project 1, FYS-STK4155
Regression and Resampling of Terrain data and the Franke function

A. G. Eriksen
(Dated: October 13, 2020)

Exploration of machine learning methods by the Regression subfamily. The methods were tested on a simulated surface generated by the Franke-function. Initially, regressed with Ordinary least squares regression before expanding to Ridge. Exploration of resampling methods also performed, with bootstrap and k-fold CV. Intent was to also implement the LASSO regression method and introduce real data to the models. This fell through.

CONTENTS

I. introduction 1

II. method 1

III. results 5

IV. conclusion 5

V. appendix 7

References 8

I. INTRODUCTION

The motivation for this project was in large part to explore regression methods as a subset of machine learning. Machine learning bases itself mainly on a frequentist approach, or frequency or proportion of data as a metric for prediction. The regression methods have a relatively intuitive setup and serves as a good entry point for exploring machine learning as well as being usefull for predicting numerical problems suspected of following an unknown function.
Three methods were decided on to explore, ordinary least squares(OLS), Ridge regression and Lasso regression. OLS was performed with results approximating the expectations for regressions of our dataset. Ridge was barely runnable, but lacked any finish w.r.t. storing and displaying the data from it. Lasso regression fell off completely.
Beyond the regressor models, we also utilized resampling methods, Bootstrap and k-fold Cross Validation. These methods allow us to make the best use out of the data already in our possession, as the resampling helps average over several smaller "data sets" sampled from a majority of the initial data. The results from these smaller sets are compared to the minority of the original which has been left untouched. The amount of tests allows us to control for e.g. variance and bias.
Once these methods were laid out and tested, the idea was to move on to real data, rather than the simulation and make use of terrain data from Norway. Due to time constraints, this did not happen.
The report here is structured into a method section where I generally go through the assumptions, models and algorithms for the methods mentioned. Following this, is a section of the few results I managed to collect and an attempt at discussing them as they relate. Finally, the conclusion attempts to conclude what can be from concluded after the run. Also here, I attempt to describe the issues with the implementation as well as the state of the project and how they might be improved.

II. METHOD

The initial assumption we make in regression is that our data set and targets are connected with some function. Thus, we can write

$$\boldsymbol{y} = f(\boldsymbol{x}) + \varepsilon. \tag{1}$$

Here, y is the targets we wish to predict and the x are our input variables. The ε represents an unknown noise in the function. Given this function, we want to approximate the targets using some model. Therefore, we set

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}, \tag{2}$$

which approximates $f(\boldsymbol{x})$. Since we want to predict the targets, we need some measurement of how well we fit the model. This comes in the form of a cost function which we can minimize to find the optimal fit.

$$C(\boldsymbol{\beta}) = \frac{1}{n}\{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^2\}, \tag{3}$$

$$\frac{\partial C(\boldsymbol{\beta}}{\partial \boldsymbol{\beta}} = 0 = \boldsymbol{X}^{\boldsymbol{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}), \tag{4}$$

$$\hat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y}. \tag{5}$$

$\hat{\beta}$ is the optimal coefficients for the model.[1] [2]

With an expression for β ready, we need a system to fit to. This comes in the way of Franke's function[3]

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left\{\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right)\right\} \\ &+ \frac{3}{4}\exp\left\{\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right)\right\} \\ &+ \frac{1}{2}\exp\left\{\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right)\right\} \\ &- \frac{1}{5}\exp\left\{\left(-(9x-4)^2 - (9y-7)^2\right)\right\}. \end{aligned} \tag{6}$$

This function, we define over an area $x, y \in [0,1]$. In addition, noise can be added to the function to obscure the functional relationship between the points in the function.

This system then consists of the input data, x, y and the output data z. Having generated the inputs and found the target, the next step is to split the data into training and test sets and define a feature matrix as a model for the system.

