An Investigation on Regression Analysis and Resampling Methods on the 2-Dimensional Franke Function and a Real Topographic Dataset

Muhammad Ali Bin Md Nazeri — Don Philip Bullecer
Department of Physics, University of Oslo, Norway
https://github.com/dondondoooooon/DONFYS-STK3155

October 27, 2022

Contents

Abstract

This article explored the use of Ordinary Least Square (OLS), Ridge, and Lasso linear regression methods; firstly on the two-dimensional Franke function obtaining three Mean Squared Error (MSE) values: $3.78{\cdot}10^{-2}, 8.47{\cdot}10^{-3}, 9.97 \cdot 10^{-1}$. We examined the difference of having noise ε in our Franke function. Furthermore, we studied and performed bias-variance analysis on the Franke function by studying the MSE value as a function of the complexity of our model, the number of data points, and resampling methods. Bootstrap and cross-validation were used to confirm the accuracy of the values we acquired via the regression methods we used on the Frank function under the Bayesian assumption that the data follows a probability distribution. For the Ridge and Lasso method, the bias-variance trade-off was also examined as a function of the hyper-parameter λ. It was concluded that **Ridge** regression, given the correct parameters, performed the best on the Franke function, followed by **OLS**, and **Lasso** regression, in that order. With the preliminary testing conducted on the Franke function, we confirmed the importance of picking the right method with the right parameters when selecting which model would fit the data set the best. For the topographical data obtained from United States Geological Survey (USGS), we performed much of the same preliminary testing to find the optimal parameters for each regression method and determined that with the correct parameters, the **OLS** and **Ridge** regression had the best performance followed closely behind by **Lasso** regression, shown by the three Mean Squared Error (MSE) values: $4.98 \cdot 10^{-1}, 4.98 \cdot 10^{-1}, 5.02 \cdot 10^{-1}$ respectively.

1 Introduction

The ability to describe, categorize and or otherwise fit a plethora of data sets to an appropriate model within its degree of error/uncertainty is a central aspect of machine learning. Regression is a category of machine learning for which the goal is to find a continuous functional relationship between the input data and a reference data set. This scientific report aims to investigate the performance of the linear regression methods Ordinary Least Squares (OLS), Ridge, and Lasso to create best-fitted models for a two-dimensional Franke function and later for a real topographic data set obtained from United States Geological Survey (USGS). Other important aspects of machine learning such as the splitting of test and training data, scaling, regularization parameters, and the bias-variance trade-off of these models shall also be discussed. The latter of which is examined using two common resampling techniques: bootstrap and cross-validation.

2 Method & Theory

This section covers the usage of OLS, Ridge and Lasso regression methods in our investigation to ascertain their compatibility with the Franke function and

the topographic data set.

2.1 Regression Analysis & Methods

The main goal of regression analysis is to construct a function such that it maps a set of input data to continuous output data. We thereby assume that our output data set is described by a continuous function $f(x)$ and an error factor that is normally distributed $\varepsilon \sim N(0, \sigma^2)$.

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

We further assume that we can parameterize our function in terms of a polynomial function of degree $n-1$ with n points.

$$y \to y(x_i) = \tilde{y}_i + \varepsilon_i = \sum_{j=0}^{n-1} \beta_j x_i^j + \varepsilon_i \tag{2}$$

where ε_i is the error in our approximation. Rewriting this general model for the fitting procedure to a linear algebra problem, we obtain this new expression

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{3}$$

where we define our vectors as described in 5.1. We approximate the function $f(x)$ in 1 with

$$\tilde{y} = \boldsymbol{X}\boldsymbol{\beta}. \tag{4}$$

We wish to find the optimal parameters for $\widehat{\boldsymbol{\beta}}$ without solving the above linear algebra problem. To do this we will need to minimize a function that gives a measure of the spread between the values of y_i that represent the exact/output values and the parameterized values $\tilde{y}_i$. This function is none other than a cost function, more specifically the MSE function.

2.1.1 Cost Function

The cost function C, also known as an error or risk or loss function, is used when modelling the data set to estimate the quality of the model $\tilde{\mathbf{y}}$. The standard definition of the cost function C is

$$\chi^2 = \frac{1}{n} \sum_{i=0}^{n-1} \frac{(y_i - \tilde{y}_i)^2}{\sigma_i^2} \tag{5}$$

, but in this investigation, we will be focused on using the MSE function, a variant of the function 5 above, as well as the R^2 score function.

2.1.1.1 MSE

The MSE function is a metric used to measure the expected value of the squared/quadratic error and is defined as

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{6}$$

Where the smaller the result, the better the fit, as it shows that the model has a low expected value of the squared error. For example, the ideal MSE for our model would be 0.00, showing that the model is a perfect fit for the data set.

2.1.1.2 R^2 score

The R^2 score function is used to calculate the coefficient of determination R^2 which signifies how well future samples are likely to be predicted by the model and can be defined as

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{7}$$

Where $\tilde{\boldsymbol{y}}_i$ is the predicted value of the $i-th$ sample and y_i is the corresponding true value. Unlike the MSE function 6, the best possible score is 1.0 but it should be noted that the R^2 score can be negative as the model can also be worse. Furthermore, a model that consistently predicts the expected value of y, thus disregarding its input features, would get a R^2 score of 0.0, indicating that there is **no** correlation between the input features and the output of the constant model.

2.1.2 OLS

OLS is a common method used to approximate the coefficients of a linear regression problem. The goal is to minimize the MSE function such that the spread of the values between y_i and $\tilde{y}_i$ are as small as possible, ideally, 0. Therefore, we set

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2, \tag{8}$$

where after setting in $\tilde{y}_i$ from 4 we get

$$C(\boldsymbol{\beta}) = \frac{1}{n}\left\{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\right\}. \tag{9}$$

Minimizing this equation by taking the first derivative, we end up with the final expression for the optimal coefficient vector

$$\widehat{\boldsymbol{\beta}}_{OLS} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{10}$$

Which is implemented in our code as such:

```python
# Function for performing OLS
def OLSlinreg(X,f):
    A = np.linalg.pinv(X.T.dot(X)) # SVD inverse
    beta = A.dot(X.T).dot(f)
    return beta # Returns optimal beta
```

Figure 1: OLS code snippet

2.1.3 Ridge

Notice that in the previous method, we were dependent on the invertibility of the design matrix $\boldsymbol{X}$, that is to say, if $\boldsymbol{X}$ is a non-invertible matrix, we could not solve for $\widehat{\boldsymbol{\beta}}$ without the use of singular value decomposition (SVD). A simple trick to mediate this issue is to add a constant value λ along the diagonal of the matrix making it invertible.

$$\boldsymbol{X}^T\boldsymbol{X} \rightarrow \boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I} \tag{11}$$

We then end up with a modified cost function for Ridge regression

$$C(\boldsymbol{\beta}) = \{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\} + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta}, \tag{12}$$

where we drop the $1/n$ parameter in front of the standard MSE equation. Note that we are basically adding a regularization parameter that is dependent on λ. In practice, this parameter actually dampens the less important features in the design matrix. The consequent optimal value for the coefficient vector would be

$$\widehat{\boldsymbol{\beta}}_{RIDGE} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{13}$$

We implement this in code as such:

```python
# Function for performing Ridge regression given a lambda
def Ridgelinreg(X,f,lmb):
    I = np.eye(X.shape[1])
    A = np.linalg.pinv(X.T @ X + lmb*I) # SVD inverse
    beta = A @ X.T @ f
    return beta # Returns optimal beta
```

Figure 2: Ridge code snippet

2.1.4 Lasso

Lasso regression is much like Ridge regression, with the difference being that the regularization parameter λ in the cost function is now expressed with a norm-1 of the vector $\boldsymbol{\beta}$. The new cost function for Lasso regression is then

$$C(\boldsymbol{\beta}) = \left\{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\right\} + \lambda||\boldsymbol{\beta}||_1. \tag{14}$$

Contrary to Ridge and OLS, after minimizing this cost function by taking the first derivative of it, we end up with the expression

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} + \lambda sgn(\boldsymbol{\beta}) = 2\boldsymbol{X}^T\boldsymbol{y}, \tag{15}$$

which leads to a non-friendly analytical equation. This equation can however be easily solved numerically, and in this investigation, we will be using the built-in SciKit-Learn function Lasso regression function *Lasso()*.

```
# Functin for performing Lasso regression given a lambda
def Lassolinreg(X,f,lmb):
    RegLasso = linear_model.Lasso(lmb)
    RegLasso.fit(X,f)
    return RegLasso.coef_
```

Figure 3: Lasso code snippet

2.2 The Franke function

The Franke function is a weighted sum of four exponentials and is shown as

$$\begin{aligned} f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}\right) \\ &+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right) \end{aligned} \tag{16}$$

Note that the function in equation 16 above will only be defined for $x, y \in [0, 1]$. We will be using a uniformly distributed set of arrays for the values for x and y. The assumption we made. This investigation will perform regression analysis of the Franke function, attempting to make a polynomial fit with an x and y dependence of the form $[x, y, x^2, y^2, xy, \dots]$ up to a chosen maximum polynomial degree where a polynomial degree ϕ is given by $\phi = a + b$ for $f(x^a, y^b)$.

