the R^2 training and test scores for OLS regression on the reduced terrain data from Møsvatn Austfjell in Norway, using 10-fold Cross Validation. The plot

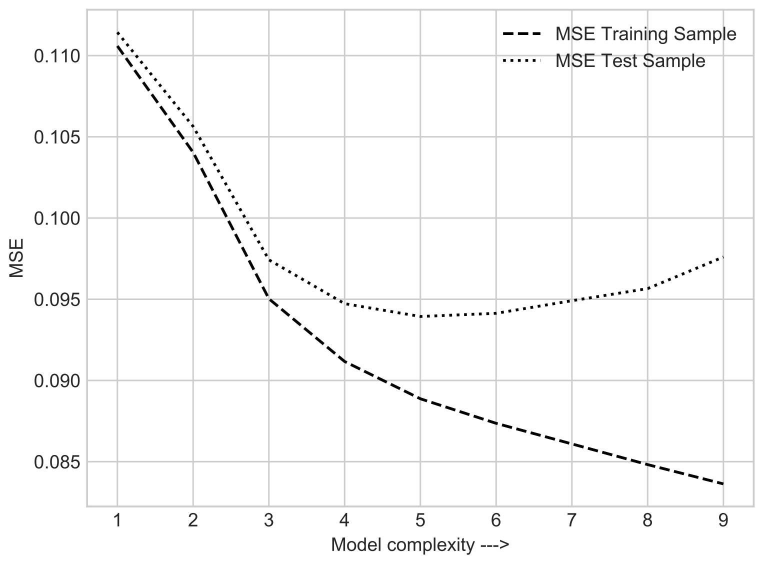

Figure 6: Comparison of kFCV ($k = 10$) mean MSE using OLS from training and test data of varying complexity, averaged over 50 data sets generated from Franke's Function on a 30×30 with noise $\sim N(0, 0.3)$.

clearly indicates that a second order polynomial fit may give the best accuracy ($R^2 \sim 0.75$) , while also indicating a complete model collapse when using a third order polynomial. This seems to be a trend, as similar geography data from Stavanger, Norway (see figure 12 in appendix 2) results in $R^2 \approx 0$ for both third, fourth and fifth degree polynomials. Higher order polynomials $(6-12)$ have slightly lower R^2 scores at $R^2 \sim 0.7$.

9 shows a visual comparison between full 3601×1801 predicted terrain, reduced training data (121×61) and true geography. In more qualitative terms, the R^2 for the prediction with respect to the true geography is 0.6522 with the shown model fit.

Ridge Regression for $p = 2$ using 10-Fold Cross Validation for 20 evenly spaced values of $\lambda \in [0, 1]$ gave an R^2 score of 0.76 ($\lambda = 0.26$). Using this λ value, the full terrain data was prediction, which resulted in an identical results for a seconder order polynomial fit (see figures 10 and 11 appendix 2):

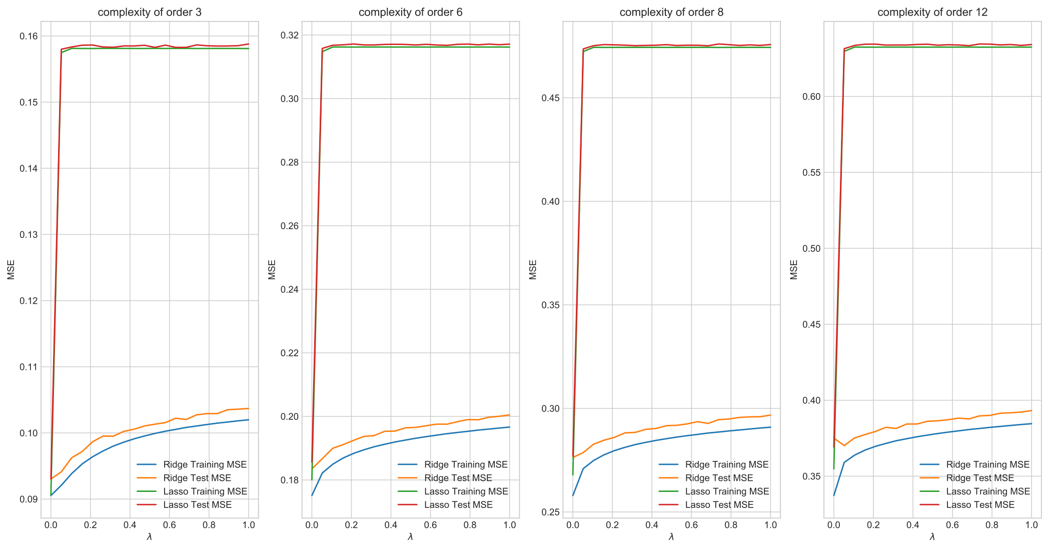

Figure 7: Test and training MSE for a range of $\lambda \in [0, 1]$, for Ridge and Lasso regression using $k = 10$ fold CV, over selected complexities