As we have settled on regression methods, we have chosen to fit z with a polynomial model, where the model complexity would represent the degree of polynomial used to set up the model. In our case the inputs amount to 2 vectors, so the algorithm for making the feature matrix becomes 1

Input: $\boldsymbol{x}$ and $\boldsymbol{y}$ and polynomial degree n
Output: Feature matrix X with dimension mxn
$n \leftarrow length(\boldsymbol{x})$
$m \leftarrow int((n+1)\cdot(n+2)/2)$
$X \leftarrow matrix.ones(nxm)$
for $i \leftarrow 1$ ***to*** $n+1$ **do**
  $q \leftarrow int((i)*(i+1)/2)$
  for $k \leftarrow 0$ ***to*** $i+1$ **do**
    $X_{(:,q+k)} \leftarrow x^{i-k}\cdot y^k$
  end
end
return X

Algorithm 1: make feature matrix X given input $\vec{x}, \vec{y}$ and dimension n

with an algorithm for the feature matrix and a set of targets generated by the Franke function we just need to fit the features and predict an output. Following this, we can apply various statistics to compare the model to the targets.

These would include

$$MSE(y,\tilde{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \mathbb{E}[(y-\tilde{y})^2], \tag{7}$$

$$R^2(y,\tilde{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i-\mathbb{E}[y])^2}, \tag{8}$$

$$\mathbb{E}[y] = \frac{1}{n}\sum_{i=0}^{n-1} y_i, \tag{9}$$

$$\mathbb{E}[(y-\tilde{y})^2] = ... = \mathbb{E}[(y-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[(\tilde{y}-\mathbb{E}[\tilde{y}])^2] + \sigma^2, \tag{10}$$

$$\textbf{Bias}(y,\tilde{y} = \mathbb{E}[(y-\mathbb{E}[\tilde{y}])^2], \tag{11}$$

$$\textbf{Var}(\tilde{y}) = \mathbb{E}[(\tilde{y}-\mathbb{E}[\tilde{y}])^2], \tag{12}$$

$$MSE(y,\tilde{y}) = \textbf{Bias}(y,\tilde{y}) + \textbf{Var}(\tilde{y}) + \sigma^2. \tag{13}$$

Moving on from this, we can begin to refine the data we have somewhat, using resampling methods. The ones we will make use of, are Bootstrap and k-fold Cross Validation. The essence here, is that we have a limited data set. To compensate for this, methods of selecting which data to run allows us to make the best use of what data we do have. The main methods discussed here are Bootstrap[2] and k-fold Cross Validation[3]

Input: feature matrix, X, targets, y, and number of bootstraps, N
Output: model fits and predictions
$\tilde{\boldsymbol{y}}^{fit} \leftarrow$ **array**(N)
$\tilde{\boldsymbol{y}}^{predict} \leftarrow$ **array**(N)
$\boldsymbol{\beta} \leftarrow$ **array**(N)
for $i \leftarrow 0$ ***to*** N **do**
  Shuffle X and y
  Split data into training and test sets
  $\beta \leftarrow OLS(X_{train}, y_{train})$
  $\tilde{y}_i^{fit} \leftarrow \boldsymbol{X_{train}\beta}$
  $\tilde{y}_i^{predict} \leftarrow \boldsymbol{X_{test}\beta}$
end
return $\boldsymbol{\beta}$, $\tilde{\boldsymbol{y}}_{fit}$, $\tilde{\boldsymbol{y}}_{predict}$

Algorithm 2: The Bootstrap method of resampling. This method makes no distinction between resamplings, using the assumption that the initial set follows a distribution the resampling is mimicing. Thus, the values approaches a better interpretation

Finally, we can also look more closely at the regression method we use. The Ordinary Least Squares solution works OK, but there are issues, for instance if $(X^TX)^{-1}$ were to be singular, then the method would fail. A solution to this lies in methods such as Ridge regression and Lasso regression. The solution to the possibility of the singular values lies in adding something along the diagonal, so $(X^TX + \lambda\mathbb{I})^{-1}$

This complicates the expression for MSE somewhat, due to this addition. A rewrite of the MSE, could be made using the ”norm-2” description to

$$\min_{\beta\in\mathbb{R}^p} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2\frac{1}{n}\|\boldsymbol{y}-\boldsymbol{X\beta}\|_2^2. \tag{14}$$