Our design matrix $\boldsymbol{X}$ for this function up to polynomial degree $= 5$ would then look like this

$$\boldsymbol{X}_i = \begin{bmatrix} 1 & x_i & y_i & x_i^2 & x_iy_i & y_i^2 & x_i^3 & x_i^2y_i & x_iy_i^2 & y_i^3 \\ x_i^4 & x_i^3y_i & x_i^2y_i^2 & x_iy_i^3 & y_i^4 & x_i^5 & x_i^4y_i & x_i^3y_i^2 & x_i^2y_i^3 & x_iy_i^4 \\ y_i^5 & & & & & & & & & \end{bmatrix} \tag{17}$$

for a given i-th row in the design matrix. We implement this in our code as

```python
# Generating Design Matrix
def create_X(x,y,n):
    N = len(x)              # No. of rows in design matrix // corr. to # of inputs to outputs
    l = int((n+1)*(n+2)/2)  # No. of elements in beta // Number of columns in design matrix
    # Frank Function 2D Design Matrix
    X = np.ones((N,l))      # Initialize design matrix X
    for i in range(1,n+1):  # Loop through features 1 to n (skipped 0)
        q = int((i)*(i+1)/2)
        for k in range(i+1):
            X[:,q+k]=(x**(i-k))*y**k  # Calculate the right polynomial term
    return X
```

Figure 4: Design Matrix code snippet

2.3 Pre-processing Data: Scaling

Scaling is a form of pre-processing that is performed on the data via re-scaling in order for our inputs to be more readable by our regression algorithms. It should be noted that re-scaling is somewhat sensitive to outlier values in the data set so care must be taken to avoid errors. We scaled our data using SciKit-Learn's *StandardScaler()* function to set the mean value to zero and variance to one for each feature we analyse, or in other words for every column in the design matrix. *StandardScaler()* does this by subtracting the mean value and dividing it by the standard deviation over the data set, for each feature. This can be shown mathematically in the equation 18 below:

$$x_j^{(i)} \rightarrow \frac{x_j^{(i)} - \overline{x}_j}{\sigma(x_j)} \tag{18}$$

where $\overline{x}_j$ and $\sigma(x_j)$ represent the mean and standard deviation of the feature x_j, in that order. This is implemented in our code below as

```python
# Scaling Data
def scale_data(xtrain,xtest,ytrain,ytest):
    scaler = StandardScaler()
    scaler.fit(xtrain)
    xtrainscaled = scaler.transform(xtrain)
    xtestscaled = scaler.transform(xtest)
    scaler.fit(ytrain)
    ytrainscaled = scaler.transform(ytrain)
    ytestscaled = scaler.transform(ytest)
    return xtrainscaled, xtestscaled, ytrainscaled, ytestscaled
```

Figure 5: Scaling code snippet

2.4 Train-Test-Split

An essential part of analysing data in machine learning is splitting the data set into train and test data. Splitting the data set is important when studying the bias-variance trade-off of the model, as it acts as a measure of the performance of the machine learning algorithms. The training set is made specifically to

train our algorithm on a separate set of data before we use the algorithm on the test data. If we attempt to model the entire data set without training it i.e without splitting, the model will most likely be biased and perform poorly when predicting new or unknown data. Thus, the train-test-split ensures that our model will be able to provide a good prediction for the true model. In our investigation, we opted to use $\frac{4}{5}$ of the data set for our training data and $\frac{1}{5}$ for the test data.

2.5 Resampling Techniques

Resampling techniques are used to assess the quality of the model by repeatedly drawing samples of data from the data set and fitting them, then analyzing the difference between each of the fits. This approach grants us a reference point for the true value of the function while also eliminating the uncertainty obtained when fitting the model only once with the training sample. We will use bootstrap to perform bias-variance trade-off analysis on our model to confirm that we're getting closer to the estimated true value we gleaned from bootstrap. Then, using cross-validation on our model, we will perform a MSE analysis with the MSE gained from cross-validation compared to the MSE we got from bootstrap. The use of resampling techniques should confirm that we're getting closer to the estimated true value of the model.

2.5.1 Bootstrap

Bootstrap is a resampling technique that only works for independent and identically distributed random variables or (*iid*) variables. We assume here that our y_i values are *iid*. While a dependent bootstrap has been developed, we will not use it in this investigation. Earlier in 2.5, we mentioned that resampling techniques help us confirm that we're approaching the estimated true value of the model. This is done by plotting the frequencies of $\widehat{\beta}$, or the optimal beta, that we get from resampling multiple times, on a histogram $p(t)$ and checking for which value of $\widehat{\beta}$ occurs the most, which we call $\widehat{\beta}^*$. Bootstrap is able to do this since the $\widehat{\beta}$ is a random variable, meaning that it has a probability distribution function (pdf) which can be plotted as $p(t)$; as $\widehat{\beta} = \widehat{\beta}(X)$ itself is a function of random variables.

Bootstrap begins with drawing n numbers, with replacement, for the variables in the data-set $x = (x_1, x_2, \ldots, x_n)$, then defining a vector x^* to contain the values drawn from x. Vector x^* is then used to calculate $\widehat{\beta}^*$ by evaluating $\widehat{\beta}$ under the observations x^*. This process is repeated k times until we have sufficient $\widehat{\beta}^*$ values to plot a histogram $p(t)$ of its relative frequency. The histogram is the estimate of the probability distribution $p(t)$ and gives us the estimated true $\widehat{\beta}^*$ of the model.

This is implemented in our code as such:

```python
# Boostrap Resampling method
def bootstraping(xtrain,ytrain):
    N = len(xtrain)
    ind = np.random.randint(0,N,size=(N,))
    xprime = xtrain[ind]
    funcprime = ytrain[ind]
    return xprime,funcprime
```

Figure 6: Bootstrap code snippet

2.5.2 Cross-Validation

We used cross-validation by first shuffling the data set randomly and splitting it into k groups that are mutually exclusive and of approximately equal size. One of the k groups is omitted and designated as the test data, k_{test}. The remaining groups are combined into a single k_{train} group and fit to a model which is then evaluated on the k_{test} group. The result is saved and the process is repeated, starting with the omission of a different k group, until each k group has had a chance to be k_{test}). The compiled results are then used to evaluate the model.

Cross-validation can be used to evaluate prediction error, relative error, MSE, and R^2 score.

2.6 Bias-Variance Trade-off

The bias-variance trade-off refers to the gain of either bias or variance that occurs in a model during testing, and this can be seen when the model under or over-fits the test data. This is often analyzed after the use of resampling techniques to assist with the assessment of the quality of the model. Bias refers to the error rate of the training data, while variance refers to the error rate of the test data. A high bias usually means that the model has insufficient data in the training set, causing it to underestimate what the true model would look like, resulting in under-fitting when modelling the test data. The opposite is true for variance, where the model overestimates what the true model would look like, possibly due to too many data points in the test data or over-tuned hyper-parameters/lambda on the training data (likely in an attempt to lower the bias for the training data), causing the model to over-compensate when modelling the true function thus resulting in over-fitting. An ideal model would be able to walk the line between under and over-fitting, giving an accurate approximation of the true function with the training and test data.

