Regression analysis and resampling methods for prognosticate terrain data

Isabel Sagen, Dorina Rustaj & Eir Eline Hørlyk

University of Oslo, Physics Department

(Dated: October 10, 2024)

The primary aim of this report is to investigate the performance of various regression and resampling methods in fitting the Franke function and real-world terrain data. The motivation is to identify the methods and parameters that minimize error, thereby enhancing accuracy in practical applications. We applied Ordinary Least Squares (OLS), Ridge, and LASSO as our regression techniques. Based on the test mean squared error (MSE), we found that the three methods performed similarly for the Franke function. However, when analyzing terrain data, Ridge regression yielded the smallest mean square error(MSE) with optimized parameters, indicating stronger performance. As for resampling methods, we looked at cross validation k-fold for $k = 5, 10$ and the bootstrap method for 1000 iterations. The k-fold exhibited a lower tendency for over fitting as shown by a decreasing MSE, presented as a function of complexity. In contrast, the bootstrap showed an increase in MSE at complexity $d \approx 8$, indicating the onset of over fitting. In comparing the Franke function with the real-world terrain data, we found that the Franke function consistently produced smaller errors across all cases we examined. This discrepancy is likely due to the irregular nature of the terrain data, which presents greater challenges for fitting.

All computational work is displaced in the Github repository.[a]

I. INTRODUCTION

Polynomial fitting is a key method in machine learning and data modeling, often used to approximate complex functions. Its applications spans various domains, showcasing its versatility and importance. It forecasts market trends in economics, analyzes stress-strain relationships in engineering, studies pollutant impacts in environmental science, determines optimal dosages in medicine, aids in calibration in physics, and facilitates smooth rendering in computer graphics. This study focuses on fitting polynomials to Franke's bivariate test function, introduced by Richard Homer Franke *(1937–2023)*, a professor known for his work in numerical analysis [8]. First presented in his 1979 paper, *A Critical Comparison of Some Methods for Interpolation of Scattered Data*, the function combines two Gaussian-like peaks, a depression, and a smaller peak, providing a varied surface to benchmark approximation methods [2].

We will first apply *Ordinary Least Squares* (OLS) regression to the Franke function, using polynomials of the from $z = f(x, y)$. Beyond OLS regression, we will extend our study to include regularization techniques like Ridge and LASSO regression, which introduce a penalty term to control model over-fitting. This allows us to analyze how the inclusion of regularization impacts model bias and variance. Furthermore, we will explore resampling techniques such as bootstrap and k-fold-cross-validation to rigorously assess model performance. Bootstrap involves creating multiple subsets from the original dataset through random sampling with replacement, allowing us to estimate the model's stability. K-fold cross-validation evaluates a model's generalization ability by dividing the dataset into k subsets, training on $k-1$ folds and testing on the remaining fold. Particular focus will be placed on understanding the Bias-Variance trade-off and MSE, which aims to minimize both bias and variance to improve the model's ability to generalize to unseen data. [9].

Finally, to demonstrate the real-world applicability of these methods, we will apply them to digital terrain data obtained from Norway. Our objective is to model this terrain data using polynomial fits and to investigate which polynomial degree yields the best approximation.

The report is structured as follows: In the *Methods* section II, we present the theoretical framework of our model along with the applied methodology. This covers both the analytical foundation and the numerical techniques utilised. The analytical calculations for the expected values are presented in *Appendices* A and B. Further we present our results in the *Results and Discussion* section III. Here we conduct a detailed analysis and reflection on the obtained results.

II. METHODS

Data

As mentioned, we aim to utilize linear regression models to predict real-world terrain data. The objective is to develop a system that takes an input vector $\mathbf{x} \in \mathbb{R}^n$ and predicts a scalar value $y \in \mathbb{R}$ as the output. The output of the linear regression model will be a linear function of the input. If $\hat{y}$ represents the predicted value for y, the

[a] Github: https://github.uio.no/isabesag/fysstk_p1

output can be expressed as follows:

$$\hat{y} = \mathbf{w}^T \mathbf{x}, \tag{1}$$

here $\mathbf{w} \in \mathbb{R}^n$ is a vector of parameters. The parameters are the values that control the behavior of the system. Thus, $\mathbf{w}$ is a set of weights that determine how each features, x_i, affects the prediction. If a feature receives a positive weight w_i, then increasing the value of that feature increases the value of our prediction $\hat{y}$. The opposite will happen if the feature receives a negative wight [4].

To begin, we will test the methods using the two-dimensional Franke function, which serves as synthetic data and is defined as follows:

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

for $x, y \in [0, 1]$. We denote n as number of points in x and y. We will add stochastic noise drawn from a normal distribution with a standard deviation of 0.1, represented as $f(x, y) + 0.1N(0, 1)$. Our grid will be 50×50.

Moreover, we scale the data to ensure all values are evaluated on the same level, preventing numerical instabilities and enhancing model performance. This step is crucial for Ridge and LASSO regression, where the regularization term penalizes large coefficients; unscaled features can result in unfair penalization and biased estimates.

We use Scikit-Learn's `StandardScalar` to scale both x and y by subtracting their mean and setting their variance to one, ensuring the features are centered around zero with a standard deviation of one. However, this method does not enforce specific maximum or minimum values in the dataset [4].

Test-Train-Split

It is typical for all machine learning studies to split the data into a training set and a test set. The **Scikit-Learn** Python library provides a function *train_test_split*, which will be utilized in this study [7]. We have decided to set 4/5 of our data as training data. In our analysis we will use $n = 50$ datasets. This approach provides a substantial training set while ensuring that the test set remains sufficiently sized. Hence we found this split to suit our analysis.

