FYS-STK4155 – Project 1[†]

Sajjad Ahmadigoltapeh[1], Jan Fredrik Kismul[2], and Silje Helene Løvbrekke[3]

[1]sajjadah@uio.no
[2]j.f.kismul@fys.uio.no
[3]siljehlo@mat.uio.no

October 7, 2019

Abstract

In this study Franke's function and one set of geographical data are used to compare the performance of the Ordinary Least Squares (OLS), Ridge and Lasso regression methods. A bias variance analytically is derived and then validated for Franke's function and geographical data set by using Bootstrap. K-fold resampling technique is applied to get better estimation for MSE and R^2 using OLS, Ridge and Lasso. For Franke's function Ridge showed the best performance with MSE$= 0.0011889$ and $R^2 = 0.977837$ while for geographical data OLS and Ridge both performed equally promising, with $R^2 = 0.970$.

Contents

[†]Program folder: `https://github.com/jkismul/MachineLearning/blob/master/scr/code/`

1 Introduction

Regression analysis is a technique that has been broadly exploited in science. The aim of this technique is interpolation between measured data, extrapolation in ranges beyond the measuring range and extraction of relations among reported data. This project focuses on widely used methods of regression analysis i.e. Ordinary Least Square (OLS), Ridge and Lasso. In other word, the above mentioned methods will be compared empirically. In this project we first use OLS, Ridge, and Lasso to analyze a data set with a known function i.e. Franke. If the model estimates the required data with minimum error, there is, nevertheless, no guarantee that it works well for any other data set. Therefore, by resampling and cross validation we can check the validity of the model for an unseen data set. In this study, k-fold cross validation technique was used for all three methods.

2 Theory and methods

We consider set of $\hat{x} = [x_0, x_1, x_2, ..., x_{n-1}]^T$ as independent variables, $\hat{y} = [y_0, y_1, y_2, ..., y_{n-1}]^T$ as the set of recorded data from experiment and n is the number of measurements, such that $y_i = f(x_i)$. We would like to find the the functional dependency between y and x which allow to make prediction for values of y which are not in the list of available experimental data. To find a functional dependency between x and y in the form of $y = f(x)$, we write:

$$y(x_i) = \tilde{y_i} + \epsilon_i = \sum_{j=0}^{n-1} \beta_i x_i^j + \epsilon_i$$

in matrix form it is shown as:

$$\hat{y} = \hat{X}\hat{\beta} + \hat{\epsilon}$$

where $\hat{X}$ is a design matrix and $\hat{y}$ is a response vector and then, we say $\hat{X}$ is linearly proportional to $\hat{y}$ with respect to $\hat{\beta}$, such that $\hat{\beta} = [\beta_0, \beta_1, \beta_2, ..., \beta_{n-1}]^T$. The residual which is shown with $\hat{\epsilon}$ is a vector representing the error in our data points. It represents how far off our prediction is from the measurements .This relation is called linear regression with respect to β. We need to choose parameters $\hat{\beta}$ with the minimum errors of $\hat{\epsilon}$. The function of linear regression parameter which is shown as $Q(\hat{\beta})$ has a unique form for each regression method which will be discussed afterward.

Here an example might help clarify the situation: lets say we have conducted an experiment where we have measured the position of a ball launched straight up into the air from a cannon. Neglecting air resistance we know that the analytical solution is on the form of a second order polynomial $x(t) = x_0 + v_0 t + at^2$, where x_0 and v_0 are the initial conditions for the position and velocity respectively. This analytical solution is our model, but the actual measured values could (and indeed probably would) differ from this simply due to errors in the measurement or other effects coming into play that the model has not accounted for (like air resistance). In any case, if we

measured the position n times our linear algebra problem could be stated like this:

$$\begin{bmatrix} x(t_0) \\ x(t_1) \\ \vdots \\ x(t_{n-1}) \end{bmatrix} = \begin{bmatrix} (t_0)^0 & (t_0)^1 & (t_0)^2 \\ (t_1)^0 & (t_1)^1 & (t_1)^2 \\ \vdots & \vdots & \vdots \\ (t_{n-1})^0 & (t_{n-1})^1 & (t_{n-1})^2 \end{bmatrix} \begin{bmatrix} x_0 \\ v_0 \\ a \end{bmatrix} + \begin{bmatrix} \epsilon_0 \\ \epsilon_1 \\ \vdots \\ \epsilon_{n-1} \end{bmatrix}$$

With some of the variable names adjusted simply to better indicate the represented values we have in this problem. What we essentially want to do is to determine the variables x_0, v_0 and a so that the error terms are minimized (the equation has solutions for all $\hat{\beta}$s, but most of those are bad fits with huge error terms). Note that there are two dimensionalities coming into play here. n is the number of measurements and determines the length of the column vectors $\hat{x}$ and $\hat{\epsilon}$. In addition there is a second dimension that determines the number of columns in the matrix $\hat{T}$ and the length of the vector $\hat{\beta}$. This number, say m, indicates the complexity of our model. When doing a polynomial fit, $m-1$ is the order of the polynomial we want to fit our data to. So in this case, where we want to fit a second degree polynomial, we have

2.1 Ordinary Least Squares

Ordinary least squares (OLS) is the simplest and more popular method and it is defined as summation of the squared differences between predicted and measured values for each data point:

$$Q = \sum_{i=0}^{n-1} \epsilon_i^2 = \hat{\epsilon}^T \hat{\epsilon} = (\hat{y} - \hat{X}\hat{\beta})^T (\hat{y} - \hat{X}\hat{\beta})$$

Q is called cost function and as it is shown above, summation of squared differences i.e. $\hat{\epsilon}^2$ can be considered as inner product of the vector with itself. We are going to use this equation to find a value for $\hat{\beta}$ which minimizes Q as cost function. To achieve this we differentiate Q with respect to $\hat{\beta}$ which gives:

$$\begin{aligned} \frac{\partial Q(\hat{\beta})}{\partial \hat{\beta}} &= 0 = \hat{X}^T(\hat{y} - \hat{X}\hat{\beta}) \\ \hat{X}^T\hat{y} &= \hat{X}^T\hat{X}\beta \end{aligned}$$

in cases where we have an invertible $\hat{X}$, the analytical solution is:

$$\hat{\beta} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\hat{y} \tag{1}$$

Furthermore, the relevant variance could be analytically determined by:

$$\text{Var}\left(\hat{\beta}\right) = \sigma^2 \left(X^T X\right)^{-1} \tag{2}$$

2.1.1 Geometric aspect of ordinary least squares (OLS)

For ordinary least square we need to find a linear combination among the $\hat{X}$ columns such that gives the closest value to $\hat{y}$. This is fulfilled when $\hat{X}\hat{\beta}$ equals to the projection of $\hat{y}$ onto column space of $\hat{X}$ and in case the $\hat{X}$ has linear independent columns the solution will be unique. Here, we can say $\hat{X}\hat{\beta}$ is the projection of $\hat{y}$ onto the column space of $\hat{X}$, therefore the error vector namely, $\hat{y} - \hat{X}\hat{\beta}$, is orthogonal to the rows of $\hat{X}^T$, namely:

$$X^T(\hat{y} - \hat{X}\hat{\beta}) = \hat{0} \implies \hat{X}^T\hat{X}\hat{\beta} = \hat{X}^T\hat{y} \tag{3}$$

if we solve the equation with respect to $\hat{\beta}$ gives:

$$\hat{\beta} = (\hat{X}^T\hat{X})^{-1}\hat{X}^T\hat{y}. \tag{4}$$

2.2 Ridge

One of the disadvantages of the OLS method is that the matrix is not necessarily invertible. We know it is required to get an inverse of $(\hat{X}^T\hat{X})^{-1}$ and when matrix is not invertible, it is impossible to model the data by this method [1]. It is clear that for cases where $\hat{X}$ is not a square matrix, and have many columns it is most likely that we get a singular matrix (zero determinant). A simple solution to this problem is to add $\lambda\hat{I}$ term, where $\lambda \geq 0$, and make the matrix invertible. In this way by superimposing a variable to the solution space we will have an invertible matrix and subsequently a unique solution [2]. The cost function of the Ridge method is:

$$Q(\hat{\beta}) = \left\| \left(\hat{y} - \hat{X}\hat{\beta}\right) \right\|_2^2 + \lambda \left\| \hat{\beta} \right\|_2^2 = \sum_{i=1}^{N}(y_i - x_{ij}\beta_j)^2 + \lambda \sum_{i=1}^{p} \beta_i^2 \tag{5}$$

