Regression Methods on Terrain Data

Steinn Hauser and Simen Håpnes
(Dated: Monday 7th October, 2019)

A study of the Ordinary Least Squares, Ridge and Lasso regression methods is presented. The three methods were applied to Franke's function and terrain data of the dead sea. The k-fold cross validation resampling method was used to quantify the optimal set of parameters for the data sets by minimizing the mean squared error. The Ridge regression method was found to be the best for Franke's function with $MSE_{min} = 0.2491$ using the hyperparameter $\lambda = 10^{-9}$ and polynomial degree $p_{deg} = 11$. For the dead sea terrain data, the Ridge regression was again found to be the best method, with $MSE_{min} = 0.0127km^2$ using $\lambda = 10^{-12}$ and $p_{deg} = 12$.

I. INTRODUCTION

Regression methods and estimation techniques have been a very central topic in statistics for a long time. More recently, massive amounts of data surfaced around the world, bringing about a data science revolution. Machine learning- and high performance computational analyses of large datasets are now the most important resources for outcome predictions - something very useful to many institutions. Examples of such predictions vary from supervised machine learning used to predict lymphoid malignancies based on pre-treatment characteristics (see medicinal work by Margaret A. Shipp et al. [7]), to forecasts of credit card holder deficiencies and defaults (see financial work by Amir E. Khandani et al. [5]). The motivation of this report is to research the abilities of such prediction methods; three methods will be presented for comparison: Ordinary Least Squares (OLS), Ridge Regression, and Lasso Regression.

Firstly, the methods are implemented numerically and applied to Franke's Function. This function is a good benchmark to assert that the methods are operating properly. The methods are also combined with resampling techniques (such as cross-validation, etc.) for proper assessment of the method characteristics (parameters which quantify the prediction abilities of the methods).

Finally, the implemented schemes will be applied to real-world terrain data provided by the **U.S. Geological Survey department**. The terrain data is chosen for visual merit, and the scheme performances will be compared and discussed in this context. A central discussion of the validity of the methods presented is the so-called bias-variance tradeoff, as it is a prominent subject of dialogue for regression methods. This will be explained in further detail in subsequent sections.

This report is separated into sections which aim to lead the reader to similar conclusions to those presented. Firstly, an introduction of the theory background needed is presented in the *theory* section, before the methods used to conduct the research are described in the *method* section. These sections are then followed by the results obtained, and discussion of those results in the *results* and *discussion* sections. Finally, a conclusion of the report is stated to summarize.

This study is a collaboration between Simen Håpnes and Steinn Hauser, and the full project code is in the following public Github Repository. In this public repository, several results figures and analysis scripts are included in an attempt to make the study sufficiently reproducible.

II. THEORY

A. Regression Methods

Regression methods are the backbone to machine learning and predictive computations. If given a list the population that have a given disease y, and several characteristics of those people $x_{age}, x_{height}, \ldots$ then, given that enough characteristics are produced, one is in theory able to pinpoint the causes of the disease and work to prevent the disease occurring in people which show early symptoms.

Three such predictive methods are presented in the following section for comparison. A basis of method assessment is needed to evaluate the strengths and weaknesses of the predictions, so a section on MSE and the R^2 score is presented subsequently.

1. *Ordinary Least Squares*

The Ordinary Least Squares (OLS) method is the most common and simplistic linear regression model. The basic concept is to predict some outcome y for some given input data X:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix} = X \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_p \end{bmatrix}, \tag{1}$$

where X is the *Design Matrix*. The design matrix can, for example in the case of a polynomial form dependence of two

parameters x_1 and x_2, be:

$$\text{N rows}\left\{\begin{pmatrix} \overbrace{1 & x_1 & x_2 & x_1^2 & x_2^2 & x_1x_2 & \dots}^{\text{p columns}} \\ 1 & x_1 & x_2 & x_1^2 & x_2^2 & x_1x_2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & x_1 & x_2 & x_1^2 & x_2^2 & x_1x_2 & \dots \end{pmatrix}\right.$$

The design matrix can take any form depending on the model which is being fit to the dataset. It is incredibly customizable, though a characteristic which they all generally share is the first column being a vector of units $\mathbb{1}$. This is for the so-called *intercept* β_0, which is not variable-dependant. The parameter which is modified to suit the data is the vector $\hat{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_{p-1}]$. This vector is found by minimizing the *residual sum of squares*:

$$RSS(\beta) = \sum_i \left(y_i - x_i^T\beta\right)^2 = (y - X\beta)^T(y - X\beta), \quad (2)$$

where x_i are the columns of X. This is a specific case of what is known as a *cost function*. The cost function is a function which is meant to quantify how well a fit $\hat{\beta}$ predicts some testing data y_i. The worse the prediction, the larger the cost function. This simplifies the regression problem to a problem of function minimization. For the specific OLS case, where the cost function is given by equation 2, minimizing this function with respect to β yields the solution:

$$\frac{\partial}{\partial\beta}RSS = 0 \Rightarrow \hat{\beta} = \left(X^TX\right)^{-1}X^Ty. \quad (3)$$

See appendix A for the full derivation of this expression. This predictor $\hat{\beta}$ can then be used to predict the results $\hat{\mathbf{y}}$ of new datasets D, $\{x_1, x_2\} \in D$:

$$\hat{\mathbf{y}} = X_D^T\hat{\beta}. \quad (4)$$

2. Ridge Scheme

The Ridge scheme is designed to be a solution to the singular X^TX matrix case. We have the linear algebra relation:

$$det\,(A) = 0 \Rightarrow A \text{ is not invertable.} \quad (5)$$

In the case of a singular (determinant equal zero) matrix X^TX, the coefficient vector $\hat{\beta}$ cannot be found using formula 3. To fix this, we need to tweak the diagonal elements of the matrix product X^TX by an amount $\lambda \neq 0$ to make it invertable again:

$$X^TX \rightarrow X^TX + \lambda I_p, \quad (6)$$

where λ is some adjustable hyperparameter (known as the Ridge parameter) and I_p is the $(p \times p)$ identity matrix. This yields the following method of finding $\hat{\beta}$ (by finding the minimum/argmin of the cost function w.r.t beta):

$$\hat{\beta}(\lambda) = \text{argmin}_\beta\left\{(Y - X\beta)^T(Y - X\beta) + \lambda\beta^T\beta\right\} \quad (7)$$

The expression for the prediction coefficients $\hat{\beta}$ is then given by (by setting $\partial C/\partial\beta = 0$ as for the OLS method):

$$\hat{\beta}_R = \left(X^TX + \lambda I_p\right)^{-1}X^Ty \quad (8)$$

This is the equation used to derive the predictive model $\hat{\beta}$ for the regression method. The components of $\hat{\beta}$ are in this case subject to the constraint region:

$$\sum_{j=0}^{p-1}|\beta_j|^2 \leq t, \quad (9)$$

where $t \sim \lambda^{-1}$.

