A Numerical Study of Linear Regression and Resampling Methods

Felicia Jacobsen
(Dated: October 10, 2020)

ABSTRACT

A comprehensive numerical analysis was applied on linear regression and resampling methods on data computed by Franke function as well as real terrain data. We included Ordinary Least Squares, as well as the regularization methods Ridge and Lasso regression. A polynomial design matrix was included in the model. Cross validation and Bootstrap resampling was performed in order to tune and validate the models for different parameters such as polynomial degree and penalization parameter. Ridge regression gave most promising results for Franke function data with a mean squared error of 0.0462 with polynomial degree 6 and penalty of 10^{-5} tuned by CV. With terrain data, the OLS gave the highest performance with MSE of 1583.918 m^2 with polynomial degree 8 tuned by CV.

I. INTRODUCTION

The main task is to make a comprehensive regression analysis on the Franke function, as well as linear regression methods such as the Ordinary least squares-(OLS), the Ridge-, and finally the Lasso regression method. Combining these tools with well-known resampling methods such as the Bootstrap- and the K-fold Cross validation (CV) method are used as a foundation for the study of bias-variance trade-off.

The motivation of this computational project is to study the relationship between input and output variables. What does this look like in practice? An example is observing genes for patients with and without cancer. What genes are common for those with cancer, and which of them are not present in patients without cancer? And can these genes reveal the impact of the cancer?

With applying regression analysis of this kind of data, we can determine if these variables have impact on the model of interest, and if so to what extent. This analysis can determine which factors matter the most, which factors can be ignored and how these factors affect each other.

Machine learning is an important tool in modern science, because it allows a system to automatically learn and to improve from additional experience without being explicitly programmed to do so. In the part of machine learning that we call "supervised learning", an algorithm can apply what has been learned in the past to new data using labeled data to predict future events.

A linear model makes a prediction by computing the weighted sum of the input features. How do we train such a model to predict new outputs? First, we need to determine a set of parameters (hyperparameters) included in the model so that it correctly fits the available data. This is where resampling methods comes in. Such methods involve drawing samples of data (training data) and refitting the model on each sample in order to find important information about the model, such as the best hyperparamteres.

The OLS regression method is such a linear model. This estimates the relationship between one or more independent variables and a independent variable. It is governed by a so-called cost function which is the sum of the squared of the distance between the observed value, and the predicted value. We want to minimize this function in order to find a model which is as close to the actual data as possible.

If the model is too well tailored onto the data which is included in the training, it will as a consequence generalize poorly onto new data. In order to reduce this form of error we introduce two regularized linear models: the Lasso and Ridge method. They are evidently similar to the OLS, but they include an additional term. This is often called the penalty term and reduces the weights of the included features in the model.

We will first exploit these regression- and resampling methods on our own generated data as our input, and Franke function as our output data. After experimenting, we will apply them onto real terrain data. Finally, we will evaluate the performance of these methods according predicting the surface of this data set.

This rapport will we sectioned into sections six parts; a theory section, method section, results, discussion, conclusion, and finally a glossary section. The theory section contain a full general description of terms and concepts relevant for this project. The method gives a full recipe of how to implement the methods used, and how the results were produced. The result section shows all of the produced plots, and results from the regression analysis. In the discussion we will explain and evaluate these results, and to justify the choices made in the method section in order to produce these result. Finally, we finalised the results with the conclusion. The last part contains the glossary, which is a small table of abbreviations used and their meaning.

II. THEORY

The Franke function needs two independent variables to compute an additional variable, which is a sum of four Gaussian terms. The function is often used as a tool to illustrate a terrain, taking two linearly independent variables, for example x- and y-data, and computes an linearly independent variable such as an z-variable.

Supervised learning

We will focus on the aspect of ML which only includes supervised learning. This means that we are training on model based on input and output (often called response or target) data that are already available. Supervised learning use labeled data in order to fit data on to desired output. In order to validate the models ability to predict, we put aside some of our original data and test how the trained model predicts the new data. The data that we hide is often called the test data, while the data that are used to train the model is of course denoted the training data.

Finally, we want to validate the performance of the trained model. With linear regression, the mean squared error and R^2 score are common evaluation tools.

Feature matrix

A feature matrix is a matrix of the features for a given response variable. Each feature characterizes this specific response. Usually, the columns in a given feature matrix is a set of data which specifies this characteristic, the rows are the specified feature data of different features which lead to a given response.

We sometimes want to give more complexity to a model. By model complexity, we refer to the number of features included in the model. Sometimes when we don't have enough feature data available, the given features that are available can serve as additional features to give higher complexity. In addition to increasing the complexity, this also adds to the dimensions of our model.

When having only one independent input variable, let's call this x. The feature matrix which follows a polynomial function of e.g. polynomial degree 3 will look like:

$$X = 1 + x + x^2 + x^3, \tag{1}$$

when the intercept is included. We now have a total of 4 features. When having two independent input variables, x and y, the feature matrix is defined as all polynomial combinations of the features with the specified degree, given as

$$X = 1 + x + y + x^2 + xy + y^2 + x^3 + x^2y + xy^2 + y^3, \tag{2}$$

and we now have a total of 10 features in our model. The features have a linear relationship to each other. In general: when having one input variable, choosing a model of polynomial degree p, we have a total of $p + 1$ features, but when we have two independent variables, a model of degree n will give us a total number of $\frac{(p+1)(p+2)}{2}$ features. Having two input variables compared to having one input variable gives us increasingly more features for increasing polynomial degree.

Suppose we have a total of n data points. When having a polynomial feature matrix up to degree p with one independent input variable, the dimension of the matrix will be of $X \in R^{n \times p+1}$. When having two independent input variables, the polynomial feature matrix will have dimension $X \in R^{n \times \frac{(p+1)(p+2)}{2}}$. For example, when p=2, we get a $n \times 21$ matrix.

Having a polynomial feature matrix has two advantages of

1. Being mathematically simple, and
2. offering simple treatment regrading the use of the least square criterion.

From point 2), since $y = \beta_0 + \beta_1 x + \beta_2 x^2 + ...\beta_{p-1} x^{p-1}$ is linear in the parameters, it can be written as $y = X \cdot \beta$. Note that this is true for feature matrices in general and not only for polynomial matrices.

Bias-variance trade-off

As mentioned earlier, the model complexity is reflected by observing the bias and variance of the model. Overfitting is a common problem when performing prediction analysis. This essentially means that the model will be well-tailored to the training data, and eventually capture some of the corresponding noise. For example when polynomial degree of a model becomes too large, they follow random fluctuations in the data. A too complex model will be overly tailored on the available data, and as a result not generalize on new data. In this case, the model overfits the data and involves an excess of optimism in evaluating the prediction error.

A model with high complexity will often result in overfitting, and thus is a poor predictive model. Overfitting is often a problem when the number of features are large compared to to number of data points. Reducing model complexity and thus reducing the number of features can be a potential solution, but this

can essentially result in loosing valuable information about the data.
An alternative approach to reduce overfitting is using regularization methods, and we will discuss these methods later.