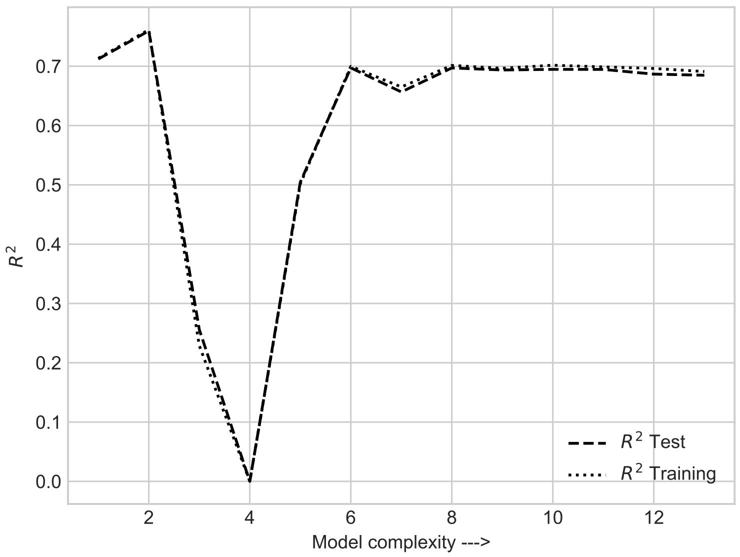

Figure 8: R^2 score for OLS fit to reduced (121×61)real terrain data (Møsvatn Austfjell, Norway) as a function of complexity

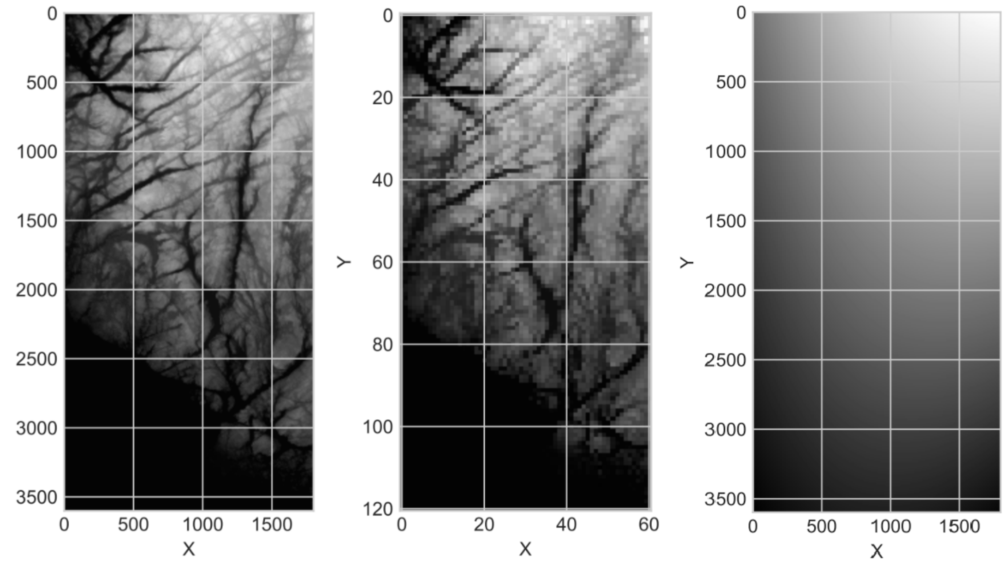

Figure 9: Predicted 3601 × 1801 terrain (right) from Møsvatn Austfjell, Norway , based on OLS regression with a seconder order polynomial on reduced (121 × 61) data (center) from complete 3601 × 1801 data of the geography.

