University of Oslo

Project 1

FYS-STK3155

15.January 2024

Abstract

The purpose of this this project is to study the applications of linear regression methods such as Ridge, Lasso and OLS (Ordinary Least Squares) to different datasets and their implications. The bootstrap and cross-validation resampling methods will also be utilized to give further insight into the model's parameters. This is accomplished by estimating the error from the real data as well as looking at the bias-variance tradeoff. Starting off, the methods and analyses are applied to a self-produced dataset by using the Franke function. Thereafter the same models will be applied to a real dataset of the digital terrain in Stavanger, Norway. Lastly, the models are compared and evaluated to see in which cases and regions they fit best, all to give an understanding of the characteristics of OLS, Ridge and Lasso regression.

Introduction

Regression analysis is an important tool which is used to study the influence of independent variables on a dependent variable. This can further be used to make a prediction on the behavior of future data and which variables impact a dataset the most. A well performed regression analysis can give the user good insight into a dataset and has many real-world applications. Suppose a company would like to estimate what effect certain variables would have on their sales, then a regression analysis would give a good picture. For this to be efficient the model created from the extracted data needs to have small deviation from the sample, but still give a good prediction on future data. Therefore, it is important to get a good understanding of different regression methods and what they entail to make sure the created model is optimal for the problem. Also creates good basis for understanding more complicated regression methods.

There are several ways to assess the quality of a model, these will be explored in this project. To get a feel for the methods the regression analysis will first be conducted on an artificial dataset created with the Franke function before going to the real dataset. First, the OLS method will be used to fit the function and an estimate of the mean-squared-error and the R2-

score as a function of polynomial degree will be presented. Then the bootstrap and cross-validation resampling techniques will be applied to graph the bias-variance and error as a function of complexity of the model. The same will be done for Ridge and Lasso regression and the three methods will be compared to find the optimal model for the Franke function dataset. Lastly the same analyses and comparisons will be done, however it will now be applied to a real dataset of digital terrain in Stavanger. To conclude the methods and their properties will be discussed and compared.

Methods

Linear Regression

Before trying to apply the linear regression in code it is important to get an intuitive understanding of the workings of the methods. Suppose the dependent variable Y can be written as a sum of a function and a normally distributed error term:

$$Y = f(x) + \epsilon$$

When performing the regression, the function "f" is assumed to have a linear relation to the "Y" data, and will be approximated by a linear function of the design matrix X. The independent data variables are stored in this matrix which has dimensionality mxn. The "m" denotes the number of data samples and "n" denotes the characteristics of the samples. In this project the characteristics represents the polynomial degree of which we will approximate Y. Each column in the design matrix will represent a possible permutation of the variables x and y within the polynomial degree range. Meaning that X can be written in vector form as $[1\ x\ y\ xy\ x^2\ y^2 \dots]$ and so on. If the specified polynomial degree is "p" then the design matrix can be written as a sum where every term is a column in X:

$$\sum_{i=0}^{p}\sum_{j=0}^{p} x^i y^j$$

Where x and y are the input vectors. The design matrix can then be fully written out as:

$$X = \begin{bmatrix} 1 & \cdots & x_0^n y_0^n \\ \vdots & \ddots & \vdots \\ 1 & \cdots & x_m^n y_m^n \end{bmatrix} = [1 \quad X_{*,1} \quad \cdots \quad X_{*,n}] = \begin{bmatrix} 1 \\ X_{1,*} \\ \vdots \\ X_{m,*} \end{bmatrix}$$

Where the "*" subscript denotes that we are looking at the whole column or row. With this knowledge, the data Y can be written as a linear equation on the form:

$$Y = X\beta + \epsilon = \beta_0 + \beta_1 X_{*,1} + \ldots + \beta_n X_{*,n} + \epsilon$$

β is the regression parameter and its purpose is to weigh the importance of each characteristic to make the optimal model of the data Y. Beta is constructed by minimizing the cost function and each linear regression method has its own derivation of beta with its unique qualities. A function which creates the design matrix has been coded according to these principles:

```
def designmatrix(x,y,n):
    if len(x.shape)>1:
        x=x.ravel()
        y=y.ravel()

    l=int((n+1)*(n+2)/2)
    X=np.ones((len(x),l))
    for i in range(1,n+1):
        q=int(i*(i+1)/2)
        for j in range(i+1):
            X[:,q+j]=(x**(i-j))*(y**j)

    return X
```

Deriving a beta expression

To derive the beta expression for OLS we first have to consider the cost function which is the residual sum of squares:

$$C(X,\beta) = \frac{1}{m}\sum_{i=0}^{m}(Y_i - f(x_i))^2 = \frac{1}{m}\sum_{i=0}^{m}(Y_i - \sum_{j=0}^{n}X_{i,j}\beta_j)^2 = \frac{1}{m}(Y - X\beta)^T(Y - X\beta)$$

The cost function is essentially the mean squared error of the model which is why it can be written as the expectation value of the error squared. To get the optimal model for Y the cost function should be as low as possible. Given this, we get an optimization problem where we want to minimize "C" for every possible beta. This can be done by taking the derivative of the cost function equal to zero:

$$\frac{\partial C}{\partial \beta} = \frac{1}{m}\frac{\partial}{\partial \beta}\left((Y - X\beta)^T(Y - X\beta)\right) = 0$$

$$-X^T(Y - X\beta) - (Y - X\beta)^T X = -2X^T(Y - X\beta) = 0$$

$$X^T(Y - X\beta) = 0$$

Above, an expression for the minima of the cost function with respect to beta has been found. To verify that this indeed is a minima, the second derivative can be observed.

$$\frac{\partial^2 C}{\partial \beta \partial \beta^T} = 2X^T X$$

Assuming the design matrix is made up of linearly independent variables (full column rank) then $X^T X$ will be positive definite and the second derivative will be positive for all values. This implies that the beta value where the first derivate is zero will be a minimum of the cost function. Furthermore, the optimal beta value can then be found by rearranging the expression for the first derivative above:

$$X^T(Y - X\beta) = 0$$

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T Y$$

Where the "hat" denotes the optimal beta for the model. A problem which may occur when implementing this method is when X reaches high dimensionality, $X^T X$ is singular. Meaning that there occurs some dependencies between the input variables and the inverse is no longer defined. This is resolved by a functionality in NumPy which computes the Moore-Penrose pseudo-inverse and is applicable to singular matrices. It is a method which uses the singular value decomposition (SVD) of X to find the inverse. With these principles it is possible to code a function which returns the optimal beta for a model.

Another way of dealing with the singularity is to add a tuning parameter, lambda, to the diagonal of $X^T X$, resulting in a matrix which is always invertible. This optimal beta is called the ridge regression estimator. It is an expansion of the OLS estimator which includes a regularization term (L2 norm). Meaning that ridge regression is essentially a method to shrink the beta coefficients from OLS using lambda as a penalizer. Consequently, to find beta, this gives rise to a constrained optimization problem.

$$\min_{\beta} \|Y - X\beta\|_2^2 \quad subject\ to\ \|\beta\|_2^2 \leq t\,, \qquad (\|x\|_2 = \sqrt{\sum_i x^2})$$

The problem highlights that the goal is to minimize the least squares residual sum of squares while keeping beta within a consistent range to prevent coefficients from blowing up. Solving the problem above using Lagranges method with "t" greater than zero derives an expression for the cost function which needs to be minimized.

$$C(X, \beta, \lambda) = \|Y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$$

To find the optimal beta, the derivative with respect to beta is set to zero and solved for beta.

$$\hat{\beta}(\lambda)_{Ridge} = (X^T X + \lambda I)^{-1} X^T Y$$

Where $X^T X$ is positive semidefinite (all non-negative eigenvalues) and lambda includes values ranging from zero upwards. The sum of these two matrices can be shown to be a positive definite matrix by Lemma 14.2.4 in … Holds only as long as lambda ranges over positive values. This implies that the sum will be an invertible matrix by the invertible matrix theorem, since all the singular values are nonzero. So, an expression for the optimal beta has been made without the use of the pseudoinverse.