In terms of overfitting, the **variance** is often mentioned. The variance is a measure of the variability the predictive model, but in terms of a sample of data points the variance is a measure of the spread of the data. A model with high variance is well tailored onto training data, and generalize poorly on new data. Models with high variance overfits, and thus has low errors on training data but high error on test data.
The variance measures how much the data point will vary from experiment to experiment. It is the amount that the estimated target (or prediction) will change if different training data was used. This is exactly the case for many resampling methods, since we use different training data for every iteration. The variance for a variable y given by

$$Var(y) = E[y^2] - E[y]^2. \tag{3}$$

The **bias** is the difference between the mean prediction value and the actual data which we want to predict. This error is not due to noise, but due to a systematic error in between the predicted and observed output because of the inaccuracy of estimating the predicted values. Mathematically, the bias is given by

$$Bias[\hat{y}] = E[\hat{y} - y], \tag{4}$$

where $\hat{y}$ denotes the estimate of y. We're often interested in a positive measurement of the bias because we often want a magnitude of this error. The squared bias is often used when observing this error.

A model of low complexity has a high bias, and as a result it pays little to no attention to the characteristics of the training data. A bias component is usually due to the lack of information about the data-generating mechanism. The model is thus oversimplified and will add to the total error of the predictive model.

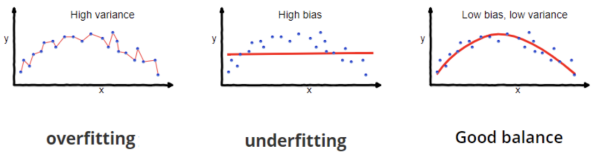

FIG. 1. Illustration of bias-variance trade-off point. Figure borrowed from [1].

Bias and variance are conflicting entities. It is impossible to simultaneously minimize both the variance and the bias, but the essential goal is to find a model which minimizes the total error, such that we need to obtain a balance between the bias and variance. In order to understand the fundamental behaviour of prediction models, it is essential to understand bias and variance.

Regression methods

The **ordinary least squares**(OLS) is a unregularized regression method. Regularized regression is a type of regression where the coefficient estimates are penalized and will either shrink(Ridge method) or sometimes shrink to zero(Lasso method). The OLS is governed by the cost function which is the squared difference between the predicted and the real/observed data. It's main task is to minimize the difference between the observed data and the prediction data. The coefficients which is in front of each feature in the feature matrix is what essentially the OLS wants to find. The set of coefficients is determined by the ones that minimizes the sum of squared errors, i.e. the cost function. The cost function of OLS is given by

$$\hat{\beta}^{OLS} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=0}^{n-1}(y_i - x_i^T\beta)^2, \tag{5}$$

where $\hat{\beta}^{OLS}$ denotes a vector of optimal coefficients in the model, y_i denotes the output variable, and x_i denotes the input data, β_i denotes a predicted coefficient, and n denotes the total number of data points. In this formulation, the explicit solution to the minimization problem of the eq. 5, is

$$\tilde{y} = X\beta = X(X^TX)^{-1}X^Ty. \tag{6}$$

When the optimal parameters $\hat{\beta}^{OLS}$ is found, the vector of fitted values is $\hat{y}^{OLS} = X\hat{\beta}^{OLS}$. The prediction $\tilde{y}$ which satisfies the cost function 5 is now denoted $\hat{y}$. Note that $\tilde{y}$ satisfies nothing, this is only predictions. The prediction and the observed data can be evaluated by the mean squared error(MSE) given by

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2. \tag{7}$$

We now observe that the cost function in OLS 5 is in other words the MSE(7). OLS is in other words governed by the mean square criterion.

Another measure of how well our model fits the data, is the R^2. The R^2 is called the coefficient of determination.

$$R^2 = \frac{\text{Explained deviance}}{\text{Total deviance}}, \tag{8}$$

$$= 1 - \frac{\text{Residual sum of squares}}{\text{Total sum of squares}}, \tag{9}$$

$$= 1 - \frac{\sum_i^n(y_i - \hat{y}_i)^2}{\sum_i^n(y_i - \overline{y}_i)^2}. \tag{10}$$

The R^2 is a measure on the fraction of variability explained by the model. A high R2 score is preferable, since a model which performs sufficiently will explain a large amount of variability. This score can at best be 1.0, which means that the given model explains 100% of the variability.

As mentioned earlier, the bias and variance is contained in the total error. We can in fact rewrite the MSE into a sum containing the variance and squared bias. Suppose we have some observed data with some additional noise

$$y = f(x) + \epsilon, \tag{11}$$

where the noise ϵ is normally distributed with a mean 0 and standard deviation σ^2. Before we start the deduction, we need to make some important statements.

1. The expected value of a sum has the property: $E[X + Y] = E[X] + E[Y]$.
2. If X and Y are independent variables, then: $E[X \cdot Y] = E[X] \cdot E[Y]$.
3. Assume $E[y] = f$.

Instead of writing the MSE as a sum, it can be written in terms of the expected value

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

$$= E[(y - \hat{y})^2] \tag{13}$$

$$= E[y^2 - 2y\hat{y} + \hat{y}^2]] \tag{14}$$

$$= E[y^2] - 2E[y\hat{y}] + E[\hat{y}^2], \tag{15}$$

we have assumed that y and $\hat{y}$ are independent, such that point 2) applies. Proceeding with subtracting and adding $E[y]^2$ and $E[\hat{y}]^2$, and applying point 1),

$$= E[y^2] - E[y]^2 + E[y^2] - 2E[y\hat{y}] \tag{16}$$

$$+E[\hat{y}^2] - E[\hat{y}]^2 + E[\hat{y}]^2, \tag{17}$$

$$= Var(y) + Var(\hat{y}) + f^2 + E[\hat{y}]^2 - 2fE[\hat{y}], \tag{18}$$

$$= \sigma^2 + Var(\hat{y}) + (f - E[\hat{y}])^2, \tag{19}$$

$$= \sigma^2 + Var(\hat{y}) + Bias(\hat{y})^2, \tag{20}$$

$$= \sigma^2 + \frac{1}{n}\sum_i^n(\hat{y}_i - E[\hat{y}])^2 + \frac{1}{n}\sum_i^n(f_i - E[\hat{y}])^2. \tag{21}$$

The **Ridge method** is a regression method of the regularized type. This essentially keeps all features, but they are reduced (or penalized). This method shrinks the magnitude of the model's coefficients, which is an efficient way of minimizing the model's variance, but it does however always add bias. Remember that we can't simultaneously minimize these two errors.

The Ridge method is an extended version of the OLS, where we have the same cost function 5, but with an additional term, often called the penalty term. Such that for coefficients with larger magnitude, the larger penalty. This means that the hyperparameter λ (also called the penalty parameter) forces the coefficients to be close to zero. Finding a tuning parameter is evidently crucial for arriving with a good predictive model, and can be found by several resampling methods such as the K-fold cross validation(CV) and the bootstrap method.

If we simply add a term to the OLS cost function:

$$\hat{\beta}^{Ridge} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=0}^{n-1}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=i}^{n}\beta_j^2, \tag{22}$$

The last term is often called the L2 penalty. This additional term will not only force the learning algorithm to fit the data but also to keep the model weights (coefficients) as small as possible. This term is only added to the model during the training of the model. Note that the bias term, β_1 is not regularized. By scaling the output data, we avoid penalizing the intercept.