Design matrix

We are using a classic linear model for our deductions.

$$z = f(x, y) + \epsilon. \tag{3}$$

This means we expect there to be a linear function z of degree d that satisfies equation (3). Here ϵ is drawn from the normal distribution $N(0, \sigma^2)$ where σ is the standard deviation of the distribution. In our case we want to describe equation (3) as a matrix equation:

$$\hat{z}_i = X\beta. \tag{4}$$

Here X is the design matrix of degree d and β is a vector containing the coefficients with length $(d + 2)(d + 1)/2$. For X, each joint will combined have maximum d degrees. For instance, for a given degree d and n points in x and y, the corresponding design matrix can be constructed as follows:

$$X = \begin{pmatrix} 1 & x_0 & y_0 & x_0y_0 & \dots & x_0^d & y_0^d \\ 1 & x_1 & y_1 & x_1y_1 & \dots & x_1^d & y_1^d \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_n & y_n & x_ny_n & \dots & x_n^d & y_n^d \end{pmatrix}. \tag{5}$$

This matrix contains the feature values for all n observations, and has dimensional $p \times n$. Here, p represents the number of polynomial features we wish to include in the fit.

Ordinary least squares

Ordinary Least Squares is a foundational method for linear regression, applied as a benchmark to compare more complex models like *Ridge* and *LASSO regression*. It estimates optimal parameters β by minimizing the residual sum of squares between the observed values z_i and the predicted values $\hat{z}_i = X\beta$. It is based on an important algorithm, *Singular Value Decomposition* (SVD), which decomposes a matrix into left singular vectors, singular values, and right singular vectors. This technique enables dimensionality reduction, noise reduction, and data compression [1]. The method works by defining a cost function and determining the values of β where its derivative is zero. The cost function is expressed as

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \hat{z}_i)^2, \tag{6}$$

$$= \frac{1}{n}(z - X\beta)^T(z - X\beta), \tag{7}$$

which represents the average squared error between the observed values z and predicted values $\hat{z}$. Here, X is the input matrix (design matrix in this case) and z is the vector of the observed data.

(6) is a convex function, so we can find the derivative as

$$\frac{\partial C(\beta)}{\partial \beta} = X^T(z - X\beta), \tag{8}$$

$$= 0. \tag{9}$$

Which gives the optimal value for β to be

$$\hat{\beta} = (X^T X)^{-1} X^T z. \tag{10}$$

The full derivation can be found in Appendix A. The expectation value of $\hat{\beta}$ is

$$\hat{\beta} = \beta, \tag{11}$$

the full derivation is given in appendix A. (11) showcase that the expected difference between $\hat{\beta}$ and β is zero, meaning that $\hat{\beta}$ is an *unbiased* estimator. Furthermore, the variance of $\hat{\beta}$ is

$$\mathrm{Var}(\hat{\beta}) = \sigma \left(X^T X\right)^{-1}. \tag{12}$$

Where again the full derivation can be found in Appendix A. The expectation value of z_i is

$$\mathbb{E}(z_i) = \mathbb{E}(X_{i,*}\beta) + \mathbb{E}(\epsilon_i),$$

$$= X_{i,*}\beta. \tag{13}$$

Moreover, the variance of z_i is defined as

$$\mathrm{Var}(z_i) = \sigma^2. \tag{14}$$

Ridge regression

Ridge regression is a regularization technique that utilizes hyper-parameters to control the behavior of the algorithm denoted $\lambda \geq 0$ [4]. The technique is particularly useful for modeling data where the features are correlated. When features are correlated, it can make it hard for a model to make good predictions because it gets confused by the overlapping information. This method adds a penalty to the model to prevent it from giving too much importance to any one feature [5]. Specifically, MSE can be penalized by the value of the intercept, indicating that different values of the regularization parameter λ can lead to varying MSE results. The λ penalizes the sum-of-squares of the parameters β. In principle this is done by finding the optimal parameters with respect to the cost function defined as

$$C(\beta) = \frac{1}{n}\|\mathbf{z} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2, \tag{15}$$

where $\|\cdot\|_2$ denotes the L_2 norm, which represents the sum of the squares of the coefficients. This technique are penalizing weights that become too large in magnitude, however we are not penalizing the offset term, w_0, since it only affects the global mean of the output, and does not contribute to over-fitting [6]. By minimizing this cost function, we seek to find the best-fitting coefficients while also keeping their magnitude small, thereby penalizing larger coefficients.

To derive the Ridge estimator, we take the derivative of the cost function (15) with respect to β and set it equal to zero. This results in the following expression for the Ridge estimator:

$$\hat{\beta}_{\text{Ridge}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{z}. \tag{16}$$

In this equation, $\mathbf{I}$ represents the $p \times p$ identity matrix. The inclusion of the $\lambda\mathbf{I}$ term helps to regularize the model, addressing issues such as multicollinearity and over-fitting by shrinking the coefficients towards zero.

LASSO regression

Least Absolute Shrinkage and Selection Operator (LASSO) regression is a linear regression method that adds an L_1 penalty to the cost function to reduce over-fitting and make the model easier to interpret. The cost function is represented by the regularized least squares loss function. Similarly to Ridge regression, this method is also valuable when dealing with datasets where features are correlated [5]. This is done because sometimes we want the parameters to not just be small, but to be exactly equal to zero [6]. The cost function for LASSO is given by:

$$C(\beta) = \frac{1}{n}\|\mathbf{z} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1. \tag{17}$$

$\mathbf{z}$ is the actual values we aim to predict, $\mathbf{X}$ is the matrix of input features, $\boldsymbol{\beta}$ are the model coefficients. The variable n denotes the number of data points. It is important to note that the inclusion of the L_1-norm makes the derivative of the cost function discontinuous, meaning there is no analytical solution for β. Therefore, we will use Python library `Scikit-learn` to perform LASSO regression.

The penalty in LASSO encourages some coefficients to be exactly zero, effectively identifying key features and simplifying the model. This approach is particularly advantageous when dealing with many features and limited data, as it helps prevent the model from fitting the noise rather than the true signal.

Bias-Variance Trade-off

As previously mentioned, the key aspect of the Bias-Variance Trade-off is to minimize both bias and variance, thereby improving the model's ability to generalize to unseen data beyond the training set. This approach ensures the model can make accurate predictions for new, previously unobserved samples.