To show that our cost function

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = (Bias\left[\tilde{\boldsymbol{y}}\right])^2 + var\left[\tilde{\boldsymbol{f}}\right] + \sigma^2 \quad (19)$$

can be rewritten in terms of bias, variance, and error variance; we first have to rewrite MSE as

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \frac{1}{n}\sum_i(f_i-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i(\tilde{y}_i-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2. \quad (20)$$

The equation 20 above can be simplified further since the variance of y and ϵ are both equal to σ^2 as shown in 5.2, the mean value of $\epsilon = 0$, and that both the function f and $\tilde{y}$ are non-stochastic variables, we get

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(\boldsymbol{f}+\boldsymbol{\epsilon}-\tilde{\boldsymbol{y}})^2\right]. \quad (21)$$

Further adding and subtracting $\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]$ we get

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(\boldsymbol{f}+\boldsymbol{\epsilon}-\tilde{\boldsymbol{y}}+\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right]. \quad (22)$$

Expanding the term above, we get

$$\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] = (\boldsymbol{y}-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i(\tilde{\boldsymbol{y}}_{\boldsymbol{i}}-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + 0^2. \quad (23)$$

or

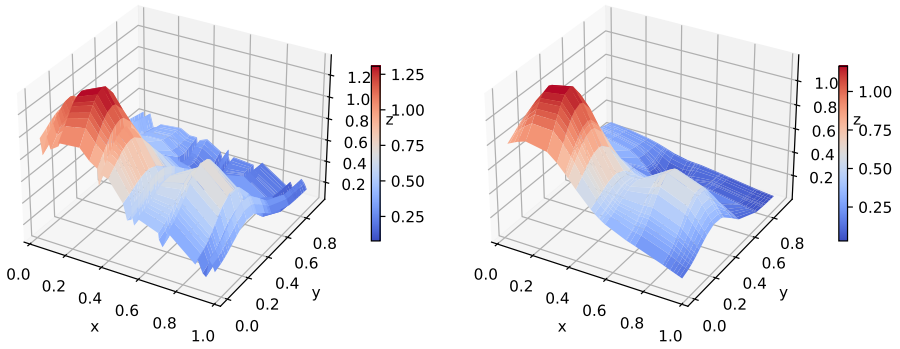
$$MSE = (Bias\left[\tilde{\boldsymbol{y}}\right])^2 + var\left[\tilde{\boldsymbol{f}}\right] + \sigma^2 = (\boldsymbol{y}-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_i(\tilde{\boldsymbol{y}}_{\boldsymbol{i}}-\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + 0^2. \quad (24)$$

3 Results & Discussion

This section contains the results obtained from the use of OLS, Ridge, and Lasso regression methods in Section 2 and a discussion of their compatibility with regard to the Franke function and the terrain data set via analysis and comparison of the bias-variance trade-off values obtained from using bootstrap and the MSE values obtained from using both bootstrap and cross-validation.

3.1 Franke function

Firstly in the Franke function, we will explore the addition of a normally distributed stochastic noise $\boldsymbol{\varepsilon} \sim N(0,1)$.



(a) Franke function with noise

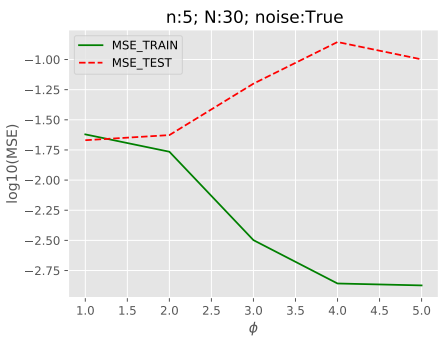
(b) Franke function without noise

Figure 7: Franke function

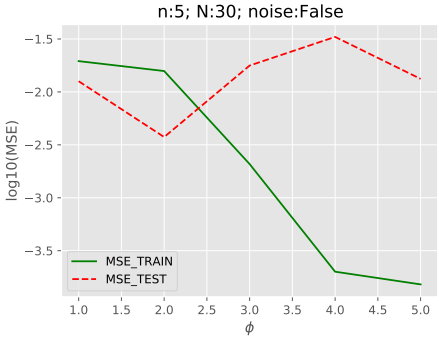
In Figure 7(a) we can clearly see that the Franke function looks slightly altered when noise is introduced, giving us an idea of what noise can do when analyzing the MSE and R^2 score later after performing linear regression on the function. Here we only used $N = 30$ for the number of data points, this also has an effect on the smoothness and resolution of the function itself without noise in Figure 7(a), but this was done to reduce computational time.

3.1.1 OLS regression

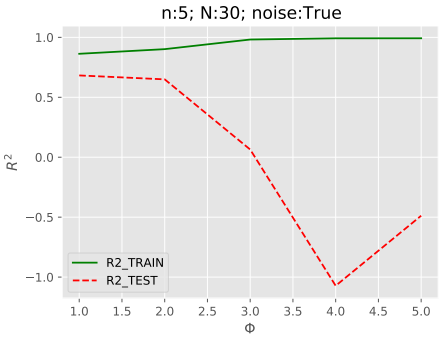
For polynomial degrees up to $n = 5$, we get the following graphs that show the MSE and R^2 score after performing OLS regression on the Franke function.



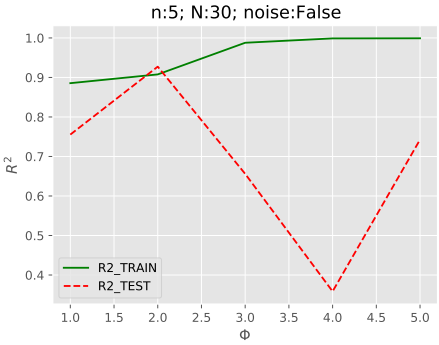
(a) MSE with noise



(b) MSE without noise



(c) R^2 score with noise

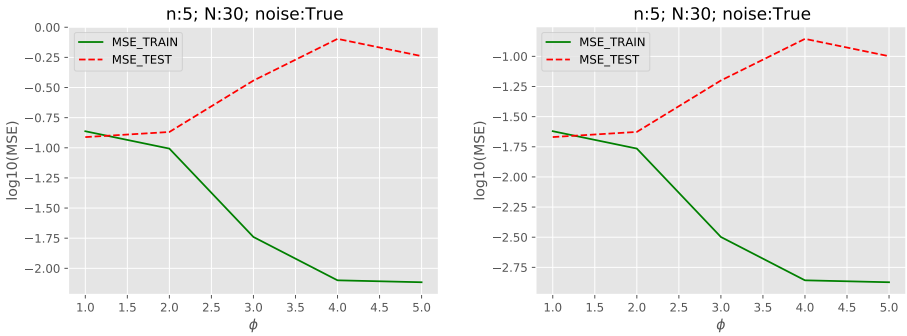


(d) R^2 score without noise

Figure 8: Plots for MSE and R^2 score with and without noise

From Figure 8 we concluded that the inclusion of noise resulted a higher MSE and a lower R^2 score, which was to be expected, due to the randomness introduced to the function by the addition of noise.

3.1.1.1 Scaling



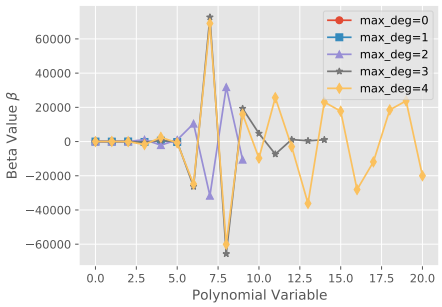
(a) Scaled Data (b) Not Scaled Data

Figure 9: Functions with and without scaling

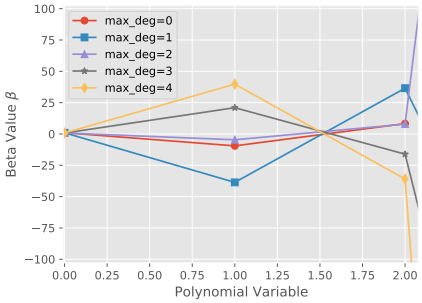
Before we delve further into the discussion of the results, we should consider whether it is necessary to scale the data. The purpose of scaling data is primarily to avoid major differences in the variance of data between feature columns in the design matrix for the data. This is most applicable to real-life regression, such as the features of age and area when considering housing data. Considering that the age of the tenants residing in a house would realistically range between 0-100 while the area of a house could be many times larger, this demonstrates the need for scaling due to the possibility of a large variance of the values between features. Above, in Figure 9, the difference between the two functions is shown: Figure 9(a) with scaling and Figure 9(b) without. The plots are both similar, leading us to conclude that the difference was negligible enough to not scale the data in the later plots for the Franke function because in our case, the Franke function's x and y variable is well defined as $x, y \in [0, 1]$.