In Ridge regression, we needs to have $\lambda \neq 0$. By setting $\lambda = 0$ we have ordinary least squares and as small as possible errors. But non-zero values of λ provides β_j results more reasonable predictors for those values of x which are not in training data set[3].

$$\frac{\partial Q(\hat{\beta})}{\partial \hat{\beta}} = 0 = \hat{X}^T\left(\hat{y} - \hat{X}\hat{\beta}\right) - \lambda\hat{\beta} \tag{6}$$

and then gives:

$$\left(\hat{X}^T\hat{X} + \lambda I\right)^{-1}\hat{\beta} = \hat{X}^T\hat{y} \tag{7}$$

finally, closed form of $\hat{\beta}_{\text{Ridge}}$ is written as:

$$\hat{\beta}_{\text{Ridge}} = \left(\hat{X}^T\hat{X} + \lambda I\right)^{-1}\hat{X}^T\hat{y}. \tag{8}$$

Furthermore, the relevant variance could be analytically determined by:

$$\text{Var}\left[\hat{\beta}^{Ridge}\right] = \sigma^2\left[X^TX + \lambda I\right]^{-1}X^TX\left(\left[X^TX + \lambda I\right]^{-1}\right)^T \tag{9}$$

2.3 Lasso

Generally, the cost function of Lasso (Least Absolute Shrinkage and Selection Operator) is written as:

$$Q(\hat{\beta}) = \left\| \left(\hat{y} - \hat{X}\hat{\beta} \right) \right\|_2^2 + \lambda \left\| \hat{\beta} \right\|_1 = \sum_{i=1}^{N} (y_i - x_{ij}\beta_j)^2 + \lambda \sum_{i=1}^{p} |\beta_i| \tag{10}$$

The only difference between Lasso and Ridge is, we take L_1-norm instead of L_2-norm in Lasso regression. This brings an advantage for Lasso regression which is, certain values of λ gives sparse solution, namely $\beta_j = 0$. As we did for other methods, it is expected to get a gradient of the cost function and set it to zero. However, as it could be seen here, the absolute function is not differentiable at zero. The derivative of absolute function is defined as:

$$\frac{d|x|}{x} = \text{sgn}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{11}$$

then, the gradient of the cost function which is mentioned above is written as:

$$\nabla Q = -2\hat{X}^T(\hat{y} - \hat{X}\hat{\beta}) + \lambda \, \text{sgn}(\hat{\beta}), \tag{12}$$

The mentioned constraint results in nonlinear solutions and there is no closed form expression of Lasso [2]. Thus, to find the minimum of cost function a dedicated minimum algorithm should be applied to find proper β_j. It seemss, the gradient descent algorithm with a proper step is convenient and Newton's method which includes **Hessian** of the cost function is another alternative. Actually, Hessian matrix for cost function of Lasso, is the second derivative of the OLS cost function, namely:

$$H_{kl} = H_{lk} = \frac{\partial^2 Q_\lambda}{\partial \beta_l \beta_k} = \frac{\partial}{\partial \beta_l}(\nabla_k Q) = \frac{\partial}{\partial \beta_l}(-2x_{ik}(y_i - x_{ij}\beta_j)) = 2x_{ik}x_{il}, \tag{13}$$

in matrix form, Hessian is written as:

$$H = 2\hat{X}^T\hat{X}. \tag{14}$$

then, for gradient we will have:

$$\nabla Q_\lambda = -2\hat{x}^T\hat{y} + H\hat{\beta} + \lambda \, \text{sgn}(\hat{\beta}). \tag{15}$$

According to the theory, it is obvious that large values of λ will force some of the β_j parameters to zero which causes non-differentiable cost function. Here, for minimizing Lasso cost function we used **scikit-learn**. Nevertheless, as a computational discussion we know evaluation of the ∇Q_λ incorporates with matrix-vector computation with a FLOPS of $\mathcal{O}(p^2)$. Due to using the second derivative, the cost function is independent of $\hat{\beta}$. Thus, we can use Cholesky method for decomposition of Hessian. Cholesky method has FLOPS of $\mathcal{O}(p^3)$. Therefore, after using Cholesky only $\mathcal{O}(p^2)$ FLOPS per iteration is required for solving set of linear equations.

2.4 Minimization

Minimization of the cost function is a crucial part of any regression method. For OLS and Ridge regression there are analytical solutions to find β_j by minimizing the formula for $Q(\hat{\beta})$. But, some methods, such as Lasso regression, need to use other minimization algorithms.

2.4.1 Newton's method

We use Taylor expansion of ∇Q around the point of $\hat{\beta}_0$. If we write Taylor expansion at a point $\hat{\beta}_0 + \delta\hat{\beta}$ then we have:

$$\nabla Q(\hat{\beta}_0 + \delta\hat{\beta}) = \nabla Q(\hat{\beta}_0) + H(\hat{\beta}_0)\delta\hat{\beta}, \tag{16}$$

here $H(\hat{\beta}_0)$ is the derivative of ∇Q, namely the Hessian of Q, evaluated at β_0. therefore, we have:

$$H(\hat{\beta}_0)\delta\hat{\beta} = -\nabla Q(\hat{\beta}_0), \tag{17}$$

and, in general, solving

$$H(\hat{\beta}_i)\delta\hat{\beta}_i = -\nabla Q(\hat{\beta}_i), \tag{18}$$

by defining $\hat{\beta}_{i+1} = \hat{\beta}_i + \delta\hat{\beta}_i$, the process can be continued until convergence.

2.4.2 Gradient descent

Some times it is not efficient to evaluate Hessian matrix. To circomvent this problem, we replace Hessian with a number which is shown as $1/\alpha$. Therefore, equation (18) is rewritten as

$$\hat{\beta}_{i+1} = \hat{\beta}_i - \alpha\nabla Q(\hat{\beta}_i), \tag{19}$$

To interpret this coefficient geometrically, we say, moving a small step in the opposite direction of the gradient gives reduction in value of cost function $\hat{\beta}$ will approach to minimum value. If we have a convex function function, small steps results convergence but slowly while larger steps may not give convergence. It is worth to note that there is no analytical solution to calculate the variance of Lasso β as we had for OLS and Ridge. Therefore, we have to run the program hundreds of time and then find the mean value and relevant variance.

2.5 Resampling methods

In this study Franke's function is used as a data set and it is modeled via OLS, Ridge and Lasso. Resampling methods are techniques that improves the accuracy of prediction and let us find estimation for variance of β via dividing the data set to **training data** and **test data**. Often you would also split off a part of the data set into **validation data**. This validation data is then used to evaluate the final model, found by training and testing on the train and test data sets. The reason for this is that evaluation becomes more biased when skill on the validation dataset is incorporated into the model configuration [4]. Note that which set is called the validation set and which is the test set varies from source to source. These techniques are mostly applied in two cases, first when data collection is difficult and it is expensive to gather new data. Second, our model could be overfitted

which means the training data is very close to the data set and starts fitting noise in the data, such that predicting any new data might fail. The idea of resampling is very simple which is required to split the data set into training and testing data. For this purpose, we can use K-fold cross-validation and bootstrapping methods.

2.5.1 K-fold

The approach for K-fold is straight forward, such that instead of perform a regression on the entire data set, first we divide the data set into K number of subsets with equal size(the last set may be smaller if the data set doesn't divide with the number of folds). It is important to note that the should be shuffled (randomized) before distributing them to the subsets. Now we have a subset as 'validation' set while the rest are the 'training' sets as it is shown in figure (1).

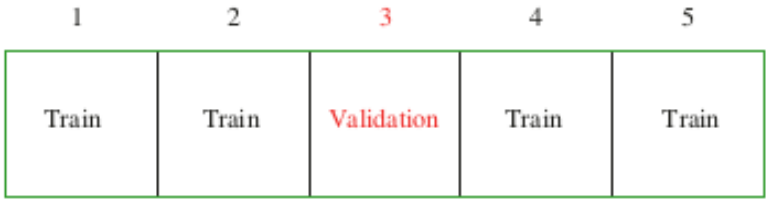

Figure 1: This figure illustrates how the training data and test data are splited in resampling [2].

2.5.2 Bootstrap

In Bootstrapping, the random selection of data set is carried out with replacement. Random selection with replacement means that each value may appear multiple times in the one sample.