3. Lasso Scheme

The Lasso scheme involves assigning a parameter λ to the product X^TX as in Ridge regression, though this time, the expression used to calculate the prediction parameter $\hat{\beta}$ is given by:

$$\hat{\beta}(\lambda) = \text{argmin}_\beta\left\{(Y - X\beta)^T(Y - X\beta) + \lambda\sqrt{\beta^T\beta}\right\} \quad (10)$$

The components of $\hat{\beta}$ are again subject to a constraint region, this time given by:

$$\sum_{j=0}^{p-1}|\beta_j| \leq t, \quad (11)$$

where $t \sim \lambda^{-1}$. This cost function expression doesn't have analytic solutions to the equation $\partial C/\partial\beta = 0$. The solution to the model $\hat{\beta}$ is therefore found by a search for the minimum of the cost function w.r.t beta by brute force. This process can be done by grid-search, though it is not very efficient; lasso algorithms are a complex subject[2], so building an algorithm from scratch is definitely less efficient than using algorithms which packages such as *sk-learn* offer. The way these are implemented will be described in further detail in the methods section.

4. Mean Squared Error and R^2 Score

After a regression method has been applied, the mean squared error (MSE) and the R^2 score functions are used to evaluate the performance of the regression. The MSE is given by

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2, \quad (12)$$

where y_i indicates a subset of the original dataset, typically a test set. It is essential that this test set was totally neglected when building the model $\hat{y}$ using the training data. $\hat{y}_i$ is the model prediction, calculated using the corresponding set X and the predictor $\hat{\beta}$.

In order to evaluate the R^2 score, the *residual sum of squares* (RSS) and the *total sum of squares* (TSS) are defined:

$$RSS(y,\hat{y}) = \sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2, \tag{13}$$

$$TSS(y) = \sum_{i=0}^{n-1}(y_i - \bar{y})^2, \tag{14}$$

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i. \tag{15}$$

The R^2 score is then given by:

$$R^2(y,\hat{y}) = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}. \tag{16}$$

The R^2 score is a measurement of how close the applied regression method is fit to the dataset. The larger the R^2 score, the better the model.

B. Resampling Techniques

Resampling techniques aim to assess methods by efficiently splitting up a single dataset into training and testing data. The method in question is trained and tested on a variety of subsets of the original dataset, producing a more accurate measurement of the methods 'true' ability to capture the aspects of the data. Two such resampling techniques are introduced and implemented for a justified comparison of the methods.

1. k-fold Cross Validation

The k-fold Cross Validation (CV) is the central resampling technique used, and aims at estimating the *expected test error*

$$Err = \mathbb{E}\left[C\left(y,\hat{y}\right)\right], \tag{17}$$

where $\mathbb{E}$ denotes the *expectation value*, e.g:

$$\mathbb{E}[x] = \sum_{k=0}^{n-1} x_k \Pr(x = x_k), \tag{18}$$

where $\Pr(x = x_k)$ is the probability of the outcome $x = x_k$. If all the outcomes of x are equal, then the expectation value is the average:

$$\mathbb{E}[x] = \sum_{k=0}^{n-1} x_k \frac{1}{n} = \frac{1}{n}\sum_{k=0}^{n-1} x_k. \tag{19}$$

The most common cost function C is the case of an MSE evaluation. In that case the cost function C is given by

$$C\big(y,\hat{y}\big) = \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \hat{y}_i\right)^2. \tag{20}$$

The k-fold CV method aims at estimating the error by splitting the data set into k subsets which are then used independently for training and testing. The method uses $(k-1)$ subsets for training (establish the prediction parameters), then uses the remaining subset for testing (calculate the MSE, R^2 score, etc.). This is done k times using every unique subset for testing exactly once:

TABLE I. An illustration of all k subsets and their role depending on the step $i \in 1, 2, \ldots, k$. There are $(k-1)$ subsets used for training for each step i, but the same subsets are not used for each step. For $i = 1$, the first subset is reserved for testing data, and for $i = 2$ second subset is reserved. This process is repeated k times until each subset has been used as testing data exactly once.

i=1	Test	Train	Train	Train	...	Train
i=2	Train	Test	Train	Train	...	Train
⋮	⋮	⋮	⋮	⋮	⋱	⋮
i=k	Train	Train	Train	Train	...	Test

Table I illustrates the all $i \in 1, 2, \ldots, k$ steps which are performed for k-fold CV. The characteristics of the prediction method are averaged over all these steps in the end, producing an evaluation of the methods abilities.

When performing cross validation in this way, it is important to ensure that the data is properly shuffled before splitting into subsets. If this is not done, then the test set data can be dramatically different from the training set data in some cases; taking as an example topic of terrain parametrization: If the $i-th$ subset of the dataset happens onto an exceptional area (such as a deep chasm), then the testing data will be very different from the rest and the resampling will not yield a proper assessment of the abilities of the model. Once the data is shuffled, the general trend of chasms and their probability is better implemented into each subset, and k-fold CV can commence.

The choice of k in k-fold cross validation should be either $k = 5$ or $k = 10$, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance[4].

C. Bias-Variance Trade-Off

The bias-variance tradeoff is an essential balance to keep in mind when applying the methods mentioned above. It regards the interplay between the bias

$$Bias^2(\hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}\left(f_i - \mathbb{E}\left[\hat{\mathbf{y}}\right]\right)^2, \tag{21}$$

and the variance

$$Var(\hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}\left(\hat{y}_i - \mathbb{E}\left[\hat{\mathbf{y}}\right]\right)^2, \tag{22}$$

where $\mathbb{E}[\hat{\mathbf{y}}]$ denotes the expectation value of $\hat{\mathbf{y}}$, $\hat{\mathbf{y}}$ denotes the prediction $\hat{\mathbf{y}} = X^T\hat{\beta}$, and f_i denotes the 'true' data function without the noise. In the parametrization of $\hat{\beta}$ described for the methods previously, the cost function $C(X, \beta)$ is introduced as the residual sum of squares (RSS) function in equation 2:

$$C(X, \hat{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 = \mathbb{E}\left[(y_i - \hat{y}_i)^2\right]. \tag{23}$$

From this, it is possible decompose the expression into a function of the bias and the variance:

$$C(X, \hat{\beta}) = Bias^2(\hat{\mathbf{y}}) + \sigma_\epsilon^2 + Var(\hat{\mathbf{y}}), \tag{24}$$

where σ_ϵ^2 is the irreducible error of the original dataset

$$y_i = f_i + \epsilon, \quad \epsilon \sim N(0, \sigma_\epsilon^2). \tag{25}$$

The proof for this relation is included in Appendix: B.

Calculating the minimum of the cost function $C(X, \hat{\beta})$ involves tuning the prediction parameter $\hat{\beta}$ to minimize the sum of the Bias^2 and the Variance of $\hat{\mathbf{y}}$ (σ_ϵ^2 is not considered in the minimization due to its irreducibility).

This implies that the balance between the bias and the variance is interlinked in the MSE of the model. If the bias is increased, the variance usually decreases and vise versa. One sense of the bias is the simplicity of the model. If the model is assumed to be linear, then the bias of the model is high and the variance is low. Increasing the complexity of the model will cause the training error to go towards zero (and bias decreases). In the case of a polynomial fit, the training error approaches zero when $p_{deg} \to N - 1$. The parameter p_{deg} represents the polynomial degree, and N is the number of points to fit to. When these two equal, then the training error equals zero according to the Stone-Weierstrass theorem (e.g. a 2nd order polynomial perfectly fits three points [1]). Though the error is zero, this model will have very weak prediction capabilities. This is due to its specialization to the specific training set. This is referred to as **overfitting** and is an extreme case of prediction modeling, where the variance is too large and the bias is too small. Similarly, the other extreme case of prediction modeling is **underfitting**. This is the case for a model with small to no variance and large bias, and is equally poor at predicting outcomes. This time, the reasoning for the inadequacy of the model is that the essence of the dataset (the 'reason' for the dataset looking like it does) is not captured; this can be something like fitting a linear model to try and predict the outcome of an exponential curve.