3.1.1.2 Parameters of β as the order of polynomial increase

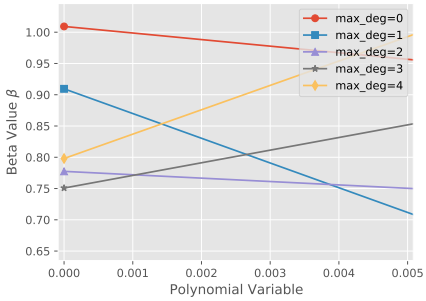
This section discusses the plots that measure the parameters of β as we increase the order of the polynomial, up to $n = 5$.



(a) Zoomed-out



(b) 0th, 1st, and 2nd polynomial coefficient



(c) Close-up of 0-th polynomial coefficient

Figure 10: Plot with β values as a function of polynomial degree for degree up to 5 for 2D-function, with close ups

Note that for $n = 5$ on the Franke function, there are 21 coefficients for all combinations of (x, y) plus the constant (0-th) coefficient. The plots in Figure 10 show us that the increase in the degree order of the polynomial for a model causes the β parameters to become more varied. It should be noted that this could be a sign of over-fitting and instability in the model where the model specifically favours odd or even polynomial fits.

3.1.2 OLS: Bootstrap & Bias-Variance Analysis

Before we discuss bootstrap and bias-variance analysis we present a reference plot that displays the test and training MSEs of the function, following (Hastie, Tibshirani, & Friedman, 2009):

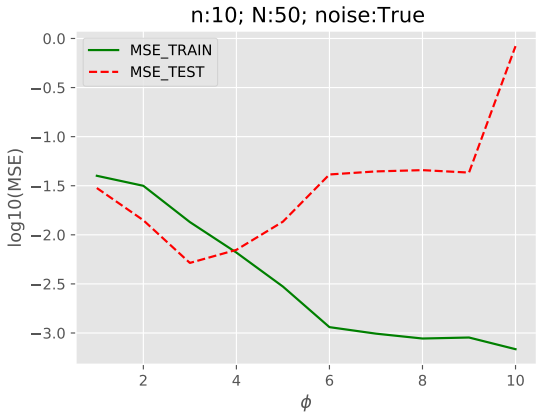
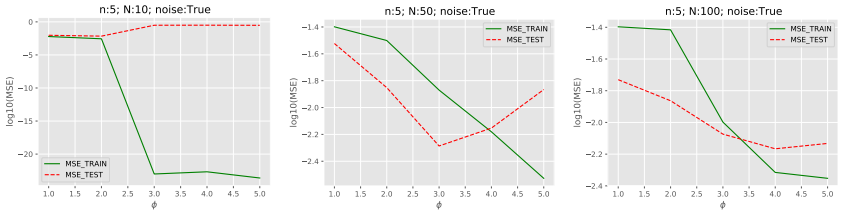


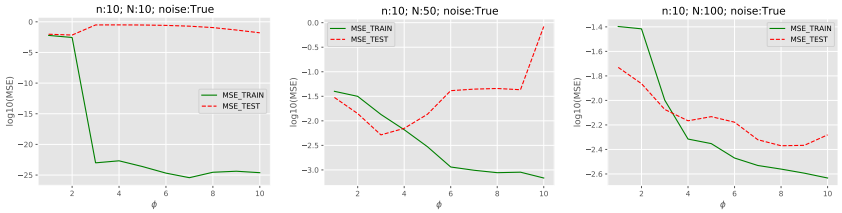
Figure 11: Training and test MSE comparison for OLS on the Franke function

The MSE plot above was done for $N = 50$ datapoints for polynomial degrees up to $\phi = 10$. As expected, the MSE for our training model decreases as the complexity of the polynomial increases while the test MSE decreases up until $\phi = 3$ where it starts to deviate and increase as complexity increases. This is because the higher complexity of the model, the better the fit will be for our model up to the point where it's fitting the training set too well (over-fitting) such that it can no longer be a reliable source of prediction for the test data. This is apparent in our MSE for the test model, where at a certain point, specifically at $\phi = 3$, the MSE is lowest but increases thereafter at higher degrees of complexity.

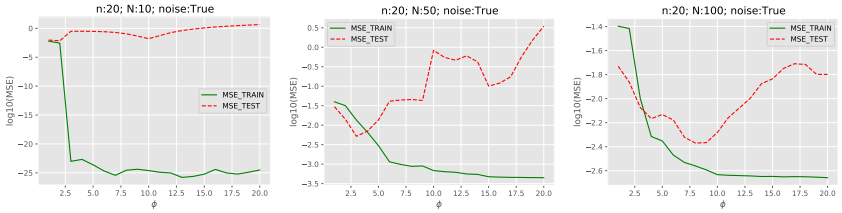
Now, onto the bias-variance trade-off analysis on our test data for the OLS regression on the Franke function.



(a) $n = 5$, $N = 10$ (b) $n = 5$, $N = 50$ (c) $n = 5$, $N = 100$



(d) $n = 10$, $N = 10$ (e) $n = 10$, $N = 50$ (f) $n = 10$, $N = 100$



(g) $n = 20$, $N = 10$ (h) $n = 20$, $N = 50$ (i) $n = 20$, $N = 100$

Figure 12: Plots for the bias-variance analysis of the OLS regression on the Franke function, with noise, varying complexity n, and data points N

Performing a bias-variance analysis of the Franke function by studying the MSE value as a function of the complexity of our model with varying complexity ϕ and data points N in Figure 12 above, we concluded that with an increase in data points N, an increase in polynomials ϕ was also needed if we were to observe over-fitting as shown in Figure 12i.

We chose to run our simulation with $N = 50$ data points for all plots after this point, because we observed that it had a good balance of stability, computational speed, and clear signs of over-fitting.

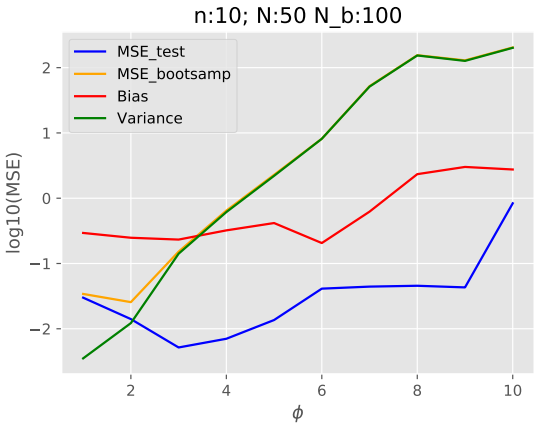
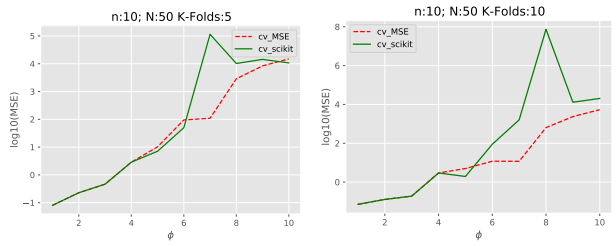


Figure 13: Error-Bias-Variance plot

In the Figure 13 above, we see the relationship between the error (MSE), bias, and variance shown in Eqn. 24. We expected the variance of the model go up as the complexity of the model went up, as we saw in Figure 10, thus being the main contributor to error in models with higher complexity. We also expected bias to be the main contributor to the error at lower complexity models and have a decreasing trend as we go up to higher complexity. The complexity ϕ where these two lines meet (bias and variance) is where our lowest MSE with the optimal beta is supposed to be. We however have been unable to properly plot bias in our code, and this has to be taken into consideration. While we unfortunately cannot give a proper visualization of this aforementioned bias-variance trade-off, Hastie et.al. has a better and smoother interpretation of this relationship (Hastie et al., 2009).

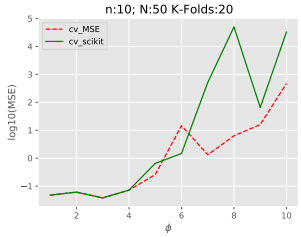
We also plotted the MSE from OLS model as a function of complexity (polynomial degree ϕ) and the MSE score from performing bootstrap resampling with 100 sampling iterations with replacement. It seems as though the OLS model without resampling performed better, but this is a less accurate model and does not quantify for the uncertainty in the sample, which is the exact reason for why we do resampling.