4 Discussion of results

4.1 OLS implementation

As the Franke's Function does not describe a planar terrain (see figure 1), the increased R^2 score and decreased MSE with increased model complexity is no surprise. The small, and approximately stationary values of β_1 and β_2 is neither surprising, as 1 suggests that there are no general drastic changes associated with x and y to the power of one. However, 1 indicates that values of x and y close to zero should produce a high response. In speculative terms, it is possible that the large positive coefficients are a result of this - especially because they correspond to high powers of the domain variables. On the large($\pm \sim 20$) confidence intervals observed on those coefficients it is, in the same uncertain terms, possible that perturbation on higher powers of x and y cause large intervals.

4.2 k-Fold CV

The initial decrease in mean MSE follows the trend seen in the OLS implementation testing. As was observed previously, and again in the kFCV test, a complexity corresponding to a fifth order polynomial appears to be the most accurate, with a minimum mean MSE ~ 0.89. This holds true for $k = 5, 10, 20, 40$ - albeit $k = 5$ exhibits significant inaccuracy relative to the other k-values, especially so when complexity further increases. Lower k values mean larger subset for both training and testing, but fever iterations. This may possibly results in less bias, as the training folds become closer to the entirety of the data. Simultaneously, this may also increase variance, although this is not explicitly shown in the results. Tendencies related to the bias-variance trade-off are however present. As increases in k are also more computationally expensive, $k = 10$ appears to be a suitable value.

4.3 Bias-Variance

Following the trend observed in the initial k-Fold Cross-Validation test, a clear indication of bias-variance trade-off is observed for complex models using OLS regression. Averaging MSE ~ 0.095 on 4th-7th order polynomials, the test data MSE then increases towards MSE ~ 1 for further increased complexity, while the training MSE continues to decrease. This is highly likely due to the better reproduction of training data at the cost of general validity on other samples from the same main population. The exact optimum polynomial degree is difficult to ascertain, although the simulations points towards a fifth order. Due to the averaging over 50 data sets, this candidate for optimal complexity is somewhat strengthened, but this is at the same time only for the specific 30×30 grid with noise $\sim N(0, 0.3)$. Due to this fact, this project reaches no specific conclusions on optimal polynomial degree. However, if the Franke's Function generated grid is somewhat representative for terrain data, polynomials above the 4th order, and below the 8th order appears to be preferable.

4.4 Ridge and Lasso

Surprisingly, $\lambda > 0$ appears to produce the increasingly inaccurate results for both Lasso and Ridge regression on the evaluation data sets. For Lasso, increasing values for λ seems to produce especially inaccurate predictions, with a significant, almost step-wise, increase after $\lambda = 0$. This may indicate that OLS regression, that is, no shrinkage, is the preferred regression method for the Franke's Function generated data. Going by the description of these procedures in the methods sections however, one would not necessarily be inclined to expect this. This means that there is a distinct possibility that the implementation of the two algorithms is faulty. In this regard, it is also worth mentioning that for $\lambda = 0$, the Lasso Method produces convergence warnings. In either case, the best MSE test scores achieved by either model for any complexity previously tested is, for values other than $\lambda = 0$, worse than OLS. As this parameter evaluation is only superficial, these results carry no implication beyond indicating that OLS is the preferred regression method when using the programs written for the purpose of this project.

4.5 Predicting real terrain data

Initial evaluation of model complexity when using OLS indicates that either a seconder order, or a higher order $(6 - 12)$ polynomial should be used in the model, with R^2 scores $R^2 \sim 0.7$ on test data from the reduced terrain data. Visual inspection of the prediction on the full geography shows that the model has captured some key aspects of the terrain, such as the darker region in the bottom left and top left, and the brighter regions top right. A rough analysis of the prediction on each 500×500 square would suggests an average of ≈ 55 meters error per 500×500 square, in a terrain with a mean height of ≈ 333 meters, which constitutes an average deviation in predicted and actual height per square of approximately 17%. However, this is clearly not the case on all such square, as the lower left corner clearly shows. The polynomial fit is too course to properly predict other easily recognizable features such as the dark streaks in the center of the geography. A R^2 score of 0.6522 seems reasonable for the chosen method. It is however counter intuitive that Ridge Regression using $\lambda = 0.26$ should produce the same result as the OLS regression, since $\lambda = 0$ corresponds to OLS - which would have been the optimal λ if OLS and Ridge were truly equal on optimal fit. This further indicates faulty implementation. In addition the calculation of negative R^2 values (set to 0 in the results), clearly indicate some fault.

