

# project 1, FYS-STK4155

## Regression and Resampling of Terrain data and the Franke function

A. G. Eriksen

(Dated: October 12, 2020)

The project motivation lies in exploring linear regression methods with regards to simulated and real terrain data.

The Ordinary Least Squares method has been explored for the simulated terrain, both with and without resampling methods.

The resampling methods used are the bootstrap and k-fold Cross validation methods.

The intent was to implement Ridge regression and Lasso regression for the simulated data and then move on to the real terrain, but time ran out and bugs kept popping up. The only data here are from the OLS

### CONTENTS

I. introduction	1
II. method	2
III. results	7
IV. conclusion	9
V. appendix	11
References	11

### I. INTRODUCTION

The aim for the project was to explore regression methods for predicting an assumed functional dependence. To build up and test the methods we used a function called the

“Franke function”[1] to simulate a terrain and provide a definitive functional dependency between the input data and the targets.

The initial regression was made using various polynomial interpolations as the basis, and studying the results of the fit for various models to try and map out error estimates in regards to the complexity of the models.

Once the simplest models were tested we added resampling of the generated data using the bootstrap method to improve and explore the accuracy of the models, exploring the bias- variance trade-off in particular.

We move on to compare with a different, popular resampling method called k-fold Cross- Validation(k-fold CV). This would provide some comparisons with the bootstrap method.

Following this, the intent was to implement a few more regression methods and explore reductions in the variance of the model predictions from penalising specific features of the model, resulting in more stable predictions. Once this was done we could bring the code to actual terrain data, whose assumed functional dependency we could not know, nor really could guarantee and go over the various regression and resampling methods for the predictions. Time ran out and this remains unexplored.

## II. METHOD

Theoretical models and technicalities

The basis for the regressions used here, is the Franke function[1]

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

The function is defined for values  $x, y \in [0, 1]$ . To make the simulation more authentic, noise can be added when generating the targets through the function, with a normally distributed noise,  $\varepsilon$ . Here  $\varepsilon$  follows a normal distribution of  $\mathcal{N}(0, 1)$ ,  $\sigma^2 = 1$ . Though the strength of the noise can be varied to highlight statistical effects.

The input  $x$  and  $y$  values are generated as arrays, either through a normal distribution of numbers in the range and then sorted, or just as a range or linearly spaced array with a given amount of elements. Though to get a proper terrain grid, we need to mesh the 2 together. Once this is done, we generally flatten the arrays before working on them to simplify things.

The regression model can be written as a matrix-vector product,

$$\tilde{y} = \mathbf{X}\beta. \quad (2)$$

This is based on an assumption that the target values are dependent on the input variables along a model like

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) + \varepsilon \quad (3)$$

This allows us to define a cost function dependent on  $\beta$ , it's derivative and the optimal  $\beta$ :[\[2\]](#) [\[3\]](#)

$$C(\beta) = \frac{1}{n} \{(\mathbf{y} - \mathbf{X}\beta)^2\}, \quad (4)$$

$$\frac{\partial C(\beta)}{\partial \beta} = 0 = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta), \quad (5)$$

$$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \quad (6)$$

With an expression for the optimal beta, we now have to find a way to build our feature matrix for a polynomial of the  $n$ -th degree, to allow for varying model complexity.

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

**Input:**  $x$  and  $y$  and polynomial degree  $n$

**Output:** Feature matrix  $X$  with dimension  $m \times n$

```

 $n \leftarrow \text{length}(x)$ 
 $m \leftarrow \text{int}((n + 1) \cdot (n + 2) / 2)$ 
 $X \leftarrow \text{matrix.ones}(n \times m)$ 
for  $i \leftarrow 1$  to  $n + 1$  do
     $q \leftarrow \text{int}((i) * (i + 1) / 2)$ 
    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  $n$

apply various statistics to compare the model to the targets. These would include

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

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

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

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

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

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

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

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]

Finally, we can also look more closely at the regression method we use. The Ordinary

**Input:** feature matrix,  $X$ , targets,  $y$ , and number of bootstraps,  $N$

**Output:** model fits and predictions