The balance between bias and variance must be studied and resolved. A model which balances both the capture of data patterns and the errors of the testing data is what is sought-after. This is why the bias-variance decomposition is so important, as the mean squared error represented by the cost function $C(X, \hat{\beta})$ is expressed by both factors, where the bias increases on one end of the spectrum (low complexity implies large bias) and the variance dominates on the other end. This means that a minimization of the mean squared error function is a good reconciliation between the bias and the variance.

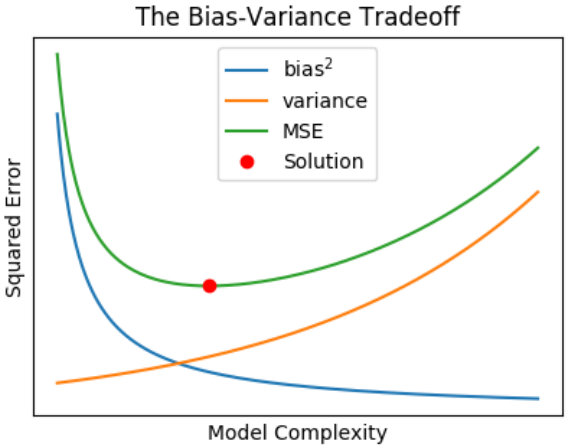

FIG. 1. Conceptualization of the bias-variance tradeoff. The bias-variance decomposition of the mean squared error (MSE) into the bias, variance and irreducible error (from equation 24) leads to this illustration of the interplay between the three. The point of minimal MSE is where the tradeoff between the bias and the variance is optimal, and is marked as the solution to the model complexity problem.

Figure 1 illustrates a conceptual case of the bias-variance trade-off. This figure intends to show the balance between the bias^2 and the variance and how they relate to the cost function $C(X, \beta)$ (set to be the mean squared error in equation 23). The optimal model complexity is found by the minimum of the cost function $\hat{\beta} = \text{argmin}_\beta\{C(X, \beta)\}$. The figure also illustrates the behaviour of the bias^2 and variance functions described previously (underfitting for low model complexity and overfitting for high model complexity). Note that the difference of the MSE plot and the variance plot with increasing complexity is not zero, but differs by the irreducible error σ_ϵ^2.

III. METHOD

The equations introduced are implemented numerically for analysis by way of code written in python 3.6 or above. The structure of the code was designed such that the regression methods were easily utilized and applied to data.

A. Code structure

The structure the project code is illustrated in figure 2:

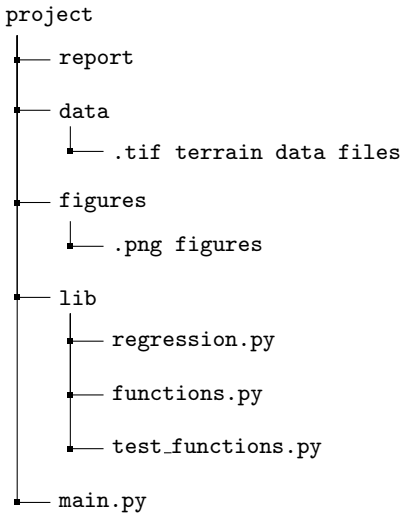

FIG. 2. Tree diagram illustrating the folder structure for the research project. The results obtained are stored as .png figure illustrations.

The work was distributed across several python scripts

- *regression.py* contains a class called *Regression*. The class contains the methods
  - *ols_fit*, for OLS regression
  - *ridge_fit*, for ridge regression
  - *lasso_fit*, for lasso regression
  - *predict*, to calculate the prediction $\hat{y}$
  - *mean_squared_error*, to calculate the MSE
  - *r2_score*, to calculate the R^2 score
  - *k_fold_cross_validation*, to perform k-fold CV.
- *functions.py* contains many different functions for various analyses using the *Regression* class. Most functions produce and save plots of data to the *figures* directory.
- *test_functions.py* includes a variety of testing functions which keep the methods used in check. Validations/tests of the linear algebra and the polynomial design matrix functions used are performed using this script. These tests are an important inclusion when using imported libraries.
- *main.py* is the only file which is executed by the user. This file controls parameters, import of data and creation of random data. The manipulation of regression objects and any applied functions are all coordinated from here.

A good way to save the attributes of regression methods is to build an object oriented code as is shown here. This *Regression* class is built very generally for application in many scenarios, making it reusable for all the research cases presented as well as other potential studies.

B. Regression Schemes

There are three regression schemes that are applied in the research presented, namely the OLS, Ridge regression and Lasso regression methods. The objective of all schemes is to produce a prediction vector $\hat{\beta}$ which is subsequently evaluated for accuracy. All of the linear algebra operations are performed using *numpy* (a fundamental scientific programming package for python) functions, as these have highly optimized accuracy and speed.

1. Ordinary Least Squares

The first scheme was implemented using *numpy*'s (version 1.16.4) linear algebra package. The OLS method is implemented by calculating the predictor using equation 3. Following are the linear algebra operators utilized:

- MMM (matrix-matrix multiplication) and MVM (matrix-vector multiplication) are both done by using *numpy*'s built-in "@" operator.
- Matrix-inversion can be done in two ways:
  1. Numpy's built-in inversion function: $A^{-1} =$ *numpy.linalg.inv(A)*.
  2. Using SVD (singular value decomposition), which requires more steps. The procedure is as follows:
     - Decompose: $UDV^T =$ *numpy.linalg.svd(A)*
     - Invert: $D^{-1} =$ *numpy.linalg.inv(D)*.
     - Multiply back: $A^{-1} = VD^{-1}U^T$.

The two matrix inversion methods are introduced as the SVD inversion method should be more numerically stable, but comes with the consequence of a larger computation time requirement. Whether this change is worth it will be discussed in further detail later [6].

2. Ridge and Lasso Scheme

Ridge and Lasso regression are implemented using the scikit-learn library for python (sklearn version 0.21.2). This library has modules that are numerically efficient for both schemes. The latter is particularly efficient, using gradient descent to evaluate equation 10. The only parameter studied for Ridge regression is the *alpha* parameter, which represents the λ hyperparameter in equation 8. Another parameter needed for sklearn's Ridge function was *normalize*, which is set to True.

For Lasso regression, there are more parameter options: The tolerance is set to 0.05, the maximum number of iterations is set to 2500, and *normalize* is set to True. *Note*: The scikit-learn module does not fit the bias (β_0) in Lasso regression. This is due to the data being shifted by the mean value $\bar{y}$ before the fit. The bias is however estimated and obtained later on using the *intercept_* attribute of the sklearn regression object.

C. Resampling

1. Split of training data and test data

When splitting the training and test data, the split is typically between 2/3rds (66% for training) and 4/5ths (80% for training) of the original data set. These fraction choices are chosen to avoid overfitting and underfitting, and are also typical conventional options[3]. In order to better understand the impact of these test sizes, several different fractions of training data are studied. The additional study involves varying the training data percentage between 10% and 90% with a step size of 5%. For each training data percentage, the OLS scheme is applied to Franke's function, and the MSE and R^2 scores are calculated. The purpose behind this analysis was to find a relation between the training data % and the degree of under/overfitting. The results from each of the training data percentages are presented and discussed in further detail later on.

