Project1: Study of linear regression methods

A. Mattiroli[1], A. Pera[1], and J. Štorek[2]

[1]University of Milano Bicocca
[2]Czech Technical University in Prague

September 2019

Abstract

In this report, a summary of results from a linear regression study performed on two data sets – 2D Franke function and Norway terrain data – can be found. Methods as Ordinary Least Squares (OLS), Ridge regression and Lasso regression have been applied in order to fit data with different degrees of polynomials. Mean Squared Error, bias and variance have been studied for each method. Ordinary Least Squares and Ridge emerge to be the best methods when dealing Franke function's data, while all the three methods happen to be equivalent for real data.

1 Introduction

This project aims to look at various regression methods including Ordinary Least Square (OLS), Ridge regression and Lasso regression. In the first part, the focus is on applying these methods to a two-dimensional function called Franke's function, using two sets of random generated values. Afterwards, resampling techniques such as Cross Validation and Bootstrap are introduced in order to better verify the estimated models, also including a study of the so-called Bias-Variance trade off. The second part of the project is an application of the methods previously explored on real digital terrain data. A background theory of the used techniques can be found in the methods section. A description of the implemented code of the study is contained in the implementation section. The analysis of results both for Franke function's data and real data can be read in the results and discussion section.

2 Methods

2.1 Linear Regression

Linear regression is a fitting procedure for predicting a quantitative response Y on the basis of explanatory variables X. If a relationship in the form $Y = f(X) + \epsilon$ is considered, it is assumed that f can be approximated by a linear function. The relationship between Y and X can be written mathematically as[1]

$$\vec{Y} = \mathbb{X}\vec{\beta} + \vec{\epsilon} \quad , \tag{1}$$

where $\vec{Y} = \begin{pmatrix} y_0 & y_1 & \dots & y_{n-1} \end{pmatrix}^T$ is the vector of the Y values, $\mathbb{X} \in \mathbb{R}^{n \times p}$ is the design matrix containing the information about the X variables, $\vec{\beta} = \begin{pmatrix} \beta_0 & \beta_1 & \dots & \beta_{p-1} \end{pmatrix}^T$ are regression parameters and $\vec{\epsilon} = \begin{pmatrix} \epsilon_0 & \epsilon_1 & \dots & \epsilon_{n-1} \end{pmatrix}^T$ is a normally distributed random error term with mean value zero.

In order to quantify the capacity of the model on fitting the data in terms of linear regression, two indices are generally used: the Mean Squared Error (MSE) and the R^2 (variance score) statistic.

The MSE calculates the mean value of the squared difference between data $\vec{y}$ and prediction $\vec{\tilde{y}}$:

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

If the predictions $\vec{\tilde{y}}$ obtained using the model are very similar to the true outcome values $\vec{y}$, MSE is close to zero and goodness of fit is supported. On the other hand, if $\vec{\tilde{y}}$ is very far from $\vec{y}$ the MSE will be large, indicating that the model does not fit the data properly.

R^2 statistic provides an alternative method to express the quality of a fit by evaluating fraction of residual (RSS) and total sum of squares (TSS). It is calculated as

$$R^2(\vec{y}, \vec{\tilde{y}}) = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2}, \tag{3}$$

where $\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$. Since R^2 can reach only values between 0 and 1 ($0 < R^2 < 1$), it allows to compare its value for different data sets. R^2 can be interpreted as amount of variability which is explained by the regression. This can be rephrased as amount of variability explained by using X. Thus if R^2 approaches one, $R^2 \sim 1$, the model explains large proportion of data variability.

2.2 Estimation methods

In order to optimize a linear model, different approaches for determination of the regression parameters are studied.

[1] All shown equations are recovered from [1], [2] and [3].

2.2.1 Ordinary Least Squares (OLS)

Aiming to optimize the $\vec{\beta}$ values, a minimization of cost function $C(\vec{\beta})$ is required. This last quantity is defined as

$$C(\vec{\beta}) = \frac{1}{n}\left\{\left(\vec{y} - \mathbb{X}\vec{\beta}\right)^T\left(\vec{y} - \mathbb{X}\vec{\beta}\right)\right\} \tag{4}$$

and it expresses the distance between the true values $\vec{y}$ and the predicted values $\vec{\tilde{y}} = \mathbb{X}\vec{\beta}$. The minimization processes needs to be implemented with respect to $\vec{\beta}$:

$$\frac{\partial C(\vec{\beta})}{\partial \vec{\beta}} = 0 = \mathbb{X}^T\left(\vec{y} - \mathbb{X}\vec{\beta}\right) \quad . \tag{5}$$

If $\mathbb{X}^T\mathbb{X}$ is invertible, $\vec{\beta}$ can be directly expressed as

$$\vec{\beta}_{OLS} = \left(\mathbb{X}^T\mathbb{X}\right)^{-1}\mathbb{X}^T\vec{y}. \tag{6}$$