If we were to add our extra parameter here, we could

$$\min_{\beta\in\mathbb{R}^p} \quad \frac{1}{n}\|\boldsymbol{y}-\boldsymbol{X\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2 \tag{15}$$

Which gives us what we call Ridge regression, or we could use ”norm-1” to set

$$\min_{\beta\in\mathbb{R}^p} \quad \frac{1}{n}\|\boldsymbol{y}-\boldsymbol{X\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1. \tag{16}$$

Input: feature matrix, $\boldsymbol{X}$, targets, $\boldsymbol{y}$, and number of folds, k
Output: model fits and predictions
$\tilde{y}^{fit} \leftarrow \textbf{array}(k)$
$\tilde{y}^{predict} \leftarrow \textbf{array}(k)$
$\boldsymbol{\beta} \leftarrow \textbf{array}(k)$
Split k folds $\boldsymbol{\mu}, \boldsymbol{\nu} \subset \boldsymbol{X}, \boldsymbol{y}$
for $i \leftarrow 0$ ***to*** k **do**
  $\boldsymbol{X}^{\boldsymbol{test}} \leftarrow \boldsymbol{X}\{\mu_i\}$
  $\boldsymbol{X}^{\boldsymbol{train}} \leftarrow \boldsymbol{X}\{\boldsymbol{\mu} - \mu_i\}$
  $\boldsymbol{y}^{\boldsymbol{test}} \leftarrow \boldsymbol{y}\{\nu_i\}$
  $\boldsymbol{y}^{\boldsymbol{train}} \leftarrow \boldsymbol{y}\{\boldsymbol{\nu} - \nu_i\}$
  $\beta \leftarrow OLS(X_{train}, y_{train})$
  $\tilde{y}_i^{fit} \leftarrow \boldsymbol{X}_{\boldsymbol{train}}\boldsymbol{\beta}$
  $\tilde{y}_i^{predict} \leftarrow \boldsymbol{X}_{\boldsymbol{test}}\boldsymbol{\beta}$
end
return $\boldsymbol{beta}$, $\tilde{\boldsymbol{y}}^{\boldsymbol{fit}}$, $\tilde{\boldsymbol{y}}^{\boldsymbol{predict}}$

Algorithm 3: k-fold Cross Validation method of resampling. This allows us to vary the combination of the shuffled indices without reusing datapoints.

Each of these represent various ways to ensure we get a minimum. The effects differ somewhat between them, where Ridge "penalizes" some betas according to the amount of variance in the system, Lasso strikes these out, setting them to 0, while keeping the unaffected β's essentially the same.

The code is set up so that a mesh of 2 matrices are fed into the franke function and then all thre are flattened into 1 dimension. Then they are fed into a resampling and regression function where the above mentioned mathematics takes place and after this, the resulting fit and predictions for the target values are subjected to various error tests, such as MSE, R2 score and bias-variance estimations

With regards to the bias-variance estimations and their relationship to the error estimate can be expressed

$$C(\hat{\mathbf{X}}, \vec{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \; = \; \mathbb{E}[(y - \tilde{y})^2] \tag{17}$$

using the values for: $y = f(x) + \varepsilon$, $f(x) \equiv f$, $\langle\varepsilon\rangle \equiv 0$ and $var(\varepsilon) \equiv \sigma^2$, as well as adding and subtracting $\mathbb{E}[\tilde{\boldsymbol{y}}]$

$$\begin{aligned}\mathbb{E}[(y - \tilde{y})^2] &= \mathbb{E}[(y - \tilde{y} + \mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] \\ &= \mathbb{E}[(f - \langle\tilde{y}\rangle)^2] + \mathbb{E}[(\tilde{y} - \langle\tilde{y}\rangle)^2] + \mathbb{E}[\varepsilon^2] \\ &+ 2\mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])] \\ &+ 2\mathbb{E}[\varepsilon(f - \langle\tilde{y}\rangle)] + 2\mathbb{E}[\varepsilon(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{y}])]\end{aligned} \tag{18}$$

Here we have 2 parts with the expectation value of our noise variable, which can be separated out ab both $\boldsymbol{f}$, $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{y}}$ are assumed deterministic. Since the noise has a mean value of 0, these parts dissapear. Additionally, $\mathbb{E}[\varepsilon] = \sigma^2$.