Our original linear regression problem was to find parameters with equation

$$\beta = (X^TX)^{-1}X^Ty, \tag{23}$$

this matrix that we want to invert can be singular if the columns in the feature matrix X is not linearly dependent. This is often the case if X is high-dimensional. A solution to this problem is to simply add a diagonal component to the matrix that we want to invert,

$$\beta = (X^TX + \lambda I)^{-1}X^Ty, \tag{24}$$

where λ is the penalty parameter, and I is the identity matrix. This is in fact what we evaluate when applying the Ridge method.

Note that λ controls the regularization. If $\lambda = 0$, the Ridge regression reduces to OLS.

The **Lasso method** is similar to Ridge in the way that it also is a regularized regression method. The penalty term in Lasso is called the L1 penalty. The main difference is that Lasso not only shrinks the coefficients, but they may also become zero. Lasso tends to completely eliminate the weights of the least important features. This is due to the penalty term in the following cost function

$$\hat{\beta}^{Lasso} = \underset{\beta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=0}^{n-1}(y_i - x_i^T\beta)^2 + \lambda\sum_{j=1}^{n}|\beta_j|. \tag{25}$$

If the penalty parameter λ becomes very large, this will force some of the coefficient to shrink to zero. This

will essentially remove those terms from the model, and therefore performs automatic feature selection. The Lasso method works better for data where the response is best modeled as a function of a small number of the predictors, however there is no guarantee that the Lasso will reduce the features. Resampling methods like CV and bootstrap can be used to tune the penalty parameter λ and therefore decide if feature reduction determines a better prediction model or not.

In figure 2, we observe the ellipses which represent unregularized MSE cost function. The circle in the left figure represent the L2 (Ridge) penalty, and the diamond represent the L1 (Lasso) penalty. The center of the ellipses represent the optimal solution for the OLS, and the circling ellipses represent badder and badder estimates. Because of the geometric properties of the circle, the ellipse will never hit the circle along the axes. However for the diamond, the ellipse contour has a high probability of hitting the diamond edges, and this represents solutions where $\beta_i = 0$.

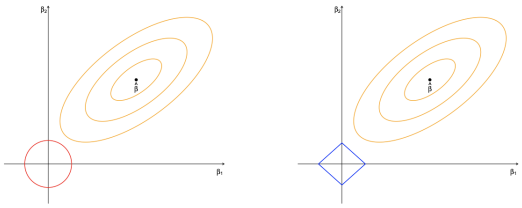

FIG. 2. Left figure: Ridge regularization. Right figure: Lasso regularization.

Resampling techniques

Resampling methods are a common tool when working with supervised learning. These methods will help with validating the model, such that they can improve the model's ability to predict unseen data.

When performing resampling methods, it is essentially important to start by splitting the whole data (input and output) into what we call the training and test set. The training set is used to train the model. We will not use the test set to no other purpose than to evaluate the trained model. We pretend that the test set is "unseen data" which the model's goal is to able to predict data closest to the data in the test set. If some information about the test set is included when training the model, the result can be an optimistic estimate of the model performance.

A resampling method involves repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain more information about the model's performance. The gain is the additional information that we don't get from fitting the model only one time.

Two common methods which we will study and apply is the K-fold CV and the bootstrap method.

K-fold cross validation is a resampling technique mainly used for tuning hyperparameters that potentially will improve a prediction model. When evaluating models with different complexity in the terms of polynomial degree in the feature matrix, it essentially compares the models of different degrees to each other. CV may also tune other parameters as well, in terms of Ridge and Lasso, it can tune the parameter λ which regularize the variance of the prediction model.

For example, for 5-fold CV we start by splitting the data, output- and input data, in training and test data. The test data is not used until the end, when we want to evaluate the predictive performance of the model. If we choose K=5, it means that we want to split the training data into 5 parts, these parts are what we call "folds." Starting with one of the five folds and renaming this as the validation set, and the 4 remaining folds are used as a temporary training set. We train our model on the temporary training set and we predict the trained model. The trained model are used to predict the validation set and a score of the model is computed, we often use the MSE. In many cases, using only one validation set will not be representative of the total training set, and the resulting score will be bad compared to if we used an other validation set. To solve this problem, each fold is in turns the validation set. We have a resulting score for every validation set used. We thus have 5 scores for every model complexity, and the total score for one complexity will be a mean of the five scores. The tuned parameters are taken from the prediction model which gives the best score. Finally, we have a prediction model based on the tuned parameters. The test set which we in the beginning sat to the side is now used to validate the final model.

The **Bootstrap resampling method** will also be used to validate and to tune hyperparameters as an alternative to the K-fold CV method. Starting by splitting the original dataset into training and test set, we now perform bootstrap on the training set.

We randomly draw an observation from the training set, and add it to a new sample that we call the bootstrap sample. After drawing one sample, we replace it back in the original sample. We repeat this until the bootstrap sample is as long as the original sample. All of the observations that are not included in the bootstrap sample is added to a smaller sample that we call out of bag observations. The bootstrap sample and the OOB observations is analogous to the temporary training data and test data from K-fold CV respectively. We train

the model based on the bootstrap samples, and validate the trained model based on OOB sample. There may be cases where the OOB sample is not representative of the whole dataset. This is when the validation of the trained model fails, this will be compensated by the fact that we perform this many times. The algorithm is built by the following steps

1. Choose size of bootstrap sample.
2. Randomly draw an observation from the dataset with replacement.
3. Add to the bootstrap sample until sample becomes the chosen size.

For every bootstrap sample we calculate the statistic on the sample. And after every iteration, the mean is calculated of the sample statistics.

III. METHOD

How to process the data

The dataset(input data) which we generate for the Franke function is uniformly distributed between 0 and 1 in both x- and y-direction. We generate a total of 1000 datapoints where each point is a set of a x and y coordinate. The Franke function which take x and y, serve as a third coordinate, which we will denote as z-data, i.e. this is our response (or target) data. The data x and y are one dimensional vectors of length n, when taking this into the feature matrix X, the matrix is of dimension $n \times \frac{(p+1)(p+2))}{2}$, for polynomial degree p which will be varied. The entire data set consists of N points, with two one-dimensional input arrays, x and y, and one one-dimensional response variable, z. However, we will extend the number of n input variables by creating a design matrix, where the columns consists of various polynomials of x and y. We use x and y to make additional features in the model, adding complexity. The aim is to find an appropriate combination of polynomials that will provide a model for the Franke function, which generalizes well.

The design matrix is determined by the polynomial degree. When the polynomial degree p is chosen, the number columns in the feature matrix(number of features) is $\frac{(p+1)(p+2))}{2}$. Thus, the matrix looks like

$$X = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & x_1^3 & x_1^2y \cdots y_1^p \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 & x_2^3 & x_2^2y_2 \cdots y_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & x_n^2 & x_ny_n & y_n^2 & x_n^3 & x_n^2y_n \cdots y_n^p \end{bmatrix} \tag{26}$$

where n=1000, the number of data points for x, y and z.