It is possible to study how close the $\vec{\beta}_{OLS}$ are to the true values $\vec{\beta}$ by calculating the average of estimated parameters

$$\mathbb{E}[\vec{\beta}_{OLS}] = \mathbb{E}[(\mathbb{X}^\top\mathbb{X})^{-1}\mathbb{X}^T\vec{y}] = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{E}[\vec{y}] = (\mathbb{X}^T\mathbb{X})^{-1}\mathbb{X}^T\mathbb{X}\vec{\beta} = \vec{\beta} \quad , \tag{7}$$

which confirms that $\vec{\beta}$ is unbiased. Moreover, the variance of $\vec{\beta}_{OLS}$ can be expressed as

$$\mathrm{Var}(\vec{\beta}_{OLS}) = \mathbb{E}[(\vec{\beta}_{OLS} - \mathbb{E}[\vec{\beta}_{OLS}])(\vec{\beta}_{OLS} - \mathbb{E}[\vec{\beta}_{OLS}])^T] = \sigma^2(\mathbb{X}^T\mathbb{X})^{-1} \quad , \tag{8}$$

where σ^2 is variance of data $\vec{y}$, $\sigma^2 = \mathbb{E}[\vec{y}\vec{y}^T] - \mathbb{E}[\vec{y}]\mathbb{E}[\vec{y}^T]$. The variance of $\vec{\beta}$ can be used to create confidence intervals - range of values that contains the true value of the parameter with probability equal to 1-α (α small). Mathematically, it can be expressed as

$$\beta_j \in \left[\tilde{\beta}_j \pm q^{\mathcal{N}(0,1)}_{1-\frac{\alpha}{2}}\sqrt{\frac{\tilde{\sigma}^2}{n}\cdot(\mathbb{X}^T\mathbb{X})^{-1}_{jj}}\right] \quad , \tag{9}$$

where q is the quantile function of a normal distribution and tilde denotes the predicted value.

2.2.2 Ridge Regression

Ridge regression is a fitting procedure which allows to estimate the regression coefficients β by shrinking the impact of least important explanatory variables in association with a given response. In particular, coefficient estimates with Ridge method are obtained similarly as in OLS by minimizing cost function

$$C(\vec{\beta}) = \frac{1}{n}\left\{(\vec{y} - \mathbb{X}\vec{\beta})^T(\vec{y} - \mathbb{X}\vec{\beta})\right\} + \lambda\vec{\beta}^T\vec{\beta} \quad , \tag{10}$$

where λ is finite parameter and $\sum_{i=0}^{p-1}\beta_i^2 \leq t$ with t finite positive. The parameter $\lambda \geq 0$, called tuning or regularization parameter, allows to tune the shrinkage penalty term. In particular,

- $\lambda = 0$ reduces Ridge to OLS method, since there is no shrinkage penalty term.
- $\lambda \to \infty$ implies a growth in the penalty term and coefficient estimates approach zero.

As a different set of coefficient estimates is produced for each λ parameter, the choice of the right tuning parameter is essential. From (10), it can be obtained that

$$\vec{\beta}_{\text{Ridge}} = \left(\mathbb{X}^T\mathbb{X} + \lambda\mathbb{I}\right)^{-1}\mathbb{X}^T\vec{y} \quad , \tag{11}$$

where the inversion of the matrix is always possible since it does not have singularity problems for finite values of the parameter λ. $\mathbb{I}$ is a $p \times p$ identity matrix. Once having all the coefficient estimates computed, predicted values of the response variable can be evaluated as $\vec{\hat{y}} = \mathbb{X}\beta_{\text{Ridge}}$

Analytical expression for the expected value and variance of β_{Ridge} can be obtained as:

$$\begin{aligned} \mathbb{E}[\vec{\beta}_{\text{Ridge}}] =& (\mathbb{X}^T\mathbb{X} + \lambda\mathbb{I})^{-1}(\mathbb{X}^\top\mathbb{X})\vec{\beta}_{\text{OLS}} \\ \text{Var}(\vec{\beta}_{\text{Ridge}}) =& \sigma^2[\mathbb{X}^T\mathbb{X} + \lambda\mathbb{I}]^{-1}\mathbb{X}^T\mathbb{X}\{[\mathbb{X}^\top\mathbb{X} + \lambda\mathbb{I}]^{-1}\}^T. \end{aligned} \tag{12}$$

From these formulas it is clear that the Ridge estimator is biased and that, for $\lambda \to \infty$, the variance of Ridge estimates goes to zero. Moreover,

$$\text{Var}(\vec{\beta}_{\text{OLS}}) \geq \text{Var}(\vec{\beta}_{\text{Ridge}}) \quad . \tag{13}$$

2.2.3 Lasso Regression

Lasso regression can be perceived as an alternative fitting method to OLS and Ridge regression which can be used to overcome the disadvantage of having a large set of explanatory variables (predictors) in building a model, since the previous two techniques do not allow to discard certain predictors (in fact, OLS includes all explanatory variables and Ridge shrinks only some parameters to zero, still inserting them into the model).

Lasso coefficient estimates are the ones which minimize the expression

$$\min_{\beta\in\mathbb{R}^p}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j X_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| \tag{14}$$

under the constraint

$$\sum_{j=1}^{p} |\beta_j| \leq s \quad , \tag{15}$$

where s is a finite positive number. The parameter $\lambda \geq 0$ is the tuning parameter. In this case, the penalty term actually manages to force some coefficient estimates to be exactly equal to zero when λ is sufficiently large (as for Ridge, it was necessary to have $\lambda = \infty$ in order to have zero impact of some variables). As it was for the previous method, it is extremely important to determine a correct value for λ, for example selecting the one which gives the best results in terms of error of the model.

2.3 Resampling techniques

Resampling is a fundamental part of statistical analysis. It involves fitting the same statistical method multiple times using different subsets of the data. Such approach allows to obtain information that is not accessible by fitting the model only once using the original training sample and also to overcome the lack of data usually faced in data analysis.