$$= \mathbb{E}[(f - \langle\tilde{y}\rangle)^2] + \mathbb{E}[(\tilde{y} - \langle\tilde{y}\rangle)^2] + \mathbb{E}[\varepsilon^2] + 2\mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}]) \tag{19}$$

Next bit requires a little bit of work, but Wolframalpha[4] lets us set $\mathbb{E}[f \cdot \tilde{\boldsymbol{y}}]$, which we can separate as they're both deterministic so, $\mathbb{E}[f]\mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}]^2$, which cancels out, seeing as both the function and the approximation share an expected value given the initial assumption of the model. This leaves us with

$$\mathbb{E}[(y - \tilde{y})^2] = \mathbb{E}[(f - \langle\tilde{y}\rangle)^2] + \mathbb{E}[(\tilde{y} - \langle\tilde{y}\rangle)^2] + \mathbb{E}[\varepsilon^2] = Bias[\tilde{y}] + Var[\tilde{y}] - \sigma^2. \tag{20}$$

The bias, variance and noise variance is in order here. The bias represents an offset between the model mean and the target values, the variance is a measure of the spread of the models with regards to the mean of all the models, while the noise variance is generally unknown and harder to pin down.

III. RESULTS

In essence, these regressions are performed to make a model of and predict the shape of (simulated) terrain. While most of the simulations here have been performed with a much more significant level of noise added on to the base function, the surface shown can be seen in figure 1

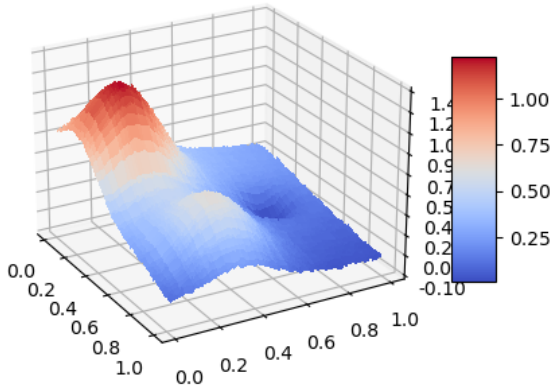

FIG. 1. Surface plot of the simulated surface generated through the Franke function The inputs and outputs are both unitless, so the axes are merely ilustrative. This figure was generated using a slight amount of noise added to the function, which is the source of the slight irregularities observed. The variance of the noise was set to 0.01 for the sake of illustrating the surface, while most of the results taken from the function uses a variance of 1, which makes the structures seen here obscured.

For the most basic implementation of regression, with no regard for the possibility of a singular matrix and with the assumption of a large enough data set that the prediction is accurate. Here, we set up the data and prepare several different models, based on the degree of the polynomial we want to use to make our fit. To visualise the accuracy of the fit, a plot of the MSE for the fit to the training data and the test data are set up.

Following after these, there was the issue of implementing resampling methods. Since these also give us several fits of the model, we can also evaluate the variance of the models across tries to produce a bias-variance plot,

Figure 3 shows a high and increasing bias and a slowly growing variance over the model complexity kept the same as in figure 2.

This is largely the extent of gathered data. Further examples of the bias variance results should be in the appendix.

IV. CONCLUSION

The setup of the Franke function as well as the surface plot of the generated data was largely taken from the project description and looks about as you'd expect. The example with no resampling and Ordinary least squares also behaves as we would expect, where the initial bias of the linear fit drops off as we introduce complexity until the point where the variance's exponential growth overtakes the dropping bias and we see the test error increase.

For some reason, this seems to not be the case when the Bootstrap method is introduced, as we see in figure 3. Here, the bias doesn't fall off for some reason and the variance hardly increases at all compared to the increase in error we saw when looking at the simplest case. This could be an error in the calculation of the new measurements, with the number of fits per model degre of complexity increases. Perhaps it's an error regarding the dimensionality along which we took the means for the difference in the bias and variance. Mixing up the rows and collumns as it were.