The expected predicted error is given by

$$\mathbb{E}[(y-\tilde{y})^2] = \text{Bias}[\tilde{y}] + \text{var}[\tilde{y}] + \sigma^2. \tag{18}$$

The error is based on a *bias* term and a *variance* term. The *Bias* term indicates how closely the average of the model´s predictions align with the actual function we aim to approximate. A model with high bias makes strong assumptions about the underlying data and is often too simple to capture the true relationship. High bias can lead to under fitting. The bias term is defined by

$$[\text{Bias}(\tilde{z}_i)]^2 = [f_i - \mathbb{E}(\hat{z}_i)]^2. \tag{19}$$

The *variance* term describes how much the model´s predictions vary around their average; it measures how sensitive the model is to changes in the training data. High variance implies that the model is overly sensitive to the specific data it was trained on, this means that small changes in the training set can result in large changes in the model´s predictions. This can happen when a model is too complex. The variance terms is defined as

$$\text{Var} = \mathbb{E}[\mathbb{E}(\tilde{F}_{\sqsupset}) - \tilde{F}_{\sqsupset}]^2. \tag{20}$$

The *error* term, σ^2, arises from the noise inherent in the data, which cannot be reduced, even when the model provides a good estimate of the true function. This type of error is inherent in the data itself and is not due to the model. From (18) we need to minimize the bias and the variance of our model to find the best possible fit to find a balance between under- and over-fitting.

Resampling

In statistics, when working with limited datasets, resampling methods offer an effective way to maximize data usage. These techniques generate additional datasets by resampling the training data, allowing us to compute test errors from regression fits based on different samples of training data. This approach enables a more accurate estimation of the dataset's statistical properties [3]. In our analysis, we apply resampling methods to examine the bias-variance-trade-off and evaluate the MSE. Specifically, we have utilised two resampling methods: the *bootstrap method* and *Cross-validation*.

Bootstrap

The bootstrap method involves an initial division of the dataset into training and test sets. Let m denote the number of data points in the training set. During each bootstrap iteration, a new training set is created by randomly selecting m data points with replacement from the original training set. The regression is performed for N bootstrap iterations which yields a more robust idea of the performance of the model. Lastly we find the test-errors from the test set used on the N previous fits meaning that we will have N different values for MSE. Lastly the averages of the different bootstrap samples are computed. Note that only the training data is resampled.

The estimated test error based on $X_{j,*}\hat{\beta} = \hat{z}_j$ is

$$\overline{\text{MSE}} = \frac{1}{N}\sum_{i=0}^{N}\left(\frac{1}{n-\text{test}}\sum_{j=0}^{n-\text{test}-1}[z_j - \hat{z}_j]^2\right).$$

The bias and variance terms are given by

$$\overline{\text{Bias}} = \frac{1}{n-\text{test}}\sum_{j=0}^{n-\text{test}-1}\left(z_j - \frac{1}{N}\sum_{i=1}^{N}\hat{z}_j^{(i)}\right)^2,$$

and

$$\overline{\text{Var}} = \frac{1}{n-\text{test}}\sum_{j=0}^{n-\text{test}-1}\left[\frac{1}{N}\sum_{i=1}^{N}\left(\hat{z}_j^{(i)} - \frac{1}{N}\sum_{i=1}^{N}\hat{z}_j^{(i)}\right)^2\right],$$

respectively.

k-fold cross-validation

In k-fold cross-validation, the dataset is randomly shuffled and divided into k equal parts (folds). One fold is designated as the test set, while the remaining k folds serve as training sets. A linear regression model is trained on the training sets and evaluated on the test set. This process is repeated until each fold has been used as the test set, and the final result is the average error from all test sets.

III. RESULTS AND DISCUSSION

We will start of by analysing and testing the Franke function for different linear regression and resampling methods. Further this expands to the Ridge and LASSO regression. Lastly we will use the terrain data to do the same analysis and compare to the Franke data. Generally we are looking for the best fit which includes small errors.

A. Franke function

Ordinary Least Squares

For this analysis we set $n = 50$ giving a grid of total of 50×50 data points. We add noise of $0.1N(0,1)$. To start off, we set max polynomial degree to $d = 5$ and perform OLS. Our goal is to analyse the R^2 and MSE as a function of the models complexity. Recall that for a perfect fit, R^2 is 1 and MSE is zero.

The results are presented in Figure. 1. As seen here, increasing complexity leads to R^2 and MSE approaching their desired values of 1 and 0, respectively. This suggests that the model is more accurate for larger d (up to $d = 5$). However, we have to consider that increasing model complexity raises the risk of over-fitting. This is when noise is perceived as part of the model which makes the model less accurate. Also, the larger MSE for small d can be a result of under-fitting, meaning when the model has not yet the complexity to capture the data correctly. Generally we want the happy medium between over- and under-fitting.

Furthermore, we visualise the β coefficients from equation (4) as a function of the model's complexity. This is showcased in Figure. 2. Here we see that for increasing d, there are larger deviations from 0. This means that the parameters are becoming larger for increasing complexity. This behavior can arise from both the model getting more sensitive to noise and/or an attempt to capture more advanced patterns in the data.

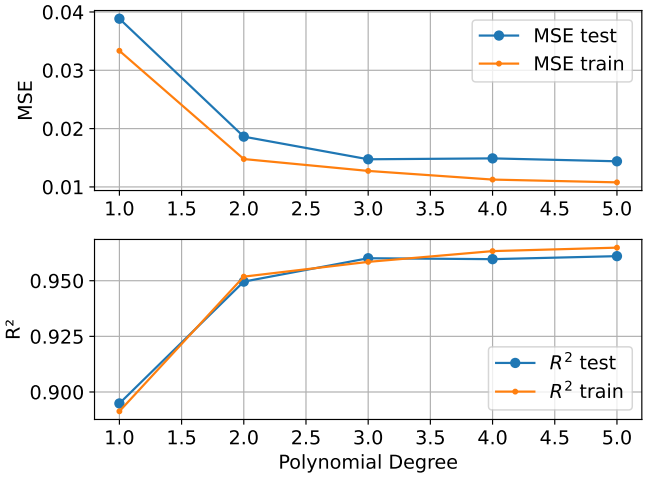

FIG. 1. The MSE and R^2 vs complexity (up to $d = 5$) for 50×50 data points with OLS regression. We observe the values are approaching 0 and 1, respectively, as complexity increases. This suggests the performance of the model improves with increasing complexity (polynomial degree).