2.3.1 Cross validation

One of the main methods used for resampling is cross-validation, which also solves problems such as overfitting and bias selection.

The process starts by splitting the data in two sets: the training set, on which it is possible to build the model, and the test set, on which predictions are made and prediction performance of the model is evaluated. In theory, it is possible to divide the data in three parts: training, test and validation, where this last one is used to estimate the prediction errors.

In particular, k-fold cross-validation is a cross-validation technique which allows to solve the problem of unbalanced data splitting. The data is divided into k subsets, approximately equally sized, which need to be exhaustive and mutually exclusive. As the process occurs iteratively, at each step one of the subsets is considered as test set and the remaining (k-1) as training sets. A specific index, such as the Mean Squared Error (MSE) is computed each time on a different test set. At the end of the k-th loop, the k-fold statistic (such as MSE) is estimated by computing the mean of these values

$$CV_k = \frac{1}{k} \sum_{i=i}^{k} (MSE_i) \tag{16}$$

2.3.2 Boostrap

The bootstrap method is a general tool for assessing statistical accuracy. Supposing there is a model which needs fit a set of training data, the basic idea is to

randomly create new data sets from the original training data set and get more precise results using the new data set. The new data set consists of randomly chosen data from the original one and has the same size as this last set, i.e. some data is chosen multiple times. This is repeated iteratively and many new data sets are created. After building each sample, a statistic (for example, MSE) is computed for every new data set. Finally, the average of all these values is calculated. When the number of the new data sets created reaches infinity, the studied statistic hits its true value for the whole dataset.

2.4 The Bias-Variance Trade-off

In the context of linear regression, starting from the assumption of a model $Y = f(X)+\epsilon$, there is an approximation of the f function in terms of parameters $\vec{\beta}$ and a design matrix $\mathbb{X}$ obtaining the model: $\tilde{Y} = \beta\mathbb{X}$. The mean squared error is then calculated as

$$MSE(\vec{y}, \vec{\tilde{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\vec{y} - \vec{\tilde{y}})^2\right] \quad . \tag{17}$$

It can also be written as

$$\begin{aligned}\mathbb{E}\left[(\vec{y} - \vec{\tilde{y}})^2\right] = \frac{1}{n}\sum_i (f_i - \mathbb{E}\left[\vec{\tilde{y}}\right])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}\left[\vec{\tilde{y}}\right])^2 + \sigma^2 = \\ = \text{Bias}[\vec{\tilde{y}}]^2 + \text{Var}[\vec{\tilde{y}}] + \text{Var}[\vec{\epsilon}]\end{aligned} \tag{18}$$

The first term can be seen as an error due to the assumptions taken while constructing the model, the second represents the variance of the model and the last one is the variance of the error term ϵ.

To prove (18) there is the need to recall that $\mathbb{E}[\epsilon] = 0$ and that $\vec{f}$ is deterministic. By adding and subtracting $\vec{f}$, we get:

$$\begin{aligned}\mathbb{E}\left[(\vec{y} - \vec{\tilde{y}})^2\right] &= \mathbb{E}\left[(\vec{y} - \vec{f} + \vec{f} - \vec{\tilde{y}})^2\right] = \\ &= \mathbb{E}\left[(\vec{y} - \vec{f})^2\right] + \mathbb{E}\left[(\vec{f} - \vec{\tilde{y}})^2\right] + 2\mathbb{E}\left[(\vec{f} - \vec{\tilde{y}})(\vec{y} - \vec{f})\right] = \\ &= \mathbb{E}\left[(\vec{\epsilon})^2\right] + \mathbb{E}\left[(\vec{f} - \vec{\tilde{y}})^2\right] + 2\mathbb{E}\left[\vec{f}\vec{y}\right] - \mathbb{E}\left[(\vec{f})^2\right] - \mathbb{E}\left[\vec{\tilde{y}}\vec{y}\right] + \mathbb{E}\left[\vec{\tilde{y}}\vec{f}\right] = \\ &= \mathbb{E}\left[(\vec{\epsilon})^2\right] + \mathbb{E}\left[(\vec{f} - \vec{\tilde{y}})^2\right]\end{aligned} \tag{19}$$

The second term can be rewritten as:

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

The bias-variance trade-off equation (18) tells that in order to minimize the MSE it is needed to select a model that simultaneously keeps low variance and low bias. This is directly connected to problems as underfitting or overfitting, which can be avoided by studying the dependence of those three indices on different polynomial degrees in the model. Note that the MSE can never be lower than σ^2, known as the irreducible error.

3 Implementation

3.1 Generation of the data sets

The Franke function is defined as

$$\begin{aligned}f(x,y) = &\frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\right)+\frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49}-\frac{(9y+1)}{10}\right) \\ &+\frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\right)-\frac{1}{5}\exp\left(-(9x-4)^2-(9y-7)^2\right) \quad .\end{aligned} \tag{21}$$

In order to produce the Franke function data, a uniform distribution between 0 and 1 generating 1000 points for x and y variables has been used. Then, a random noise with mean 0 and coefficient 0.01 has been added with the code

```
z_1 = z +0.01*np.random.randn(n)
```

The noise is used to explore its impact on R^2 and MSE. As the noise component becomes larger, the mean squared error of the model increases and the variance score decreases. Also by exploring different polynomial degrees, the noise helps to detect where the overfitting is reached.