2. k-fold Cross Validation

The k-fold CV is implemented into the Regression class in order to effectively evaluate MSE and R^2 for a regression scheme. This method takes four arguments:

1. The desired number of subsets k.
2. A choice of method: OLS, Ridge or Lasso.
3. A hyperparameter λ for the Ridge and Lasso schemes.
4. A boolean which declares whether or not to use the SVD matrix inversion method for OLS.

The procedure for k-fold CV is then as follows:

1. Shuffle the entire data set (both X and y, row by row).
2. Split the set into k equally sized subsets. **Note:** if $(N \textbf{ mod } k) \neq 0$, then the remainder of the data set is not included in the CV.
3. Repeat the following step k times, using all subsets as test sets exactly once:
   - Set aside the ith subset as a test set and use the chosen model on the remaining subsets to produce a prediction $\hat{\beta}$. Evaluate MSE_i and R_i^2 for this prediction using the test set.
4. Evaluate a final estimate for MSE and R^2 by taking the average of all MSE_i and R_i^2 for all $i \in [1, 2, \ldots, k]$

This method of k-fold CV yields an average measurement of the MSE and R^2 scores which are characteristic to the regression method applied to the data set. This sets a good foundation for the comparison between the models, as the dataset used is the same for all methods.

D. Analysis of the schemes

For all Franke's function regression analyses, a data set is generated using random x_i and y_i coordinates in the range $[0, 1]$. These produce an output $z_i = f(x_i, y_i) + \sigma^2$, where f is Franke's function and σ^2 is the variance of the noise in the data. All three schemes are analysed with various degrees of polynomial complexity using a fixed N data points and irreducible error σ^2.

- Additionally for OLS: the impact of the number of data points N is studied.
- Additionally for Ridge/Lasso: the impact of the hyperparameter λ is studied.

The k-fold CV is used to produce a reliable evaluation of the MSE and R^2 scores for each scheme. This utlimately yields the optimal degree of complexity p_{deg}, optimal number of data points (in the case of OLS) N, and optimal hyperparameter λ (for ridge and lasso).

1. Ordinary Least Squares

Two OLS analyses are conducted, both of which calculated the MSE in two ways: One of which used the entire data set as training data and the other used 5-fold cross validation.

The first analysis intended to explore the optimal polynomial degree/complexity of the model. Franke's function with noise is generated with a fixed number of data points and fixed irreducible error σ^2. A design matrix is set up for different polynomial degrees in the range between 2nd and 11th degree. In this case, it ranges from component x_2 up to component $x_1^3 x_2$ (sometimes simply called x and y) of the two-parameter polynomial design matrix presented. For each increasing degree in this matrix, the MSE is evaluated in two separate ways: As a prediction based on the entire data set as training set (no testing data reserved), or evaluated using 5-fold cross validation.

The second analysis is similar to the first: Vary the polynomial degree (model complexity) and evaluate MSE using either 100% training data (no data reserved for testing) or with 5-fold CV. The second analysis additionally has a variation of the data points n: n is varied to have the values $\{10^3, 10^4, 10^5, 10^6\}$, and for each of these values, a predictor $\hat{\beta}$ is generated.

Two goals are kept in mind with regards to this analysis:

1. Investigate the importance of k-fold CV to avoid the problem of overfitting, something which is sure to happen when no data is reserved for testing.
2. Investigate how the number of data points n and the model complexity p_{deg} impacts performance of the model (MSE and R^2 score).

2. Ridge Regression and Lasso Regression

For these two schemes, the analysis conducted is slightly different from the OLS analysis presented. The analyses for these methods include a variation of the hyperparameter λ, as well as a variation of the degree of complexity p_{deg}. The number of data points is set to a constant $n = 10^4$ throughout. The purpose of varying these parameters is to model the surface of prediction errors to find a minimal, optimal $\hat{\beta}$ for each method.

A data set is generated using Franke's function, and the Ridge and Lasso regression methods are applied for various polynomial degrees p_{deg} and hyperparameters λ. The central goals of this analysis is:

1. See the impact which the polynomial degree and hyperparameter have on the MSE.
2. Ultimately compare all three models to each other, when n and σ^2 are fixed.

For the sklearn's lasso method, there are additional parameters required for the gradient descent: The *tolerance* and the *max iteration*. These both determine when a sufficient solution to $\hat{\beta}$ is found (when the difference of the 'tweaks' to the predictor are less than the tolerance value), and determine when a search for a better $\hat{\beta}$ should be called off (when a max iteration is reached). These parameters are set to $tol = 0.05$, and $maxiter = 2500$ for this study.

E. Terrain Data

The study conducted on the terrain data is identical to the research presented so far. Only this time the data is not generated by Franke's function, but is rather data gathered by satellites in orbit. The terrain data is downloaded from the **U.S. Geological Survey department** site, and some specifications of this data are needed for the figures and plots which are produced.

The datasets come in a grid of 3601×3601 data points which describe the height from a reference; the grid of points are separated from each other by 1 arc-second on both the longitutal and latitudal axis. This implies that the entire image covers $1 \times 1\text{deg}^2$ of the earth. Declaring the earths radius to be $3,671km$ yields the following resolution of the geological images:

$$\frac{2\pi \cdot 3,671km}{360^\circ} \approx 111.2km \tag{26}$$

This means that the images cover an area of $(111.2)^2 km^2$. The separation between the points in the grid is then given by:

$$\frac{111.2km}{3601pts.} \approx 30\frac{m}{pt.} \tag{27}$$

These are required for the axes and the MSE analyses which are conducted for the terrain data.

To increase the computational efficiency, the resolution of the points is reduced by 66% for each axis, which is every third data points in both dimensions. This means that the number of grid points is 1200×1200, such that each grid point is separated by approximately $90m$ instead of $30m$.

The terrain data was parametrized by the three regression schemes, OLS, Ridge and Lasso, with polynomial design matrices. In order to study the best parameters, the polynomial degree and hyperparameter (for Ridge and Lasso) were varied. For all combinations of parameters, the MSE was found by using k-fold cross validation with $k = 5$.

IV. RESULTS

A. Franke's Function

Figure 3 shows a fit on Franke's function (with noise) with OLS by using a 5th degree polynomial.

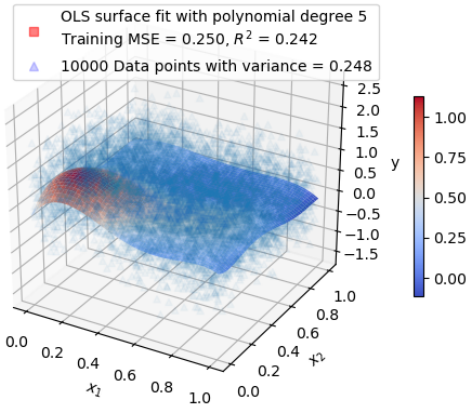

FIG. 3. 5th degree polynomial fit on Franke's function data with noise by using the OLS regression method. The mean squared error and R^2 score are evaluated on the training data.

Figure 4 shows the confidence intervals of the β parameters for the same 5th degree polynomial fit as in figure 3. The confidence intervals are illustrated with one σ uncertainty.