```

 $\tilde{y}^{fit} \leftarrow \text{array}(N)$ 
 $\tilde{y}^{predict} \leftarrow \text{array}(N)$ 
 $\beta \leftarrow \text{array}(N)$ 
for  $i \leftarrow 0$  to  $N$  do
    | Shuffle  $X$  and  $y$ 
    | Split data into training and test sets
    |  $\beta \leftarrow OLS(X_{train}, y_{train})$ 
    |  $\tilde{y}_i^{fit} \leftarrow X_{train}\beta$ 
    |  $\tilde{y}_i^{predict} \leftarrow X_{test}\beta$ 
end

return  $\beta, \tilde{y}_{fit}, \tilde{y}_{predict}$ 

```

**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

Least Squares solution works OK, but there are issues, for instance if  $(X^T X)^{-1}$  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  $(X^T X + \lambda \mathbb{I})^{-1}$

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

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

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

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

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

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

**Input:** feature matrix,  $X$ , targets,  $y$ , and number of folds,  $k$

**Output:** model fits and predictions

$\tilde{y}^{fit} \leftarrow \text{array}(k)$

$\tilde{y}^{predict} \leftarrow \text{array}(k)$

$\beta \leftarrow \text{array}(k)$

Split  $k$  folds  $\mu, \nu \subset X, y$

**for**  $i \leftarrow 0$  **to**  $k$  **do**

$X^{test} \leftarrow X\{\mu_i\}$

$X^{train} \leftarrow X\{\mu - \mu_i\}$

$y^{test} \leftarrow y\{v_i\}$

$y^{train} \leftarrow y\{\nu - v_i\}$

$\beta \leftarrow OLS(X_{train}, y_{train})$

$\tilde{y}_i^{fit} \leftarrow X_{train}\beta$

$\tilde{y}_i^{predict} \leftarrow X_{test}\beta$

**end**

**return**  $\beta, \tilde{y}^{fit}, \tilde{y}^{predict}$

**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  $\beta$ '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 three 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

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