We want to utilize resampling methods to perform a bias-variance trade-off analysis for more complex fitting. To first get an insight, we visualise the training and test MSE as a function of complexity, shown in Figure. 3. Here we use the bootstrap resampling method on OLS for 1000 iterations. We see that up to $d = 5$, the test and train data remain very similar. Beyond this the training MSE continues to decrease and the test MSE starts to increase. This can indicate that the test data is getting over-fitted, while the train data is continuing to improve. Perhaps if complexity increased even more we would see the training MSE starting to increase. Moreover, for small d when both train and test are larger, we expect

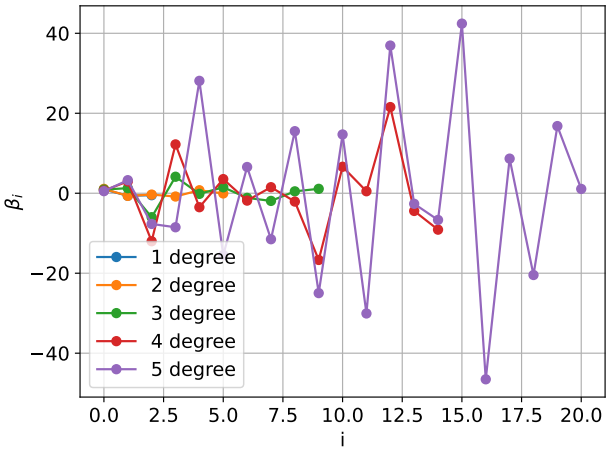

FIG. 2. The β_i values for different d values. i is the index of the β vector.We have used OLS with 50×50 data points and added noise to the Franke function. We see with increasing d, the β value deviates more from 0.

that the bias will be large as well. Conversely, for larger d when test increases and train decreases, the variance is expected to increasing.

To confirm our predictions we visualise the bias-variance-trade-off in Figure. 4. We observe the following: for smaller d we have larger bias and for greater d we get larger variance. Notably at beyond $d = 14$ the variance surpasses the bias and begins to increase more rapidly. This can indicate a point where over-fitting becomes very pronounced. From the figure. we can also asses that $d \in [4, 7]$ will give the best fit. This is because in this region the error is at a minimum and the bias and variance remain relatively stable without any rapid increase. The range $d \in [4, 7]$ also aligns with our analysis of Figure. 3.

Before we continue this exploration we look at how n affects our model. For larger values, $n = 100$ we needed go beyond $d = 15$ to observe over-fitting. This also took a lot of computational power and time. For smaller values like $n = 25$ we saw that the variance surpassed the bias at around $d = 12$. As this is similar to the $n = 50$ case we concluded that $n = 25$ would still be sufficient. however we continue using $n = 50$ for the sake of continuity.

To finalise our study of the error for OLS we analyse how the cross validation k-fold method compares to the bootstrap method. We do this by examining the test MSE over complexity for both methods. The result is shown in Figure. 5. In this analysis, we use 1000 bootstrap iterations and $k = 5$ and 10 folds, considering complexities up to $d = 12$.

We first observe that both resampling methods gives

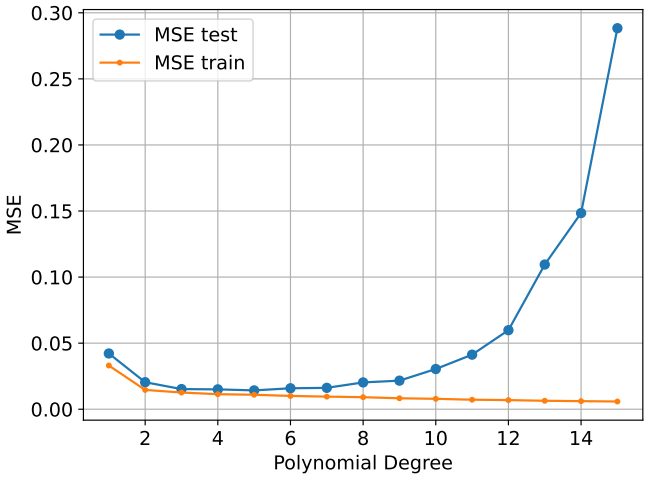

FIG. 3. The train and test MSE for different d values using OLS. Here we have used 50×50 data points, 1000 bootstrap iterations and added noise to the Franke function. We see the test and train set start deviating from each other at around $d = 6$.

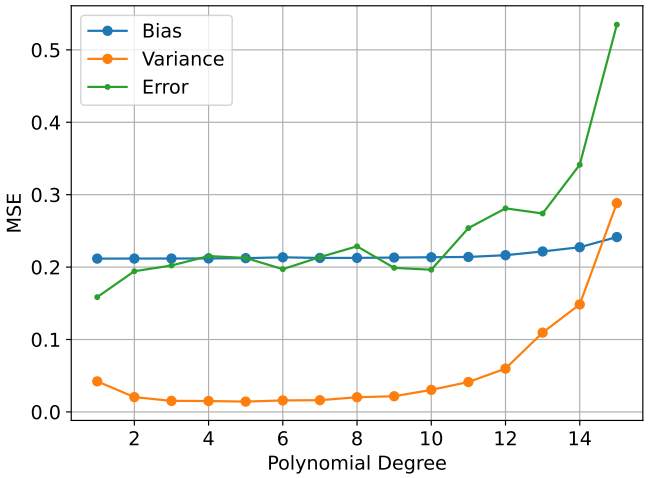

FIG. 4. The bias, variance and MSE for different d values. Here we have used 1000 bootstrap iterations for OLS. There are 50×50 data points and noise added to the Franke function.

similar results for $d = 6$ and lower. Past $d = 7$ the MSE for the bootstrap becomes slightly larger than that for k-fold. For $d = 8$ and onwards they diverge significantly. The bootstrap increases rapidly while the k-fold continues to decrease slightly but evenly. This can indicate that k-fold has less tendency to over-fit at higher complexities. However we should be cautious about drawing this conclusion as it can be a mere coincident. We can also note that $k = 5$ gives slightly better results than $k = 10$. This is not as expected as we would assume more samples performs better due to larger dataset. Nevertheless, this could also be coincidental, and we proceed under the assumption that larger k generally improves regression performance since it provides more data.