To get insight into how ridge differs from OLS we can look at the SVD of X for each of the methods.

$$X = U\Sigma V^T$$

U and V are orthogonal m x n and n x n matrices where the columns span X´s column- and row space respectively. Sigma is a n x n diagonal matrix where the entries are the singular values of X in descending order. Using the SVD we can reformulate the OLS predictor:

$$X\widehat{\beta}_{OLS} = X\left(X^T X\right)^{-1} X^T Y = U\Sigma V^T \left(V\Sigma^2 V^T\right)^{-1} V\Sigma^T U^T Y$$

Using the properties of orthogonal matrices this gives:

$$X\widehat{\beta}_{OLS} = UU^T Y$$

This result tells that the optimal OLS model is essentially a projection of Y onto the column space of X. Now, the same will be done for the ridge predictor to give a comparison.

$$X\widehat{\beta}(\lambda)_{Ridge} = X(X^T X + \lambda I)^{-1} X^T Y = U\Sigma V^T \left(V\Sigma^2 V^T + \lambda I\right)^{-1} V\Sigma^T U^T Y$$

$$X\widehat{\beta}(\lambda)_{Ridge} = U\Sigma\left(\Sigma^2 + \lambda I\right)^{-1} \Sigma U^T Y = \sum_{i=0}^{n} U_{*,i} \frac{\sigma_i^2}{\sigma_i^2 + \lambda} U_{*,i}^T Y$$

The lowercase sigma denotes the non-zero singular values of X. Like the OLS predictor, the ridge predictor projects Y onto the column space of X. However, the ridge includes a term which shrinks the Y vector depending on lambda, and as the singular value decreases. The condition for ridge regression is that lambda is positive definite which forces the middle term in the expression to range from 0 to 1. Consequently, it acts as a shrinking factor.

In addition to ridge regression there exists another linear regression method which also acts as a shrinking method. This method is called LASSO (least absolute shrinkage and selection operator) regression. Similarly to ridge, the lasso estimator is simply the addition of a regularization term to the OLS estimator. However, this method employs an L1 penalty instead of the L2 as in ridge. The beta expression for lasso is derived analogously since it also is a constrained optimization problem.

$$\min_{\beta} \|Y - X\beta\|_2^2 \quad subject\ to\ \|\beta\|_1 \leq t\,, \qquad (\|x\|_1 = \sum_i \|x_i\|)$$

$$C(X, \beta, \lambda) = \|Y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

The previous L2 constraint was a circle (in 2D) while the new L1 acts as a square (in 2D). The substitution with the L1-norm changes the nature of the problem and therefore changes the properties of the beta model. Since L1 is a square, it has points on its borders which are discontinuous which will make finding an analytical solution for the optimal beta model difficult. Therefore, lasso regression will be coded with the preexisting functionalies in sci-kit which due to the geometry of the L1, when optimizing the beta vector some components are zeroed out rather than shrunk as in ridge. An easier way to see the effects of the different regression coefficients is to study the case when X is the identity matrix.

$$\hat{\beta}_{OLS} = (X^T X)^{-1} X^T Y = Y$$

$$\hat{\beta}(\lambda)_{Ridge} = (X^T X + \lambda I)^{-1} X^T Y = (I + \lambda I)^{-1} Y = \frac{\hat{\beta}_{OLS}}{1 + \lambda}$$

As stated earlier and seen from the equation above, the ridge estimator is basically a method for shrinking the beta coefficients of OLS. The expression is missing the scaling effect of the singular values but does a good job of showing the main concept. As for the lasso the cost

function first needs to be tweaked to include the properties of the design matrix and then optimized.

$$C(X,\beta,\lambda) = \|Y - X\beta\|_2^2 + \lambda\|\beta\|_1 \ = \|Y - \beta\|_2^2 + \lambda\|\beta\|_1$$

$$\frac{dC}{d\beta} = -2\|Y - \beta\|_2 \ + \lambda\frac{\beta}{\|\beta\|_1} = 0$$

$$\hat{\beta}(\lambda)_{Lasso} = \begin{cases} Y_i - \frac{\lambda}{2}, & Y_i > \frac{\lambda}{2} \\ Y_i + \frac{\lambda}{2}, & Y_i < -\frac{\lambda}{2} \\ 0, & \|Y_i\| \leq \frac{\lambda}{2} \end{cases}$$

Now the zeroing effect of the lasso is more visible. All the coefficients within a range from 0 to $\frac{\lambda}{2}$ are set to zero while the outer betas are shifted $\frac{\lambda}{2}$ either up or down. To sum, ridge and lasso are regression methods which shrink the contribution of the OLS beta coefficients.

Mean Squared Error, Bias and Variance

To get an understanding of when each method performs best and how they behave it is informative to look at statistical properties such as the error, bias and variance. As stated earlier the data Y can be approximated by linear regression and written in matrix form as equation… The expectation value for the data Y using this approximation can be shown to be:

$$\mathbb{E}[Y] = \mathbb{E}[X\beta + \epsilon] = \mathbb{E}[X\beta] + \mathbb{E}[\epsilon] = \mathbb{E}[X\beta]$$

Previously it was mentioned that the error was set to be normally distributed $\epsilon \sim \mathcal{N}(0,\sigma^2)$ (sigma is not the singular value). This is why the expectation value term for the error disappears. Going further the expectation value for each Y element can be expressed as:

$$\mathbb{E}[Y] = \frac{1}{m}\sum_{i=0}^{m-1} Y_i \qquad , \qquad Y_i = \sum_{j=0}^{n-1} X_{i,j}\beta_j + \epsilon_i = X_{i,*}\beta + \epsilon_i$$

$$\mathbb{E}[Y_i] = \mathbb{E}\left[X_{i,*}\beta\right] + \mathbb{E}[\epsilon_i] = \mathbb{E}\left[X_{i,*}\beta\right] = X_{i,*}\beta$$

$X_{i,*}\beta$ is a non-stochastic variable which is why the expectation value will be the value itself. This result tells us that the mean value of the data sample Y is given by the product of the X and beta matrices. Now, looking at the variance of each element of Y this gives:

$$var[Y_i] = \mathbb{E}[(Y_i - \mathbb{E}[Y_i])^2] = \mathbb{E}\left[{Y_i}^2\right] - \mathbb{E}[Y_i]^2$$

$$var[Y_i] = \mathbb{E}\left[\left(X_{i,*}\beta + \epsilon_i\right)^2\right] - (X_{i,*}\beta)^2 = \mathbb{E}\left[(X_{i,*}\beta)^2\right] + 2\mathbb{E}\left[X_{i,*}\beta\epsilon_i \right] + \mathbb{E}[{\epsilon_i}^2] - (X_{i,*}\beta)^2$$

$$var[Y_i] = \mathbb{E}[{\epsilon_i}^2] = var[\epsilon_i] = \sigma^2$$

Where the properties of the error (mean value equal zero) are used to derive an expression for the variance of Y. As seen from the equation above, the data Y has the same variance as that defined for the error previously. This result combined with the expectation value tells us that the linear model for Y follows a normal distribution that has mean value $X\beta$ and the variance is due to the error term ($Y_i \sim \mathcal{N}(X_{i,*}\beta, \sigma^2)$). General insight into the linear model has now been attained through studying the distribution of Y, however to gain further information about the specifics of each linear method the different estimators need to be analyzed.