4.6 Conclusions

This project observed that polynomial fits using linear regression of order four to eight appeared to give the most accurate predictions on the Franke's Function generated data for a 30×30 geography with MSE ~ 0.89 when the response suffered from noise $\sim N(0, 0.3)$. The project simultaneously observed R^2 scores$\leq$

0 for polynomials of the third, fourth and fifth order when the same core program was used in predicting real world terrain on a scale of kilometers with 3601×1801 resolution, based on a training set over the same area with resolution 121×61. The project is therefor inconclusive on these polynomial complexities for the selected geographies, and assumes an implementation error in the core program developed for the purpose of this project. After further testing on the same data, this project goes on to conclude that linear regression using second order polynomials are capable of of predicting geography with a resolution of 3601×1801 terrain based on training data with 121×61 resolution up to a level of accuracy corresponding to predicting an average of 16% percent deviation on terrain height for a 500×500 square. Further examination on choice of regression models is needed in order to draw any decisive conclusions on subject, due to strong indications of implementation errors in the core code used in the project.

References

[1] Uri M. Ascher and Chen Greif. Linear least squares problems. In *A First Course on Numerical Methods*, pages 141–166. Society for Industrial and Applied Mathematics, 2011.

[2] Richard Franke. A critical comparison of some methods for interpolation of scattered data. Technical report, Naval Postgraduate Sschool Montery CA, 1979.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer Series in Statistics. Springer New York, New York, NY, second edition edition, 2009.

[4] Morten Hjorth-Jensen. Lectures notes in fys-stk4155. data analysis and machine learning: Linear regression and more advanced regression analysis, Sep 13 2019.

[5] Gareth James. *An Introduction to statistical learning : with applications in R.* Springer texts in statistics. Springer, New York, corrected at 8th printing 2017. edition, 2017.

[6] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, 2019.

[7] Kevin P. Murphy. *Machine learning : a probabilistic perspective.* Adaptive computation and machine learning. MIT Press, Cambridge, 2012.

Appendices

Appendix 1.

Leaning on literature such as [3][p.223], the relationship

$$\begin{aligned}\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] &= \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2 \\ &= Bias^2(\tilde{\boldsymbol{y}}) + Var(\tilde{\boldsymbol{y}}) + Error\end{aligned}$$

May be shown using that $E(f) = f, E(y) = f$:

$$\begin{aligned}E[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2] &= E[(\boldsymbol{y}-\boldsymbol{f}+\boldsymbol{f}-\tilde{\boldsymbol{y}})^2] = E[(\boldsymbol{y}-\boldsymbol{f})^2] + E[(\boldsymbol{f}-\tilde{\boldsymbol{y}})^2] + 2E[(\boldsymbol{f}-\tilde{\boldsymbol{y}})(\boldsymbol{y}-\boldsymbol{f})] \\ &= E[(\boldsymbol{f}+\boldsymbol{\epsilon}-\boldsymbol{f})2] + E[(\boldsymbol{f}-\tilde{\boldsymbol{y}})^2] + 2E[\boldsymbol{f}\boldsymbol{y}-\boldsymbol{f}^2-\tilde{\boldsymbol{y}}\boldsymbol{y}+\tilde{\boldsymbol{y}}\boldsymbol{f}] \\ &= E[\epsilon^2] + E[(\boldsymbol{f}-\tilde{\boldsymbol{y}})^2] + 2(\boldsymbol{f}^2-\boldsymbol{f}^2-\boldsymbol{f}E[\tilde{\boldsymbol{y}}]+\boldsymbol{f}E[\tilde{\boldsymbol{y}}]) \\ &= \sigma^2 + E[(\boldsymbol{f}-\tilde{\boldsymbol{y}})^2] + 0 \\ &= E[(\boldsymbol{f}+E[\tilde{\boldsymbol{y}}]-E[\tilde{\boldsymbol{y}}]-\tilde{\boldsymbol{y}})^2] + \sigma^2 \\ &= E[\boldsymbol{f}-E[\tilde{\boldsymbol{y}}]]^2 + E[\tilde{\boldsymbol{y}}-E[\tilde{\boldsymbol{y}}]]^2 + \sigma^2 \\ &= \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2 \\ &= Bias^2[\tilde{\boldsymbol{y}}] + Var[\tilde{\boldsymbol{y}}] + error\end{aligned}$$

Appendix 2.