For polynomial degree p, the design matrix is made by:

1. vector of x^0 or intercept terms is multiplied with a vector $[1, y, y^2, ..., y^p]$
2. vector of x is multiplied with $[1, y, y^2, ..., y^{p-1}]$
3. vector x^2 terms is multiplied with $[1, y, y^2, ..., y^{p-2}]$
4. vector x^n terms is multiplied with $[1, y, y^2, ..., y^{p-n}]$,
   $\vdots$

this procedure continues until $n = p$.

When **splitting** the feature matrix and output data into training and test set, we use a module from Scikit-learn which does this for us. Specifying that 20% of the data should be in the test set, we now take out random observations and put these into a subset called the training set until this set includes 80% of the total data points. The feature matrix X is split into training and test set, and the output data z is split with respect to the split of X, such that correct input data corresponds to the correct output data.

The **standardizing** of the feature matrix X is done by taking the mean of each vector column, and then subtracting the mean of each column, respectively. Then, the standard deviation is calculated for every column, and every element in that specific column is divided by the standard deviation. What standardizing does is centering the data around zero and scale to unit variance. Let's say for element i in column j, when performing basic standardizing:

$$\frac{(x_i - \mu_j)}{\sigma_j} \tag{27}$$

The standardizing of data is done with respect to the training data, such that the scaling is done after the splitting of data into training and test set. Making unit variance will avoid huge values. Since our data already lies in an interval [0,1], we do not standardize such that we obtain unit variance, and we only subtract the mean from ever column, this is done by using **preprocessing.StandardScaler(with_std=False)** from Scikit-learn.

Note that we scale training- and test set of X with respect to the scaling of training set of X, such that the training set does not contain information about the test set. We do the same for the output data z.

Implementing regression methods

The **OLS** is implemented by implementing the eq. 6. In our case, y is the output data from the Franke function z, and X is our feature matrix.
The coefficients β_i are found by fitting the OLS on the

training data, but when applying CV and bootstrap, this will serve as the temporary data. The product of coefficients computed by fitting the model on the training data X, and the test data X will serve as the prediction of the unseen output data. We need to use **numpy.linalg.pinv()** in order to compute the inverse of $(X^T X)$, this is also use to increase numerical stability. This matrix is often singular for large numbers of data. If the determinant of the matrix is zero, the inverse does not exist, and we say that the matrix is singular. **numpy.linalg.pinv()** returns the inverted matrix if the matrix is not singular, but will return the pseudo-inverse when it is singular.

The pseudo-inverse which it returns is in fact called the Moore–Penrose inverse, and it is a generalization of the inverse matrix. It is calculated using the singular value decomposition.
When applying resampling techniques, the X in this matrix we want to invert is actually the temporary training subset of the feature matrix X.

The **Ridge method** is an extension of the implementation of the OLS. With this method, the matrix that we want to invert has an additional term, $(X^T X + I\lambda)$.
When performing Ridge, we standardize both the X- and z-data, after splitting into training and test sets in order to avoid penalizing the bias term, this may cause large errors when training the model.

The **Lasso method** is implemented by using **sklearn.linear_model.Lasso** since Lasso has a non analytical solution. What this module does is to apply the gradient descent method with the Lasso cost function. It starts with the unregularized MSE cost function, and continues to iterate until it converges towards a regularized solution. We therefore need to set the number of iterations and a tolerance for this module. Maximum iterations are set to 10000, and tolerance is 10^{-4}.
When applying the Lasso method, we standardize both the X- and z-data, after splitting into training and test sets. This is to avoid singularities and penalization of the bias term in the model.

Implementing resampling methods

When performing **K-fold CV**, a number of 5 folds is chosen. The training set is split into 5 equal folds. We loop over a total of 5 times, and in iteration i we train on the temporary set and validate the model by the validation set. We repeat this 5 times, until every fold has been used as a validation set.

Since each observation in X is row-wise, the split of folds is along vertical axis (axis 0 in numpy). When making the temporary set by the four remaining folds, we need to concatenate the matrices along the horizontal axis, which is axis 0 in numpy.

The MSE is computed for every validation. For each polynomial degree, we have 5 MSE's, which we compute the mean of. We can also compute the MSE with the a prediction on the trained data. The resulting MSE vectors is computed with respect to the polynomial degree of feature matrix X. The best parameters after tuning the parameters with either bootstrap or CV, are with respect to the minimum computed MSE, as it satisfies the cost function.

The **bootstrapping** is applied after splitting the data into training and test set. This method is implemented by making an array of random indices. These indices are gathered by using **boot_index = np.random.randint(0, N, N)**, where N is the number of observations in the data sample that we want to make a bootstrap sample for. This set of indices can cotain several counts of the same index. In fact, on average around 64% of the unique indices make up this entire set, where some of them counts several times.

The out of box samples are generated using the remaining set of indices which was not used in the bootstrap sample, **OOB = [x for x in range(N) if x not in boot_index]**. This sample is the remaining unique indices which wasn't chosen, they contain approx. 36% of the unique indices.

We apply these two set of indices on the training set, such that bootstrap sample is X_temp = X_train[boot_index], which is what we use as a temporary training set. X_vali = X_train[OOB], which we use as a validation set of each bootstrap. The predicted model based on bootstrap samples is validated 100 times for OLS, Ridge and Lasso.

Generating plots

How to make a **bias variance plot** showing the clear characteristics of this trade-off? We make a plot of the variance and bias based on predicted output data z. This plot is only computed in the bootstrap method. The variance of one point is the total variance of predicted output from every bootstrap, for the same data point. Since we need to evaluate the variance on a fixed data for every bootstrap, this can't be done on the validation output because we need the same unique data point. The test output which is put away in the beginning of the bootstrap loop. The final array of variances is the mean of every variance that we computed for every bootstrap for a specific output data point. We now have a variance corresponding to every model complexity.

Something similar is done when computing the bias.

We need to compute based on the test data, which need to be fixed as the bias is evaluated with respect to a specific data point. We can't compute the bias of OOB samples since this is different for every bootstrap iteration. When computing bias, we need to remove the noise from the real output data as referred to in equation 21, so that we only take the Franke function data into account.

We compute the MSE and R2 of the test- and training set by implementing the algorithm eq. 7 and eq. 10 as referred to in the theory. When computing the MSE for the training data, we need to check if this decreases for increasing model complexity before we proceed. The test MSE must increase for the increasing complexity as well, in order for our results to be consistent with the bias variance trade-off as described in the theory.

In order to make an analysis of Ridge and Lasso, we created a heatmap. This is one of many ways to preset a target dependent on two parameters. The heatmap consistst of the test MSE visualized as a color gradient with respect to the different values of penalization λ and degree p. The color scale will go from red to dark green. If the MSE is large, the respective section will have a red color, if the MSE is medium, the color will be light yellow, and if the MSE is very small, the color will be dark green.

Including actual terrain data

Lastly, instead of studying the Franke function with a known underlying function, we chose to study actual terrain data (height above sea lever) for a piece of area (in square meters) in Lyndal, south of Norway.W e download the terrain from **this** open-source database. We now have data of the height, denoted z. Data points in other dimensions x and y are generated using the **numpy.linspace** function from numpy, these are unit spaced points along the respective axes. The original dataset consists with over 6 million points, which will take a lot of unavailable CPU power. Corresponding x, y and z data points were reduced down into three one-dimensional arrays, which consists of a random sample of the entire terrain landscape. The data is shortened using the **numpy.random.permutation()** module, and we only use 10000 points for x, y and z. Since the loaded z-data is not square, the x-data needs to be as long as the z-data along axis=0, and the same for y-data along axis=1.