The OLS estimator which is expressed by equation … can be used to study the distribution of the beta coefficients for this method. Starting off the mean value needs to be expressed:

$$\mathbb{E}\left[\hat{\beta}_{OLS}\right] = \mathbb{E}[(X^T X)^{-1} X^T Y] = (X^T X)^{-1} X^T \mathbb{E}[Y]$$

Using the properties that X is a non-stochastic variable, and the previous result for the expectation value we get:

$$\mathbb{E}\left[\hat{\beta}_{OLS}\right] = (X^T X)^{-1} X^T X\beta = \beta$$

This result shows that the OLS estimator is unbiased. Furthermore, the variance of the OLS estimator can be expressed as:

$$var\left[\hat{\beta}_{OLS}\right] = var[(X^T X)^{-1} X^T Y] \ , \qquad B = (X^T X)^{-1} X^T$$

$$var\left[\hat{\beta}_{OLS}\right] = var[BY] = B\ var[Y]\ B^T = B\ \sigma^2\ B^T$$

Where the previous result that the variance of Y can is the variance of the error has been applied. Substituting for B and using the properties of symmetrical matrices now gives:

$$var\left[\hat{\beta}_{OLS}\right] = (X^T X)^{-1} X^T \sigma^2\ X(X^T X)^{-1} = \sigma^2 (X^T X)^{-1} X^T X (X^T X)^{-1} = \sigma^2 (X^T X)^{-1}$$

With this result, a confidence interval for the beta values can be made through the standard deviation. This can also be done analytically for the ridge estimator (Lasso not possible to find analytically) to give a comparison between OLS and the linear shrinking methods. To start, the expected value of the ridge estimator will be shown:

$$\mathbb{E}\left[\hat{\beta}_{Ridge}\right] = \ \mathbb{E}[(X^T X + \lambda I)^{-1} X^T Y] = (X^T X + \lambda I)^{-1} X^T X\beta$$

Since the expectation value is not equal to the estimator itself for any $\lambda > 0$ the ridge estimator is biased. When lambda is zero the ridge is equal to the OLS. By the same methods of calculation, the variance of the ridge can be expressed as:

$$var\left[\hat{\beta}_{Ridge}\right] = \ var\left[W\hat{\beta}_{OLS}\right] = W\ var\left[\hat{\beta}_{OLS}\right] W^T, \quad W = (X^T X + \lambda I)^{-1} X^T X$$

$$var\left[\hat{\beta}_{Ridge}\right] = \sigma^2 (X^T X + \lambda I)^{-1} X^T X ((X^T X + \lambda I)^{-1})^T$$

Similarly to the mean value, the variance of the ridge estimator tends to zero as lambda increases. This is to be expected by eq … the beta coefficients tend to zero as lambda gets large. Therefore, the variance of the ridge is always lower or equal to (when $\lambda = 0$) the OLS. Although there is no analytical expression for the Lasso, the variance behaves in a similar

fashion to ridge. Due to the fact that Lasso also uses the lambda to shrink the OLS estimator. Therefore, when doing further comparisons between the methods the Lasso will be presumed to have similar traits to ridge.

$$var\left[\hat{\beta}_{OLS}\right] - var\left[\hat{\beta}_{Ridge}\right] = \sigma^2(X^TX)^{-1} - \sigma^2(X^TX + \lambda I)^{-1}X^TX((X^TX + \lambda I)^{-1})^T$$

Another useful parameter to measure the quality of the linear model is the mean squared error. The MSE is defined the same way as the cost function for OLS (eq …) and can be rewritten as the expectation value of the square of the model's deviation from the data.

$$C(X,\beta) = \frac{1}{m}\sum_{i=0}^{m}\left(Y_i - f(x_i)\right)^2 = \mathbb{E}[(Y - X\beta)^2]$$

The MSE can be rewritten again by adding and subtracting the expected value of $X\beta$ on both sides.

$$\mathbb{E}[(Y - X\beta)^2] = \mathbb{E}[(f(x) + \epsilon - X\beta)^2]$$

$$\mathbb{E}[(Y - X\beta)^2] = \mathbb{E}[(f(x) + \epsilon - X\beta + \mathbb{E}[X\beta] - \mathbb{E}[X\beta])^2]$$

Remembering that the expectation value of the squared error is the variance, and using the formula for the variance the MSE can be decomposed into terms:

$$\mathbb{E}[(Y - X\beta)^2] = [(f(x) - \mathbb{E}[X\beta])^2] + var[X\beta] + \sigma^2$$

The first term is called the bias term and gives the square of the model's deviation from the data points distribution (without error). It arises due to the model trying to align with the true function of the data. For example, if the model is linear but the true distribution is non-linear, the model will have high bias. The variance of the model gives the error as the complexity increases by showing how much the data spreads from the mean of the model. The last term is the irreducible error which comes from the noise and gives a lower estimate of the error. An ideal model has both low variance and low bias. However, to lower the bias one would increase the model complexity (increase the polynomial degree) but this will result in

overfitting (high variance) and make future predictions unreliable. A good model is one which has a good balance between the bias and variance. Therefore, after training the model with a training set of data a MSE analysis will be done on the test sample to look for over- or underfitting (high error for training data). Generally, accuracy can be improved by giving the model more data samples if possible.

Resampling methods

The resampling methods which will be employed are the bootstrap and k-fold cross validation. Both will be used to perform a bias-variance analysis of the model. The bootstrap method uses random sampling with replacement to estimate the value of an estimand, which in this case is the error decomposed into bias and variance. According to the central limit theorem (put source here) it is possible to make an estimate of the true bias-variance distribution of the data even though there is a limited dataset. A code for the bootstrap will be made by resampling from a set of training data and using it to train the model. Then the error for test data will be calculated using beta expressions derived from the training data, an excerpt of the code is shown below:

```
for i in range(len(p)):

    X=designmatrix(x,y,i+1)

    X_train, X_test, Y_train, Y_test= train_test_split(X,Y,test_size=0.2)
    X_scaler = StandardScaler()
    X_scaler.fit(X_train)
    X_train_ = X_scaler.transform(X_train)
    X_test_ = X_scaler.transform(X_test)

    z_tilde=np.empty((Y_train.shape[0],bootstraps))
    z_pred=np.empty((Y_test.shape[0],bootstraps))

    for j in range(bootstraps):
        X_, z_ =resample(X_train,Y_train)
        beta=beta_OLS(X_,z_)
        z_tilde[:,j]=X_@beta
        z_pred[:,j]=(X_test@beta)


    error2[i]=np.mean( np.mean((Y_train[:,np.newaxis] - z_tilde)**2, axis=1, keepdims=True) )
    error[i]=np.mean( np.mean((Y_test[:,np.newaxis] - z_pred)**2, axis=1, keepdims=True) )
    bias[i]=np.mean( (Y_test[:,np.newaxis] - np.mean(z_pred, axis=1, keepdims=True))**2 )
    variance[i]=v+np.mean( np.var(z_pred, axis=1, keepdims=True) )
```

K-fold cross validation is a structured way of splitting the data as opposed to the bootstrap method. By doing this the creation of similar datasets due to sampling with replacement is avoided. Consequently, CV structures the data by splitting the original dataset into *"k"* number of datasets. Each dataset is then once assigned as the test set while the other are training sets and the error is calculated. The total error is then presented as the mean of all the test errors, an excerpt of the code for cross-validation is given below. To find the optimal value for the number of folds it, the estimation will be done with k between 5-10.