2.6 Bias and variance trade off

In data analysis using supervised learning methods, such as the models studied here, OLS, Ridge and Lasso, will have a conflict between minimizing two of the sources of error namely bias and variance. Minimizing one means maximizing the other. Therefore, the best performance happens when we find a balance between the bias and variance, where the mean squared error (or cost function) is at its minimum. Below is the equation for error estimation, mean squared error:

$$\begin{aligned} E[(y-\hat{f}(x))^2] &= \left(E[f(x)-\hat{f}(x)]^2\right) + \left(E[\hat{f}(x)^2] - E[\hat{f}(x)]^2\right) + \sigma^2 \\ &= \text{Bias}(\hat{f}(x))^2 + \text{Var}(\hat{f}(x)) + \sigma^2 \end{aligned} \tag{20}$$

σ^2 is the first term representing the noise and it is indisputable. The second term is the deviation of the average prediction from the true value, noise-free model, which is the (squared) bias. The last term is the so-called variance and shows how much the predictions vary Mathematically, we can

say if we have $D = (\vec{x}_i, y_i)_{i=1}^N$ as the data set, which is produced by a function such as Franke's function, then we have:

$$y_i = f(\hat{x}_i) + \varepsilon_i = f_i + \varepsilon_i$$

where the variables ε_i are independent and normally distributed with variance σ^2 around zero. By using any regression method it is possible to predict $\tilde{y}_i$ from the data set then, the expected mean square error will be:

$$\begin{aligned}
E_{D,\varepsilon}\left[\left\|\hat{y}-\hat{\tilde{y}}_D\right\|_2^2\right] &= E_{D,\varepsilon}\left[\left\|\hat{y}-\hat{f}+\hat{f}-\hat{\tilde{y}}_D\right\|_2^2\right] \\
&= E_{D,\varepsilon}\left[\left\|\hat{y}-\hat{f}\right\|_2^2+\left\|\hat{f}-\hat{\tilde{y}}_D\right\|_2^2+2(\hat{y}-\hat{f})\cdot(\hat{f}-\hat{\tilde{y}}_D)\right] \\
&= E_{D,\varepsilon}\left[\left\|\hat{y}-\hat{f}\right\|_2^2\right]+E_{D,\varepsilon}\left[\left\|f-\hat{\tilde{y}}_D\right\|_2^2\right]+2E_{D,\varepsilon}\left[(\hat{y}-\hat{f})\cdot(\hat{f}-\hat{\tilde{y}}_D)\right] \\
&= E_{D,\varepsilon}\left[\|\hat{\varepsilon}\|_2^2\right]+E_{D,\varepsilon}\left[\left\|\hat{f}-\hat{\tilde{y}}_D\right\|_2^2\right]+2E_{D,\varepsilon}\left[\hat{\varepsilon}\cdot(\hat{f}-\hat{\tilde{y}}_D)\right] \\
&= \sigma^2+E_{D,\varepsilon}\left[\left\|\hat{f}-\hat{\tilde{y}}_D\right\|_2^2\right]+0\cdot E_D[(\hat{f}-\hat{\tilde{y}}_D)] \\
&= \sigma^2+E_D\left[\left\|\hat{f}-\hat{\tilde{y}}_D\right\|_2^2\right]
\end{aligned}$$

adding and subtracting the average prediction, $E_D\left[\hat{\tilde{y}}_D\right]$ gives:

$$\begin{aligned}
E_D\left[\left\|\hat{y}-\hat{\tilde{y}}_D\right\|_2^2\right] &= \sigma^2+E_D\left[\left\|\hat{f}-E_D\left[\hat{\tilde{y}}_D\right]+E_D\left[\hat{\tilde{y}}_D\right]-\hat{\tilde{y}}_D\right\|_2^2\right] \\
&= \sigma^2+\left\|\hat{f}-E_D[\hat{\tilde{y}}_D]\right\|_2^2+E_D\left[\left\|E_D\left[\hat{\tilde{y}}_D\right]-\hat{\tilde{y}}_D\right\|_2^2\right] \\
&\quad +2E_D\left[\left(\hat{f}-E_D\left[\hat{\tilde{y}}_D\right]\right)\cdot 0\right. \\
&= \sigma^2+\left\|\hat{f}-E_D\left[\hat{\tilde{y}}_D\right]\right\|_2^2+E_D\left[\left\|E_D\left[\hat{\tilde{y}}_D\right]-\hat{\tilde{y}}_D\right\|_2^2\right] \qquad (21)
\end{aligned}$$

As it is seen above, the same result as equation (20) is given mathematically. The complex model minimizes the second term due to better fitting the true model (f). Lastly, the third term increases with model complexity. Because, the fitting procedure is more sensitive to noise in the different data sets. The equation (21) that shows bias-variance decomposition needs to have the underlying model namely f. More mathematical computation including adding and subtracting $E_D\left[\vec{\tilde{y}}_D\right]$ gives the expression as:

$$E_D\left[\left\|\hat{y}-\hat{\tilde{y}}_D\right\|_2^2\right] = \left\|\hat{y}-E_D\left[\hat{\tilde{y}}_D\right]\right\|_2^2+E_D\left[\left\|E_D\left[\hat{\tilde{y}}_D\right]-\hat{\tilde{y}}_D\right\|_2^2\right] \qquad (22)$$

This is another form of the bias-variance formula which does not need any information about the underlying model. Division by N gives MSE on the left-hand side.

The bias – variance tradeoff is also affected by model complexity, number of data points, training set and testing set. When we fit a function to our data and start by exploring a function with low complexity p, we might find that the bias is high and the variance relatively low (underfitting). This leads to the problem of the model ignoring important features of the training data. Conversely if p becomes too large, our function will follow random fluctuations in the data (overfitting), leading to an increase in variance and no important gains in bias. If we increase the complexity too much we'll reduce the bias and increase the variance [5]. A model with higher complexity is not able to predict the reasonable values for test data. Therefore, we strive to find the best balance between bias and variance. Number of data points will also affect this balance, too small data set and you're in the danger of underfitting. If this is the case, resampling methods can be applied. If we have a large data set we can apply other strategies to avoid the opposite problem, overfitting. One strategy is to split our data in training and test sets (sometimes training, test, and validation sets). A randomly selected portion of our data is used as training set, is used to fit the various candidate models. The remaining portion is used as test set to evaluate the performance of the available models and choose the most accurate one. This can be used to find the best value of p.

2.7 Mean Squared Error (MSE) and R^2 score

In all regression methods, the value of the cost function is minimized, but its value is not necessarily meaningful. To be specific, the cost functions in Ridge and LASSO regression depend on the hyperparameter λ,and to determine the optimal λ one runs the evaluation process over a range of λ values. There are other functions, which are used to interpret the performance of a given regression method.

Mean square error (MSE) which should be as small as possible and it is defined as:

$$\text{MSE}(\hat{\beta}) = \frac{1}{N}\left\|\hat{y} - \hat{\tilde{y}}\right\|_2^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2, \tag{23}$$

The value of **MSE** presents the quality of our prediction. As it is presented in equation (23), this value prospects the average of the square of the residuals. Apparently, from definition in equation (23), it could be realized that MSE can never be negative and lower MSE value means a better prediction. For regression evaluation the R^2 is considered as a golden standard to assess quality of fitting. R^2 score which is defined as:

$$\text{R}^2(\hat{\beta}) = 1 - \frac{\left\|\hat{y} - \hat{\tilde{y}}\right\|_2^2}{\|\hat{y} - \bar{y}\|_2^2} = 1 - \frac{\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{24}$$

Here, y_i are the data we want to fit (indexed response variables), $\bar{y}$ is the mean of the measured values and $\tilde{y}_i$ is the predictor variables (from model). In addition, we define ϵ_i as $\epsilon_i = y_i - \tilde{y}_i$. As it is shown in equation (24), to calculate R^2, a fraction of **MSE** over **Variance** should be deduced from one. The average of the response variables is denoted $\bar{y}_i$. Let's interpret the formula step by

step: if the residual sum of squares becomes low so, we have a good fit. But, it should be compared with the spread of response variables. After all, if the response variables are all nicely distributed close to the mean then getting a low value for residual sum of squares could not be impressive. In the other hand, if the model fits well, then residual sum of squares would be zero and the R^2 becomes one. In this sense we have a span of possible R^2 scores between zero and one. Therefore, the R^2 value shows us how good our model is at predicting future samples.