That said, there were several things wrong in the code, and while I eventually managed to finish a calculation using the k-fold cross validation, the amount or repeated code showed it's toll in strange errors. Though the code

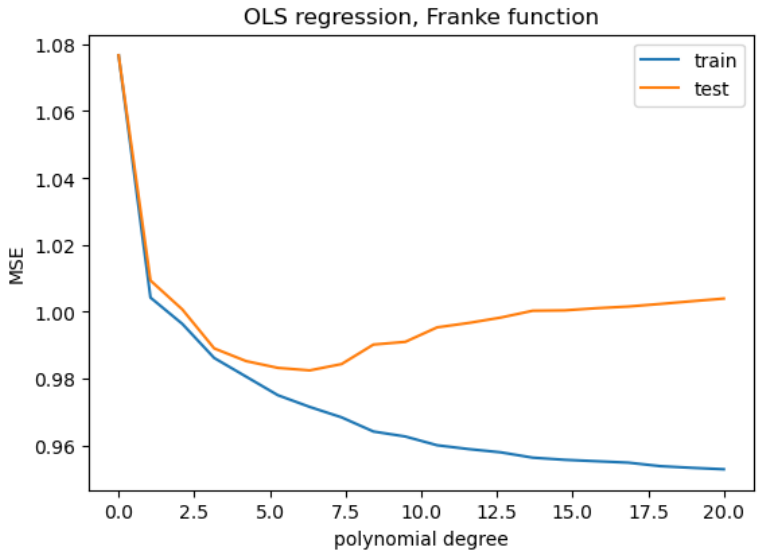

FIG. 2. Plot of the MSE for OLS regression of the Franke function with no resampling techniques active. The worst fit is for the straight line case, where the MSE lies at about a 1, with a rapid reduction which smooths out around 5th degree polynomial for the test set, while the training fit continues to improve. Couple of notices here, is the floating point numbers that represent the model complexity or polynomial degree. They should be integers. In addition, there was no real fit for the 0th degree polynomial, and the starting point here is not accurate

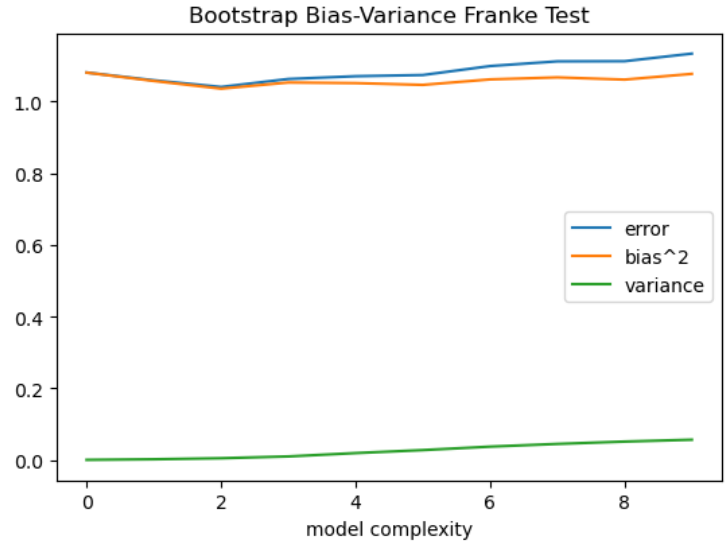

FIG. 3. Here we have the bias and variance plot of a run with bootstrap, running $1e3$ resamplings over $1e6$ data points with a noise strength of 1 and fitting to the training set. We see that the bias starts out very high and gradually increases. The variance begins very small, and then slowly increases. The summed up error roughly follows the 2, but there is a distinct difference in the behaviour for the error here and in figure 2 with regards to the error's behaviour over the model's increasing complexity.

was attempted cleaned up and object oriented in order to reduce copying code and making the usecase more general, especially wrt. including the real terrain data. These attempts did not bear fruit, unfortunately, as when I finally managed to get a working example of the simplest case, the results were strange, even just the measure of the MSE.

Another issue I encountered was difficulties in selecting which data to keep. The β coefficients would definitely be a good idea. Perhaps also the train and test indices, as well as the training and test fits of the model. I did not settle on a method I was happy with. Further usefull data, not the least in the case of plots were the MSE, bias, etc.

which would need to be stored or used for each iteration of each of the varying parameters.