Results

Linear regression methods on Franke function

To begin, the linear regression methods were applied to a self-created dataset by the franke function which is defined by function:

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

The function made data with the x and y variables ranging from 0 to 1 with a uniform distribution of 101 points and adding stochastic noise with 0.1 variance. With the created dataset and the given x and y variables a design matrix was made according to the methods explained. Then a fit of the data was made using the optimal beta with varying polynomial degree. A plot of the MSE and R2-score as well as the beta coefficients are given in the figures below. Before making the fit, both the design matrix and data have been scaled using the Standardscaler from scikit-learn. It scales the data by subtracting the mean as well as dividing by the variance for each feature/column. This results in data which has standard deviation equal to 1 and mean equal to zero. Doing this ensures that all the features are of the

same scale and negates the varying magnitude as the complexity of the model increases. In addition it also allows for a more qualitative analysis by comparing with other models.

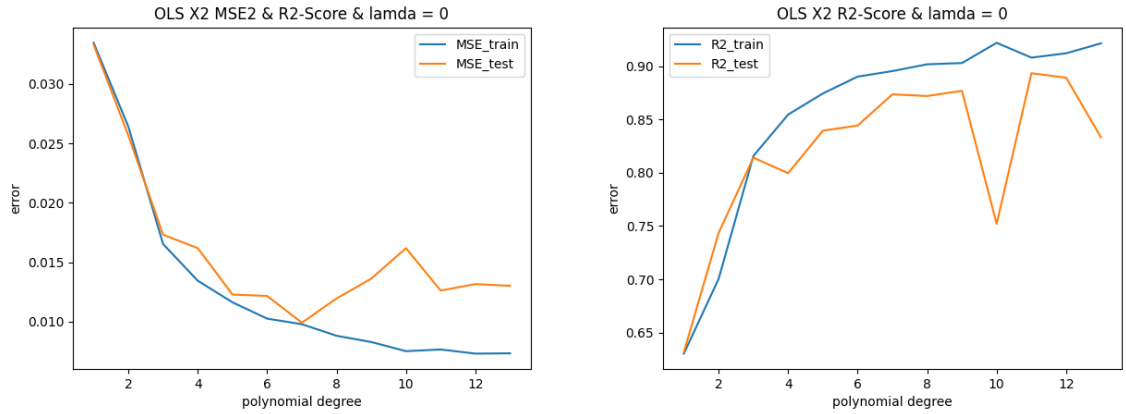


Figure 1: OLS MSE and R2-score as a function of polynomial degree when fitted to the franke function

Before using OLS, the data was split into training and test sets to give a more accurate result of how the method would work for new unseen data. As seen from the results, the training error decreases as the degree increases due to the polynomial being able to make a closer fit to all the datapoints. However, with the test error it deviates from the training error at around the 3rd degree. This is because, while the model is becoming more trained to the training set, the model will become specific to that set. Meaning that the model reaches overfitting territory as the complexity increases. The R2-score is essentially a measure of the MSE but scaled by the models variance. Meaning that it is a measure of how much of the variance in the data in accounted for by the model. From looking at both the MSE and R2-score it seems that 7th degree polynomial is the optimal OLS fit.

Now, the same analysis will be performed on Lasso and Ridge for selected values of lambda.

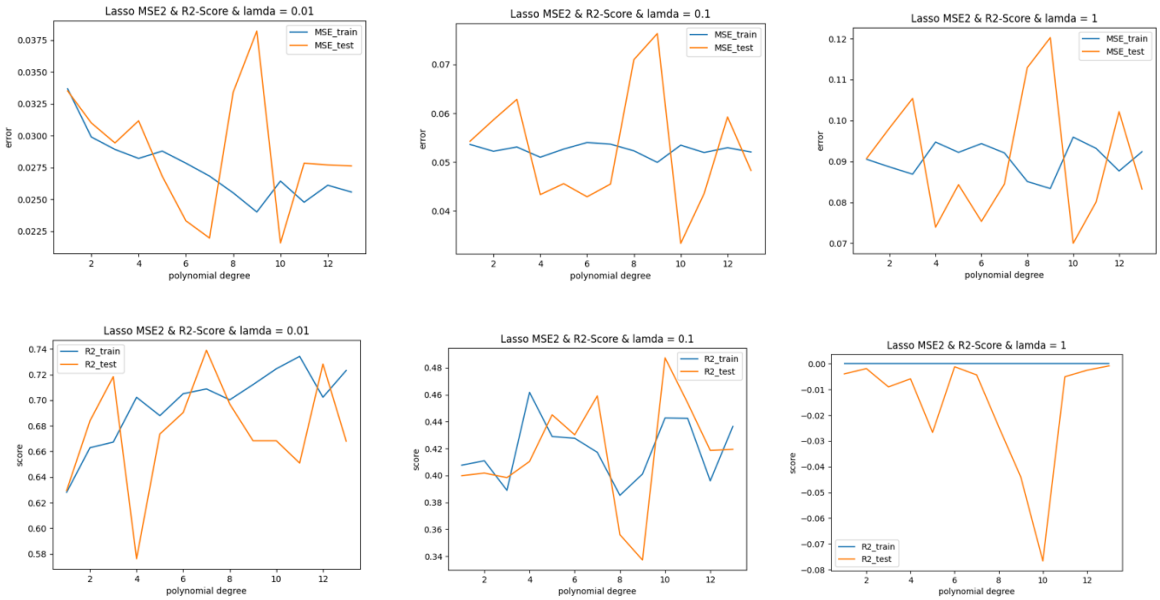


Figure 2: MSE and R2 as a function of Lasso model complexity with lambda ranging from 0.01-1