2.8 Implementations

In this project three regression methods i.e. OLS, Ridge and Lasso are implemented.

Comments on limitations in the code

When writing the code, there were a couple of shortsighted implementations, and fixing these might affect the total code adversely. Therefore, we leave it as is due to time constraints, and list the problems below.
One issue is that the program was created considering only square data sets, so that dimensions in the x and y direction must be the same.
The other issue is that the program is not very granular, in that if the length selected for x and y does not divide nicely for k-fold splitting, dimensions may not add up while calculating cost functions. As above, changing this could affect the whole rest of the program. It is thus left as is, with a warning to the user that errors may occur!

2.8.1 Ordinary Least Squares

Initially, coded equation (1) which calculates β value by applying OLS method. In this method we used usual inversion matrix from numpy package as it is shown in the following code:

```
beta = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(z_n)
```

Then, coded equation (2) to calculate the variance of β multiplied by σ^2 as it is presented bellow:

```
def VarOLS_betas(X, sigma):
        covar = np.linalg.inv(X.T.dot(X))
        vari = np.diagonal(covar)
        return vari*(sigma**2)
```

Afterward, we applied a fruitful linear algebraic decomposition namely SVD for OLS [1], in order to reach higher order polynomials, as it is shown below:

```
def ols_svd(X:np.ndarray, z:np.ndarray ,_lambda,method='ols')->np.ndarray:
    u, s, v = scl.svd(X)
    if method=='ols':
        return v.T @ scl.pinv(scl.diagsvd(s, u.shape[0], v.shape[0])) @ u.T @ z
```

The relevant results are printed in tables 1,2,3,4 and 5 and discussed in section (3.1).

2.8.2 Ridge Regression

we coded equation (8) and equation (9) which calculates β and β-variance for Ridge via inversion matrix as printed bellow:

```
def VarRidge_betas(X,_lambda,sigma):
    XX = X.T@X
    invers = np.linalg.inv(XX+_lambda*np.eye(len(XX)))
    return np.diagonal(invers)*(sigma**2)
```

The relevant results are printed in tables 6,7,8,9 and 10 and discussed in section (3.3).

2.8.3 LASSO Regression

Lasso regression also is implemented via using `sklearn` with k-fold as it is shown as a pseudo code in algorithm (1). Moreover, the relevant results are printed in tables 11,12,13,14 and 15 and discussed in section (3.4)

Algorithm 1: Lasso Regression with k-fold

if method == Lasso
    lassoing $=$ **Lasso(alpha**$=$ **lambda, fit-intercept = False, tol** $= 10^{-7}$**, max-iter**$= 10^{9}$**)**
    lassoing.fit(X-train,z-train) ▷ **lassoing**$\rightarrow$ **to have different name by sklearn default**
    β **= lassoing.coef** ▷ **collecting** β

2.8.4 Bias variance trade off

Calculating the bias-variance trade off proved a challenge using k-fold resampling, so we decided to use bootstrap. Bellow we outline bootstarp methods used in the attempt to get a valid bias-variance as it is shown as a pseudo code in algorithm (2). Furthermore, the relevant graphs are plotted in figure (6) and discussed in section (3.2).

Algorithm 2: Bootstrap for bias variance trade off

Input (N) ▷ **N**$\rightarrow$ **Bootstrap Iteration No.**
n = length (training set)
for i=0 : N
    sampling from D(x,y) with replacement ▷ **D(x,y)**$\rightarrow$ **refers to data set**
    cal. predicted MSE, Bias ,Var
end for
cal. mean-value over N

2.8.5 K-fold cross validation

For implementing K-fold cross-validation we made a main function so-called `kFolds`. This main function includes sub-functions of `Shuffler` and `kSplitter` for fulfillment of shuffling and

splitting respectively. Then, through a `for` loop the splitted data are centred and finally evaluation of MSE and R^2 is carried out. Algorithm (3) presents the pseudo code of above concept.

Algorithm 3: K-fold cross validation

main function (kFolds)
    call Shuffler sub-function
    call kSplitter sub-function
    for i : (N) ▷ **N→ refers to no. of folds**
        centering data
        cal. MSE, R^2
    end for
Return mean value of MSE,R^2 over N

One important point that should not be forgotten is, for regularization methods such as Ridge and LASSO, one must first center the data. By centering the data set (around the mean), you will subtract the mean, to ensure that $E[y_i] = 0$. Thus, we make sure that the algorithm is not biased by the inclusion of the mean of the training data set.
Centering is done by subtracting the mean of each column from it's respective column, and subtracting the mean of the targets from the target values. Note that the first column of the model matrix now will be zeros, creating a singular matrix, so this column should be deleted. This also makes all $\beta_0 = 0$, so they will not be included when finding confidence intervals of β values.
The value to center around, the mean in our case, is taken from the training data set and for centering test data, we subtract the mean again. Centering is implemented in the code as following:

```
X_train = X_train_ - np.mean(X_train_,axis=0)
z_train = z_train_ - np.mean(z_train_)
X_test  = X_test_  - np.mean(X_train_,axis=0)
z_test  = z_test_  - np.mean(z_train_)
X_train = np.delete(X_train,0,1)
X_test = np.delete(X_test,0,1)
```

It is important to note that before calculating MSE or R^2, it is required to add the previously subtracted value back to the predicted target values.

2.8.6 Model Performance

The equation (23) which presents **MSE** definition is coded for test and train data inside `kFolds` main function as bellow:

```
mse_test_eval += np.mean((z_test_ - z_test_pred[:,c])**2,axis=0,keepdims=True)
mse_train_eval += np.mean((z_train_ - z_train_pred[:,c])**2, axis=0,
                  keepdims=True)
```

Furthermore, the equation (24) which presents $\mathbf{R}^2$ definition is coded for test and train data inside `kFolds` main function as bellow:

```
r2_test_eval +=1-np.sum((z_test_-z_test_pred[:,c])**2,axis=0,keepdims=True)
                /np.sum((z_test_-np.mean(z_test_)**2),axis=0,keepdims=True)
r2_train_eval +=1-np.sum((z_train_-z_train_pred[:,c])**2,axis=0,keepdims=True)
                /np.sum((z_train_-np.mean(z_train_)**2),axis=0,keepdims=True)
```

3 Results and discussion

3.1 OLS

In OLS, two methods for inverse matrix computation is used. The first one was usual matrix inversion and the second was using SVD decomposition. As it can be observed in figure 2 and 3 SVD decomposition facilitates to use a polynomial with higher degree. In addition figure (4) we notice the expected deviation between train and test data. Namely, the more complex model will learn the noise of training data leading to worse prediction. In Figure (5), without resampling, it is tried to show the effect of noise, which is marked by **n**, on the MSE.

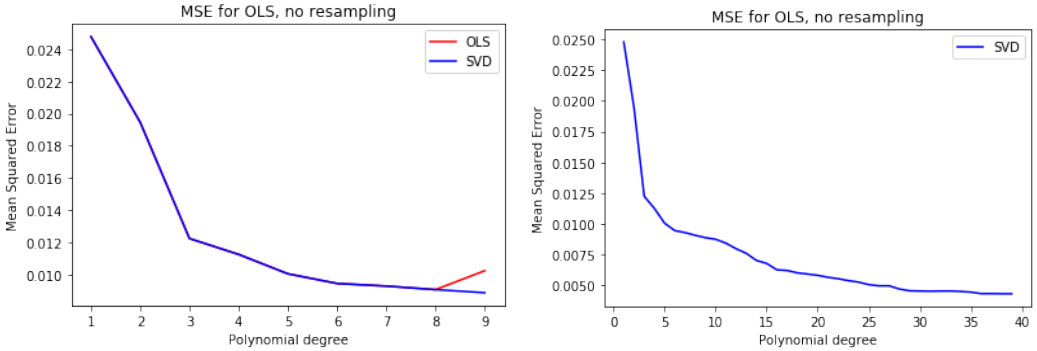

Figure 2: Comparison of MSE for OLS method with using usual inverse matrix and SVD decomposition.