Polynomial degree	R^2	MSE
1	0.7405	0.0176
2	0.8140	0.0126
3	0.9115	0.0060
4	0.9561	0.0030
5	0.9840	0.0011

Table 2: MSE and R^2 scores for OLS on Franke's Function generated responses using a 30×30 grid without noise

Polynomial degree	R^2	MSE
1	0.0865	0.9102
2	0.0999	0.8968
3	0.1099	0.8868
4	0.1145	0.8822
5	0.1285	0.8683

Table 3: MSE and R^2 scores for OLS on Franke's Function generated responses using a 30×30 grid with noise $\sim N(0, 1)$

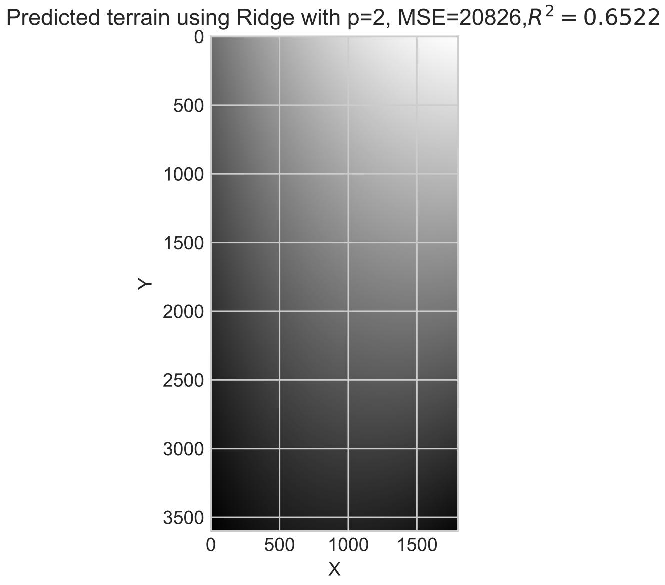

Figure 10: Ridge Prediction on full geodata $p = 2$

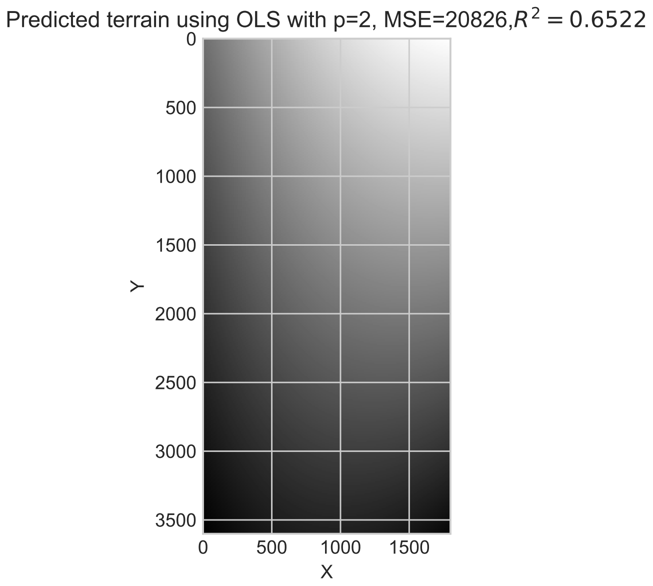

Figure 11: OLS Prediction on full geodata $p = 2$.

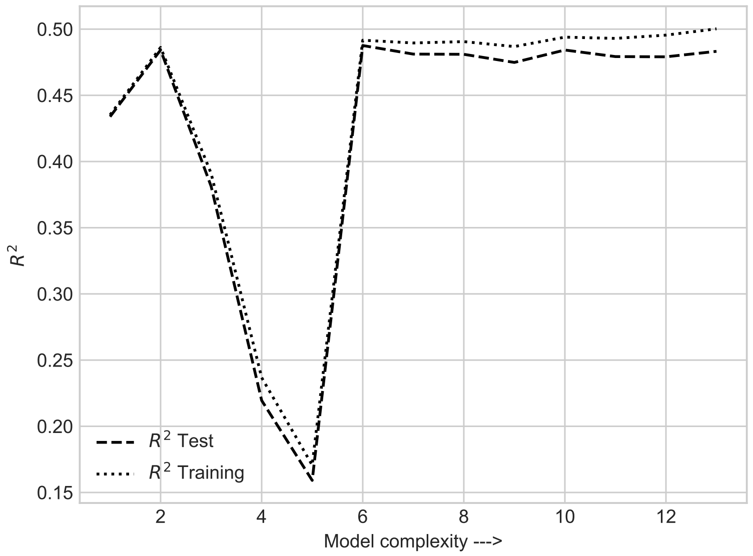

Figure 12: R^2 score for OLS fit to reduced (121×61)real terrain data (Stavanger Norway) as a function of complexity