We have now preprocessed the terrain data. The regression analysis is done by preparing 5-fold CV and tuning over polynomial degrees for OLS, and tuning over both degree and penalization λ for Ridge and Lasso. The terrain data is split into training and test set. The training data will be shuffled in each fold. The resulting parameters which corresponds to the minimum MSE will be printed for each model. We will also analyse the test MSE as a function for every hyperparameter for each of the models.
The model which is determined as the best for this specific data in the analysis, with its tuned parameter will be used to predict the terrain surface. The predicted surface and actual terrain is computed in 3D. In reality, these surfaces will go through one anther at certain points, but will not happen because the plotting package that we use (pyplot from matplotlib) does not allow it.

IV. RESULTS

These results shows the performance of the linear regression models tuned with CV and bootstrap. We will first look at the performance of OLS without tuning. Afterwards we will analyse OLS tuned with bootstrap and CV, and observing the test and training MSE. We will do the similar for Ridge and Lasso with including penalization as an additional tuning parameter. The MSE's is represented through heatmaps.
Afterwards, we will observe all three methods tuned with CV on the terrain data. We use OLS with tuned degree to predict surface of terrain data, the predicted surface is plotted with the terrain.

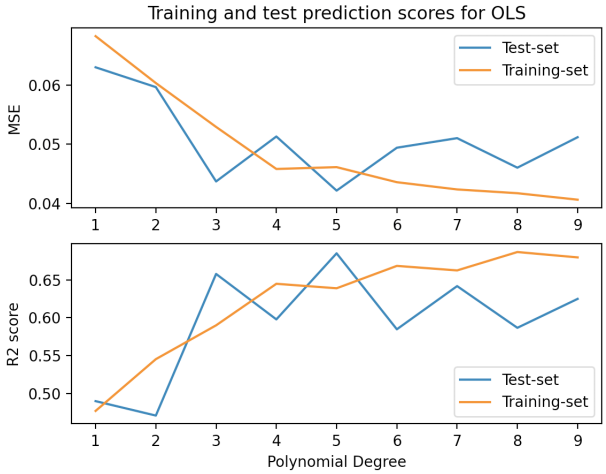

FIG. 3. This is the MSE's and R2 score for OLS model on Franke function with added noise following a normal distribution with mean=0 and standard deviation=0.2.

From figure 3, best complexity for prediction model for test data is degree 5, with a corresponding MSE of 0.04 and r2 score of 0.69.

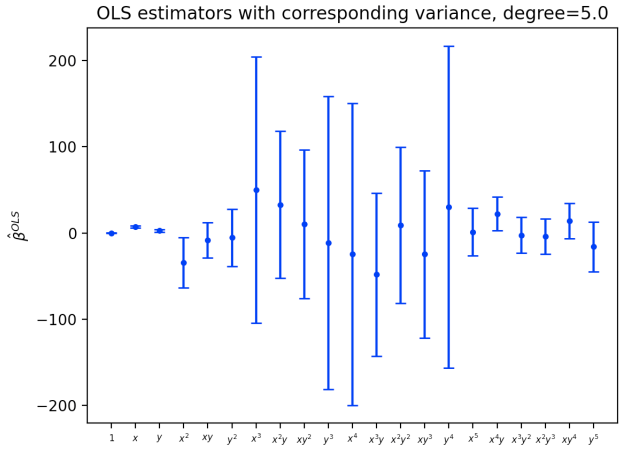

FIG. 4. Variance of each coefficient computed by OLS model with polynomial feature matrix of degree 5. The x-axis represents each of the features, and the y-axis represents the value/weight of each coefficient in the model. The errorbar denotes the variance corresponding to the coefficient.

In figure 4 we're observing estimated coefficients and their corresponding variance for OLS regression model of 5th degree.

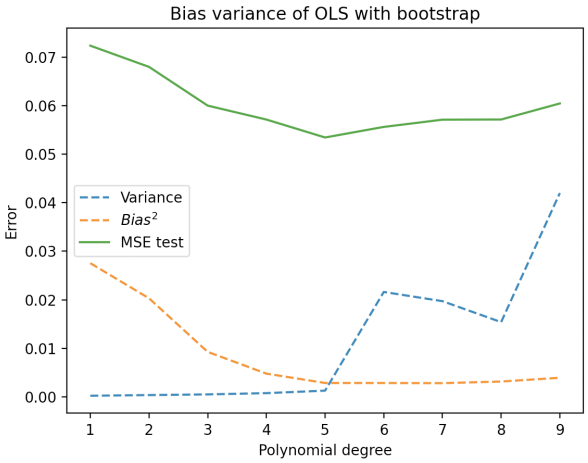

FIG. 5. Blue dotted line represents computed variance for every point for each bootstrap, these variances are calculated as the average that correspond to one polynomial degree. The squared bias is computed similarly and is illustrated by the orange dotted line. The total error is the green line.

Figure 5 illustrates bias variance trade-off with additional total error (MSE) for OLS model predicted on test set.

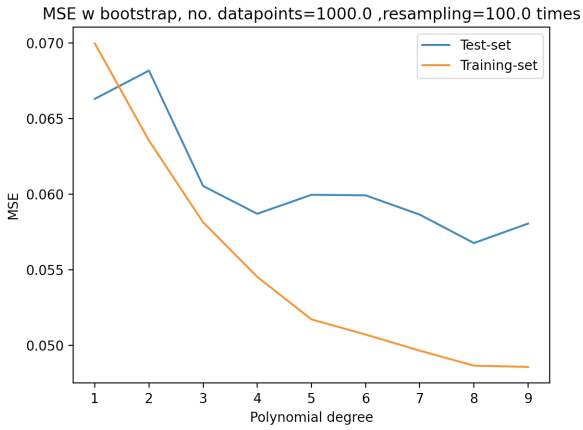

FIG. 6. Orange line is the mean MSE of trained OLS model predicted on temporary training set (bootstrap sample) for 100 bootstraps for every polynomial degree. The blue line denotes the mean MSE of trained OLS model's predicted validation (OOB) output of 100 bootstrap resamples.

Figure 6 shows OLS trained on bootstrap samples and validated on OOB samples 100 times. Resulting best polynomial degree is 8 with corresponding MSE of 0.0568 based on prediction of test set.

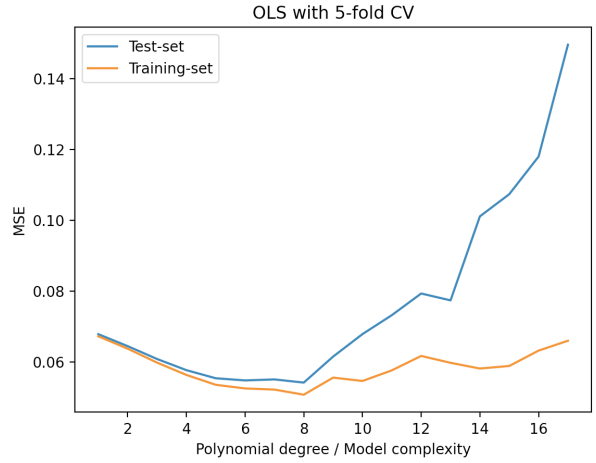

FIG. 7. MSEs for OLS model on Franke function predicted on test- and training set using 5-fold CV. Orange line denotes mean MSE of predicted temporary training set of Franke function for every fold corresponding to a specific polynomial degree. The blue line denotes OLS model predicting validation set of Franke function.