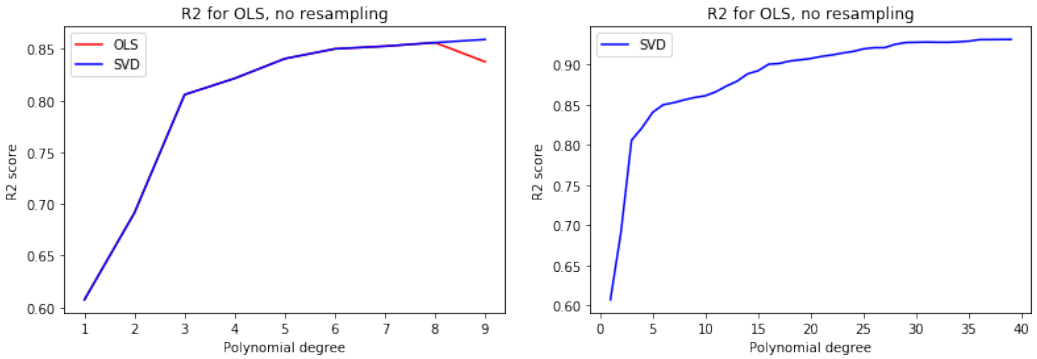

Figure 3: Comparison of R^2 for OLS method with using usual inverse matrix and SVD decomposition.

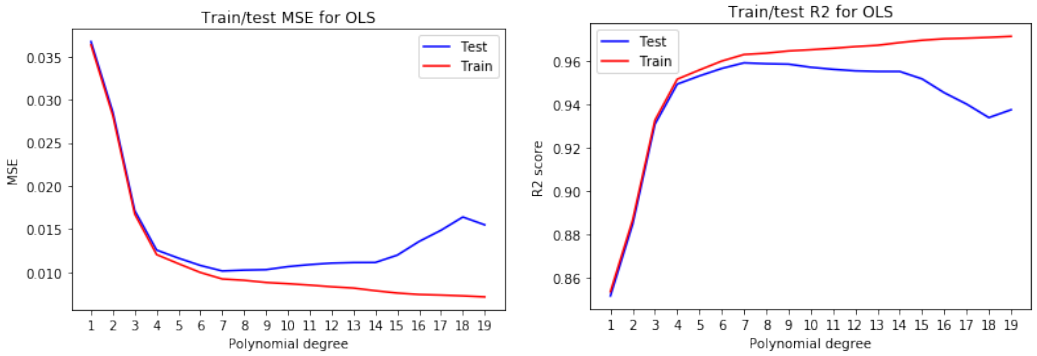

Figure 4: Illustration of MSE and R^2 for OLS method including test and train data with resampling technique.

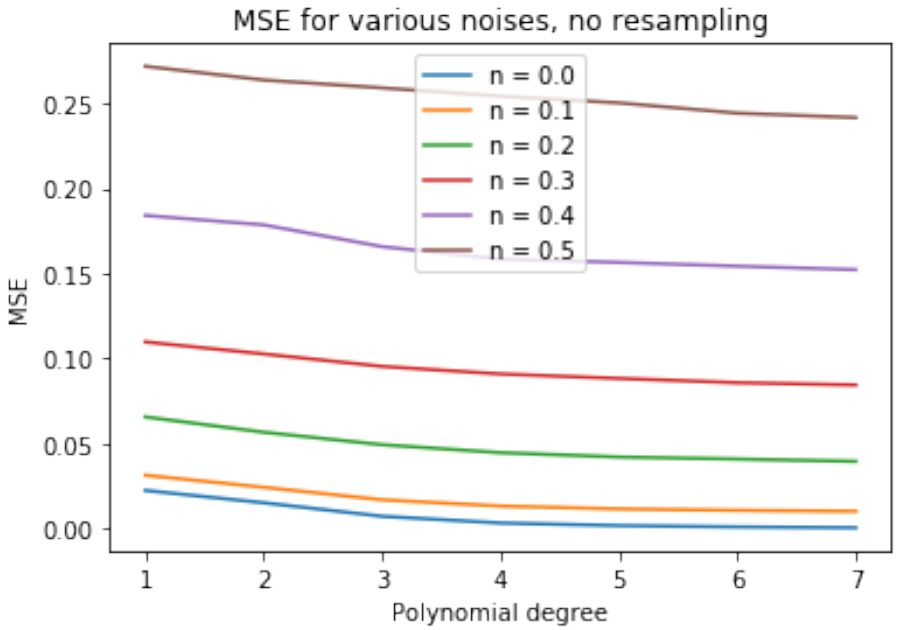

Figure 5: This figure explores the effect of the stochastic noise on MSE for Franke's function. Using OLS method without resampling.

Table 1: OLS method, polynomial degree 1

Coefficient	Confidence interval = $\beta_{mean} \pm 2{*}\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-0.8120637	0.0016981
β_2	-0.5090383	0.0026968

Table 2: OLS method, polynomial degree 2

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-2.2701904	0.038767
β_2	0.9549401	0.0273738
β_3	0.0972064	0.0506055
β_4	1.2761754	0.0373837
β_5	-1.7208892	0.0521155

Table 3: OLS method, polynomial degree 3

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-5.8379154	1.1390995
β_2	7.7571017	4.5733096
β_3	-3.7583408	1.5879232
β_4	0.9292325	0.4334452
β_5	3.1313521	1.2400603
β_6	-1.5624547	0.6040347
β_7	-5.6322713	2.099585
β_8	-0.1538759	0.7278582
β_9	3.30478	1.2159054

Table 4: OLS method, polynomial degree 4

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-3.8678853	44.4785117
β_2	0.8295333	418.9885384
β_3	7.776172	625.4456379
β_4	-6.1867029	113.7415205
β_5	3.4510194	2.7142567
β_6	3.0935539	36.5902364
β_7	-15.3990628	111.7753067
β_8	8.959124	36.5117611
β_9	-14.2196957	38.3532326
β_{10}	20.2113342	39.607065
β_{11}	-1.5138869	11.8712746
β_{12}	6.0856737	126.6070448
β_{13}	-15.5039699	16.89626
β_{14}	4.3834358	51.4322346

Table 5: OLS method, polynomial degree 5

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	43.7705189	697.4239451
β_2	-170.220085	10550.6456961
β_3	285.0178338	33576.3469056
β_4	-216.7843553	24103.5291587
β_5	61.1322632	2584.725178
β_6	13.4067284	28.5022744
β_7	-81.700258	1449.1467921
β_8	201.2193161	11462.3570988
β_9	-214.4461846	15435.2088038
β_{10}	81.1884193	2695.6115334
β_{11}	-26.2018974	729.2309321
β_{12}	87.618918	1413.7417741
β_{13}	-94.2332346	2567.1543296
β_{14}	42.6312212	743.9501859
β_{15}	12.8337397	7404.0392154
β_{16}	-55.4741576	2014.6198219
β_{17}	17.2872233	280.6549939
β_{18}	10.6663092	15145.8319818
β_{19}	13.5955444	708.2450101
β_{20}	-7.5563235	4087.7887192

3.2 Bias variance trade off

In top part of figure (6) we illustrate the ideal behaviour of bias, variance and `MSE` in theory. Below it, we have plotted the `MSE` together with bias and variance for our fit to the Franke function based on the method that is described in section (2.8.4). Using this plot we can determine which model has the lowest mean squared error thus getting a balance between the bias and the variance. In this case we see that the mean squared error has a dip at what looks like polynomial degree 7. It looks like the model is saturated, meaning that the model cannot learn more from the data set, after degree 12 and we start seeing evidence of overfitting with low bias and increasing variance as the model is fitting to noise in the data. Note that this figure was picked out of twenty plots with various data set sizes due to it's ideal portrayal of the bias variance trade off!

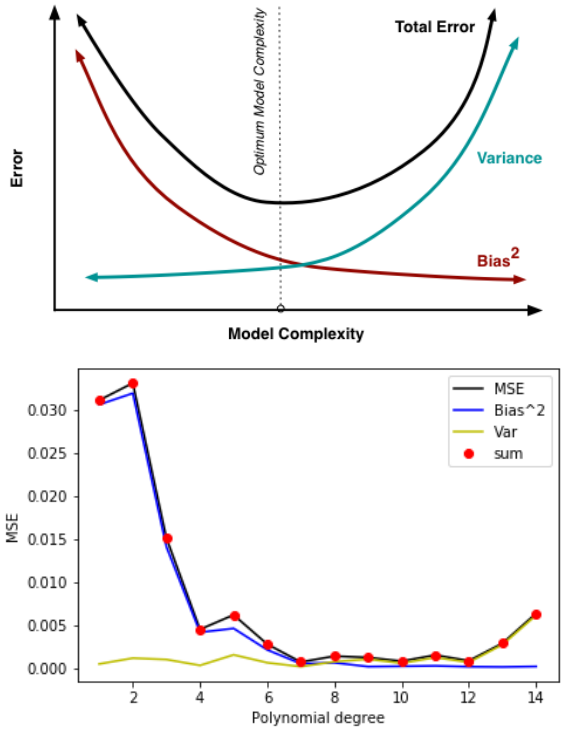

Figure 6: Bias variance trade off illustration. The figure in top is ideal and theoretical sketch of bias variance behavior (ref:`http://scott.fortmann-roe.com/docs/BiasVariance.html`). The second figure is illustration of bias variance trade off using OLS on constructed data with the Franke function using bootstrap to calculate MSE, bias and variance. This method is described in section (2.8.4).