3.1.2.1 OLS: Cross-Validation



(a) Cross-validation with 5 k-folds

(b) Cross-validation with 10 k-folds



(c) Cross-validation with 20 k-folds

Figure 14: Plot of MSE function after cross-validation with 5, 10, and 20 k-folds

Another resampling method is cross-validation, which is arguably better than bootstrap. The plots above are evaluations of the MSE for the OLS model of the Franke function as a function of complexity up to $\phi = 10$ with cross-validation resampling with 5, 10, and 20 k-folds.

From Figure 14 above, we can deduce that Figure 14(c) performs the best, showing that it has the lowest MSE at $\phi = 3$. We also see that our cross-validation code follows SciKit-Learn's code rather well up until $\phi = 5$ where it starts to become unstable, varying noticeably from SciKit-Learn's MSE at higher complexities. We suspect this has to do with their function being much more efficient and optimized at calculating the MSE.

Comparing the lowest MSE obtained from our own cross-validation code with 20 k-folds to the MSE obtained from our bootstrap code with 100 samples with replacement, we get:

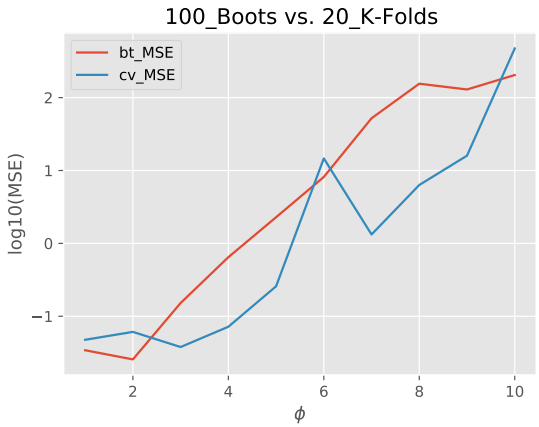


Figure 15: Comparing cross-validation MSE to bootstrap MSE

Here we can see that our cross-validation code performs better overall than our bootstrap approach on our OLS model as it has a lower average MSE as ϕ increases. However, bootstrap does give us a lower minimum MSE at $\phi = 2$ than the minimum MSE obtained through cross-validation at $\phi = 3$. A possible reason for this could be because bootstrap samples the data set with replacement, which could have skewed the MSE in its favour.

3.1.3 Ridge

3.1.4 Ridge: λ dependency

Comparing the MSE values of the test and training data with our Ridge model, firstly with a low arbitrary value for $\lambda = 7.54 \cdot 10^{-15}$, without resampling, as in (Hastie et al., 2009) to see what we're working with:

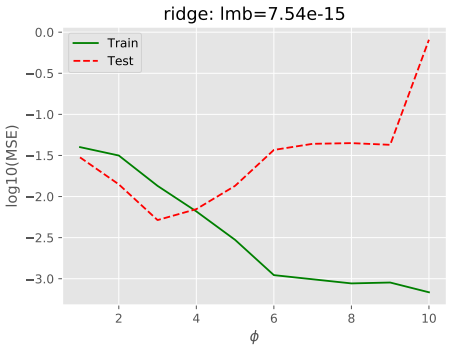


Figure 16: Training and test MSE comparison for Ridge with $\lambda = 7.54 \cdot 10^{-15}$, without resampling

In Figure 16, we see the general trend of the training data getting better MSE scores as we go up in complexity, while the training data set begins to show signs of over-fitting after $\phi = 4$.

To visualize the dependence of our model on the regularization parameter λ more effectively we decided to use a heat map with λ on the y-axis and ϕ on the x-axis with MSE as depth, or on the z-axis.

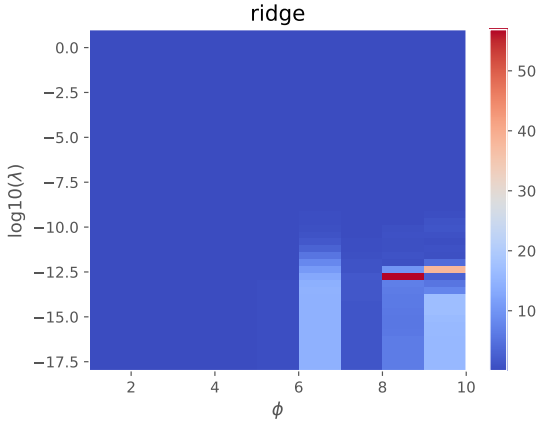


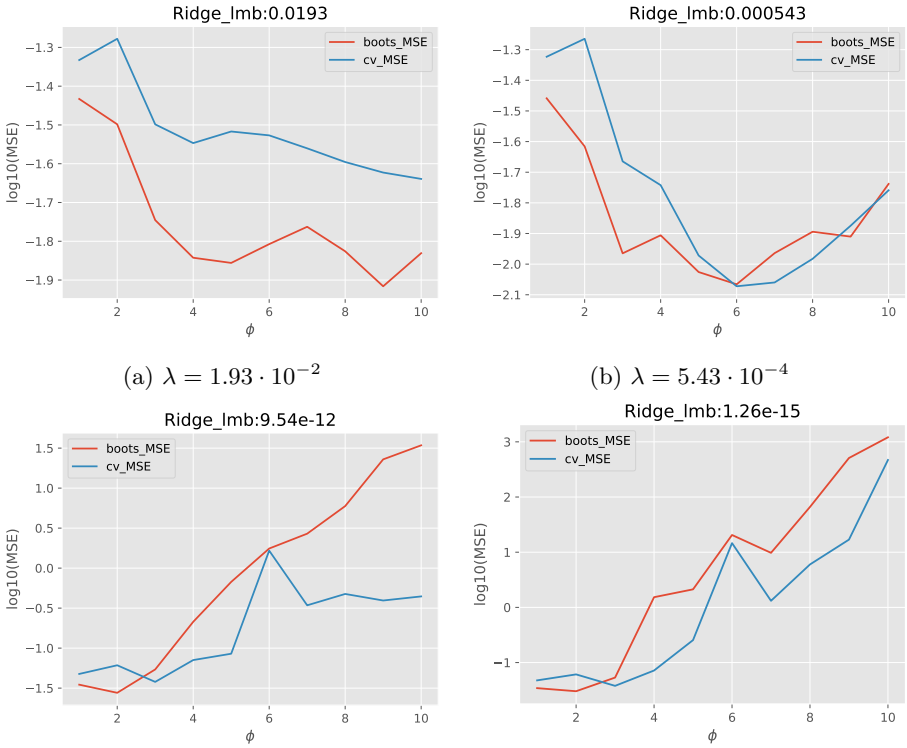
Figure 17: Visualization to find the best value for λ for our Ridge model on the Franke function

The heat map in Figure 17 shows us approximately where we could find the best λ for each complexity, giving us a better idea of what values of λ to use for our Ridge model when comparing the MSE scores for bootstrap and cross-validation in Section 3.1.4.1 where we discuss the overall dependence our Ridge model has

on λ, as well as bootstrap vs. cross-validation as resampling methods.

3.1.4.1 Ridge: Bootstrap and Cross-validation analysis

Comparing the MSE scores for our Ridge model as a function of model complexity up to $\phi = 10$, with $N = 50$ data points, for bootstrap with 100 samples and resampling against cross-validation with 20 k-folds, with $\lambda = 1.93 \cdot 10^{-2}, 5.43 \cdot 10^{-4}, 9.54 \cdot 10^{-12}, 1.26 \cdot 10^{-15}$ (lambda values obtained from the heatmap in Figure 17) we get the following plots:



(a) $\lambda = 1.93 \cdot 10^{-2}$

(b) $\lambda = 5.43 \cdot 10^{-4}$

(c) $\lambda = 9.54 \cdot 10^{-12}$

(d) $\lambda = 1.26 \cdot 10^{-15}$

Figure 18: MSE scores for both bootstrap and cross-validation resampling with varied λ values to study the dependency on the regularization parameter on the Ridge model

The plots in Figure 18, showcase the effect of manipulating the hyper- or regularization parameter λ on the model. It can be seen that an increase of λ increases the bias as higher λ reduces the differences between values in the model, resulting in a lower variance. Note that manipulating λ too much could lead to both under and over-fitting, depending on how much it was manipulated. Here, we decided to continue using the value of $\lambda = 5.43 \cdot 10^{-4}$ as we found it to be the

best estimate for an optimal λ for the Ridge model based on the heat map in Figure 17. When $\lambda = 5.43 \cdot 10^{-4}$, our lowest MSE score was 0.00847, at $\phi = 6$ before the the model becomes over-fitted.