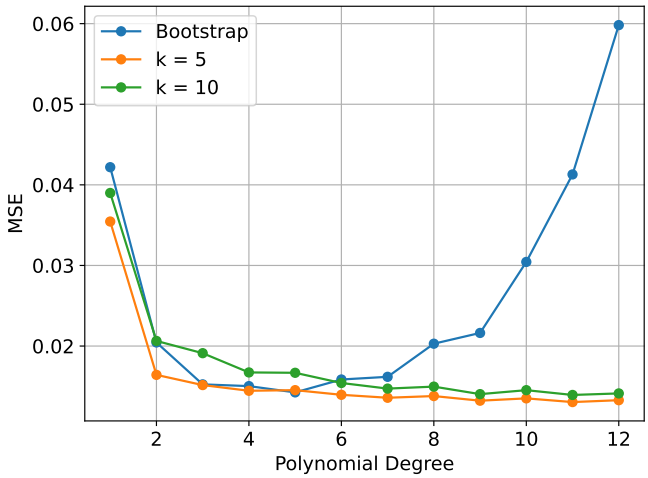

FIG. 5. The MSE for different d values comparing the bootstrap method and cross-validation k-fold. Here we have used 1000 bootstrap iterations and $k = 5$ and 10 folds. We see around $d = 7$ both k-folds outperforms the bootstrap.

Ridge

We explored Ridge regression to find the d and λ value that gives us the best fit. To assert this we search for the smallest the MSE's over different d-λ combination. This is done using k-fold resampling for $k = 10$ as we assume this gives greater stability. The result is shown in Figure. 6

Here we see that the smallest MSE has a value of 0.0142. This corresponds to $d = 5$ and $\lambda = 0.1$. Generally we can observe that $d \approx 1$ and $d \approx 15$ shows larger MSE which can allude to under- and over-fitting, respectively. Further we observe that smaller λ values give largest MSE for larger d. The region around $\lambda \approx 1$ is shown to give smaller MSE values.

LASSO regression

Similar to our approach with Ridge regression, we executed LASSO regression on the Franke function dataset, employing a grid search to find the most suitable polynomial degree and hyper-parameter λ. In Figure. 7, we display the MSE minimization related to model complexity and λ through 10-fold cross-validation. The desired parameters were identified as $d = 6$ and $\lambda = 0.001$, resulting in an MSE of 0.0140 (last digit did not make the plot). This MSE is very similar to that obtained from Ridge regression. However the two have quite dif-

ferent λ values. This may be a result of how the different regression methods operate.

From these results, it appears that LASSO regression performs slightly better than Ridge regression. We can also compare this the MSE for OLS being 0.0140 for $d = 5$(value extracted from terminal, not in plot). Note this is probably a mere coincident and does not suggest OLS and LASSO performs identically. Although the margins are very small, our findings suggest that LASSO and OLS has a slight edge in performance. However this conclusion can not be drawn from our single regression.

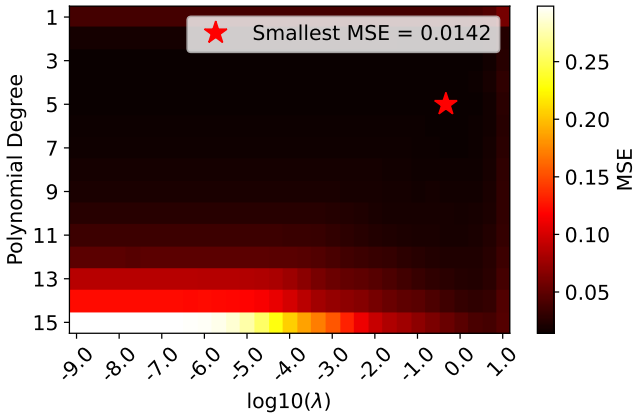

FIG. 6. The MSE for different λ and d values for the Ridge regression. Here we see that the smallest MSE value is 0.0142 corresponding to $d = 5$ and $\lambda \approx 0.1$.

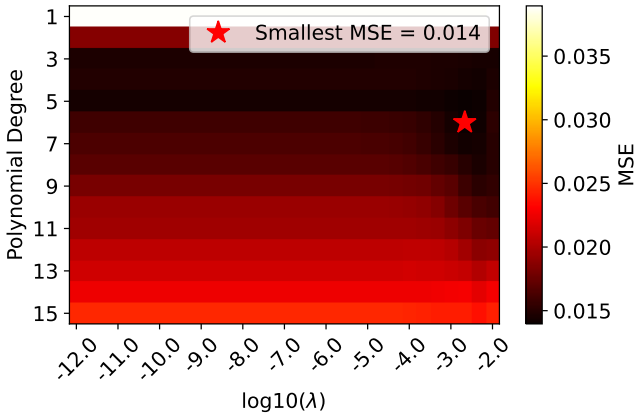

FIG. 7. The MSE for different λ and d values for the LASSO regression. Here we see that the smallest MSE value is 0.0140 corresponding to $d = 6$ and $\lambda \approx 0.001$.

B. Terrain data

Next, we aim to perform the same analysis using real-world terrain data. The selected subset contains 50×50 data points. The dataset used for this analysis represents the geographic terrain in Norway. We decided to indirectly scale the design matrix by scaling vectors x and y, as we did with the Franke function. The terrain height was normalized by taking the mean away and dividing by the standard deviation. This process helps create a more stable algorithm by ensuring that the input and output variables share a consistent scale. We note that the terrain data will be less smooth and contain more complex structures than the Franke function. Hence we are likely to think this data is more difficult to fit.

Ordinary Least Squares

Similarly as for the Franke function, we start of by looking at the MSE and R^2 as a function of complexity up to $d = 7$. We increased from $d = 5$ to $d = 7$ to capture the flattering trend of the data. The analysis is for a single OLS regression, shown in Figure. 8. It shows that as MSE decreases, R^2 increases for both the training and testing datasets, aligning with theoretical predictions. However, when comparing these results to those obtained from the Franke function in Figure. 1, we see the numerical values are smaller for the latter. This suggests that OLS struggles more to fit the terrain data compared to the Franke function, which is what we expected. Notably, at $d = 3$, there is a slight hump in the results that was not observed for the Franke function. This is likely due to the inherent irregularities in the terrain data.