3.3 Ridge regression

Figure (7) shows MSE and R^2 behaviour with respect to polynomial degree by using resampling technique. We can see up to polynomial degree four, train and test plots overlap and they deviate afterward. The reason is, train data learns the noise on the data set, and for more complex models due to overfitting the `MSE` trend to zero. On the other hand, for a data set on the same form, but with different noises (test data), the model will fit worse the more it overfits. The β parameters, and their 95% confidence interval, for Ridge regression on Franke's function are printed in tables (6,7,8,9,10). Figure (8) presents MSE behaviour with respect to different polynomial degree at various λ values. The value of MSE is minimum when $\lambda = 5.6e-7$ (red curve) and polynomial degree is between 5 and 10. But for polynomial degrees exceeding 11, it can be seen that $\lambda = 1e-5$ (purple curve) is the one that minimizes the MSE.

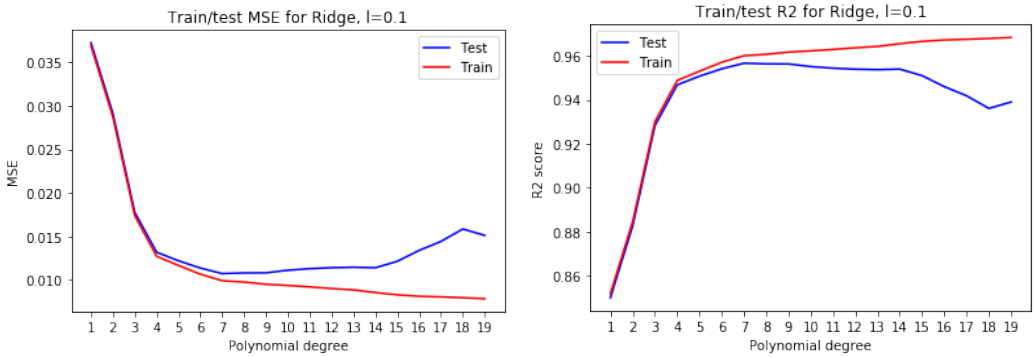

Figure 7: Illustration of MSE and R^2 for Ridge method including test and train data with resampling technique. Here $\lambda = 0.1$

Table 6: Ridge method, polynomial degree 1

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-0.8012228	0.0016942
β_2	-0.5022427	0.0026869

Table 7: Ridge method, polynomial degree 2

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-2.2398837	0.035543
β_2	0.9421918	0.0252954
β_3	0.0959087	0.0446809
β_4	1.2591386	0.0347637
β_5	-1.6979156	0.046676

Table 8: Ridge method, polynomial degree 3

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-5.7599801	0.1204545
β_2	7.6535456	0.4244766
β_3	-3.7081675	0.1623068
β_4	0.9168274	0.0938681
β_5	3.089549	0.32629
β_6	-1.5415961	0.2296172
β_7	-5.5570813	0.3632077
β_8	-0.1518217	0.2729143
β_9	3.2606617	0.2599308

Table 9: Ridge method, polynomial degree 4

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	-3.8162496	0.1521785
β_2	0.8184591	0.4302951
β_3	7.6723613	0.4540348
β_4	-6.1041114	0.2417359
β_5	3.4049488	0.1098387
β_6	3.0522554	0.3307088
β_7	-15.1934877	0.4930858
β_8	8.839521	0.3223182
β_9	-14.0298649	0.3752065
β_{10}	19.9415159	0.4909576
β_{11}	-1.4936768	0.4622582
β_{12}	6.0044309	0.4671236
β_{13}	-15.2969942	0.4555009
β_{14}	4.3249176	0.385747

Table 10: Ridge method, polynomial degree 5

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	σ_{mean}
β_1	43.1861892	0.1548577
β_2	-167.9476728	0.4418717
β_3	281.212889	0.475492
β_4	-213.8903172	0.4943329
β_5	60.3161568	0.288428
β_6	13.2277506	0.1120666
β_7	-80.609572	0.3562734
β_8	198.5330688	0.5326374
β_9	-211.5833607	0.5204628
β_{10}	80.1045663	0.4174515
β_{11}	-25.852106	0.3826369
β_{12}	86.4492188	0.5281323
β_{13}	-92.9752353	0.5962702
β_{14}	42.0621009	0.5178181
β_{15}	12.6624112	0.5128513
β_{16}	-54.733586	0.5745178
β_{17}	17.0564415	0.5614923
β_{18}	10.5239156	0.5097852
β_{19}	13.414046	0.5535765
β_{20}	-7.4554477	0.4925491

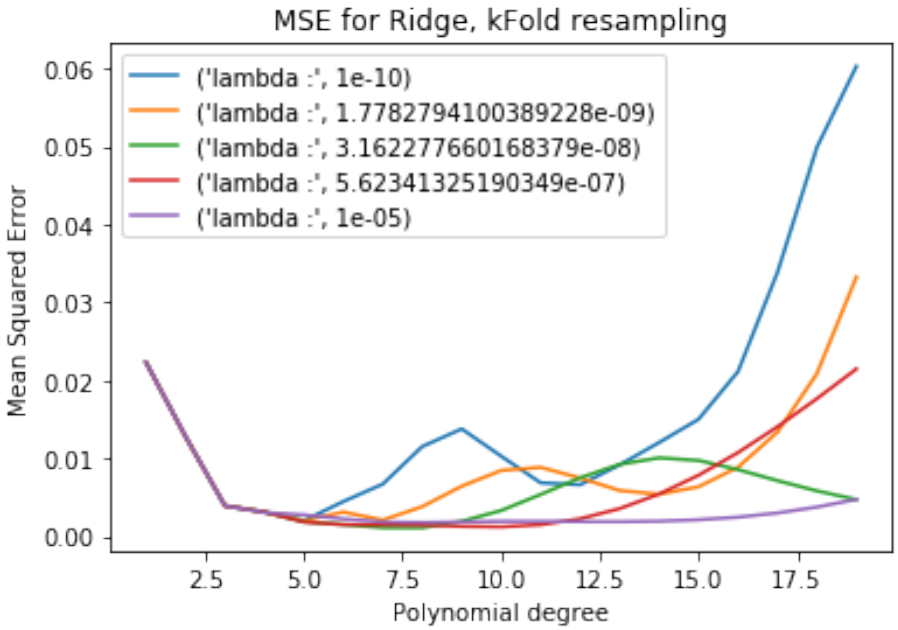

Figure 8: Illustration of MSE behaviour with respect to different polynomial degree at various λ values. The value of MSE is minimum when $\lambda = 5.6e-7$ (red curve) and polynomial degree is between 5 and 10. But for polynomial degrees exceeding 11, it can be seen that $\lambda = 1e-5$ (purple curve) is the one that minimizes the MSE.

3.4 Lasso regression

The printed the β parameters with 95% confidence interval, for Lasso regression on Franke's function are tabulated in tables (11,12,13,14) and (15). It is worth to note that in table (15), the LASSO method has forced coefficient β_4 to zero, thus β_4 is disappeared. This is due to the automatic feature selection property of Lasso regression. In figure (9), calculated `MSE` and R^2 for Lasso method are presented with respect to polynomial degree at $\lambda = 0.0001$. As it can be seen the train and test data have the same behaviour up to around polynomial degree three. However, They deviate in more complex models. The reason is simple, as it has been mentioned earlier, model learns the noise and fits to higher polynomial degree with smaller `MSE` due to overfitting. The choice of $\lambda = 0.0001$ was made for initial execution of code and it is not an optimized value. To estimate the optimized value of λ, figure (13) is plotted which shows `MSE` variation for test data versus polynomial degree. In this figure each color belongs to a specific λ value. Here, the left figure shows the results of LassoCV (cross validation is carried out by scikitlearn itself). The right figure shows the results of Lasso (cross validation is performed using the code that we developed). As it could be seen the ranges of `MSE` in both plots are approximately similar.