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  $\mathbb{E}[\tilde{y}]$

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

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

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

Next bit requires a little bit of work, but Wolframalpha[4] lets us set  $\mathbb{E}[f \cdot \tilde{y}]$ , which we can separate as they're both deterministic so,  $\mathbb{E}[f]\mathbb{E}[\tilde{y}] - \mathbb{E}[\tilde{y}]^2$ , 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

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

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

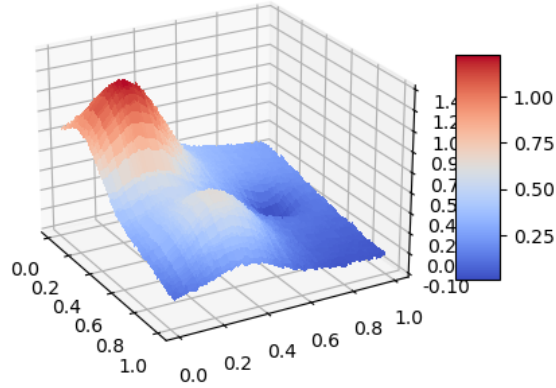


FIG. 1. Surface plot of the simulated surface generated through the Franke function. The inputs and outputs are both unitless, so the axes are merely illustrative. This figure was generated using a slight amount of noise added to the function, which is the source of the slight irregularities observed. 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 8 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.



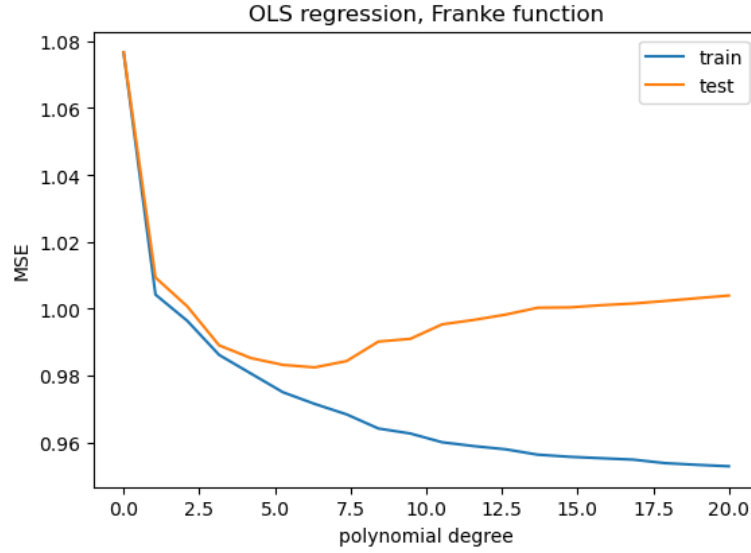


FIG. 2. Plot of the MSE for OLS regression of the Franke function with no resampling techniques active. The worst fit is for the straight line case, where the MSE lies at about a 1, with a rapid reduction which smooths out around 5th degree polynomial for the test set, while the training fit continues 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

#### 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 8. 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 degree of complexity increases. Perhaps it's an error

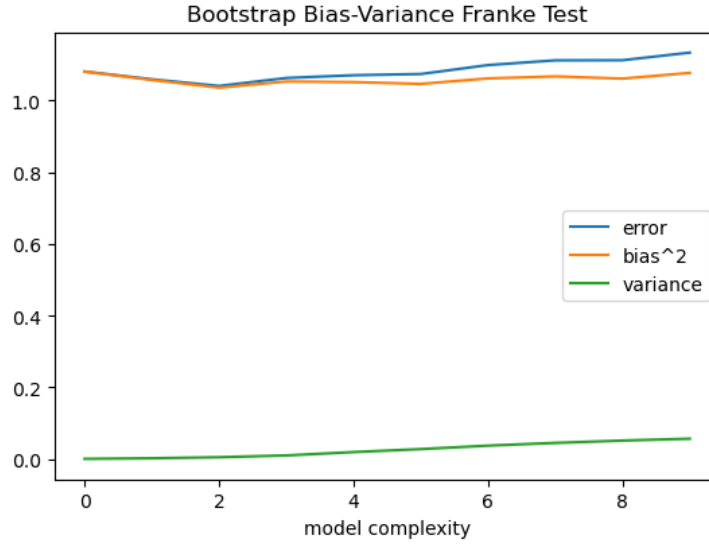


FIG. 3. Here we have the bias and variance plot of a run with bootstrap, running  $1e3$  resamplings over  $1e6$  data points with a noise strength of 1 and fitting to the training set. We see that the bias starts out very high and gradually increases. The variance begins very small, and then slowly 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.