For the second part of the project, the digital terrain data have been read from the link [4] with the fuction imread() (included in imageio library) and standardized before being used in order to avoid outliers. The values contained in a 3601x1801 table have been raveled, obtaining 65341 observations. Then x and y values have been arranged as

```
x = np.arange(0,1801,10)
y = np.arange(3601,0,-10)
x, y = np.meshgrid(x,y)
```

for then being also raveled before creating the design matrix. On the total of the observation a selection has been applied as

```
my_cols = list(np.arange(0,1801,10))
my_rows = list(np.arange(0,3601,10))
z = z[my_rows,]
z = z[:,my_cols]
```

In this way only one value every ten has been taken into account as part of a sample. The sample has then took the place of the original data set during the whole study because of its largeness and high requirements on time of computation.

3.2 Design matrix $\mathbb{X}$

The design matrix for the model is determined by implementing a function which takes x, y and length of x vector n=len(x) as inputs and returns the design matrix as an output. In particular, the matrix is built by a loop which allows to place all possible combination of x and y powers in the matrix itself, given the polynomial degree which is set equal to 5. Each column of the design matrix is obtained by increasing the power of x while decreasing the power of y, and viceversa. For example, the first column contains x^0y^0, the second x^1y^0, the third x^0y^1, the fourth x^2y^0, the fifth x^1y^1, the sixth x^0y^2 and so on. The function performing this operation in the study is called Design_Matrix_X and can be found at the GitHub link [5] with all material.

3.3 Linear regression with OLS

Linear regression with ordinary least squares (OLS) has been performed according to the section 2.2.1, checking that the design matrix is not singular. The effectiveness of the calculations produced is checked by computing the same quantities applying Scikit-Learn functions, such as LinearRegression().fit() and predict(). 95% confidence intervals are found for the parameters β's by calculating the variance of the estimated coefficients β_{OLS}, extracted form the diagonal of the covariance matrix with the code

```
var_beta_OLS = 1*np.linalg.inv(X.T.dot(X))
var_diag=np.diag(var_beta_OLS)
l1_OLS = beta - 1.96*np.sqrt((var_diag)/(X.shape[0]))
l2_OLS = beta + 1.96*np.sqrt((var_diag)/(X.shape[0]))
```

The result is an interval in which the true value of the parameter lays with a confidence level equal to 95%.

The quality of the fit produced by applying OLS is evaluated through the computation of mean squared error and variance score. In particular, after implementing the functions

```
def MSE (ydata, ymodel):
    n = np.size(ymodel)
    y = (ydata - ymodel).T@(ydata - ymodel)
    y = y/n
    return y

def R2 (ydata, ymodel):
   return 1-((ydata-ymodel).T@(ydata-ymodel))/
            ((ydata-np.mean(ydata)).T@(ydata-np.mean(ydata)))
```

the results are checked by using Scikit-Learn functions mean squared error and R^2 score.

3.4 K-Fold cross validation – MSE and R^2

In order to evaluate the model built with cross validation, it is necessary to split the data into training and testing subsets. Following this purpose, a k_fold cross-validation function is implemented, dividing x, y and z into k=5 folds and considering as a testing subset the 20% of the total data available.

Firstly, a loop is set in order to determine different steps with range equal to the number of available folds.

In each step, a function renamed train_test_splitdata (see GitHub link [5]) is applied. Its operation is to change the fold used as test subset at every loop. It does it by applying two commands

```
x_learn=np.delete(x_,i)
x_test=np.take(x_,i)
```

The first one deletes a subset from the all data and keeps the left part as training data while the second one does the opposite, picking the exact part of data removed before as a single test set. By changing the interval i it is sure that different subsets will be chosen as training and test at each loop.

Afterwards, two design matrices are created by calling the Design_Matrix_X function, one for the test data and the other for the train data.

A linear regression model is then determined by estimating $\vec{\beta}$ using the train data and calculating the predicted values of the response variable by multiplying $\vec{\beta}$ for the design matrix of the test data, $\tilde{Y} = \mathbb{X}_{\text{test}}\beta$.

Secondly, the mean squared error and the variance score between the test set and the predicted values are calculated for each loop. At the end of all steps, the averages of these quantities are calculated and given as result of the function. The implemented code is then compared to the Scikit-Learn K-fold cross-validation in order to check the validity of the obtained results.

3.5 Bias – Variance tradeoff using bootstrap

The so-called bias-variance tradeoff allows to explore the dependence of bias and variance on the polynomial complexity. The entire data set is shuffled and divided into train and test parts using the Scikit-Learn function train_test_split with given test size equal to 0.2. Setting the number of studied degrees equal to 20, for each degree a bootstrap resampling technique is applied on the train part of the data and design matrices are created for both train and test sets. Mean squared error, variance and bias values for the test data are computed for each degree and stored into specific arrays, which are then used in order to plot the MSE, bias and variance with respect to all different possible polynomial degrees. Together with the quantities calculated for the test data, the MSE is also computed for the train subset, in order to compare train and test errors in a plot which shows the polynomial degree for which overfitting is reached. The described procedure can be found at the GitHub link [5].

In particular, the number of bootstraps used is set equal to 100 for Franke's function data, while results with both 10 and 30 bootstraps are explored dealing with real data, not being able to increase the number of iterations even more due to long computational time.

3.6 Ridge regression

Ridge regression method is implemented by considering the polynomial complexity as well: each degree determines a specific lambda value which is optimal in terms of MSE minimization. In particular, a list of different values of lambda is set as lamdas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 1].