Table 13: LASSO method, polynomial degree 3

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	$2*\sigma_{mean}$
β_1	-5.2406627	0.1136768
β_2	6.5553014	0.1740586
β_3	-3.0699937	0.0948168
β_4	0.7769565	0.1529274
β_5	2.9926657	0.2401626
β_6	-1.4259681	0.1745757
β_7	-5.1152699	0.2437003
β_8	-0.1512917	0.2313806
β_9	2.8904714	0.1703509

Table 11: LASSO method, polynomial degree 1

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	$2*\sigma_{mean}$
β_1	-0.8108641	0.0033872
β_2	-0.5062484	0.0046289

Table 12: LASSO method, polynomial degree 2

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	$2*\sigma_{mean}$
β_1	-2.264361	0.0157907
β_2	0.950219	0.0112648
β_3	0.0988542	0.0230772
β_4	1.2791781	0.0293351
β_5	-1.7250534	0.0231478

Table 14: LASSO method, polynomial degree 4

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	$2*\sigma_{mean}$
β_1	-4.3117265	0.3480649
β_2	3.9334044	1.011631
β_3	0.5792741	1.2400912
β_4	-1.7123863	0.53298
β_5	2.0608263	0.2391357
β_6	-0.0218436	0.5401111
β_7	-1.4435573	0.4747637
β_8	0.0093784	0.2481275
β_9	-7.517076	0.4398744
β_{10}	9.9471908	0.9622081
β_{11}	-0.1100235	0.4357784
β_{12}	0.0280185	0.2498239
β_{13}	-8.4619005	0.6717276
β_{14}	5.3313621	0.381844

Table 15: LASSO method, polynomial degree 5

Coefficient	Confidence interval = $\beta_{mean} \pm 2*\sigma_{mean}$	
	β_{mean}	$2*\sigma_{mean}$
β_1	-3.1833821	0.3351377
β_2	0.9625122	0.8586192
β_3	2.6524628	0.7879324
β_5	-1.853067	0.2251063
β_6	2.1963419	0.3003637
β_7	0.317427	0.7112762
β_8	-2.2377142	0.6571323
β_9	-0.0854882	0.260766
β_{10}	0.5133118	0.327317
β_{11}	-7.4207495	0.5323528
β_{12}	7.5008502	0.8457855
β_{13}	-0.0001916	0.0090451
β_{14}	0.7203894	0.7300029
β_{15}	-0.0116448	0.1630717
β_{16}	-0.0044055	0.086347
β_{17}	-1.9464006	1.2645272
β_{18}	1.7793998	1.0721749
β_{19}	-5.3451333	1.2759316
β_{20}	3.7619003	1.0739524

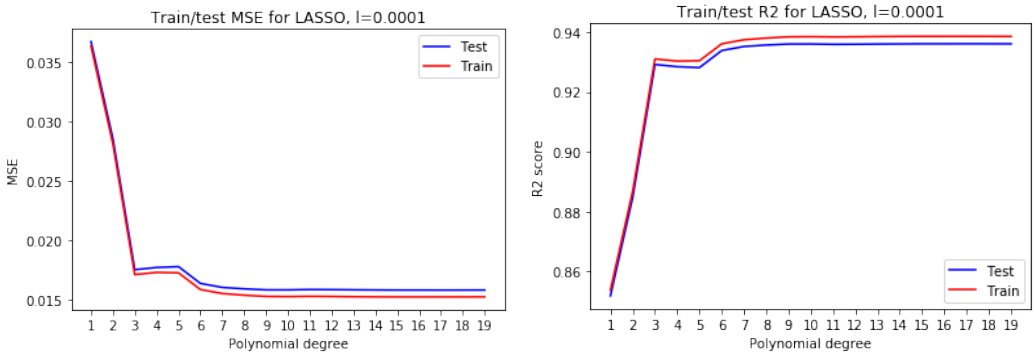

Figure 9: MSE and R^2 variation with respect to polynomial degree for Lasso method including test and train data with resampling technique.($\lambda = 0.0001$)

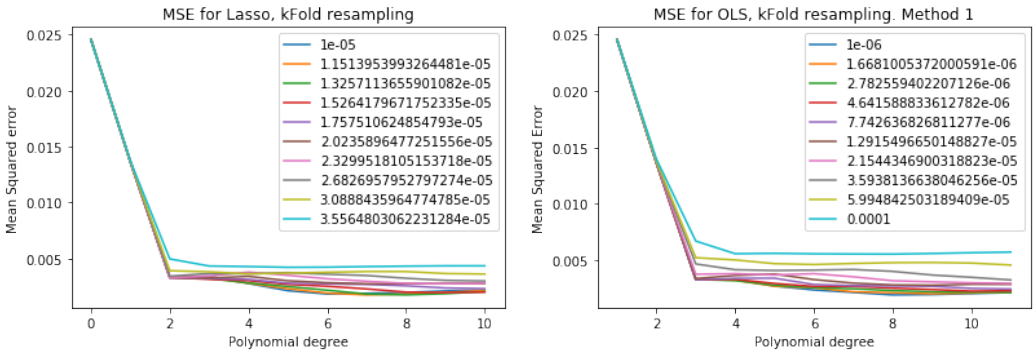

Figure 10: MSE with respect to polynomial degree based on Lasso regression. The left figure shows outcome of LassoCV (cross validation is carried out by scikitlearn itself). The right figure shows outcome of Lasso (cross validation is achieved by the code that we developed). As it could be seen the ranges of MSE in both plots are approximately similar

3.5 Comparison of OLS, Ridge and Lasso

In table (16), one can compare the performance of training and test data which are sampled from Franke's function based on OLS, Ridge and Lasso. In this regard, at the first glance one may expect the OLS shows better performance for trained data i.e. Ridge should result higher MSE and lower R^2 for trained data, but it does not. Therefore, for this data set with this parameters Ridge regression is the best fit. Beside of this, at the first run we considered **tolerance** $= 10^{-8}$ and **iteration** $= 10^9$ which needed 4 hours for convergence. Therefore, we decided to increase it to **tolerance** $= 10^{-3}$ with **iteration** $= 1000$.

Method	MSE(train)	R^2(train)	MSE(test)	R^2(test)
OLS	0.000585	0.988858	0.0021412	0.959281
Ridge	0.0002058	0.9960936	0.0011889	0.9778371
Lasso	0.0006745	0.9869749	0.002109	0.9501375

Table 16: Comparison the performance of OLS, Ridge and Lasso methods on Franke's function. Ridge regression shows the best fit.

Method	MSE	Bias+Variance	Bias	Variance
OLS	0.000124982	0.000124982	0.000119858	5.12393420e-06
Ridge	0.005489025	0.005489025	0.005431349	5.76760195e-05
Lasso	0.031897187	0.031897187	0.031727567	0.000169620

Table 17: Illustration of bias-variance trade off for Franke's function using Bootstrap method. For Ridge and Lasso the $\lambda = 0.01$, and polynomial degree is 10.

3.6 Earth data

Earth data is imported from the file SRTM_data_Norway_1.tif. Running the code on whole data set will take a long time, so it is required to select a part of the picture. But we should be careful that any small size of the picture does not have enough quality and we need to reduce the size with keeping the quality. To understand how the quality of picture changes with size, it is required to implement SVD decomposition on terrain data and study the variation of singular value plot. Figure (11) depicts the behaviour of singular value (σ) with respect to the matrix size. It can be observed that σ of terrain data experience a sharp reduction at around 20. Therefore, the size of the selected picture should no be less than 20X20. In our case, a suitable size of picture namely a square block with size of 100X100 was chosen. By a data set including a square block of 100X100, the code was executed for polynomial degree 1 to 10, with several values of λ. Here, for Lasso to convergence, it is required to center data first, then normalize it (Note that we did not normalize in this project, which was a major contributor to the LASSO code converging slowly, or not at all for small tolerances) and finally toggle the tolerance (scikit learn parameter for Lasso function) to 0.01. The relevant graphs that presents `MSE` for OLS, Ridge and Lasso are plotted in figure (12). As it is noticed in this figure the OLS and Ridge method follow each other closely, while LASSO stabilizes

at around MSE$= 1000$. Similar behaviour of OLS and Ridge is not surprising because the relevant λ value for Ridge is considered 10^{-5} which is very close to zero. It is worth to note that $\lambda = 10^{-5}$ is the best hyperparameter value for polynomial degree 10. The MSE for Ridge and OLS are close enough that one cannot make a proper selection between them without making more tests. Lasso prediction as it is illustrated in figure (12) and reported in table (18) is not reliable for earth data prediction. Table (18) confirms that OLS and Ridge are best fit for terrain data with $R^2 = 0.970$. Figure (13) and (14) illustrate variation of R^2 score with respect to polynomial degree and λ, based on Lasso and Ridge respectively which performed on terrain data .