I'd think the main improvement for this project would be a more carefully setup, object oriented design so that the varying resampling and regression methods would be more easily changed without retyping so many initial variables and data, as these quickly accumulated errors and made the code very tiresome to navigate. Further, this would make entering in the actual terrain data much easier and would help guarantee the results could be comparable between the to usecases given no changes were made to the code running on the data sets.

V. APPENDIX

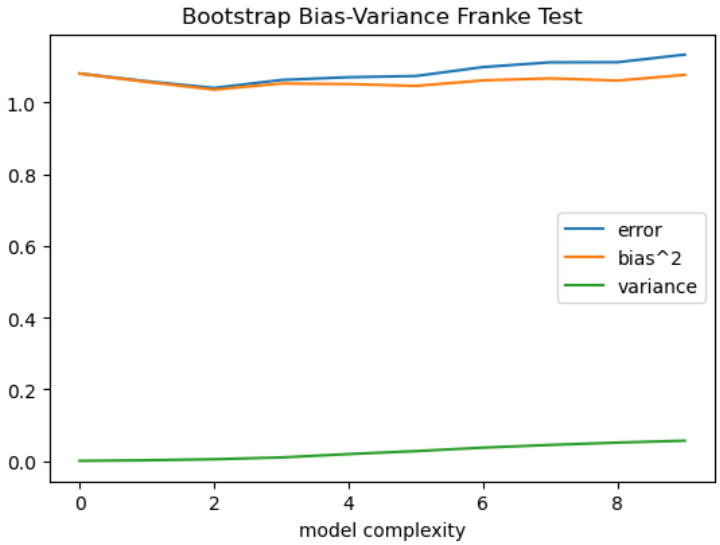

FIG. 4. Bias, variance and error for the bootstrap resampling of the franke data fitted to the test set. $1e3$ bootstraps and $1e4$ data points input.

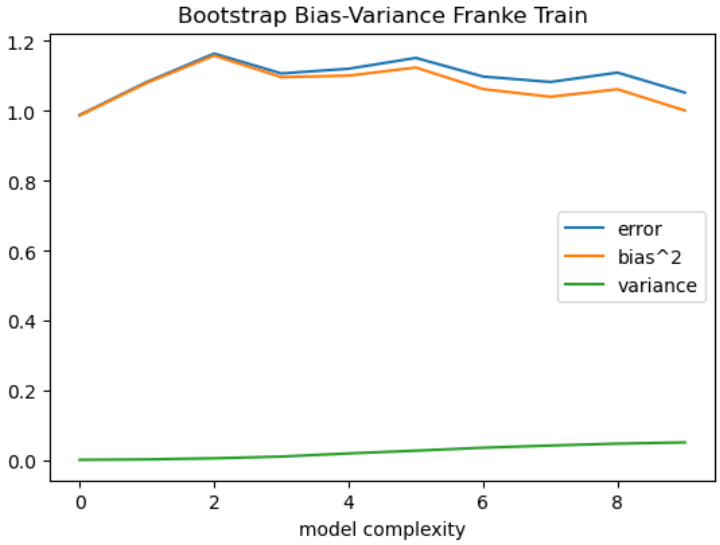

FIG. 5. Bias, variance and error for the bootstrap resampling of the franke data fitted to the train set. $1e3$ bootstraps and $1e4$ data points input.