Firstly, for each degree a design matrix, estimated parameters and predicted values are determined. In particular, estimated values of the parameters are computed by inverting the matrix $\mathbb{X}^T\mathbb{X} + \lambda * \mathbb{I}$ (after analyzing its singularity), where $\mathbb{I} \in \mathbb{R}^{p\times p}$ is the identity matrix having dimensions $p \times p$, for p parameters in the model. The implemented code is

```
beta_r = np.linalg.inv(X.T.dot(X)
        +lamda*np.identity(int((m+1)*(m+2)/2))).dot(X.T).dot(z_1)
zridge = X @ beta_r
```

The validity of the method used is then checked by computing the same values using Scikit-Learn functions skl.Ridge, for alpha=lambda and predict().

As escribed in the OLS implementation section, confidence intervals of the parameters are determined by calculating the variance of $\vec{\beta}$s. The difference is in the presence of the lambda parameter added as shown in the code

```
M = np.linalg.inv(X.T.dot(X)+lamda*np.identity(int((m+1)*(m+2)/2)))
var_beta_ridge = M.dot(X.T).dot(X).dot(M.T)
var_b_ridge = np.diag(var_beta_ridge)
l1_Ridge = beta_r - 1.96*np.sqrt((var_b_ridge)/(X.shape[0]))
l2_Ridge = beta_r + 1.96*np.sqrt((var_b_ridge)/(X.shape[0]))
```

The real values of the parameters can be found between the two bounds computed at a confidence level equal to 0.95.

Secondly, for each degree the value of lambda which gives the lowest MSE of the model is stored in a dictionary[2] in order to be able to compare the best lambda for each polynomial degree.

3.7 Lasso regression

Lasso regression is implemented similarly to Ridge, in terms of exploring which lambda value allows to have the lowest MSE for each polynomial degree. In particular, a list of lambda is set as lamdas = [0.0001, 0.001, 0.01, 0.1, 1]. Smaller lambda values are not considered due to convergence problems.

By appling scikit-learn functionality skl.Lasso and placing alpha=lambda, the predicted values and estimations of the parameters $\vec{\beta}$ are computed. Moreover, by storing the best lambda for each polynomial degree in a dictionary, differences in terms of which value minimizes the MSE for each degree are explored. It has to be said that scikit-learn function automatically optimizes the learning rate for Lasso but, at the same time, requires an high number of iterations in order not to get convergence issues. Therefore, the results obtained using Lasso as a fitting method through the scikit-learn code are not completely reliable.

4 Results and discussion

In order to display a higher amount of details all the graphs are shaped on logarithmic scale on their y axes which allows the changes of very small numbers to be more easily seen. Firstly, OLS, Ridge and Lasso results for Franke function are discussed, secondly the ones for terrain data.

4.1 Random generated values and Franke's function

In Table 1, mean squared error (MSE) and R^2 values obtained by fitting by a fifth grade polynomial model are summarized for different regression methods. By comparing the statistics, it can be affirmed that OLS and Ridge turn out to be the best methods to fit the Franke's function data. They return the lowest MSE value and the highest R^2, followed by the Lasso regression method. Those outcomes are endorsed by calculating the statistics through a k-fold cross validation, as visible in Table 2. In order to see if the results obtained with the implemented code are reliable, it is useful to compare them with the scikit-learn values in Table 3, obtained by applying LinearRegression, Ridge and Lasso functions from that library.

In Table 8 in appendix A, values of $\vec{\beta}$ elements are shown with their 95% confidence intervals. Percentage of this interval with respect to the β_i value is calculated to determine reliability of β_i value.

[2] associative array in Python

	OLS	Ridge	Lasso
MSE	0.0020510889	0.0020673085	0.0087683865
R^2	0.9737030822	0.9734951306	0.8937880494

Table 1: Statistics obtained with our own code

	OLS	Ridge	Lasso
MSE	0.0022951092	0.0020454841	0.0089400624
R^2	0.9699985517	0.9737485463	0.8914546373

Table 2: Statistics obtained with cross validation our own code

	OLS	Ridge	Lasso
MSE	0.0020510889	0.0020673088	0.0087683865
R^2	0.9737030822	0.9734951268	0.8937880494

Table 3: Statistics obtained with scikit-learn functions

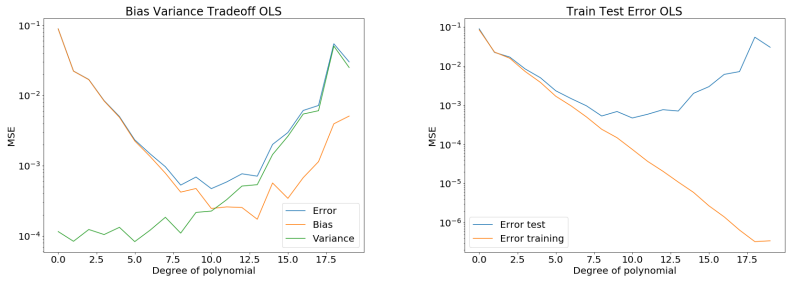

Figure 1: Left: Bias variance tradeoff dependence on degree of polynomial for OLS method. Right: Mean squared error dependence on degree of polynomial for OLS training and test data set. Both graphs are shown for Franke function.

By studying the bias-variance tradeoff while changing the polynomial degree, overfitting and underfitting problems have been quantitatively evaluated. These things are strictly connected. It can be seen from the left graph in Figure 1 that, to low degrees of polynomial, high values of error and bias, but very little variance, correspond. Bias and error decrease by adding more complexity to the model, while the variance increases. As we have seen in the theory section, according to the bias-variance trade-off equation (18), it is needed to select a model that simultaneously keeps low variance and low bias in order to minimize the MSE. As described in the implementation of the code section, the model is estimated on a set of training data.