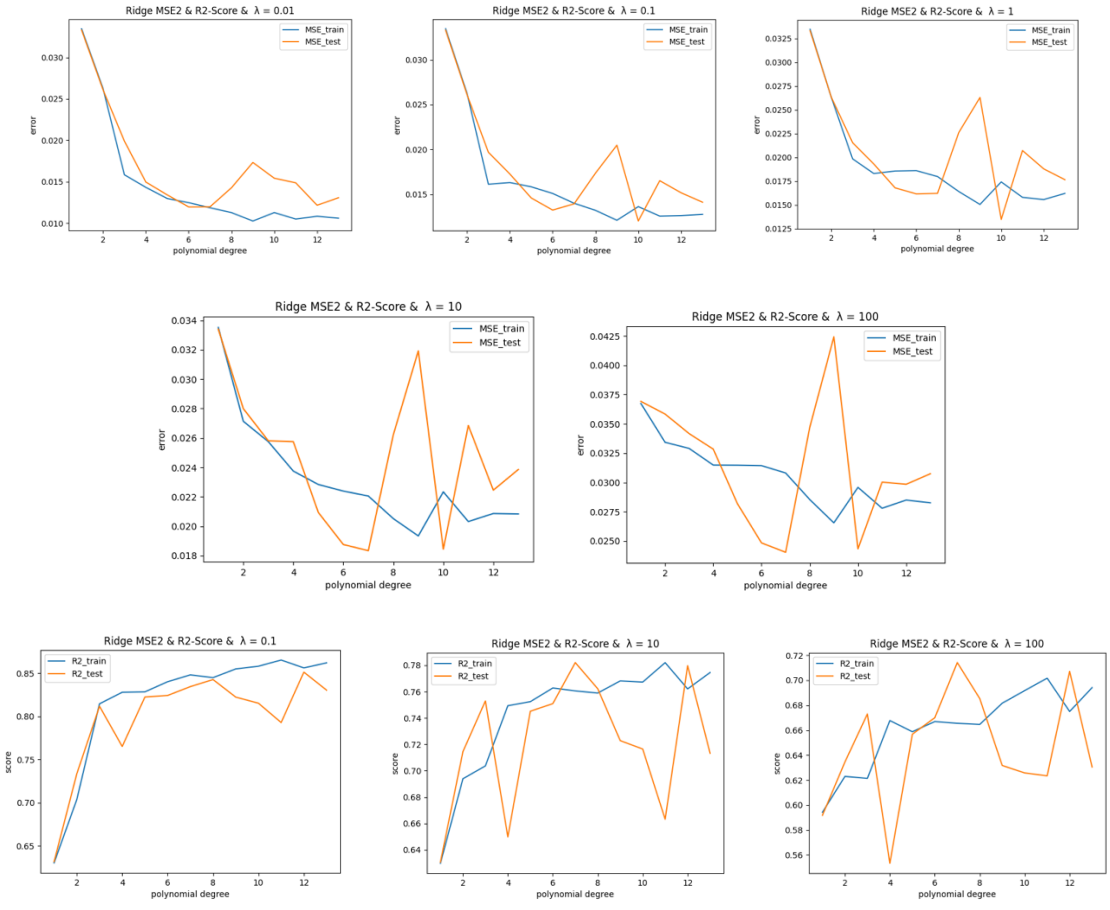


Figure 3: MSE and R2 with Ridge model complexity, lambda ranging from 0.01-100

These results show how feature penalization affects the fit to the data. Generally, for both Lasso and Ridge, the quality of the fit tends to decrease with increasing lambda which especially is apparent when comparing the R2-scores. The Lasso shows to deviate quicker from the OLS than Ridge resulting in higher errors. This is due to the discontinuous way Lasso penalizes the estimator. All parameters within a set range are nullified and only those outside the region make up the estimator. When lambda is large enough, all estimators will be zero and the result will be similar to figure 2 for lambda equal to one. In contrast, Ridge only decreases each OLS estimator by a factor given by the singular values, which is why the decreasing quality of the model is continuous for higher lambda values. Below the beta coefficients have been printed out for a 1st ,2nd and 3rd degree polynomial for different lambdas. The behavior of the different estimators can be seen to act as described previously with increasing lambda.

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.000000
1	-0.144630	-0.144625	-0.134549
2	-0.191279	-0.191273	-0.181198

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.000000
1	-0.144630	-0.144584	-0.043824
2	-0.191279	-0.191219	-0.098474

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.0
1	-0.144630	-0.140202	-0.0
2	-0.191279	-0.185448	-0.0

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.000000
1	-0.258224	-0.258048	-0.158441
2	-0.184397	-0.184306	-0.083958
3	-0.006410	-0.006520	-0.000596
4	0.193037	0.192941	0.044732
5	-0.135951	-0.135973	-0.128773

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.000000
1	-0.258224	-0.256481	-0.040518
2	-0.184397	-0.183503	-0.014073
3	-0.006410	-0.007494	-0.000000
4	0.193037	0.192075	-0.000000
5	-0.135951	-0.136169	-0.077680

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.000000
1	-0.258224	-0.165429	-0.0
2	-0.184397	-0.137557	-0.0
3	-0.006410	-0.053734	-0.0
4	0.193037	0.127619	-0.0
5	-0.135951	-0.135820	-0.0

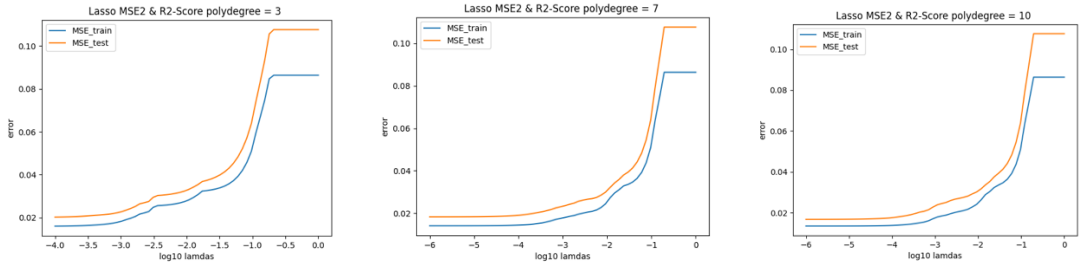
	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.000000
1	-0.131526	-0.134698	-0.150662
2	0.391510	0.373140	-0.095832
3	-0.438785	-0.431385	-0.029580
4	0.392192	0.394152	0.000000
5	-1.954344	-1.909126	-0.113115
6	0.232667	0.227943	-0.000000
7	0.117574	0.117705	0.062938
8	-0.295215	-0.297146	0.000000
9	1.325410	1.297722	-0.000000

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.000000
1	-0.131526	-0.165569	-0.039513
2	0.391510	0.243781	-0.011933
3	-0.438785	-0.377879	-0.000000
4	0.392192	0.401167	-0.004012
5	-1.954344	-1.584353	-0.069870
6	0.232667	0.192821	-0.000000
7	0.117574	0.120562	-0.000000
8	-0.295215	-0.306777	-0.000000
9	1.325410	1.096723	-0.000000

	β_OLS estimator	β_Ridge estimator	β_Lasso estimator
0	0.000000	0.000000	0.0
1	-0.131526	-0.149546	-0.0
2	0.391510	-0.139826	-0.0
3	-0.438785	-0.088985	-0.0
4	0.392192	0.051396	-0.0
5	-1.954344	-0.155952	-0.0
6	0.232667	-0.010487	-0.0
7	0.117574	0.124774	-0.0
8	-0.295215	-0.036615	-0.0
9	1.325410	0.046089	-0.0

Figure 4: Beta coefficients for lambda ranging from 0.01, 0.1, 10 (left to right)

In contrast to OLS, finding the optimal polynomial for Lasso and Ridge is more complex since the fit also depends on the chosen lambda values. However, as has been studied, the model tends to worsen as lambda is increased which suggests that the optimal complexity should be the one given by OLS (7th degree polynomial). Thereby, it remains to study the model's dependence on lambda to give a definite answer to which model performs best. Below such an analysis of Lasso and Ridge is presented for chosen polynomials.