Moreover, Figure. 9 illustrates the relationship between the test and train MSE against polynomial degree d. This is done for d up to 15 for 1000 bootstrap iteration on OLS regression. Here we see the train MSE ascends evenly and approaches 0. In contrast, the test MSE behaves less consistently: it closely follows the training MSE up to $d = 9$, after which the two curves diverge, with the test MSE rising rapidly. This is an indication of over-fitting. When compared to the Franke function, the overall error for the terrain data is notably higher. The terrain data presents a greater challenge for fitting, as reflected by the larger magnitude of the error across all polynomial degrees.

Again we want to compare the bootstrap with the cross-validation k-fold method. This is shown in Figure. 10. The plot displays the test MSE as a function of complexity for 1000 bootstrap iterations, 5 and 10 k-folds. As expected, we see a similar plot that of the Franke function. At $d = 9$ the bootstrap starts to increase, while the k-fold continues to decrees slightly. As mentioned before, this can indicate that the k-fold

method is less likely to over-fit. However, further investigation is needed to confirm this.

Generally, the results we obtained are less smooth compared to those from the Franke function, which likely stems from the irregular and rough nature of the terrain data. The jagged characteristics of the terrain make it harder to achieve a smooth fit, and this unevenness is reflected in our findings. In all three OLS cases, the terrain data exhibits errors that are approximately 10 times larger than those observed with the Franke function. This highlights the added difficulty of modeling real-world terrain data compared to man made data like the Franke function.

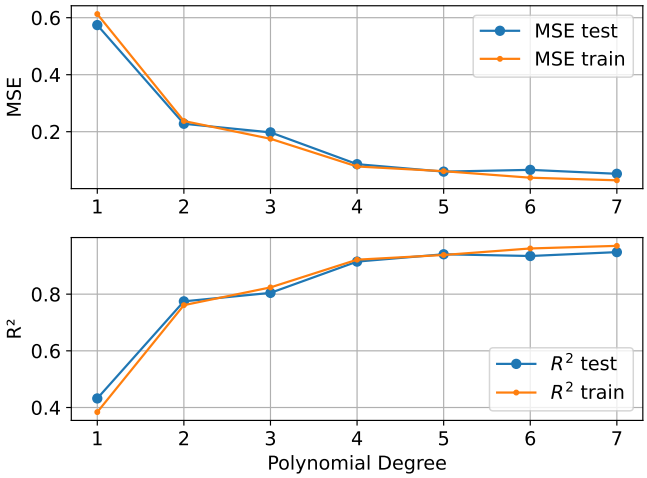

FIG. 8. The MSE and R^2 vs complexity (up to $d = 5$) for 50×50 data points. We used OLS and 4/5 test-train-split. We observe the values are approaching 0 and 1, respectively, as complexity increases. This suggests the performance of the model improves with increasing complexity (polynomial degree).

Ridge Regression

Utilizing our earlier work, we fitted polynomial regression models using Ridge regularization. To find the best parameters, we performed a grid search. Figure. 11 shows the MSE calculated from k-fold cross-validation with $k = 10$. The ideal parameters identified were $d = 9$ and $\lambda \approx 0.01$, resulting in an MSE of 0.026.

LASSO Regression

In the same vein, we applied LASSO regression to the terrain dataset, utilizing a grid search to identify the best parameters. The results of this analysis are displayed in Figure. 12. In this scenario, the best parameters are identified as $d = 11$ and $\log \lambda = 10^{-4}$,

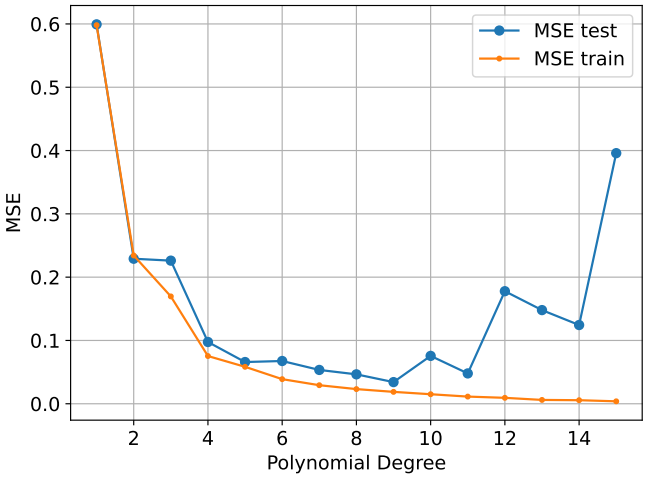

FIG. 9. The train and test MSE for different d values using OLS. Here we have used 50×50 data points and 1000 bootstrap iterations. We see the test and train set start of similar but split notably at around $d = 9$.

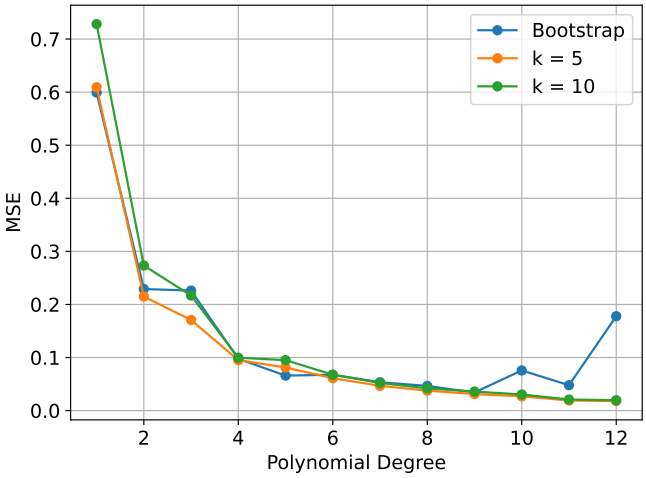

FIG. 10. The MSE for different d values comparing the bootstrap method and cross-validation k-fold for OLS. Here we have used 1000 bootstrap iterations and $k = 5$ and 10 folds. We see around $d = 9$ the k-fold outperforms the bootstrap.

resulting in an MSE of 0.033. LASSO regression yielded a significantly worse fit in this instance compared to Ridge. Additionally, it needed a notably high polynomial degree and smaller λ for optimal parameter adjustment.