Figure 7 shows resulting MSEs with OLS predicted on temporary training set and validation set of training. Resulting best degree is 8, with MSE 0.0542 from validation.

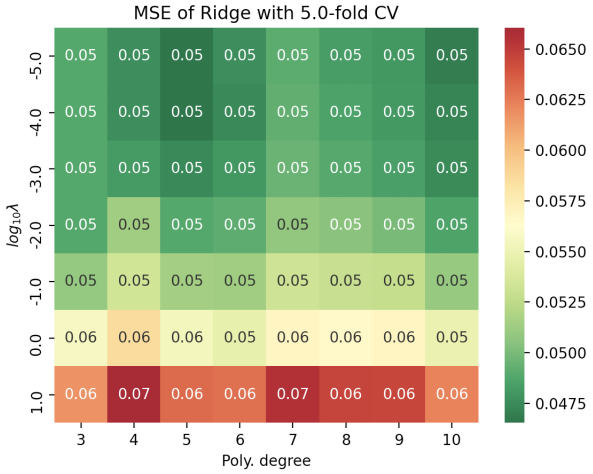

FIG. 8. Heatmap of MSE of Ridge predicted on validation set of Franke function output for the corresponding hyperparameters λ and polynomial degree using 5-fold CV. Each MSE is the mean validation MSE for every fold. The x-axis denotes polynomial degree ranging from 3 to 10. The y-axis denotes log10 of λ, ranging from 10^1 to 10^{-5}.

Figure 8 shows Ridge regression applied for different penalties and polynomial degrees. This is a heatmap of MSE of Ridge predicted on test set for the corresponding hyperparameters. Polynomial degree 6 and $\lambda = 10^{-5}$ is the tuned parameters corresponding to the minimum MSE.

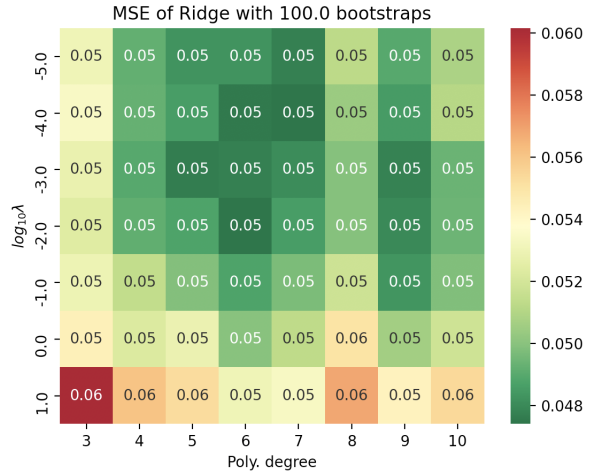

FIG. 9. Heatmap of MSE of Ridge predicted on OOB sample from Franke function using 100 bootstrap resamples. Each MSE is the mean MSE of OOB sample for 100 bootstraps. The x-axis denotes polynomial degree ranging from 3 to 10. The y-axis denotes log10 of λ, ranging from 10^1 to 10^{-5}.

Figure 9 is a heatmap of test MSE with Ridge applying with 100 bootstrap resamples. Best degree=7, with best lambda $\lambda = 10^{-4}$.

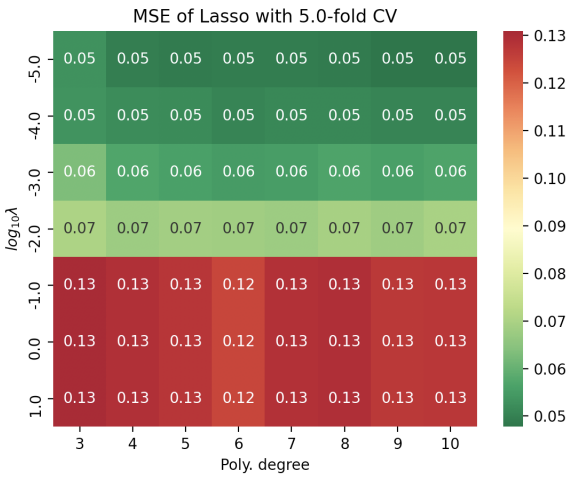

FIG. 10. Heatmap of MSE of Lasso regression predicted on validation set of Franke function for the hyperparameters degree and penalization using 5-fold CV. The hyperparameters are shown along the axes. The x-axis denotes polynomial degree ranging from 3 to 10. The y-axis denotes log10 of penalization parameter λ, ranging from 10^1 to 10^{-5}.

With Lasso regression on test set with 5-fold CV gives a tuned degree=9, with $\lambda = 10^{-5}$ in terms of smallest MSE.

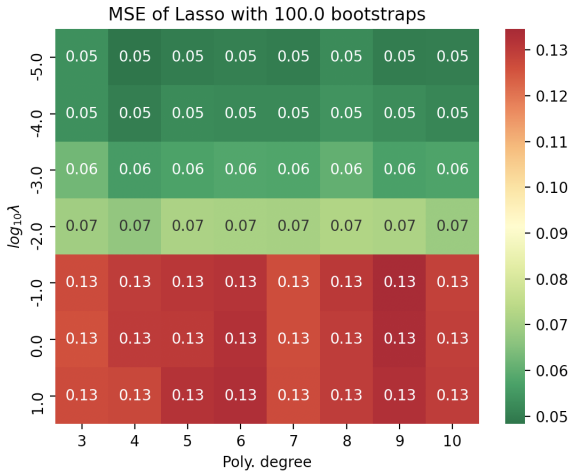

FIG. 11. Heatmap of MSE of Lasso regression predicted on validation set (OOB samples) for the corresponding hyperparameters, polynomial degree in feature matrix and penalization λ, using 100 bootstrap resamples.

With Lasso regression on test set with 5-fold CV gives best degree=4, with best $\lambda = 10^{-5}$.

TABLE I. Full overview of minimum MSE of test set given for all 3 regression methods, with tuned hyperparameters by CV and bootstrap

Resampling	OLS	Ridge	Lasso
CV	0.0542	0.0462	0.0478
Bootstrap	0.0568	0.0474	0.0483

TABLE II. Tuned polynomial degree of OLS on Franke function by CV and boostrap, as well as tuned degree and penalization of Lasso and Ridge on Franke function data by bootstrap and CV.