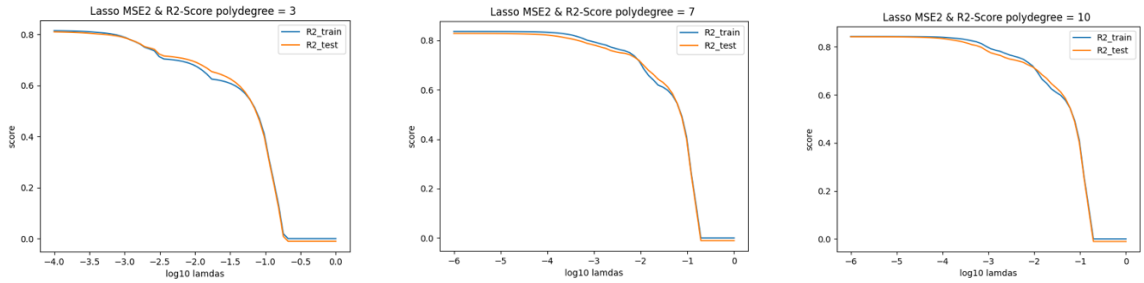


Figure 5: Lasso MSE as a function of lambda with polydegrees 3,7 and 10

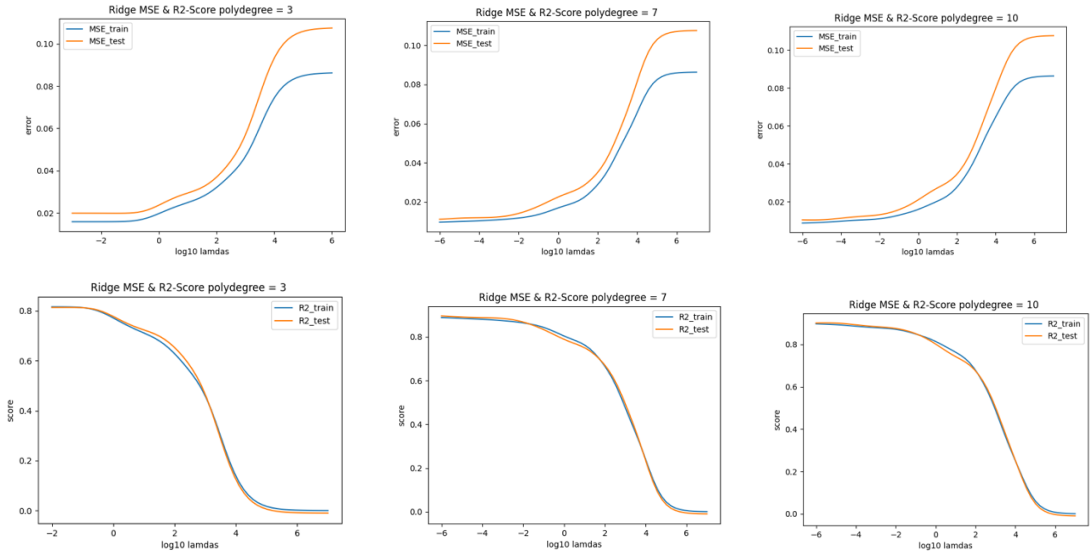


Figure 6: Ridge MSE as a function of lambda for polydegree 3,7 and 10

These plots show what was observed previously, the error increases as lambda increases and consequently results in a bad fit. As shown by the R2-score, the variance of the training model goes to zero while the test model has a negative score, implying that it is a worse fit than the mean. The shape of the error plots also reflects the penalizing nature of the different estimators. Lasso being discontinuous has flat areas on the graph due to the zeroing, while Ridge plateaus due to scaling by lambda and the singular values. From this sample of polynomials, it seems that beyond the 7^{th} degree the models do not give smaller errors, as seen by comparing the 7^{th} and 10^{th} degrees. All these results point to the OLS method to be the best fit to the data.

Next, a bias-variance analysis was made of the OLS regression using the bootstrap method. As stated earlier the error can be decomposed into bias and variance, which will be studied as a function of the model complexity.

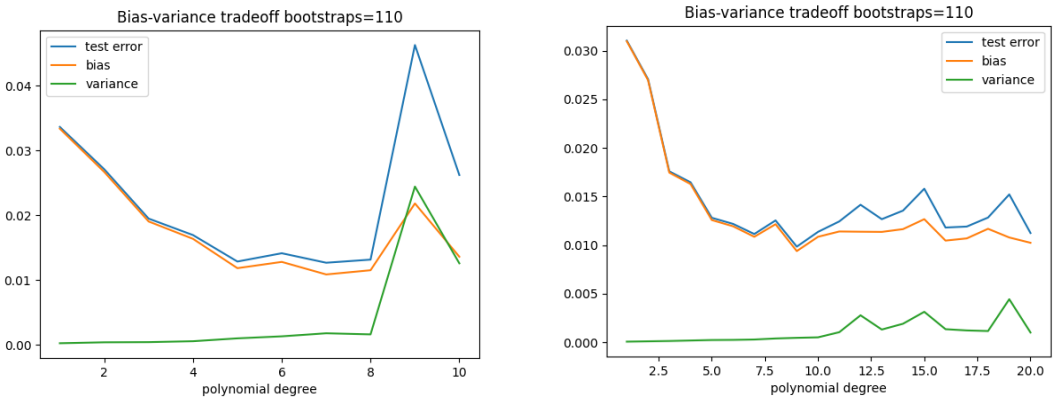
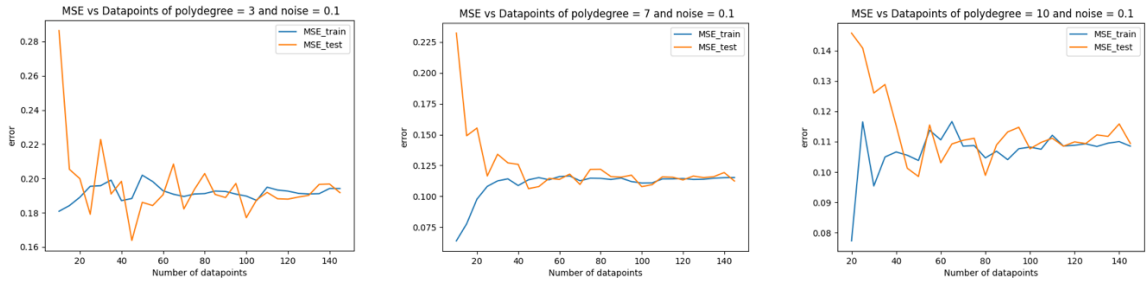


Figure 7: OLS Bias-Variance tradeoff as function of complexity. Left plot is with grid of 20x20 datapoints, right is grid of 40x40 datapoints.

To find out what makes the error rise it is composed into bias and variance. The results in figure 7 show that as the complexity increases, further increase in the error is mainly due to variance. This tells that the function is approaching overfitting. At lower degrees the variance is insignificant because the error arises from the bias. Previously the optimal complexity was found by primarily looking at the MSE, but now the optimal fit should be one that has the lowest variance and bias possible for the purpose of the model. Generally, are flexible models ones with high variance and low bias while ones with the opposite are rigid. For the 20x20 grid this is still the 7th degree however, the total error has decreased and the test error deviates later. More datapoints imply more accurate training of the model and since the data here is artificially created by a function, the test error deviation will occur at a higher complexity. Unless, the error term is adjusted to the number of datapoints. Figure .. shows how the number of datapoints and noise affects the error of the model (Only for OLS).



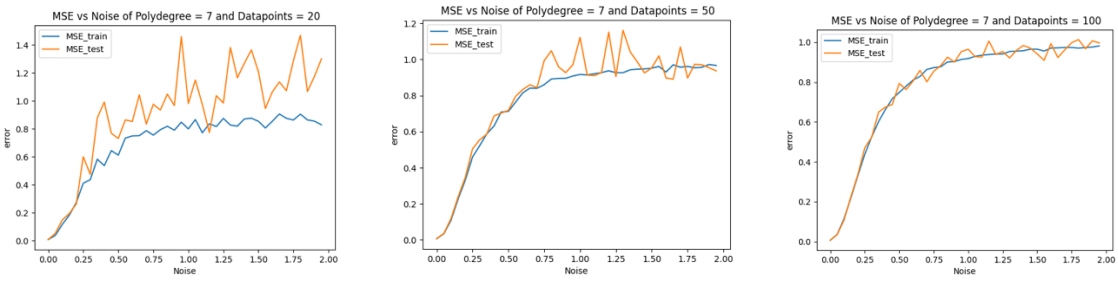
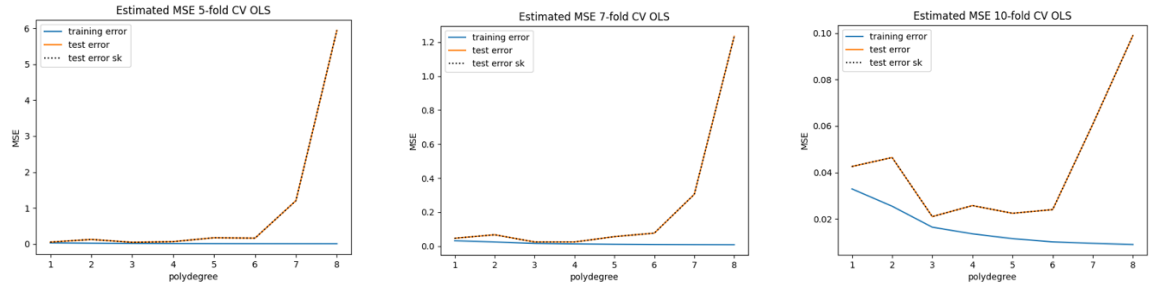


Figure 8: MSE vs Datapoints in the top row which shows how beyond 40x40, the errors converge. MSE vs Noise at the bottom which shows how the error converges with increasing datapoints.