For OLS we obtained $MSE = 0.316$ for $d = 9$, which was printed in the terminal (not plotted). Comparing the three methods — OLS, Ridge, and LASSO — Ridge, performs best based on test MSE for the terrain data. Since this result differs from what we observed with the Franke function, we may infer that Ridge regression performs better for datasets with greater

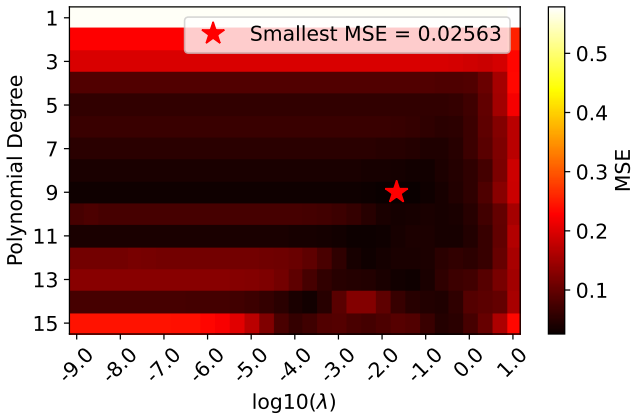

FIG. 11. Figure showcasing colour map representing for different λ and d values. We are using Ridge regression on our terrain data with 50×50 data points. Here we see that the smallest MSE value is 0.026 corresponding to $d = 9$ and $\lambda \approx 0.01$.

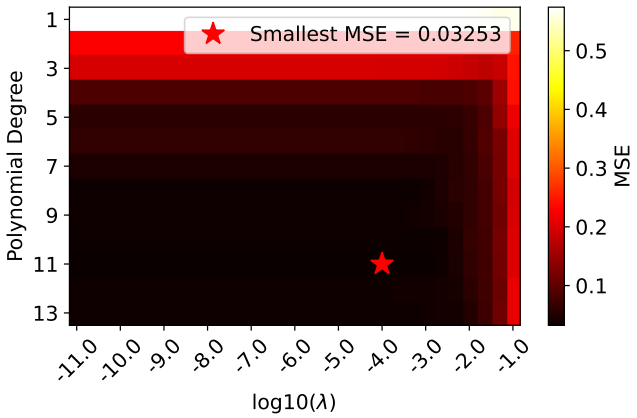

FIG. 12. Figure showcasing colour map representing for different λ and d values. We are using LASSO regression on our terrain data with 50×50 data points. Here we see that the smallest MSE value is 0.033 corresponding to $d = 11$ and $\lambda = 10^{-4}$.

irregularities. This can be explained by the nature of the two methods. While both Ridge regression and LASSO penalize their coefficients, Ridge regression does not exclusively select one variable among correlated features and shrink the others to zero, as LASSO does. This characteristic allows Ridge regression to retain all predictors, thus preserving valuable information that might otherwise be lost.

Comparing and Evaluating the Models

In general, we observed that results were significantly affected by both data type and size. The study we conducted was almost only for 50×50 dataset and we observed that varying n had substantial effects on the results. This suggests that our findings are highly data dependent, meaning they may not generalize well to other datasets or sizes. For now our conclusions stays specific to the data we studied. However, we can still assess how different regression methods perform in our specific cases.

For the Franke function, OLS, Ridge, and LASSO achieved a similar minimum MSE of approximately 0.0140 ($d = 5$), 0.0142 ($d = 5, \lambda = 0.1$) and 0.0140 ($d = 6, \lambda = 10^{-3}$), respectively. However, this performance did not carry over to the terrain data, where Ridge outperformed the others with an MSE of 0.026 ($d = 9, \lambda = 0.01$), while OLS and LASSO gave 0.032 ($d = 9$) and 0.033 ($d = 11, \lambda = 10^{-4}$), respectively. Since the terrain data produced the largest errors and is more complex, we can conclude that it was the hardest to fit in our analysis. Given that Ridge regression achieved the lowest MSE for this dataset, it can be considered the most effective model for handling our data.

One might question the benefits of Ridge and LASSO regression, especially since polynomial features for terrain data, derived from combinations of x and y coordinates, typically show low correlation. However, our findings suggest that Ridge regression outperforms both OLS and LASSO in providing the most accurate fit for real-world terrain data used. This indicates that the advantages of achieving more stable predictions (reducing variance) outweigh the potential drawbacks of slightly decreased accuracy (increased bias) that come from shrinking the model's coefficients.

Our evaluation demonstrates that regularization techniques, like those used in Ridge and LASSO regression, gives the best results, particularly for the complex data structures that is the terrain.

IV. CONCLUSION

In this paper, we evaluated the performance of several regression models; OLS, Ridge, and LASSO, on both synthetic, the Franke function, and real-world terrain data. For the Franke function, OLS and LASSO produced the lowest MSE. However, when applied to the more complex and irregular terrain dataset, Ridge regression outperformed both OLS and LASSO.

Our findings indicate that Ridge regression's strength in reducing variance and producing more stable predictions outweighs the slight increase in bias from shrinking the model's coefficients. This makes Ridge especially useful for datasets with complex patterns, such as the terrain data, where regularization enhances performance. However, this advantage does not extend to the Franke function, where the $MSE_{Ridge} = 0.142$ is slightly higher than both $MSE_{LASSO} = MSE_{OLS} = 0.140$. In con-

trast, for the terrain data, Ridge performs better with $MSE_{Ridge} = 0.026$, lower than $MSE_{LASSO} = 0.033$.

When comparing resampling methods, we found that k-fold cross-validation and bootstrapping performed similarly at lower complexities. However, at higher complexities, k-fold cross-validation was better at resisting over-fitting, as seen by the lower test MSE for higher polynomial degrees.