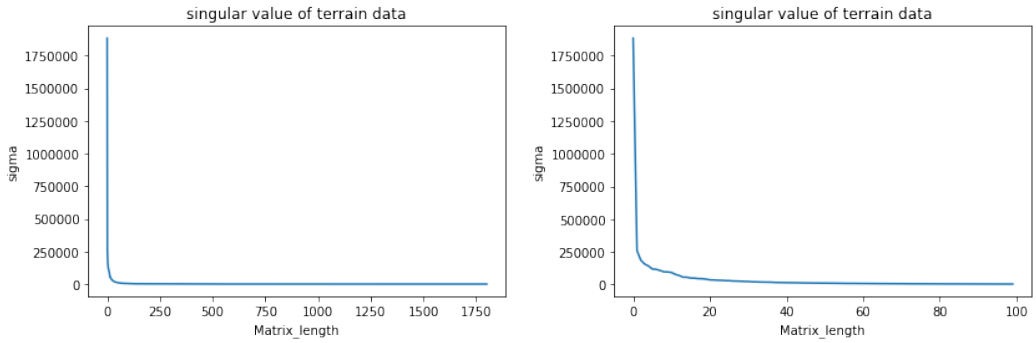

Figure 11: Variation of singular values (σ) of terrain data with respect to the number of elements. As it can be seen in left side the (σ) initially shows very sharp reduction and it is flatten til end. In the right side, it is depicted that the graph is stable from 20th element til 100th

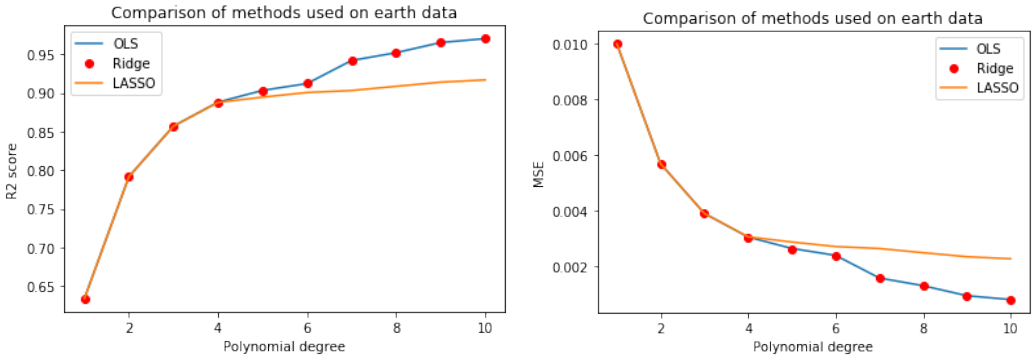

Figure 12: Comparison of MSE and R^2 for OLS, Ridge and Lasso as a function of polynomial degree on terrain data. OLS and Ridge show a complete overlap while Lasso decreases for some certain degrees of polynomial and finally stabilizes.

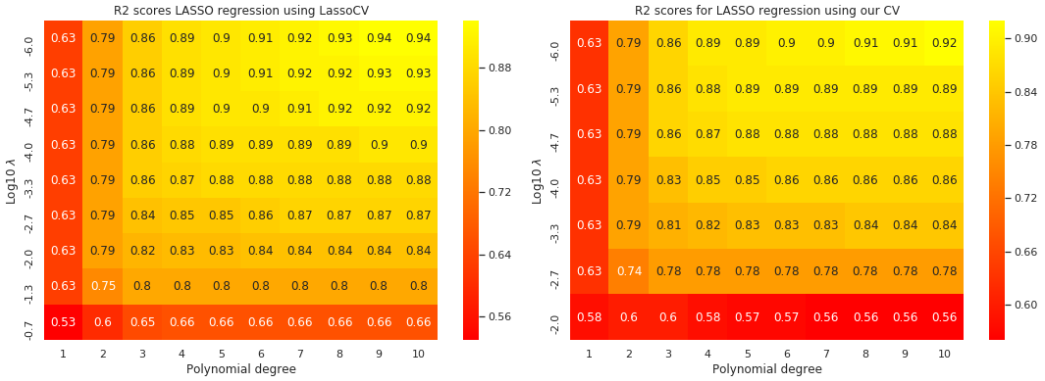

Figure 13: R^2 score with respect to polynomial degree based on Lasso regression performed on terrain data. The left figure shows outcome of LassoCV (cross validation is carried out by scikitlearn itself). The right figure shows outcome of Lasso (cross validation is achieved by the code that we developed). As it could be seen the ranges of MSE in both plots are approximately similar.

Method	MSE(train) [m]	R^2(train)	MSE(test)[m]	R^2(test)
OLS	142.13	0.971	145.17	0.970
Ridge	142.18	0.971	144.72	0.970
Lasso	997.64	0.795	999.58	0.795

Table 18: Illustration of performance for OLS, Ridge and Lasso methods. The data set includes a square block 100X100 from terrain data. For Ridge, the $\lambda = 10^{-5}$ which is the best hyperparameter for polynomial degree 10 and for LASSO $\lambda = 1$.

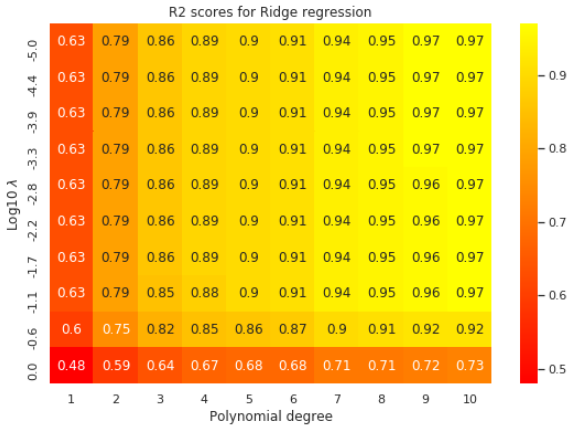

Figure 14: R^2score with respect to polynomial degree and λ based on Ridge regression performed on terrain data. This figure shows outcome of Ridge with cross validation (cross validation is achieved by the code that we developed).

4 Conclusion

We found in this study, in the first section Franke's function is used as a data set and then OLS, Ridge and Lasso techniques are applied to model the data set. In addition, resampling and cross validation are used to increase the performance of model (`MSE` and R^2) and investigate the validity of the model for unseen data set(test data). Beside of this, Bootstrap method is used to investigate the bias-variance trade off. Compared with Lasso and OLS, Ridge performs well for Franke's function considering the added stochastic noise.
Furthermore, it seems the gradient descent method that is used for minimizing the Lasso is inefficient due to this mathematical fact that Lasso is not differentiable. Therefore, execution of Lasso converges poorly compared with OLS and Ridge that work well in terms of performance. To make Lasso converge, it is required to increase the tolerance which leads to low accuracy.
Lastly, the geographical data set so-called earth data is modeled. For earth data Bootstrap is applied to check bias-variance trade off and K-fold is used for model validity investigation. It is realized that the performance of Ridge and OLS for prediction is the best and they both had promising results. However, Lasso performs poorly for earth data prediction.

References

[1] M. Hjorth-Jensen, Project 1, Department of Physics, University of Oslo, Norway, **2019**.

[2] J. Friedman, T. Hastie, R. Tibshirani, *The elements of statistical learning*, *Vol. 1*, Springer series in statistics New York, **2001**.

[3] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, D. J. Schwab, *Physics reports* **2019**.

[4] J. Brownlee, *Machine Learning Mastery* **2017**.

[5] A. Azzalini, B. Scarpa, *Data analysis and data mining: An introduction*, OUP USA, **2012**.