Figure 8 tells us that after around 40x40 datapoints, the training and test errors converge if the noise is constant. This implies that there will not be any significant change in the test and training error beyond a 40x40 grid for increasing complexities. Which is a consequence of the data being highly correlated because it is produced from a function. Only an increase in the noise will disturb this correlation and thereby cause a divergence which will show regions of overfitting. However, as the grid increases a larger noise is required to observe the split as shown by the bottom row in figure 8. As stated previously, the datapoints and noise for the vanilla data was chosen to be 20x20 and 0.1 respectively. This is because the effects of over/underfitting wanted to be highlighted, as well as it being easy and quick to compute.

Another way of studying the error is by the k-fold cross validation resampling method. A study of the error was conducted and the results for the different regression methods is given below for fold 5-7-10. For the OLS method the error was found as a function of the complexity, while for the Lasso and Ridge, it was by the lambda values.



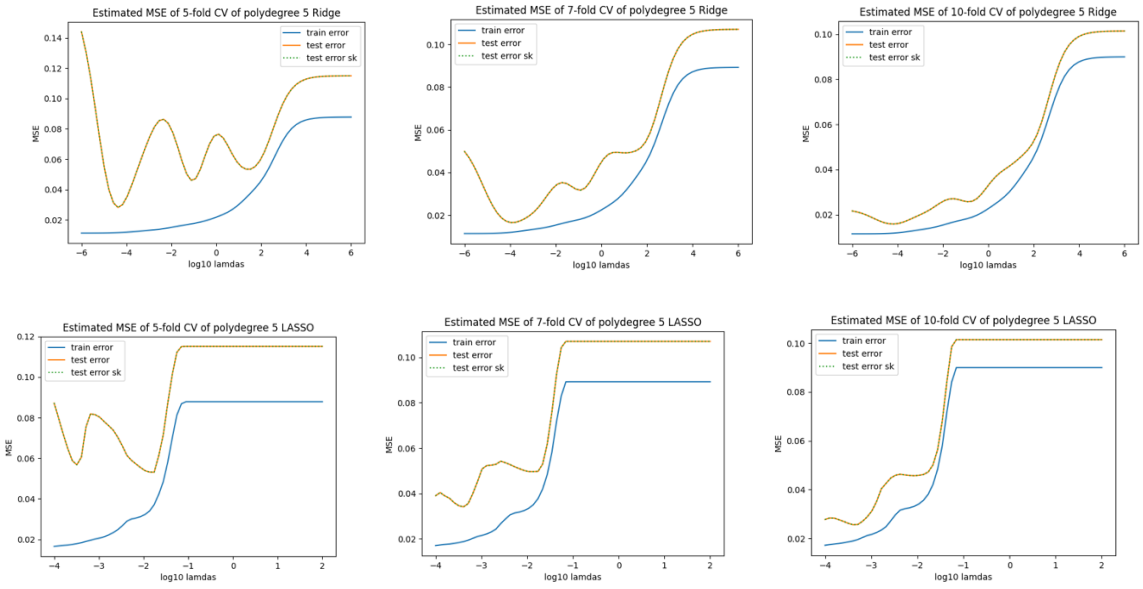


Figure 9: Upper row: OLS error as a function of complexity using 5/7/10-fold cross validation. Middle row: Ridge error as a function of lambda using 5/7/10-fold cross validation on 5th degree polynomial. Lower row: Lasso error as a function of lambda using 5/7/10-fold cross validation on 5th degree polynomial.

The results for every method vary depending on the number of folds which shows that the CV is sensitive to the number of folds. By looking at the OLS results one can see how the bootrap method works compared to k-fold CV. More folds makes the error take similar shape to the previous MSE analysis without CV. The purpose of bootstrapping is to find the true distribution of statistical variables to give an accurate representation of something like the MSE. The purpose of CV is to find measure and compare the performance of different models and in this case also to find optimal an optimal hyperparameter.

Looking at the OLS results from both, one can see that with increasing folds the CV approximates the bootstrap resampling (even though it doesn't resample with resplacement). Larger number of folds estimates the error more accurate but it does also require more computing. The CV shows that the minimal error is for low values of lambda for both Ridge and Lasso (Around 10 to the power of -4) but it would probably smooth out like in figure 5 and 6 if more folds were used. With this and all the previous results, OLS seems to be the best regression method to approximate the Franke function.

Linear regression methods on Real Data

Now, as an understanding of the linear regression methods has been constructed it will be used as a basis to evaluate real terrain data of an area near Stavanger, Norway. The same analyses will be performed to evaluate which regression model fits the data best.

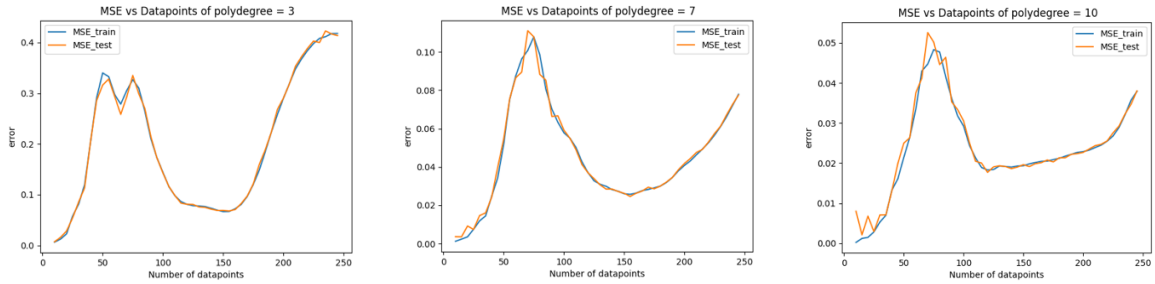
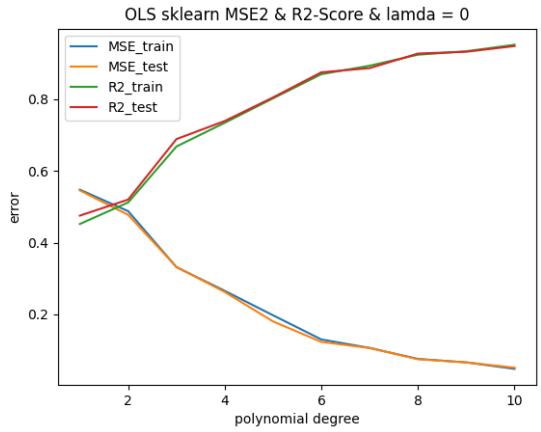


Figure 10: MSE vs Datapoints of terrain dataset. Error has maxima around 75x75 grid and minima between 100-150 grid

Before choosing the number of datapoints to proceed with an OLS error analysis was conducted. Figure 10 shows how this error varies. The criteria should be an area where there is a noticeable difference in training and test error as the complexity changes while not requiring high computational cost. The local maxima (50-75) are good candidates and are chosen as datapoints for further investigation.