The variance expresses how the model would change if the estimation is done by using a different training data set. Ideally, the estimate for the model

should not vary too much between training sets. As can be seen from the graph, more flexible statistical methods have higher variance. That is when overfitting happens. In the case studied, to use a polynomial model with a degree higher than ten is inconvenient because the model performes overfitting, which means that it does not learn just from the training data but also starts learning the noise and memorizing it instead of just reproducing it.

As described in the implementation of the code section, the model is estimated on a subset of the total data (training data) and the rest is considered to be test data which used to evaluate the model built. Left graph in Figure 1 shows how the MSE calculated on the train data and the one calculated on the test data vary depending on the polynomial degree of the model. The two quantities are almost the same considering lower degrees, while for degrees greater than ten, the training error keeps decreasing at the same rate and the test error starts increasing. This confirms what has been already said about the overfitting problem and showed in the bias-variance study. A more flexible model maximizes its performance on the set of training data, but fails to fit other data sets or predict future observations.

For Ridge regression, the best lambda for each polynomial degree is always the same, equal to 10^{-5}. It can be seen in left histogram in Figure 2 that MSE reaches its minimum for $\lambda = 10^{-5}$. As has been expected, for every λ value lower MSE for higher polynomial degrees is observed. In graphs in Figure 3, bias-variance tradeoff and MSE for test and train subset are shown. To obtain these plots, $\lambda = 10^{-5}$ has been used. Same trend as for OLS has been reproduced. However, unlike OLS, behaviour has been studied until 50th degree of polynomial. For $\lambda > 10^{-5}$ variance is observed to be low even for high degrees of polynomial but this trend disappears with greater λ (see Figure 9 in appendix A).

Regarding the plot of MSE dependence on the training and test error (right graph in Figure 3), overfitting starts around 10th degree of polynomial as well as it has been observed in OLS case. Most interesting differences, however, can be seen after the tenth degree, having error test and error train as diverging lines which reflect the overfitting model behaviour.

In Figure 2 (right), Bias variance tradeoff dependence on degree of polynomial and lambda for Lasso method can be seen. $\lambda = 10^{-4}$ has been chosen as the best one (i.e. the one which allows to have the lowest MSE). For this choice of λ, bias-variance tradeoff and mean squared error for test and train dataset graphs have been produced (Figure 4). Until the studied 20th degree of polynomial, overfitting is not observed. Moreover, training error is observed to be bigger than test error for the studied degrees of polynomial, even though this difference is not very visible.

It also has been found that Lasso method is much more time demanding than OLS or Ridge method. This is caused by gradient descent which is not used in OLS and Ridge because they offer analytical solutions of minimization equations. As a consequence, higher polynomial degrees are not studied due to an excessively long computational time.

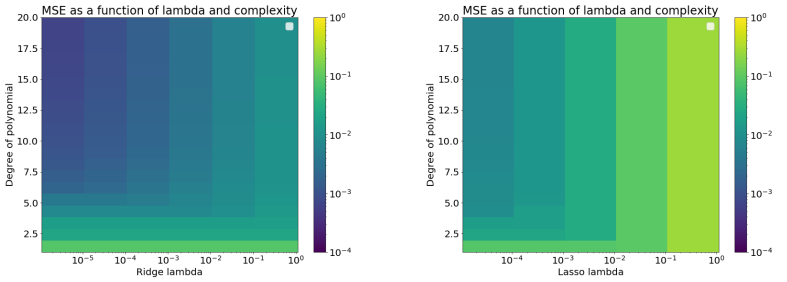

Figure 2: Mean squared error dependence on degree of polynomial and lambda parameter for Ridge (left) and Lasso (right) method shown in 2D histogram. Both graphs are shown for Franke function.

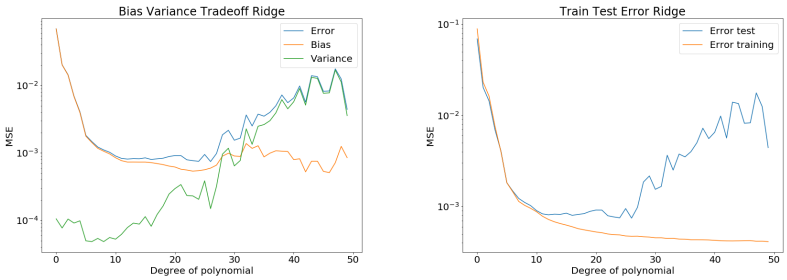

Figure 3: Left: Bias variance tradeoff dependence on degree of polynomial for Ridge method. Right: Mean squared error dependence on degree of polynomial for Ridge training and test data set. Both graphs are shown for Franke function.

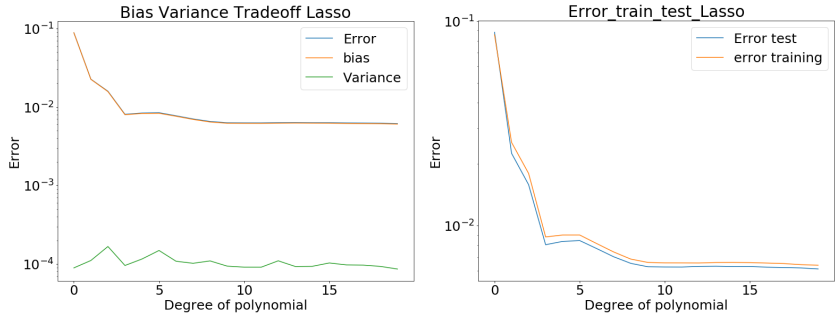

Figure 4: Left: Bias variance tradeoff dependence on degree of polynomial for Lasso method. Right: Mean squared error dependence on degree of polynomial for Lasso training and test data set. Both graphs are shown for Franke function.