Resampling	OLS	Ridge	Lasso
CV, best degree	8	6	9
CV, best λ	-	10^{-5}	10^{-5}
Bootstrap, best degree	8	7	4
Bootstrap, best λ	-	10^{-4}	10^{-5}

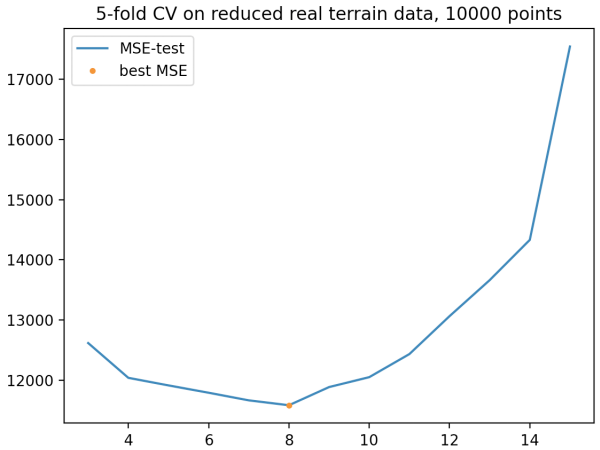

FIG. 12. OLS with 5-fold CV for variuos polynomial degree in design matrices applied on reduced real terrain data of Lyngdal, Norway.

Best degree with OLS on reduced terrain data is degree=8, with MSE=11583.918 m^2.

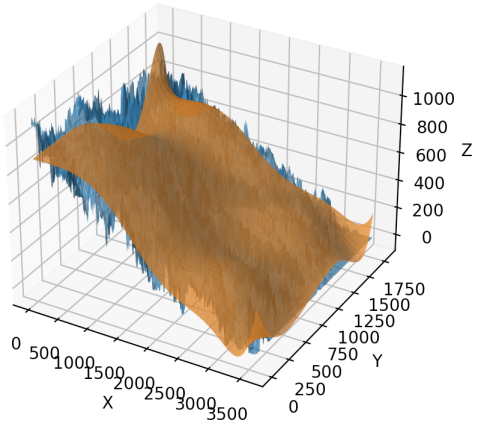

FIG. 13. Two surfaces where blue surface is real raw terrain data of Lyngdal, Norway. Orange surface is predicted height over seal level (meters above seal level) for the entire data set grid, but with using the model that was trained on the training set.

Figure 13 shows a blue surface which is actual terrain data over Lyngdal. The orange is the predicted surface with tuned polynomial degree (degree 8).

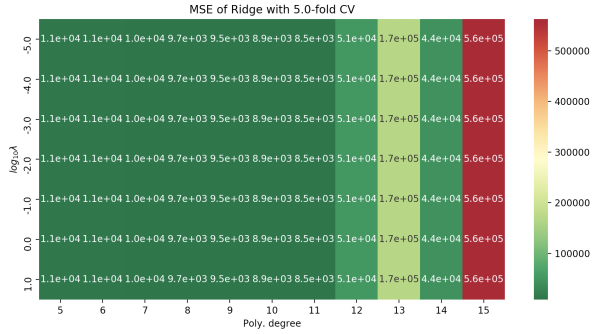

FIG. 14. Ridge with 5-fold CV on reduced terrain data of Lyngdal landscape. We have two tuning parameters: polynomial degree included in feature matrix and penalization λ.

Ridge CV : Best degree=11, with best $\lambda = 10^1$.

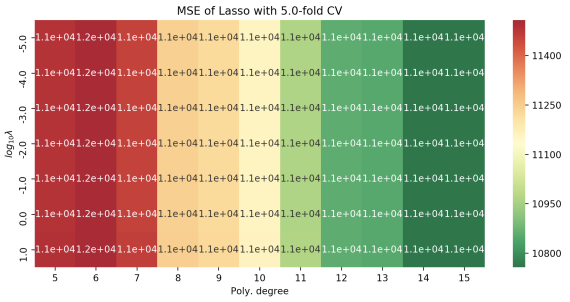

FIG. 15. Lasso with 5-fold CV on reduced terrain data of Lyngdal landscape. The axes represent penalization parameter and polynomial degree included in the feature matrix.

Lasso CV : Best degree=14, with best $\lambda = 10^{-5}$.

TABLE III. Table contains tuned parameters: degree and penalization tuned by 5-fold CV of OLS, Ridge and Lasso regression models.

Tuned Parameter	**OLS**	**Ridge**	**Lasso**
Polynomial degree	8	11	14
Penalization, λ	-	10^1	10^{-5}

V. DISCUSSION

From figure 3, we're observing the general trend of bias variance trade-off. We observe from the figure that the training set MSE and R2 improves for increasing degree, whereas the test MSE and R2 worsens. When increasing the polynomial degree for training data, the model fits more and more onto the noise of the data. The model is too tailored onto the training data for high complexities, and will capture the information given in the training set as expected from what we reviewed in the theory. A consequence of this is that while the model fits the training data very well, this generalizes poorly onto new data, we recognize the model as overfitted. While for test data, the trained model of high complexities, degrees>5, generalize poorly onto this unseen (test) data and the MSE increases. We observe that the MSE of the training data will continue to decrease for higher degrees, since the fit gets better and better. However, we expect the MSE for the model predicted on new data to decrease as the model is too tailored to the training data which consistent with the theory of overfitting a model. For models with low complexity (p<5), the high MSE is due to that the model captures too little information about the data, and this error is due to high bias which is consistent with the theory of a underfitted model.
The R2 score also reflects this trade-off. When increasing the polynomial degree, the model predicted on the training data explains more and more variability. For the model predicted on test data, the R2 score will only decrease after a polynomial degree of 5. A model with degree 5 achieves lowest MSE and highest R2.

When observing the variance of each corresponding β-coefficient in figure 4, we're observing the increasing variance for terms with high polynomial degree. The figure illustrates the variance corresponding to each coefficient for a model with feature matrix with polynomial degree 5. Observing the figure tells us that standalone x and y parameters have higher variance, since coefficients belonging to terms like x^3, y^3, y^4, and x^4. This might be due to the fact that the underlying data generating mechanism, the Franke function is not constant along x and y, and thus the mixed terms should in almost all cases work better due to this. This is exactly what we observe for models with high complexity - in fact the contributions from these terms is what essentially result in the high total variance of the model. When including higher polynomial degrees, we involve increasingly more variance in total. The overall worst terms (in case of high variance) are those of 3rd and 4th degree, whereas 5th degree has smaller variance compared to the 2nd degree. This is perhaps due to the fact that the Franke function have "bumps" in the surface, and the 5th degree produce approximately 2 bumps. Observing the variance in each of the weights, this will be a reason to include regularization to the model in order to reduce the variance in each of the individual weights. The bias terms (1, x and y) is the terms with significalty lower variance.

In figure 1 we observe the characteristics of bias-variance tradeoff. For models of low complexity, the bias is high due to the lack of information about the overall data, and as a result we observe underfitting of the data. For higher polynomial degrees, the variance will increase since the model is more sensitivity to small variations in the data. The resulting high variance is due to overfitting of the data, and the model is too well-tailored on the training data such that the model will not capture the characteristics of the test data. While for lower polynomial degrees, there is a lack of information included in the model, and the bias is increasing for decreasing complexity which shows characteristics of a underfitted model. We observe the balance point to be at 5th degree. This is because the test MSE curve which is the sum of the squared bias, variance and the stochastic noise added to the Franke function. (green curve) has a minimum at this degree.