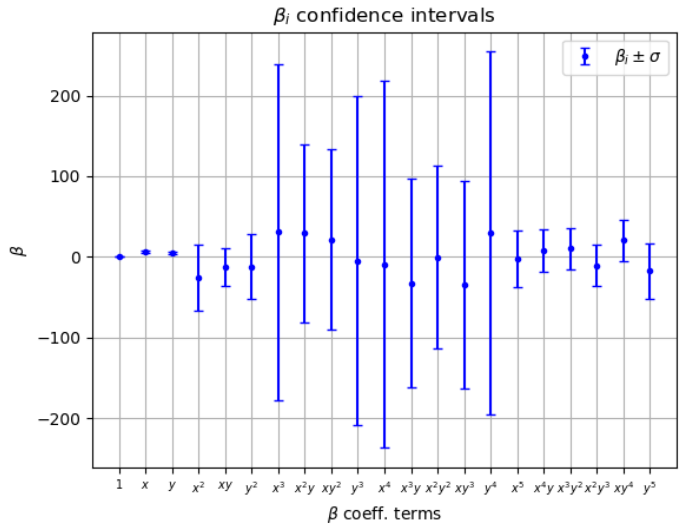

FIG. 4. β_i coefficients for a 5th degree polynomial, with errorbars corresponding to $\pm\sigma$ (one standard deviation).

Figure 5 presents the model complexity analysis conducted on the OLS method. Here, an increasing polynomial degree corresponds to increasing the model complexity.

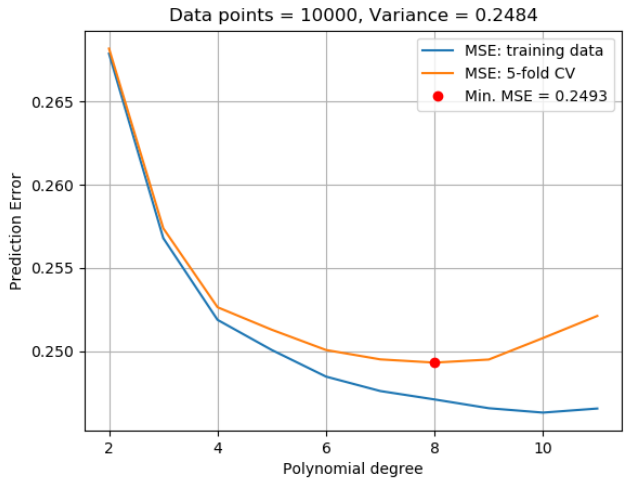

FIG. 5. An illustration of the prediction error of Franke's function as a function of the polynomial degree. Both MSE cases with and without resampling are included. For the training error, the MSE uses 100% of the data for training and evaluation. For the resampling method, the MSE is estimated by setting $k = 5$ in k-fold CV. The minimum MSE of the resampled data vs. polynomial degree is included.

Figure 6 shows the MSE evaluated with both the training set and with k-fold CV for various polynomial degrees and various number of data points.

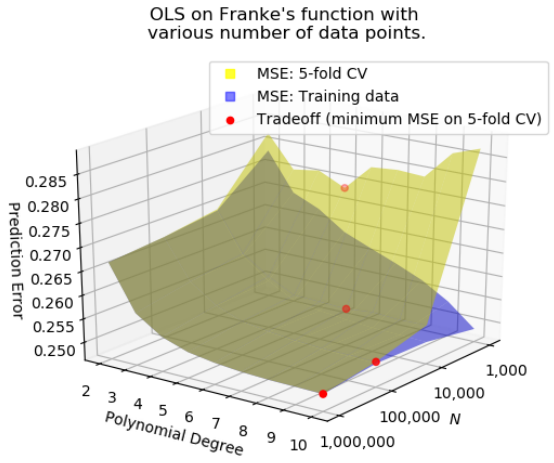

FIG. 6. OLS on Franke function data with noise. The training error (MSE on training set) and test error (MSE evaluated with 5-fold CV) is shown. Additionally, the bias-variance trade-off is shown for each N.

Figure 7 and 8 shows various estimations of the MSE and R^2-score is shown. The regression method is OLS on Franke function data with noise.

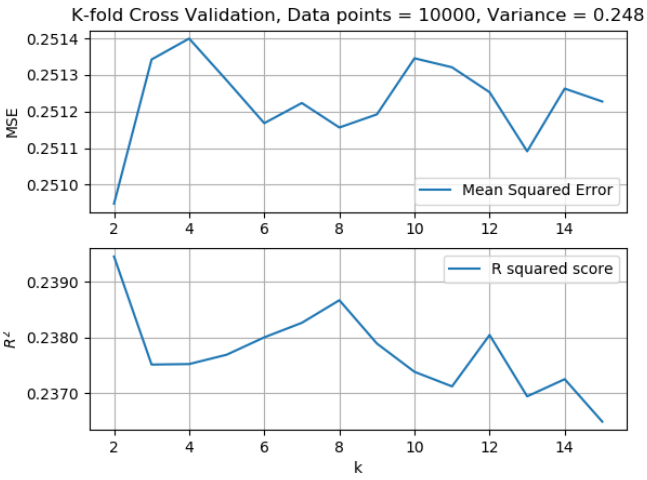

FIG. 7. OLS on Franke function data with noise: the figure shows the evaluated MSE and R^2-score for various choices of k in k-fold CV.

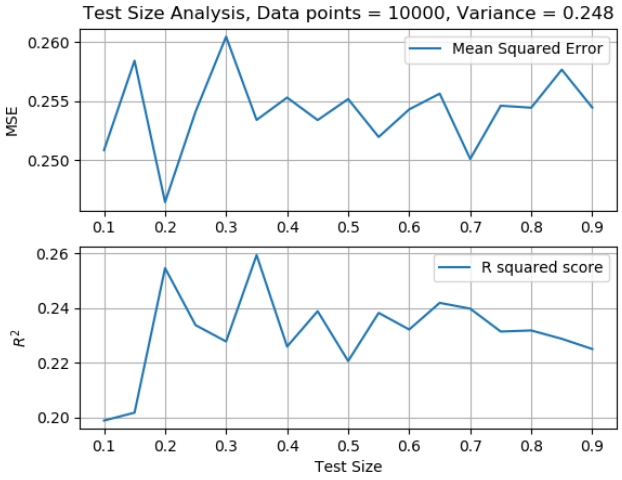

FIG. 8. OLS on Franke function data with noise: the figure shows MSE and R^2-score evaluated on the test set with various test set sizes.

Figure 9 shows MSE for Ridge regression on Franke function data with noise applied on various polynomial degrees and λ-parameter.

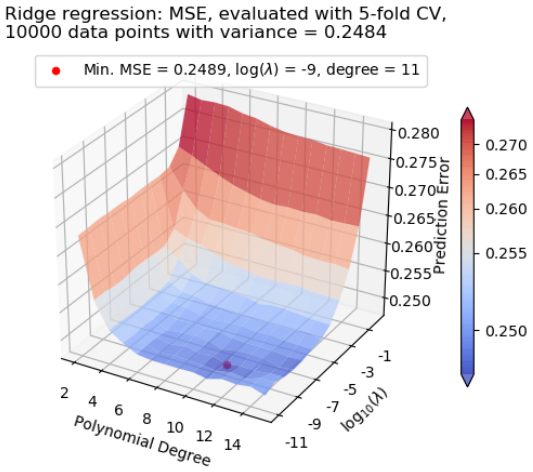

FIG. 9. Ridge regression on Franke function data with noise: the MSE is evaluated by using k-fold CV with $k = 5$ for various polynomial degrees and λ-parameters. The Ridge regression was done with sklearn with parameter: normalize=True.