Further note that our bootstrap code performed slightly better than our cross-validation code up until the point of over-fitting, providing a lower MSE score as seen in Figure 18(a) where our bootstrap plot becomes over-fit at $\phi = 5$, while the cross-validation plot becomes over-fit at $\phi = 4$. However, we chose to stick with cross-validation later on when we compare the different models because we believe it to be a stronger resampling technique as opposed to bootstrap, due to the fact that bootstrap resampling is reliant on the central limit theorem. We also had to consider the computational time when we running these simulations. Whether we were able to have reached that limit or an acceptable value of it by only 100 sampling iterations is a subject for discussion.

3.1.5 Lasso

3.1.6 Lasso: λ dependency

As before in Ridge, we compare the MSE scores of the test and training data with our Lasso model, firstly with the same low arbitrary value for $\lambda = 7.54 \cdot 10^{-15}$, without resampling, as in (Hastie et al., 2009) to see what we're working with.

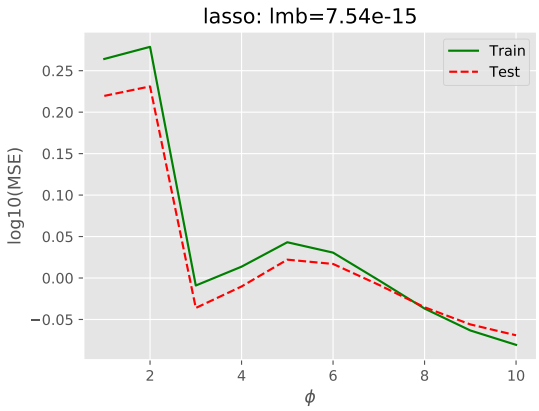


Figure 19: Training and test MSE comparison for Ridge with $\lambda = 7.54 \cdot 10^{-15}$, without resampling

We noticed that the MSE scores for both the train and test data set had the same trend, i.e. the training data set also showed over-fitting and or instability, much in the same form as the training set. We suspect this had to do with either the λ we chose, or the nature of the Lasso model, effectively diminishing less important features in the design matrix as opposed to dampening less important features in Ridge model.

As we did in to find the best λ for our Ridge model, we used a heat map with λ on the y-axis and ϕ on the x-axis with MSE as depth, or on the z-axis to give us a better idea of what values of λ to use for our Lasso model when comparing the MSE scores for bootstrap and cross-validation in Section 3.1.6.1.

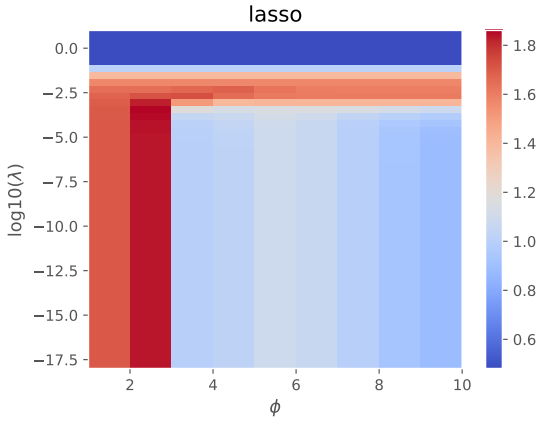


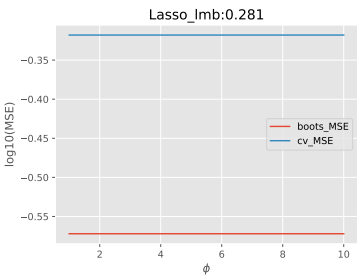
Figure 20: Visualization to find the best value for λ for our Lasso model on the Franke function

The heatmap in Figure 20 shows us approximately where we could find the best λ for each complexity. This turned out to be $\lambda = 4.29 \cdot 10^{-7}$.

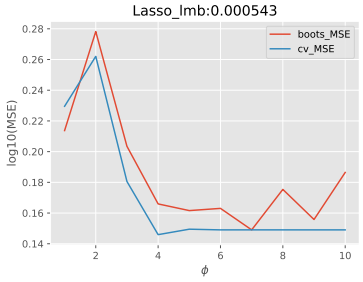
3.1.6.1 Lasso: Bootstrap and Cross-validation analysis

Comparing the results for the MSE scores via bootstrap and cross-validation with different λ values in our Lasso model we can see that the MSE of the model reaches a constant when using bigger λ values. The plots below, in Figure 21, show generally higher MSE scores as opposed to the Ridge model. We were still able to observe the trend where increasing λ increases the bias, but this time we believe the effect is somewhat faster or more immediate, due to the fact that the Lasso model has the property of diminishing less important features in the design matrix, thus leaving not much variance in the model at all.

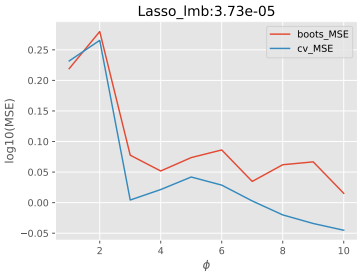
We noticed this time cross-validation showed better MSE scores than bootstrap, which was what we were expecting since bootstrap samples with replacement, and that 100 sampling iteration might not have been enough, as is shown in the slight difference results in bootstrap vs cross-validation in the Ridge model (see Figure 18).



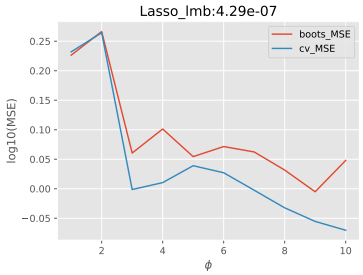
(a) $\lambda = 2.81 \cdot 10^{-1}$



(b) $\lambda = 5.43 \cdot 10^{-4}$



(c) $\lambda = 3.73 \cdot 10^{-5}$



(d) $\lambda = 4.29 \cdot 10^{-7}$

Figure 21: MSE scores for both bootstrap and cross-validation resampling with varied λ values to study the dependency on the regularization parameter on the Lasso model

3.1.7 Best fit model & Evaluation

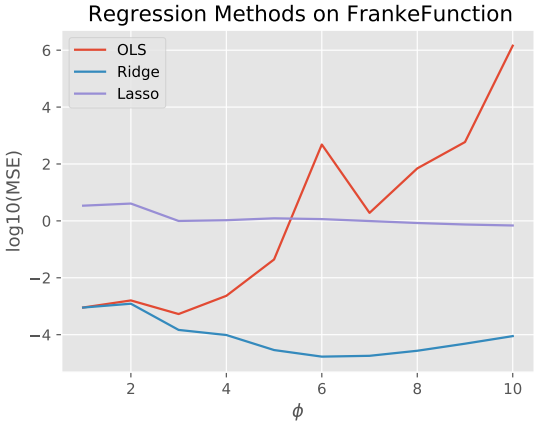


Figure 22: Comparison of the 3 regression methods on the Franke function using cross-validation resampling method with 20 k-folds

Before moving onto the real terrain data, we had to come to a decision for which regression method performed the best on the Franke function.

Comparing all the regression methods, we see that the Ridge regression model performed the best, as it had the lowest MSE out of the three regression methods as shown above in 22, also in tabular form in Table 1 below.

Table 1: MSE, λ, n degree comparison

	OLS	Ridge	Lasso
n degree	3	6	3
λ	N/A	$5.43 \cdot 10^{-4}$	$4.29 \cdot 10^{-7}$
MSE	$3.78 \cdot 10^{-2}$	$8.47 \cdot 10^{-3}$	$9.97 \cdot 10^{-1}$

Seeing all the results together, we suspect the reason to the Lasso model's poorer performance was due to its nature of diminishing less important features, resulting in a poorer MSE score. The result for Ridge made sense to us, as the use of Ridge regression dampens the less important features in the design matrix, improving the MSE score, causing it to perform better than the OLS model. A sort of middle ground between Lasso and OLS.

The creation of the plots for what we needed in the Franke function ran relatively fast, the results of the plots were roughly what we expected and the comparison of our code to sk-learns code for cross-validation shows us that we are on the right path.