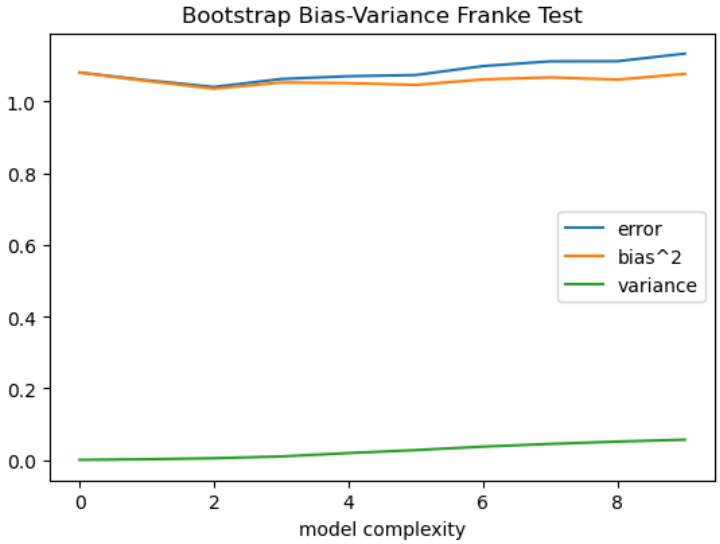

FIG. 6. Bias, variance and error for the bootstrap resampling of the franke data fitted to the test set. 1e3 bootstraps and 2e4 data points input.

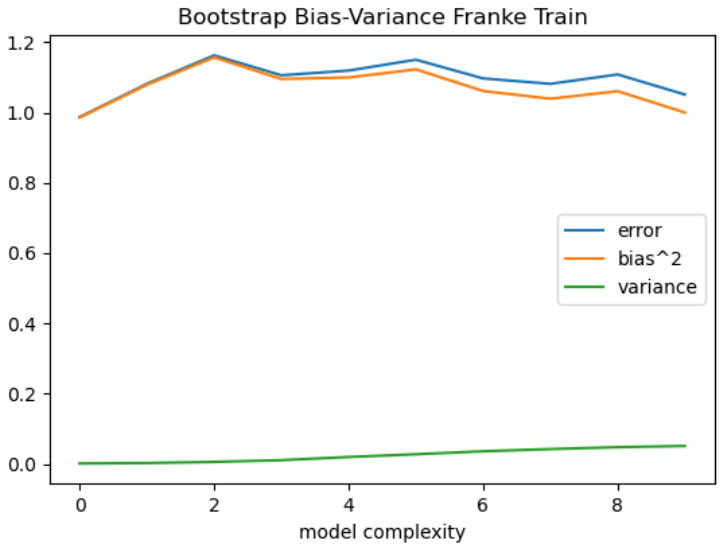

FIG. 7. Bias, variance and error for the bootstrap resampling of the franke data fitted to the train set. 1e3 bootstraps and 2e4 data points input.

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction* (Springer Science & Business Media, 2009).
[2] M. Hjorth-Jensen, Lecturenotes fys-stk4155 – applied data analysis and machine learning (2020).
[3] R. Franke, *A critical comparison of some methods for interpolation of scattered data*, Tech. Rep. (NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 1979).
[4] E. W. Weisstein, "expectation value." from mathworld–a wolfram web resource.

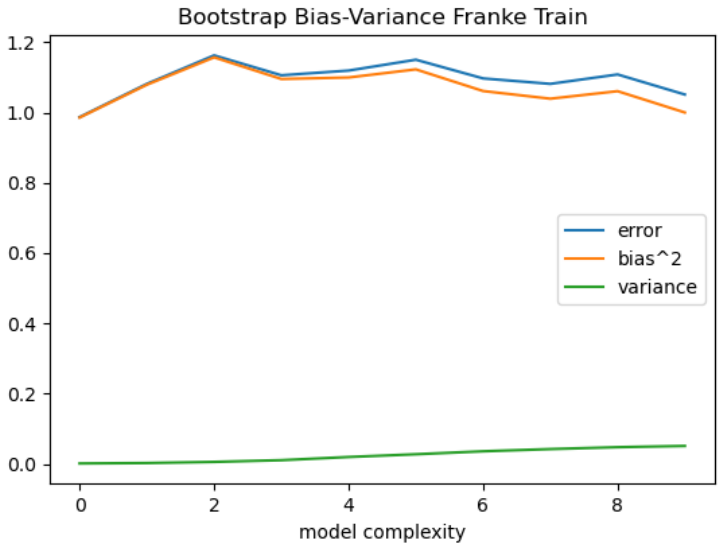

FIG. 8. Bias, variance and error for the bootstrap resampling of the franke data fitted to the train set. 1e3 bootstraps and 1e6 data points input.