Figure 10 shows MSE for Lasso regression on Franke function data with noise applied on various polynomial degrees and λ-parameter.

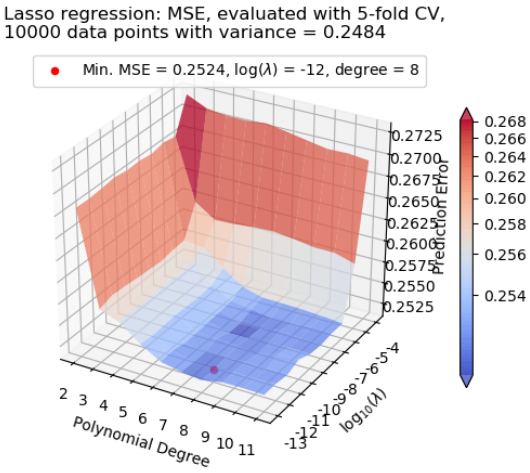

FIG. 10. Lasso regression on Franke function data with noise: the MSE is evaluated by using k-fold CV with $k = 5$ for various polynomial degrees and λ-parameters. The Lasso regression was done with sklearn with parameters: tol=0.05, max_iter=2500 and normalize=True.

Table II lists the results of the comparison of the three methods of regression on Franke's function.

TABLE II. Table listing the final results and comparisons of the regressional methods applied to Franke's function with 10,000 data points.

Scheme	MSE minimum	p_{deg}	log(λ)
OLS	0.2493	8	–
Ridge	0.2489	11	-9
Lasso	0.2524	8	-12

B. Terrain Data

Figure 11 shows the MSE evaluated with 5-fold CV with the OLS scheme.

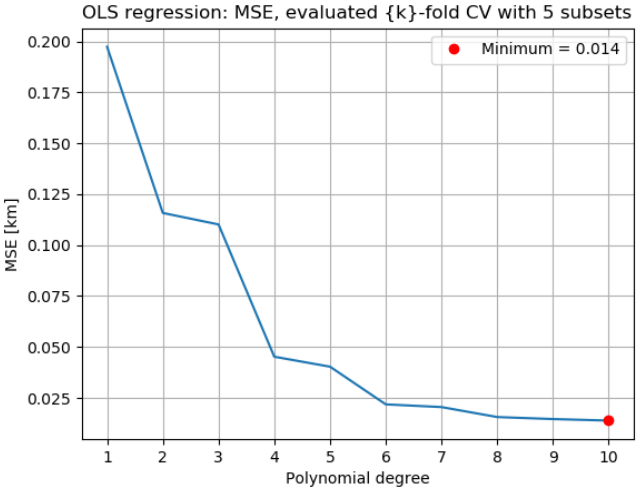

FIG. 11. OLS regression: MSE with k-fold CV with $k = 5$ vs. polynomial degree.

Figure 12 shows the MSE evaluated with 5-fold CV with Ridge regression for various polynomial degrees and hyper-parameters.

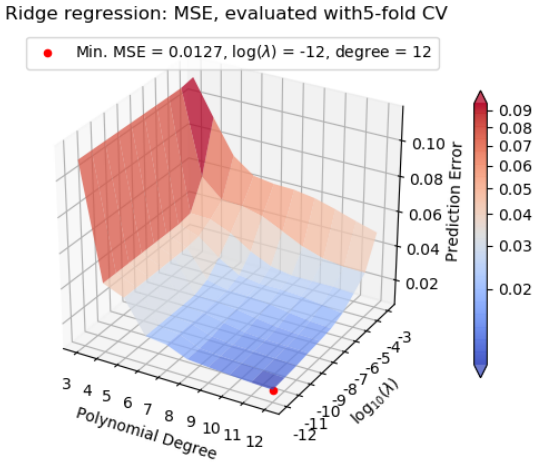

FIG. 12. Ridge regression: MSE with k-fold CV with $k = 5$ vs. polynomial degree and $\log_{10}(\lambda)$.

Figure 13 shows the MSE evaluated with 5-fold CV with Lasso regression for various polynomial degrees and hyper-parameters.

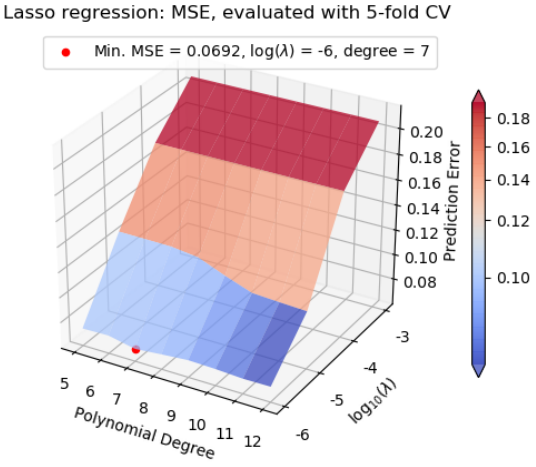

FIG. 13. Lasso regression: MSE with k-fold CV with $k = 5$ vs. polynomial degree and $\log_{10}(\lambda)$.

Table III lists the results of the comparison of the three methods of regression applied to the Terrain data.

TABLE III. Table listing the final results and comparisons of the regression methods applied to the terrain data.

Dead Sea Terrain			
Scheme	MSE minimum	p_{deg}	$\log(\lambda)$
OLS	0.014	10	–
Ridge	0.0127	12	-12
Lasso	0.0692	7	-6

V. DISCUSSION

A. Franke's Function

1. *Ordinary Least Squares*

Figure 3 shows the OLS regression using a 5th degree polynomial on noisy data of Franke's function. The entire data set was used for training in this case. The training error was found to be 0.250, which doesn't say much about the model. The R^2-score was found to be 0.242, signifying that the model was quite poor at capturing the variability of the data around Franke's function. This is might be due to a slightly low polynomial degree.

In figure 4, the $\hat{\beta}$ confidence intervals for the 5th degree polynomial fit is shown. Both the intercept β_0 and the first degree terms are small in terms of both value and uncertainty. The 2nd and 5th degree terms are larger both in value and uncertainty. Finally, the $\hat{\beta}$ coefficients for the 3rd and 4th degree polynomials have much larger confidence intervals than the other degrees. This trend of increasing errorbars $\pm\sigma$ for increasing p_{deg} is reversed in

the 5th degree polynomial coefficients, all of which have errorbars similar to that of the 2nd degree polynomials. Another feature which is shown in the figure is that the confidence intervals $\pm\sigma_\beta$ are largest for the coefficients which don't represent mixed variables (e.g the confidence interval for x^3 and y^3 are greater than x^2y and xy^2). This makes sense, because the Franke function consists of exponential terms which are a mixture of both x and y.

Figure 5 shows the train- and test error as a function of model complexity. The theory states that the training error should decrease indefinitely, because the model overfits the data. Figure 5 shows a decreasing training error, as the theory predicts. However, the training error increases from the 10th to the 11th polynomial degree. This last increase is likely to be due to numerical instability in the matrix inversion methods. The test error, which was evaluated with 5-fold CV behaves differently from the training error. It decreases until it reaches its minimum MSE at an 8th degree polynomial, and then it increases. This is in good agreement with the theory presented, illustrated in figure 1. The minimum MSE of the test error, which was obtained at an 8th degree polynomial, finds the optimal bias-variance trade-off for this case. The minimum MSE (using training data) was found to be $MSE(p_{deg} = 8) = 0.2493$. This value does not differ considerably from the neighboring polynomial degrees $p_{deg} = 7$ to $p_{deg} = 9$, allowing for some choice of polynomial complexity. Seeing as $p_{deg} = 7$ has a lower requirement for computational power, the error difference could be neglected to potentially increase the number of data points of the analysis (and keep the same computation time). Whether the error difference between polynomial degrees 7 and 9 remains equally negligible for larger data sets is a subject which would require further elucidation.