regarding the dimensionality along which we took the means for the difference in the bias and variance. Mixing up the rows and columns 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 of repeated code showed its toll in strange errors. Though the code 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  $\beta$  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

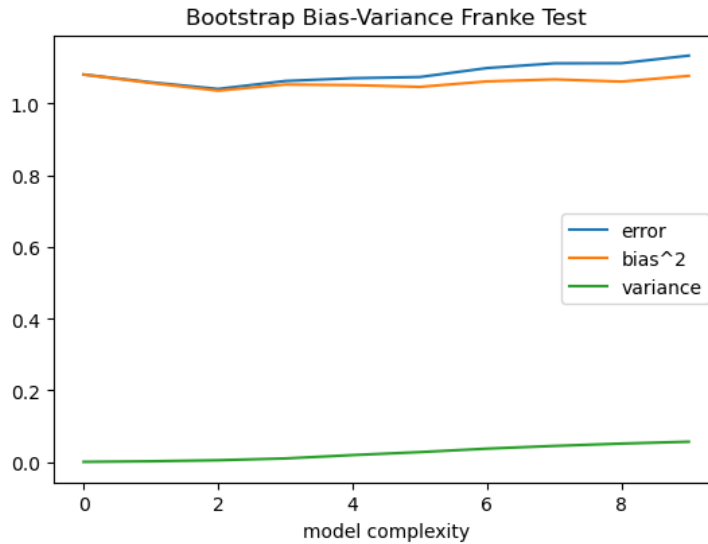


FIG. 4. Bias, variance and error for the bootstrap resampling of the franke data fitted to the test set.  $1e3$  bootstraps and  $1e4$  data points input.

- 
- [1] R. Franke, *A critical comparison of some methods for interpolation of scattered data*, Tech. Rep. (NAVAL POSTGRADUATE SCHOOL MONTEREY CA, 1979).
  - [2] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction* (Springer Science & Business Media, 2009).

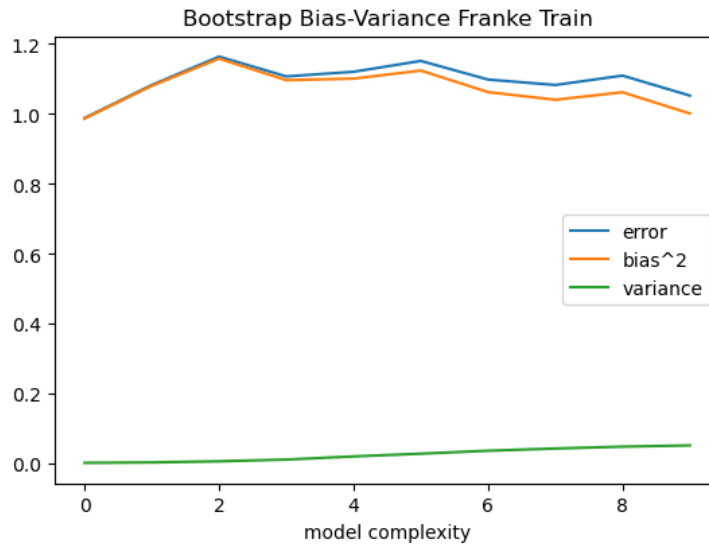


FIG. 5. Bias, variance and error for the bootstrap resampling of the franke data fitted to the train set.  $1e3$  bootstraps and  $1e4$  data points input.

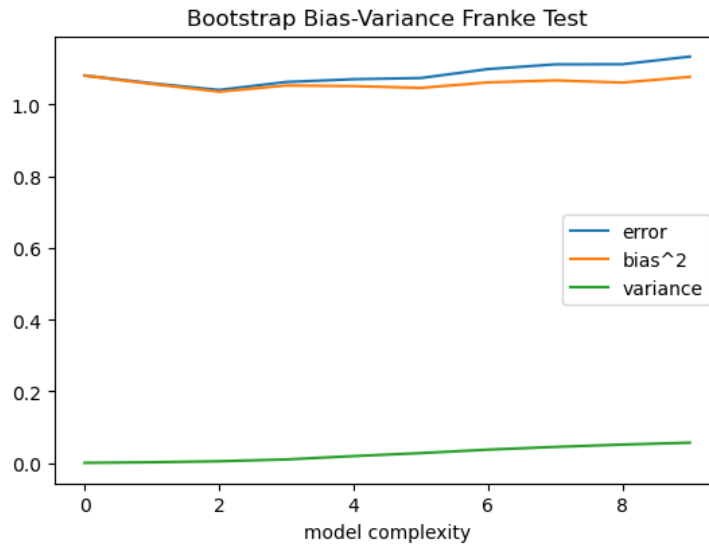


FIG. 6. Bias, variance and error for the bootstrap resampling of the franke data fitted to the test set.  $1e3$  bootstraps and  $2e4$  data points input.

[3] M. Hjorth-Jensen, Lecturenotes fys-stk4155 – applied data analysis and machine learning (2020).

[4] E. W. Weisstein, "expectation value." from mathworld—a wolfram web resource.

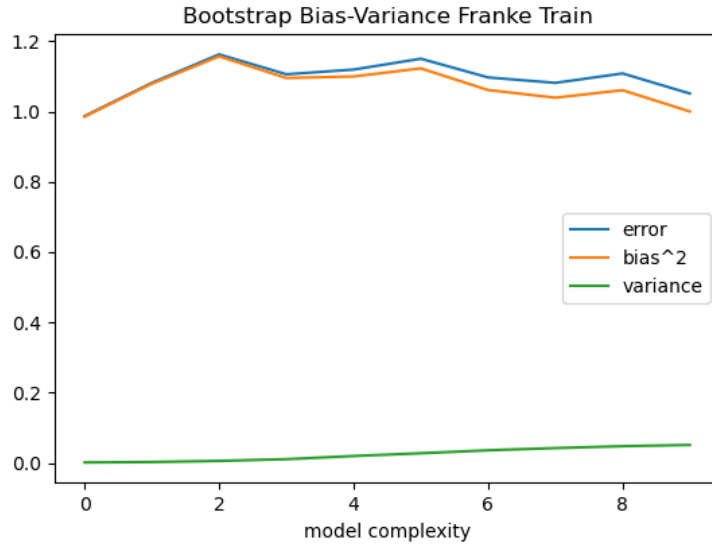


FIG. 7. Bias, variance and error for the bootstrap resampling of the franke data fitted to the train set.  $1e3$  bootstraps and  $2e4$  data points input.

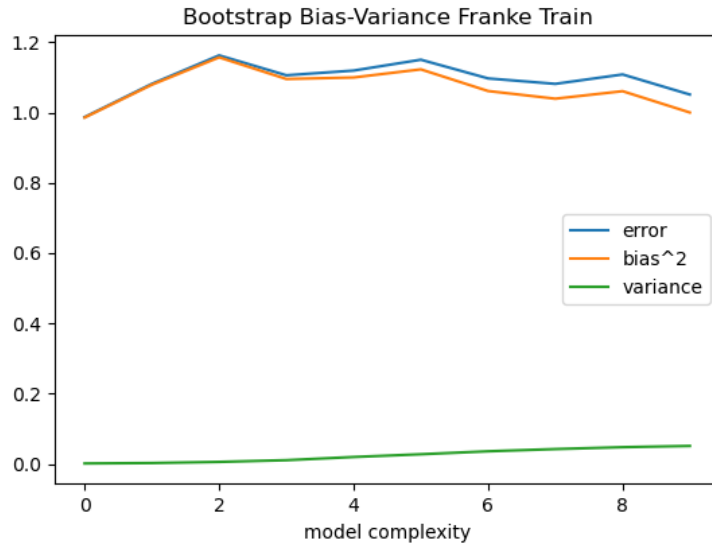


FIG. 8. Bias, variance and error for the bootstrap resampling of the franke data fitted to the train set.  $1e3$  bootstraps and  $1e6$  data points input.