3.2 Topographical Data

We chose to work with a terrain image of somewhere in Norway. The original terrain data set was too large, with 6,485,401 pixels/data points to be exact and is shown by the plot below.

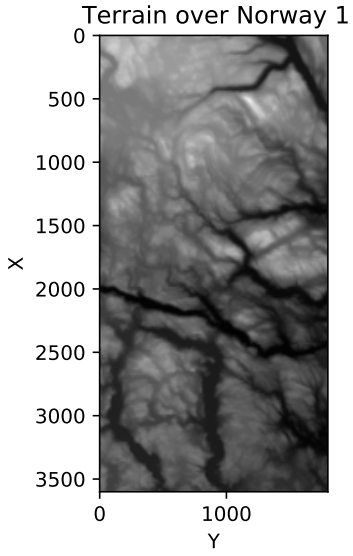
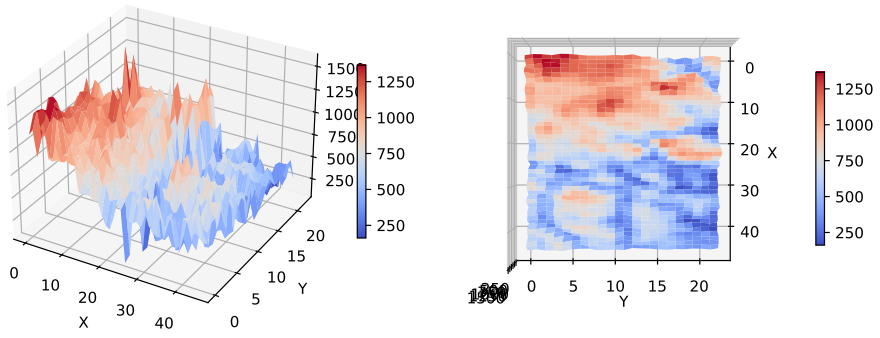


Figure 23: Raw Terrain Data

Due to a large number of data points, several pre-processing techniques had to be used. We decided to scale it by a factor of 10^{-4} to get a total of 1058 data points, resulting in faster computational time at the cost of poorer resolution on the image as seen in the plots of the scaled terrain data below in Figure 24.



(a) Scaled terrain data

(b) Top view of scaled terrain data

Figure 24: Scaled Terrain Data

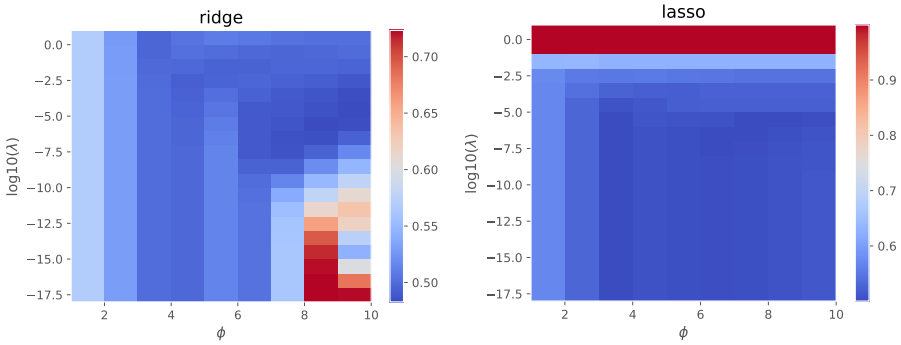
At this point we will be repeating most of the operations we performed on the Franke function, now with the terrain data set. As such, we will be using cross-validation with 20 k-folds as our resampling method for all three regression methods: OLS, Ridge, and Lasso, for polynomial degree up to $\phi = 10$.

It is important to note that since now we are dealing with real data, where the values of the data within the design matrix $\boldsymbol{X}$ and original function itself $\boldsymbol{y}$ can vary a lot. At this point, we have applied scaling on both $\boldsymbol{X}$ and $\boldsymbol{y}$.

3.2.1 λ for Ridge and for Lasso

For Ridge and Lasso regression, we first must find a suitable λ value to be used. As in Figure 17 and Figure 20, we used a heat map to approximate where we could find the optimal λ for our Ridge and Lasso model.

As we see in Figure 25, we again use a heat map to approximately find the optimal λ for the terrain data for Ridge model to be $\lambda = 2.07 \cdot 10^{-8}$, while for Lasso model to be $\lambda = 2.34 \cdot 10^{-4}$.



(a) Optimal λ for Ridge regression (b) Optimal λ for Lasso regression

Figure 25: Visualization to find the optimal value for λ for Ridge and for Lasso model on the terrain data

3.2.2 Best fit model & Evaluation

Comparing all the regression methods, we saw that both OLS and Ridge model performed equally well, having the lowest MSE value (before over-fitting) at both at polynomial degree $n = 4$, as shown in Figure 26 and in tabular form in Table 2.

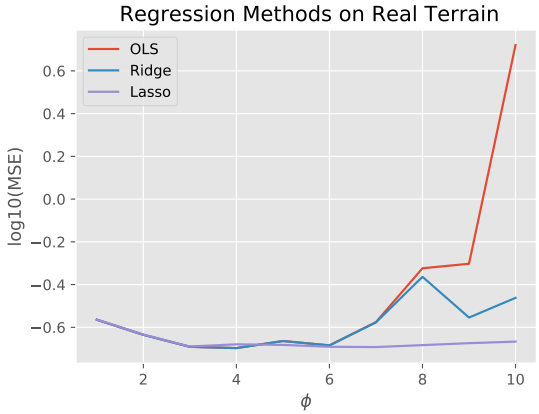


Figure 26: Comparison of the 3 regression methods on the terrain data using cross-validation resampling method with 20 k-folds

Table 2: MSE, λ, n degree comparison

	OLS	Ridge	Lasso
n degree	4	4	3
λ	N/A	$2.34 \cdot 10^{-4}$	$2.07 \cdot 10^{-8}$
MSE	$4.98 \cdot 10^{-1}$	$4.98 \cdot 10^{-1}$	$5.02 \cdot 10^{-1}$

From the Table 2 we notice that firstly, the error is at the same order of magnitude, although our Lasso performed slightly worse than both OLS and Ridge. These results were noticeably different from the results we obtained from our Franke function in Table 1. This might have been due to several reasons, the main one being that we're now dealing with real terrain data that was first and foremost scaled down in resolution by a factor of 10^{-4}, so the error from this vs. the error from an already known function (Franke function with stochastic noise) is expected to be greater.

3.3 Critique

So, whether this is a good enough model of the terrain data is up for discussion. Although we achieved a relatively low MSE score of around $5 \cdot 10^{-1}$, this is for a terrain image with poor resolution. If one were to use it in real life, it might not pass as a very reliable model of a given terrain. We constrained ourselves to use relatively low number of data points, as if we had used a higher number of data points (especially for the Franke function) it would require a higher complexity ϕ before we could observe over-fitting as well as longer computational time.

Overall, we definitely feel like we've gained a deeper insight into the workings of linear regression, bias-variance trade-off, resampling techniques, hyperparameter dependencies, and working with real data. One thing to note regarding the coding for the bias-variance trade-off analysis, the code example for bias given in the lectures (which used an arbitrary 1D example) seem to be a bit off because for some reason when implemented in our project it seemed like it did not give the right form for the bias.

So a definite improvement for this project, would be to accurately ascertain the bias so we can better visualize the bias-variance trade-off. When investigating which lambda to use, it was quite difficult to base the decision solely on where the lowest MSE score was on the heat map plot because the model could easily have just become unstable. So although they did give an approximate image of where we could begin our search, we ended up having to plot MSE vs. ϕ with promising values of lambda. Further improvement on our project could be done by saving the value of the optimal beta $\widehat{\beta}$ for a regression method to then plot the model function for both Franke function and real terrain data could be interesting.

Perhaps something to note about the the task could be its structure, as it felt like we were going in circles sometimes, and losing sight of the overall aim of

the task. However, it was nice to see our report take form and improve after each session, and we believe the report as it is to be a good demonstration of what we learnt.

4 Conclusion

Throughout this investigation, we delved deep into central aspects of machine learning such as test-train splitting, scaling, MSE score as a function of complexity of the model, resampling, regularization parameter tuning, and overall bias-variance trade-off. All of which are equally important to fully understand the scope of what we aim to do: create a best fit model for a given data set.