Figure 6 shows the training and test error for the OLS method applied to Franke's function with noise as before. The addition of this figure is that the number of data points N is varied. As can be seen in the figure, the test and training errors deviate from each other far more for lower values of N. This is due to the bias-variance trade-off being reached quicker (at a lower complexity) for lower values of N. This is straightforward to deduce by considering the extreme case of a few data points N. In that case, a low polynomial degree can almost exactly fit the data regardless of the original function (noise also has a far larger impact in the low N limit).

Figures 7 and 8 show the two analyses which estimate the MSE and R^2 scores. In figure 7, the k parameter in k-fold CV is varied between 2 and 15. The figure shows that both the MSE and R^2 are fairly stable, regardless of the choice of k. This is except for $k = 2$, which provided the largest deviation from the other results. The case of $k = 2$ might give the misconception of a good model, but this is not for certain at all. In figure 8, it is shown that the MSE and R^2 scores vary more than for k-fold CV with different k's. This means that the k-fold CV provides a more stable estimate for MSE and R^2 than one single split into training and test set. This is as was expected.

2. Ridge Regression

Figure 9 shows the estimated MSE for Ridge regression on Franke's function data set with noise. The surface illustrates the MSE produced by 5-fold CV. The surface was prodcued by varying the complexity p_{deg} and hyperparameter λ for the Ridge regression scheme.

The figure shows that Ridge regression is sensitive to values of λ from 10^{-4} and larger, where the MSE becomes increasingly larger almost regardless of model complexity. The figure also indicates that with λ values lower than 10^{-4}, the result is remarkably stable. Concerning the polynomial degree, the model is almost at its best already at the 6th degree and the MSE is very stable at all degrees from the 6th and above. Unlike the OLS regression, the MSE does not increase much at higher model complexities due to an increasing variance.

The minimum $MSE = 0.2489$ was the lowest of all three regression schemes, and it is noticeably close to the irreducible error of 0.2484 (variance in noise on Franke's function data). This minimum was obtained at an 11th degree polynomial with $\lambda = 10^{-9}$.

3. Lasso Regression

Figure 10 shows the estimated MSE for the lasso regression applied to the Franke's function data set with noise. The surface illustrates the MSE produced by 5-fold CV. The surface was prodcued by varying the complexity p_{deg} and hyperparameter λ for the lasso regression scheme.

The figure shows that this scheme is quite stable to λ values on and below 10^{-6}, but values larger than this gives a far worse result, as the MSE quickly increases. Regarding the polynomial degree, this scheme is fairly stable from about the 7th degree and above. Similar to Ridge regression and unlike OLS regression, the MSE does not increase remarkably at higher polynomial degrees due to increasing variance. The reasoning for this is hard to say, as the lasso regression was performed by a complex scikit-learn library function. The minimum $MSE = 0.2524$ was obtained at an 8th degree polynomial (same as OLS) with hyperparameter $\lambda = 10^{-12}$.

For all three schemes on this data set, this is the worst result, although it is not far behind OLS and Ridge regression. It is worthy to mention that the gradient descent method, which sklearn uses, might need a higher number of iterations than the 2500 which was used. At this number, the algorithm already had a far slower execution time than the other two schemes. The convergence criterion

was not met, meaning that it was the number of iterations which decided how good result this method yielded, not the tolerance. Regardless, the maximum number of iterations was chosen to not be incremented due to the execution time, which was remarkably slow for the higher polynomial degrees. A more detailed study of the lasso scheme would be interesting to conduct, as the execution time limited the extent of this particular paper.

B. Terrain Data

Figure 11 illustrates the OLS regression applied to the dead sea terrain data. The MSE decreases when the polynomial degree increases. There are many more features in the terrain data than there are in Franke's function, so it's therefore reasonable that the optimal model complexity is higher in this case. The analysis ended at $p_{deg} = 10$, because higher polynomial degrees caused large instabilities. It would be interesting to see where this MSE begins to increase again if a more stable inversion method than the two presented in this report is used.

Figure 12 illustrates the ridge regression method's MSE performance on the terrain data as a function of the polynomial degree p_{deg} and hyperparameter λ. Both these parameters have a trend of decreasing the MSE for larger p_{deg} and smaller λ, and the search for a minimum did not exceed $p_{deg} = 12$ and $\lambda = 10^{-12}$. It could very well be that the MSE keeps decreasing for larger p_{deg} and smaller λ, though these calculations required a large amount of computational power and were not conducted for this research.

Figure 13 illustrates the Lasso regression method's MSE performance on the terrain data as a function of the polynomial degree p_{deg} and hyperparameter λ. The MSE clearly decreases with a smaller λ. This research did not include λ values smaller than 10^{-6}, as the execution time was very slow for smaller λ values. The lowest MSE was achieved at a 7th polynomial degree, but this might be affected by the maximum number of iterations and the tolerance for convergence (set to 2500 and 0.05, respectively). A further improvement for the research concerning the Lasso method would be to increase the maximum number of iterations and to decrease the tolerance.

Table III lists the complete comparisons of the three methods applied to the terrain data. The ridge method found the lowest MSE of the three at MSE= 0.0127 with $p_{deg} = 12$ and $\lambda = 10^{-12}$. Not very far behind, is the OLS method, with MSE= 0.014. The Lasso method however, achieved at best, MSE= 0.0692, which is much higher than the other two methods.

The methods varied largely in computational time, something this study did not account for in the comparison. This factor is essential in a high performance computing context, and would be interesting to look further into.

VI. CONCLUSION

The study of the data set produced by Franke's function provided useful insights about the regression schemes. The k-fold cross validation method was found to be a very useful resampling tool to provide consistent estimates of the characteristics of the regression schemes presented. The study revealed that larger data sets required higher polynomial degrees to find the optimal regression prediction $\hat{\beta}$. The study found that the Ridge regression scheme gave the most accurate prediction of both Franke's function and the terrain data. For Franke's function, the Ridge scheme achieved MSE= 0.2489 using $p_{deg} = 11$ and $\lambda = 10^{-9}$. For the Terrain data, the Ridge scheme achieved MSE= 0.0127 using $p_{deg} = 12$ and $\lambda = 10^{-12}$.

There are several further improvements which can succeed the study presented; a potential follow-up study could involve an increase in the polynomial degrees and hyperparameter variants which were considered in this report. Additionally, a larger design matrix diversity can be considered, as the regression conducted is very specific to the two-variable design matrix polynomials $(x_1 + x_2)^{p_{deg}}$. Another central element to consider is the execution time, which was strikingly different for the three schemes. This is an important aspect, because both the model performance and computational efficiency are two essential considerations when choosing a regression method. A study comparing the performance of methods in relation to their computational time would be an even more thorough assessment of the methods in a high performance context.

REFERENCES

[1] Bishop, E. (1961). A generalization of the stone-weierstrass theorem. *Pacific Journal of Mathematics*, 11(3):777–783.