4.2 Digital terrain data

In Table 4 it can be seen how the values of MSE and R^2 are the same for all the three implemented methods dealing with a fifth order polynomial. The lambda used during Ridge regression was 0.00001. It is known that when lambda is set equal to zero OLS and Ridge correspond. Therefore, in this case, due to the value of lambda approaching zero and due to some approximations their statistics correspond, having Lasso being equivalent as well.

	OLS	Ridge	Lasso
MSE	0.5171407481	0.5171407481	0.5171407481
R^2	0.4838395938	0.4838395938	0.4838395938

Table 4: Statistics obtained with our own code

	OLS	Ridge	Lasso
MSE	0.5171407481	0.5171407481	0.5171407481
R^2	0.4838395938	0.4838395938	0.4838395938

Table 5: Statistics obtained with sklearn functions

	OLS	Ridge	Lasso
MSE	0.5174907697	0.5174619137	0.5176508750
R^2	0.4833970783	0.4835012555	0.4831934296

Table 6: Statistics obtained with our own cross validation code

However, there is the need to state that an R^2 of 0.48 does not correspond to a great fit, because it means that only the 48% of the variability of the data is represented by the model. It could be overall argued that a linear model might not be the best for representing the analyzed digital terrain data.

In order to check the evaluation, it is possible to have a look at the statistics calculated by implementing scikit-learn functions in Table 5. The values obtained agree with the poor fitting of the linear model and confirm the previous calculations for the indices.

When computing MSE and R^2 indices by applying the k-fold cross validation (see Table 6) , similar results are obtained, thus the three methods are found to be almost equivalent.

The graphs in Figure 5 (first row) are obtained by setting the number of bootstraps equal to 10. It can be seen that, by increasing the number of bootstraps to 30, different trends are displayed (see Figure 5, second row). In particular, the higher is the number of iterations, the more precise will the indices studied be.

Looking at the bias-variance plot for OLS, it can be seen that variance tends to be very small for all the degrees explored, even though it increases when

the degree increases. The plot shows that the bias-variance equation (18) is satisfied. This happens to be true for both bias-variance plots in the first and second row of figure 5. As for the difference between MSE for train and test subsets, in the plot in the first row of figure 5 (10 bootstraps) it is clear that the test error is lower than the train one at first and tends to overcome it after the seventh degree. After that point, the difference between the two quantities can be explained as a result of overfitting, which actually starts already after the third degree. The plot in the second row (30 bootstraps) shows a different trend: the training error is lower than the test one for every degree and it is possible to notice a slightly increasing tendency after the third degree.

Confidence intervals for $\vec{\beta}$ parameters have been also studied, see Table 9 in appendix A. Compared to Franke function, β_i elements are of the several orders of magnitude lower and have got negligible relative error, unlike β_i for Franke.

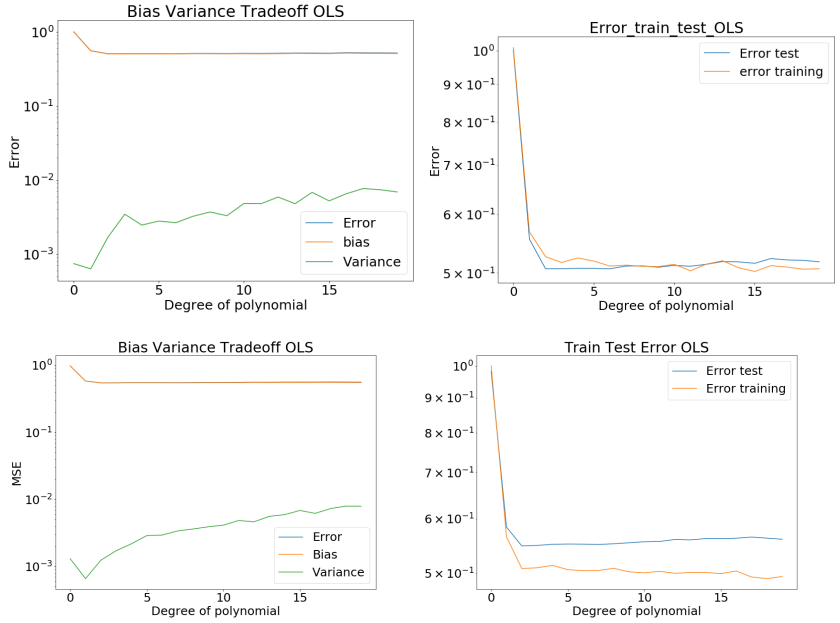

Figure 5: Left column: Bias variance tradeoff dependence on degree of polynomial for OLS method. Right column: Mean squared error dependence on degree of polynomial for OLS training and test data set. Both graphs show terrain data. In first row, graphs obtained for 10 bootstrap loops can be seen, in second one that for 30 bootstrap loops.