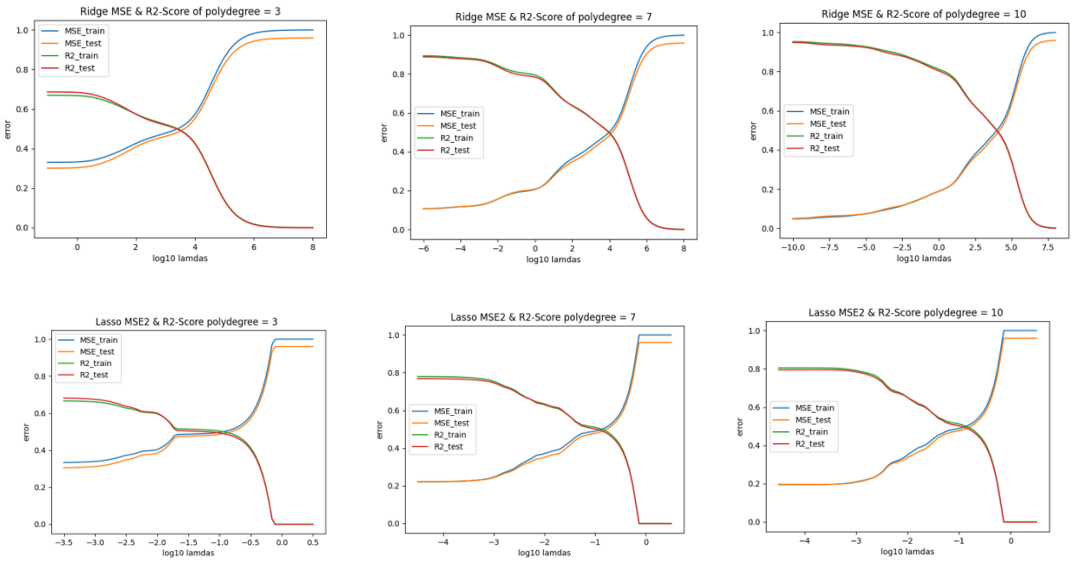


Figure 11: OLS, Ridge and Lasso MSE & R2-score on 75x75 grid

An evaluation of the MSE & R2-score for the regression methods is presented in figure 11. First, for the OLS method in contrast to the vanilla data, the test error decreases with the training error. Meaning that the model gets more accurate with higher complexity without seeming to suffer from overfitting. The Ridge and Lasso results support this observation when looking at the total MSE and R2-scores. This could however be because the selected data does not give the true representation of the error distribution. Therefore, will the next step be to verify these results by introducing resampling.

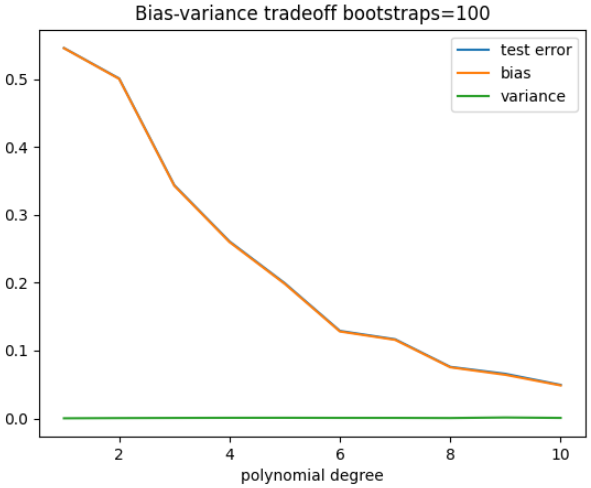


Figure 12: OLS bias-variance tradeoff analysis using bootstrap resampling

Using the bootstrap resampling gives a bias-variance tradeoff as given by figure 12. The variance of the model is more or less constant and close to zero. This means that the error is mainly due to the bias. The results from resampling confirm the behavior of the model observed previously. To give a final confirmation to this, k-fold CV analysis was conducted and is given in figure 13.

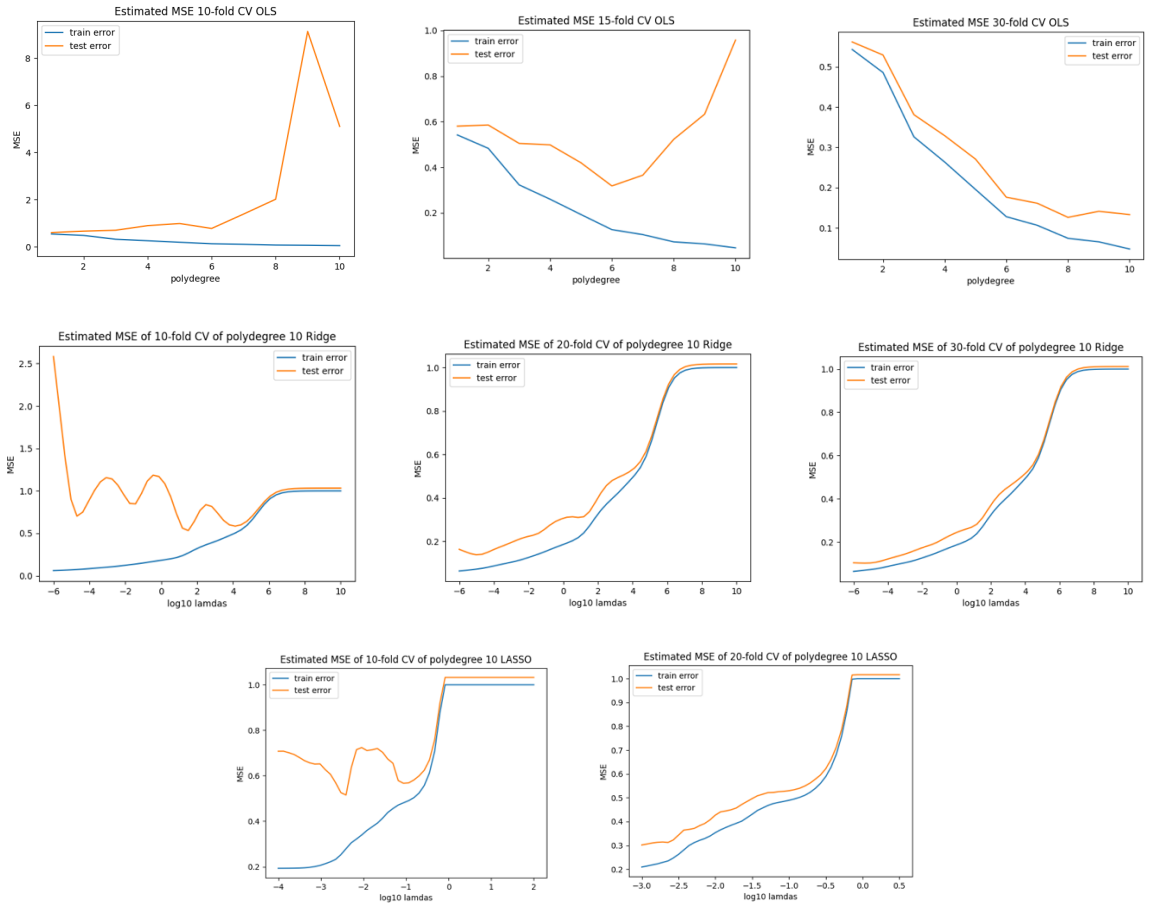


Figure 13: OLS, Ridge and Lasse k-fold cross validation

With increasing folds the error tends to take the shape of the previous results. All these results point towards the OLS method being the optimal method for the terrain data since it gives the best R2-score and the optimal hyperparameter is one which makes the Ridge and Lasso methods behave like OLS (lambda goes toward zero). Like with the Franke function, OLS is the best fit to the terrain data. This is probably because of the similar nature of the data. When the model is well trained on an adequately sized training set, it will perform well on new terrain data because the terrain type has been approximated by the model. New terrain data of the same area can often be predicted by looking at the surroundings and assuming it follows the trend. Consequently, it means that for such dataset, new data tends to have relatively strong correlation to the previous.

Conclusion

In this project the three linear regression models OLS, Ridge and Lasso have been explored by evaluating their performance statistically. This was done to gain an understanding of the workings of these methods to build a foundation for further learning of machine learning techniques. To accomplish this, a vanilla dataset made by the franke function was first modelled and then real terrain data was analyzed. Through testing it was found that for these specific datasets the OLS gave the optimal fit.