[2] Chichignoud, M., Lederer, J., and Wainwright, M. J. (2016). A practical scheme and fast algorithm to tune the lasso with optimality guarantees. *Journal of Machine Learning Research*, 17(229):1–20.

[3] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[4] James, G. (2017). *An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics)*. Springer.

[5] Khandani, A. E., Kim, A. J., and Lo, A. W. (2010). Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, 34(11):2767–2787.

[6] Morten, H.-J. (2015). Lecture notes fall 2015.

[7] Shipp, M. A., Ross, K. N., Tamayo, P., Weng, A. P., Kutok, J. L., Aguiar, R. C., Gaasenbeek, M., Angelo, M., Reich, M., Pinkus, G. S., Ray, T. S., Koval, M. A., Last, K. W., Norton, A., Lister, T. A., Mesirov, J., Neuberg, D. S., Lander, E. S., Aster, J. C., and Golub, T. R. (2002). Diffuse large b-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning. *Nature Medicine*, 8(1):68–74.

APPENDIX A: OLS PREDICTOR DERIVATION

Setting the Residual Sum of Squares (RSS) partial derivative with respect to the predictor β equal zero yields:

$$\frac{\partial}{\partial \beta}\ RSS = 0 \tag{28}$$

Inserting the expression for RSS as described in equation 2 yields:

$$\frac{\partial}{\partial \beta}\ (y - X\beta)^2 = 0, \tag{29}$$

rewriting as:

$$\frac{\partial}{\partial \beta}\left((y - X\beta)^T (y - X\beta)\right) = 0, \tag{30}$$

expanding:

$$\frac{\partial}{\partial \beta}\left(y^T y - y^T X\beta - (X\beta)^T y + (X\beta)^T (X\beta)\right) = 0, \tag{31}$$

removing the factors which are independent of β and utilizing the rule

$$\left(A^T B\right)^T = B^T A \tag{32}$$

yields:

$$\frac{\partial}{\partial \beta}\left(-y^T X\beta - \beta^T X^T y + \beta^T X^T X\beta\right) = 0, \tag{33}$$

Also, seeing as the product $y^T X\beta$ is a scalar, we have that:

$$y^T X\beta = \left(y^T X\beta\right)^T = \beta^T X^T y \tag{34}$$

Since the transposed of a scalar equals the scalar. This allows us to rewrite equation 33 to

$$\frac{\partial}{\partial \beta}\left(-2\beta^T X^T y + \beta^T X^T X\beta\right) = 0, \tag{35}$$

Differentiating with respect to β yields:

$$-2X^T y + \frac{\partial}{\partial \beta}\left(\beta^T X^T X\beta\right) = 0 \tag{36}$$

One more matrix-vector differentiation formula is needed for the final product:

$$\frac{\partial(a^T A a)}{\partial a} = 2Aa = 2a^T A \tag{37}$$

for vectors a and a symmetric matrix A. We can now derive the final formula for β:

$$-2X^T y + \frac{\partial}{\partial \beta}\left(\beta^T X^T X\beta\right) = 0 \tag{38}$$

$$-2X^T y + \frac{\partial}{\partial \beta}\left(\beta^T A\beta\right) = 0 \tag{39}$$

$$-2X^T y + 2A\beta = 0 \tag{40}$$

$$-2X^T y + 2X^T X\beta = 0 \tag{41}$$

$$-X^T y + X^T X\beta = 0 \tag{42}$$

by the declaration of $X^T X = A$ being symmetric. This finally yields:

$$X^T X\beta = X^T y \tag{43}$$

$$\beta = \left(X^T X\right)^{-1} X^T y. \tag{44}$$

APPENDIX B: COST FUNCTION RELATION

Expanding the cost function

$$C(X, \beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 = \mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] \tag{45}$$

by addition and subtraction in the expectation value of the squared component yields:

$$\mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] = \mathbb{E}\left[\mathbf{y}^2 - 2\mathbf{y}\hat{\mathbf{y}} + \hat{\mathbf{y}}^2\right] \tag{46}$$

expansion of the expectation value:

$$\mathbb{E}\left[\mathbf{y}^2 - 2\mathbf{y}\hat{\mathbf{y}} + \hat{\mathbf{y}}\right] = \mathbb{E}\left[\mathbf{y}^2\right] - \mathbb{E}\left[2\mathbf{y}\hat{\mathbf{y}}\right] + \mathbb{E}\left[\hat{\mathbf{y}}^2\right] \tag{47}$$

Utilizing the following relations:

$$\mathbb{E}\left[\epsilon^2\right] = Var(y) \tag{48}$$

$$\mathbb{E}\left[\hat{\mathbf{y}}^2\right] = Var(\hat{\mathbf{y}}) + \mathbb{E}\left[\hat{\mathbf{y}}\right]^2 \tag{49}$$

$$\mathbb{E}\left[\epsilon\right] = 0 \tag{50}$$

$$\mathbb{E}\left[\mathbf{y}^2\right] = \mathbb{E}\left[(f + \epsilon)^2\right] = \mathbb{E}\left[f^2 + 2f\epsilon + \epsilon^2\right] \tag{51}$$

$$= \mathbb{E}\left[f^2\right] + 2\mathbb{E}\left[f\epsilon\right] + \mathbb{E}\left[\epsilon^2\right] \tag{52}$$

$$= f^2 + \sigma_\epsilon^2 \tag{53}$$

$$\mathbb{E}\left[\mathbf{y}\hat{\mathbf{y}}\right] = f\mathbb{E}\left[\hat{\mathbf{y}}\right] \tag{54}$$

$$\tag{55}$$

allows us to rewrite formula 47 to:

$$\mathbb{E}\left[\mathbf{y}^2\right] - \mathbb{E}\left[2\mathbf{y}\hat{\mathbf{y}}\right] + \mathbb{E}\left[\hat{\mathbf{y}}\right] = f^2 + \sigma_\epsilon^2 - 2f\mathbb{E}\left[\hat{\mathbf{y}}\right] + \mathbb{E}\left[\hat{\mathbf{y}}^2\right] \tag{56}$$

and furthermore:

$$\mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] = f^2 + \sigma_\epsilon^2 - 2f\mathbb{E}\left[\hat{\mathbf{y}}\right] + Var(\hat{\mathbf{y}}) + \mathbb{E}\left[\hat{\mathbf{y}}\right]^2. \tag{57}$$

Recreating the square:

$$\mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] = f^2 - 2f\mathbb{E}\left[\hat{\mathbf{y}}\right] + \mathbb{E}\left[\hat{\mathbf{y}}\right]^2 + \sigma_\epsilon^2 + Var(\hat{\mathbf{y}}) \tag{58}$$

$$= (f - \mathbb{E}\left[\hat{\mathbf{y}}\right])^2 + \sigma_\epsilon^2 + Var(\hat{\mathbf{y}}). \tag{59}$$

$$= Bias^2(\hat{\mathbf{y}}) + \sigma_\epsilon^2 + Var(\hat{\mathbf{y}}). \tag{60}$$

Utilizing the definition of the bias of $\hat{f}$. Rewriting the bias and variance definitions in terms of the components of the vectors yields:

$$= \frac{1}{n}\sum_{i=0}^{n-1}(f_i - \mathbb{E}\left[\hat{\mathbf{y}}\right])^2 + \sigma_\epsilon^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\hat{y}_i - \mathbb{E}\left[\hat{\mathbf{y}}\right])^2 \tag{61}$$