Using cross-validation with 20 k-folds to assess which model fits the data best, we determined for the Franke function that the Ridge model at polynomial degree $\phi = 6$ using $\lambda = 5.43 \cdot 10^{-4}$ performed the best with an MSE score of $8.47 \cdot 10^{-3}$. Whilst for the chosen terrain data, both OLS model and Ridge model with $\lambda = 2.34 \cdot 10^{-4}$ at polynomial degree $\phi = 4$ performed equally well with an MSE score of $4.98 \cdot 10^{-1}$.

Whether or not this was a good enough model was discussed, and we conclude with an open ended interpretation for the readers, where we believe that perhaps an MSE score of $4.98 \cdot 10^{-1}$ on a terrain model is not acceptable for real life application, especially if it was modelled on a low resolution image of the terrain.

Nevertheless, the investigation has given us a deeper insight into the importance of selecting the proper tools; OLS, Ridge, and or Lasso, with the right resampling method; bootstrap and cross-validation, with certain optimal parameters to successfully create a best fit model for both the two-dimensional Franke function and for real topographical data set.

5 Appendix

This section contains a number of explanations and derivations for relevant equations that were employed through the course of our investigation.

5.1 Vector Definition

We define our vectors for the general equation for linear regression where the output data is defined as

$$\boldsymbol{y} = [y_0, y_1, y_2, \dots, y_{n-1}]^T,$$

the polynomial coefficients as

$$\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_{n-1}]^T,$$

the noise factor as

$$\boldsymbol{\epsilon} = [\epsilon_0, \epsilon_1, \epsilon_2, \dots, \epsilon_{n-1}]^T,$$

and the design matrix $\boldsymbol{X}$ containing our input data as

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_0^1 & x_0^2 & \dots & \dots & x_0^{n-1} \\ 1 & x_1^1 & x_1^2 & \dots & \dots & x_1^{n-1} \\ 1 & x_2^1 & x_2^2 & \dots & \dots & x_2^{n-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1}^1 & x_{n-1}^2 & \dots & \dots & x_{n-1}^{n-1} \end{bmatrix} \tag{25}$$

5.2 Mean Values and Variances in Linear Regression

Assuming that our data can be described by a function $f(\mathbf{x})$ and a normal distributed error $\varepsilon \sim \mathcal{N}(0, \sigma^2)$

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

we can approximate this function $f(\mathbf{x})$ with our model $\tilde{\mathbf{y}}$ from the solution of OLS where we minimized $(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2$ with

$$\tilde{\mathbf{y}} = \boldsymbol{X}\boldsymbol{\beta},$$

giving rise to the design matrix $\boldsymbol{X}$.

Writing out $\mathbf{y}$ for a given element i, we get

$$y_i = \sum_{j=0}^{d-1} \boldsymbol{X}_{ij}\boldsymbol{\beta}_j + \varepsilon_i = \mathbf{X}_{i,*}\beta + \varepsilon_i$$

we rewrite the summation of the multipilication of the j-th row of $\boldsymbol{X}$ and j-th value of $\boldsymbol{\beta}$ into a shorthand notation

$$\mathbf{X}_{i,*}\beta + \varepsilon_i.$$

5.2.1 Mean Value of y_i

The mean value of y_i can then be expressed as

$$\mathbb{E}(y_i) = \mathbb{E}(\mathbf{X}_{i,*}\beta + \varepsilon_i) \tag{26}$$

The equation 26 above can be rewritten due to the mean value's property of linearity, giving us the new equation for the mean

$$\mathbb{E}(y_i) = \mathbb{E}(\mathbf{X}_{i,*}\beta) + \mathbb{E}(\varepsilon_i). \tag{27}$$

The first term in equation 27 is purely deterministic, therefore the mean value is just the value/s itself. Meanwhile, as per definition, the second term is 0, because we have made the assumption at the start that our data can be described with a function $f(\boldsymbol{x})$ and a normal distributed error $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2)$.

Thus we end up with an expression for the mean value

$$\mathbb{E}(y_i) = \mathbf{X}_{i,*}\beta. \tag{28}$$

5.2.2 Variance of y_i

The variance of y_i can be expressed as

$$\mathrm{Var}(y_i) = \mathbb{E}([y_i - \mathbb{E}(y_i)]^2)$$

We expand the expression to get

$$\mathrm{Var}(y_i) = \mathbb{E}(y_i^2 - 2y_i\mathbb{E}(y_i) + \mathbb{E}(y_i)^2),$$

and by the property of linearity, we are able to further rewrite this as

$$= \mathbb{E}(y_i^2) + \mathbb{E}(-2y_i\mathbb{E}(y_i)) + \mathbb{E}(y_i)^2$$

$$= \mathbb{E}(y_i^2) - 2\mathbb{E}(y_i)\mathbb{E}(y_i) + \mathbb{E}(y_i)^2$$

$$= \mathbb{E}(y_i^2) - \mathbb{E}(y_i)^2$$

$$= \mathbb{E}((\mathbf{X}_{i,*}\beta + \varepsilon_i)^2) - (\mathbf{X}_{i,*}\beta)^2$$

$$= \mathbb{E}((\mathbf{X}_{i,*}\beta)^2 - 2\mathbf{X}_{i,*}\beta\epsilon_i + \epsilon_i^2) - (\mathbf{X}_{i,*}\beta)^2$$

$$= \mathbb{E}((\mathbf{X}_{i,*}\beta)^2) + \mathbb{E}(-2\mathbf{X}_{i,*}\beta\epsilon_i) + \mathbb{E}(\epsilon_i^2) - (\mathbf{X}_{i,*}\beta)^2$$

$$= (\mathbf{X}_{i,*}\beta)^2 - 2\mathbb{E}(\mathbf{X}_{i,*}\beta\epsilon_i) + \mathbb{E}(\epsilon_i^2) - (\mathbf{X}_{i,*}\beta)^2$$

From this we can ascertain that y_i is approximately $N_X i\beta\sigma^2$

5.3 Expectation value of β

Show that $\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$

Using OLS expression for the optimal parameters $\widehat{\beta}$, first we know that the expectation value of $\hat{\beta}$ is the same as

$$\mathbb{E}[(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}].$$

Which we can rewrite as

$$\mathbb{E}[(\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}]. = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{Y}]$$

Since we know that $\mathbb{E}[\mathbf{Y}] = \mathrm{X}\boldsymbol{\beta}$, we can plug it into the last equation to make

$$(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}[\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

Finally, since

$$(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}$$

We can conclude that

$$\mathbb{E}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}$$

Meaning that the estimator of the regression parameters is unbiased.

5.4 Variance of $\hat{\beta}$

The variance of $\hat{\beta}$ is

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

Which can be expanded to

$$\mathbb{E}\{[\boldsymbol{\beta} - \mathbb{E}(\boldsymbol{\beta})][\boldsymbol{\beta} - \mathbb{E}(\boldsymbol{\beta})]^T\} = \mathbb{E}\{[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y} - \boldsymbol{\beta}]\,[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y} - \boldsymbol{\beta}]^T\}.$$

Which can be rearranged into

$$\mathbb{E}\{[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y}]\,[(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y}]^T\} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T.$$

Expanding again we get

$$\mathbb{E}\{(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\mathbf{Y}\,\mathbf{Y}^T\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1}\} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T.$$

Rearranging so that we can plug in the property $\mathbb{E}(\mathbf{Y}\mathbf{Y}^T) = \mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T + \sigma^2\,\mathbf{I}_{nn}$ we get

$$(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbb{E}\{\mathbf{Y}\,\mathbf{Y}^T\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T.$$

After plugging in $\mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T + \sigma^2\,\mathbf{I}_{nn}$

$$(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\{\mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T + \sigma^2\}\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T.$$

Which expands into

$$(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbf{X}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{X}^T\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} + \ \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbf{X}\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^T.$$

After simplifying the above equation we get

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \boldsymbol{\beta}\,\boldsymbol{\beta}^T + \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1} - \boldsymbol{\beta}\,\boldsymbol{\beta}^T$$

to finally arrive at

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2\,(\mathbf{X}^T\mathbf{X})^{-1}$$

References

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning* (2nd ed.). Springer. Retrieved from /bib/hastie/Hastie2001/ESLII_print10.pdf,http://www-stat.stanford.edu/~tibs/ElemStatLearn/,/bib/hastie/Hastie2001/weatherwax_epstein_hastie_solutions_manual.pdf