In figure 6 we use 100 bootstrap resamples to estimate MSE's for every polynomial degree. The MSE curve of test is not smooth and seems to bounce from one degree to the next. The fact that this curve is not smooth will influence the credibility of the results, and it makes it less certain since we're dealing with slight instabilities. We have a common minimum at degree 8, with a MSE of 0.06. After degree 8 we expect the test MSE to only increase, since we expect overfitting for models with high complexity. Because the curve is not smooth, it

does not guarantee that the MSE will increase for higher polynomials. The training MSE expect to only decrease for increasing polynomial degrees since the model will capture more of the characteristics of the training data, but is not guaranteed due to instabilities.
In figure 7 we move away from the bootstrap and apply the 5-fold CV method. The degree which corresponds to the lowest MSE is degree 8, with a MSE of 0.05. The test MSE has decreased with 17% compared to the MSE with the bootstrap method as seen in table I. In this figure, we clearly observe the overfitting effect, since the test MSE increases after 8th degree. Both resampling methods give states that the same hyperparameter for the model has the highest predictive performance, but the CV gives a slightly smaller MSE error for the given model. Bootstrap achieves a more precise estimate of the true MSE, but I don't know if it's lower or higher, only that it should be closer to the truth.

We now observe the regularized linear regression method Ridge in figure 8. The heatmap illustrates the resulting MSE of test set for 7 different values of λ, and polynomial degrees from 3 to 10. The resulting MSE is estimated by CV. For higher penalties, $\lambda = 10, 1, 10^{-1}$, the resulting MSE is large compared to lower penalties. This is a strong indication that high shrinkage of the individual weights in the model is not preferable to increase its predictive power.

The minimum MSE of test set obtained by Ridge with CV tuned parameters is low compared to the other MSE's as observed in table I, this indicated that Ridge tuned by 5-fold CV gives predictions which are closest to the target of interest, even when the penalization is small. Using this method tuned by Bootstrap also gives smaller MSE compared to Lasso tuned with Bootstrap and Lasso with CV.

Applying bootstrap on Ridge will give high errors for models of low complexity and seemingly regardless of penalty λ. Still, the bootstrap method chooses a higher polynomial degree and a higher penalty compared to the CV method as the parameters which give the lowest MSE. High penalties such as, 10, 1 and 0.1 gives givers higher errors. This behaviour was also observed for the bootstrap method as observed is figure 9. It is unknown why bootstrap and CV result in different tuning parameters as observed in table II, but the model has a characteristic behavior regardless of resampling method indicated in the figure 8 and 9. The resulting MSE's as seen in table I is not very different regardless of the different tuned parameters, but it is clear that CV obtains the parameters which give the minimum MSE.

For the Lasso method, large penalization parameters (10, 1 and 0.1) gives high errors as seen in figure 10 and 11. This method chooses a small λ and an even higher polynomial degree as the best parameters based on the minimum MSE compared to the Ridge method. Large values of λ will most probably result in feature reduction, and the large errors are a strong indication that the model prefer to keep all of its features for all model complexities for this dataset. High error for high values of penalization is due to underfitting, because the bias increases for decreasing complexity. For instance when two features are highly correlated, the Lasso method tends to shrink one of these features. These results indicate that each individual feature is not correlated for every complexity, since including them in the model improves the model's prediction ability. Even though the bootstrap method chooses a much smaller complexity than the CV method, their MSE's do not differ largely. The model shows the same behaviour for both CV and bootstrap as we expect, and this is also reflected on the resulting heatmaps.
Since large penalty parameters give large MSE, both Lasso and Ridge choose small penalties in order to minimize this error. When the penalty is very small, then the Lasso and Ridge method should be somewhat similar to OLS. For $\lambda = 0$ equation 25 and 22 will exactly the same as 5. Because both regularization methods give less test MSE compared to OLS tuned with CV and Bootstrap, this indicates that the small regularization gives a positive effect. Regularization could be an improvement of the model and thus will increase it's prediction ability.

Based on the the results from Ridge and Lasso regression, they both choose a minimal penalty parameter 10^{-5} in order to minimize the MSE. Reducing many features in Lasso lead to high MSE's which indicates that each feature included in the model, even for high polynomials, are uncorrelated in the sense that they linearly independent. But since the two MSE's using Lasso is smaller compared to OLS, as observed in table I, this therefore indicates that the small penalty gives an positive effect. The model suffers from too high penalty, and could be due to too strong suppression of the beta-terms. The complete reduction may be a better model when features has perfect correlation.

When comparing resampling methods based on table I, we observe that 5-fold CV results in a smaller MSE's of unseen data when compared to models tuned with Bootstrap. This indicates that 5-fold CV is a better method for estimating hyperparameters for this specific dataset, when compared to Bootstrap.

From figures 14 and 15 there is no variation of MSE in the different penalty values. This indicates that the regularization has no effect for either Ridge and Lasso, and therefore the OLS is observed to obtain the highest performance compared to the regularization methods. It is however observed that Ridge and Lasso favours different polynomial degrees. Ridge regression has smallest MSE for degree 5 up to degree 11, but

Lasso have small MSE for degree 14 and 15.

When performing OLS on the terrain data, we have a minimum MSE of 1583.918 m^2 and degree 8 as seen in 12. The OLS model of degree 8 is thus used to predict the surface of the terrain as seen in figure 13 which is the orange surface.

Regularization methods has no effect for the terrain data. The penalty gives no effect to the MSE and thus the prediction. The penalty term is to reduce the weights of the individual features, these results indicate that the regularization does not improve the prediction performance of the model. For L1 penalty which in fact shrinks some features to zero for high values and will thus largely decrease the model complexity, but this does not worsen or improve the MSE. Regularization is one of the most important prerequisites for improving the model with predictors of high correlation, but it is not a solution to every problem. Data sets, such as terrain data, which contain high irregularity seems to be one of the reasons why these regularization methods does not work. By observing the results, L1 and L2 does not seem to be beneficial for this type of data set.

VI. CONCLUSION

A data set computed by the Franke function was modeled by applying three linear regression methods and with polynomial design matrices, OLS, and two regularized methods, namely Ridge and Lasso regression. The Ridge method was observed to give the highest performance in terms of the smallest MSE of 0.0462 with a polynomial degree of 6 with $\lambda = 10^{-5}$ tuned with 5-fold CV.

When introducing real terrain data, the regularization methods, Ridge and Lasso regression, had no effect on the model in terms of penalization. It therefore seems that regularization fails on this type of data which contain high irregularities, but the methods in itself does not fail. However, it is observed that Ridge and Lasso favour different polynomial degrees in terms of the MSE. Since regularization gives no improvement in terms of MSE, OLS seems to have the highest performance out of all three regression method, because it gives the smallest MSE of 1583.918 m^2.

REFERENCES

[1] Singh, S. (2018). Understanding the bias-variance tradeoff.

VII. APPENDIX A: GLOSSARY

A comprehensive glossary list is presented. The intention of table IV is to serve as a lookup table for whenever needed. It is not necessary to know all of these abbreviations and technical words by heart.

Abbreviation	**Full word**
CV	Cross validation
ML	Machine learning
OLS	Ordinary least squares
MSE	Mean squared error
SD	Standard deviation

TABLE IV. A glossary list containing often used abbreviations.