Notes for future work

To improve the model, future work could involve testing on other synthetic datasets and training on larger amounts of data. This would help better handle overfitting and improve the model's accuracy. Additional research using different datasets and broader resampling techniques would also be needed to confirm the generalizability of these results. As our results were very data dependent, it could be beneficial to perform a more thorough study on how our metrics are effected by changing the data parameters. Note that there is also a lot of room the explore different metrics then MSE, and numbers of resampling iterations. Expanding the d and λ values would also be insightful.

REFERENCES

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Discusses Singular Value Decomposition. Springer, 2006. Chap. 12, pp. 598–610.

[2] Richard H. Franke. *A Critical Comparison of Some Methods for Interpolation of Scattered Data*. Tech. rep. ADA081688. Accessed: 2024-09-26. Naval Postgraduate School, 1979. URL: https://apps.dtic.mil/sti/pdfs/ADA081688.pdf.

[3] Phillip Good. *Resampling Methods: A Practical Guide to Data Analysis*. New York: Springer Science & Business Media, 2006, pp. 1–300. ISBN: 978-0387333980.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer, 2009, pp. 63–68.

[6] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022, pp. 381, 385. URL: probml.ai.

[7] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[8] SIAM. "Obituary: Richard Homer Franke". In: *SIAM News* (2023). Accessed: 2024-09-26. URL: https://www.siam.org/publications/siam-news/articles/obituary-richard-homer-franke/.

[9] Stanford University. *The Bias-Variance Tradeoff*. Tech. rep. Accessed: 2024-09-26. CS229 Lecture Notes, 2017. URL: http://cs229.stanford.edu/notes/cs229-notes3.pdf.

We used ChatGPT to help with fine tuning some parts of the writing of the report.

Appendix A: Mean values and variances in linear regression method

$x_{i,j}$ are individual elements of the design matrix. Given regression model (with error):

$$y_i = x_{i,1}\beta_1 + ... + x_{i,p}\beta_p + \epsilon_i$$

we can write it in vector form as

$$y_i = \sum_j x_{i,j}\beta_j + \epsilon_i$$

thus the expectation value becomes

$$\mathbb{E}[y_i] = \mathbb{E}[\sum_j x_{i,j}\beta_j + \epsilon_i]$$

Since $x_{i,j}$ is non-random, we get that $\mathbb{E}[\epsilon] = 0$, and
$\mathbb{E}[y_i] = \mathbb{E}[\sum_j x_{i,j}\beta_j]$
At last, we implement matrix notation as follows

$$\mathbb{E}[y_i] = X_{i,*}\beta$$

where

$$\mathbf{X}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,k} \end{bmatrix} \text{ is the } i\text{-th row of the design matrix } \mathbf{X},$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix} \text{ is the column vector of coefficients.}$$

Variance for y

$$\begin{aligned}
\mathrm{Var}(y_i) &= \mathbb{E}\left([y_i - \mathbb{E}(y_i)]^2\right) \\
&= \mathbb{E}(y_i^2) - [\mathbb{E}(y_i)]^2 \\
&= \mathbb{E}\left[(X_{i,*}\beta + \epsilon_i)^2\right] - (X_{i,*}\beta)^2 \\
&= \mathbb{E}\left[(X_{i,*}\beta)^2 + 2\epsilon_i X_{i,*}\beta + \epsilon_i^2\right] - (X_{i,*}\beta)^2 \\
&= (X_{i,*}\beta)^2 + 2\mathbb{E}(\epsilon_i)X_{i,*}\beta + \mathbb{E}(\epsilon_i^2) - (X_{i,*}\beta)^2 \\
&= \mathbb{E}(\epsilon_i^2) \\
&= \mathrm{Var}(\epsilon_i) = \sigma^2.
\end{aligned}$$

Expectation value with the OLS expressions for the optimal parameter $\hat{\beta}$

$$\begin{aligned}\mathbb{E}(\hat{\beta}) &= \mathbb{E}[(X^TX)^{-1}X^Ty] \\ &= (X^TX)^{-1}X^T\mathbb{E}[y] \\ &= (X^TX)^{-1}X^TX\beta\end{aligned}$$

We can use that $(X^TX)^{-1}X^TX = 1$

$$= \beta$$

Variance for $\hat{\beta}$

$$Var(\hat{\beta}) = \mathbb{E}[\beta - \mathbb{E}(\beta)][\beta - \mathbb{E}(\beta)]^T$$

We can define $\beta = (X^TX)^{-1}y$ and from the last exercise $\mathbb{E}(\beta) = \beta$

$$\begin{aligned}Var(\hat{\beta}) &= \mathbb{E}[(X^TX)^{-1}X^Ty - \beta][(X^TX)^{-1}X^Ty - \beta]^T \\ &= (X^TX)^{-1}X^T\mathbb{E}yy^TX(X^TX)^{-1} - \beta\beta^T\end{aligned}$$

We can use that $\mathbb{E}yy^T = X\beta\beta^TX^T + \sigma^2I_{nn}$

$$\begin{aligned}&= (X^TX)^{-1}X^TX\beta\beta^TX^T + \sigma^2X(X^TX)^{-1} - \beta\beta^T \\ &= \beta\beta^T + \sigma^2(X^TX)^{-1} - \beta\beta^T \\ &= \sigma^2(X^TX)^{-1}\end{aligned}$$

Appendix B: Analytical Bias-variance trade-off

Our true data is generated from $y = f(x) + \epsilon$ and an approximation $\tilde{y} = X\beta$. This gives us the cost function:

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

We will use the definitions:

$$\text{var}[\tilde{y}] = \mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right]$$

and

$$\text{Bias}[\tilde{y}] = \mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right].$$

First we look at a term for the variance of the model:

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(f + \epsilon - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(f + \epsilon - \tilde{\boldsymbol{y}} + \mathbb{E}[\tilde{y}] - \mathbb{E}[\tilde{y}])^2\right]$$

$$= \mathbb{E}\left[(f + \epsilon - \tilde{\boldsymbol{y}} + \mathbb{E}[\tilde{y}] - \mathbb{E}[\tilde{y}])^2\right]$$

After doing some (a lot) calculations on paper, we get the following:

$$= \mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \sigma^2$$

$$= \text{Bias}s[\tilde{y}] + \text{Var}\left[\tilde{\boldsymbol{y}}\right] + \sigma^2$$