Degree	0 - 13	14	15	16	17 - 19
Best lambda	1e-05	0.0001	1e-05	0.0001	1e-05

Table 7: Ridge regression lambda values

Plots for Ridge regression in Figure 6 are computed using $\lambda = 0.00001$, since that value turns out to be the best for most of the polynomial degrees (see Table 7).

In particular, the bias-variance tradeoff plot in Figure 6 is quite similar to the one obtained for OLS, since λ is close to zero. The error test/error train plot shows that the MSE for the test data starts to increase slightly after the third degree, underlining an overfitting tendency which cannot be properly stated only by looking up to the twentieth degree. In particular, this is true both for the plots in the first and second row of Figure 6, even though more precise results in terms of increasing tendencies can be noticed in the 30 bootstraps plot (second row), also due to higher polynomial degrees studied.

Graphs for other λ values can be seen in Figure 10 in appendix A. In these, no change in bias-variance tradeoff is observed. For train and test data, training error appears to be greater than test error up to order of polynomial 29 where both lines cross each other. This crossing might indicate reaching overfitting point.[3] Even though it could have been useful, evaluations and plots for higher degrees of the polynomial have not been performed, due to an excessive duration of the run for obtaining them. Based on Franke function, it is needed to increase the degree of polynomial 2 or 3 times to obtain a similar shape of curves.

Concerning the Lasso regression, the best lambda (giving the lowest MSE) for every degree of the polynomial model is found to be equal to 0.0001. Therefore, graphs in Figure 7 are computed applying that value of the parameter, studying bias-variance tradeoff and train/test error comparison. The bias-variance tradeoff plot is similar to the one displayed for the Ridge method, both for the first and the second row (respectively, plots obtained with 10 and 30 bootstraps). Referring to the plot in the first row, an overfitting tendency is observed from degree of polynomial 3, even though the test error shows just a slighly increasing trend. Looking at the second row plot, it is possible to notice that the test error is always higher than the train one (the trend was inverted with 10 bootstraps) and that the former quantity increases faster than how it does with the 10 bootstraps analysis, showing again an overfitting tendency after the third degree of polynomial.

For a more detailed analysis of bias-variance tradeoff, higher degrees of polynomial need to be studied. Unfortunately, these were not performed due to the duration of the run higher than 10 hours of computation.

5 Conclusion

Three fitting methods (Ordinary Least Squares, Ridge regression and Lasso regression) have been applied in order to fit two different datasets: one obtained from a 2D-function known as Franke function and the other being from a real digital terrain sampling. The used procedures perform differently for Franke function's data and real digital terrain data. In particular, the analysis has underlined that OLS and Ridge are the best linear regression fitting methods for

[3] Graph has been obtained for 30 bootstrap loops.

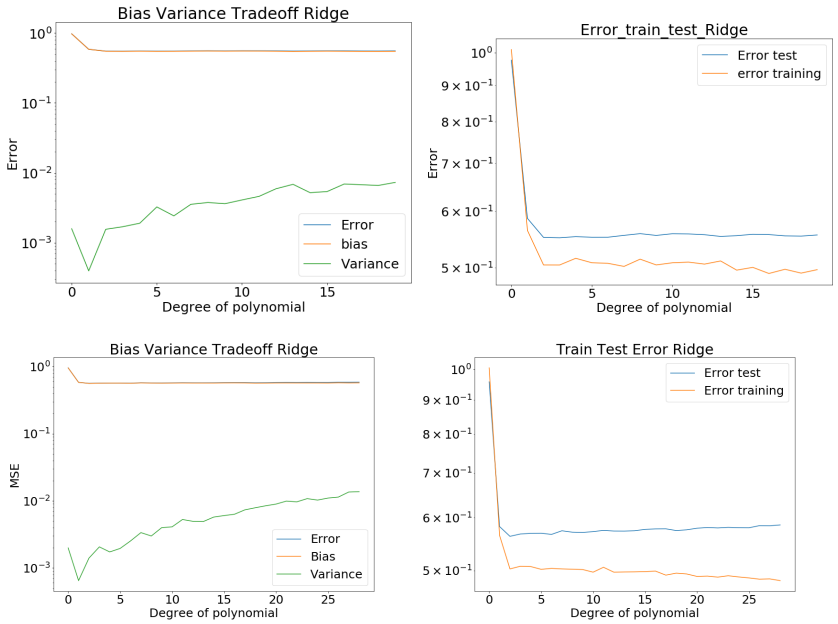

Figure 6: Left column: Bias variance tradeoff dependence on degree of polynomial for Ridge method. Right column: Mean squared error dependence on degree of polynomial for Ridge training and test data set. Both graphs show terrain data. In first row, graphs obtained for 10 bootstrap loops can be seen, in second one that for 30 bootstrap loops.

Franke's data, while OLS, Ridge and Lasso have a similar behaviour regarding results obtained for real data. It has also been found out that increasing number of bootstrap loops makes an observable difference in resulting plots. Moreover, in both data cases it has been recorded that Lasso method is much more time demanding than the other ones. From the analysis of the terrain data, linear regression does not seem to be an optimal regression method for the data itself. It would be interesting to explore more regression methods, for example to study how some other generalized linear models behave for case considered. In conclusion, it could be stimulating to try fitting a model using more advanced gradient descent methods and comparing results to the ones obtained with the procedures used in this project.

References

[1] James, G., Witten, D., Hastie, T., Tibshirani, R. (2013). An Introduction to Statistical Learning. Springer Texts in Statistics. Springer New York. https://doi.org/10.1007/978-1-4614-7138-7

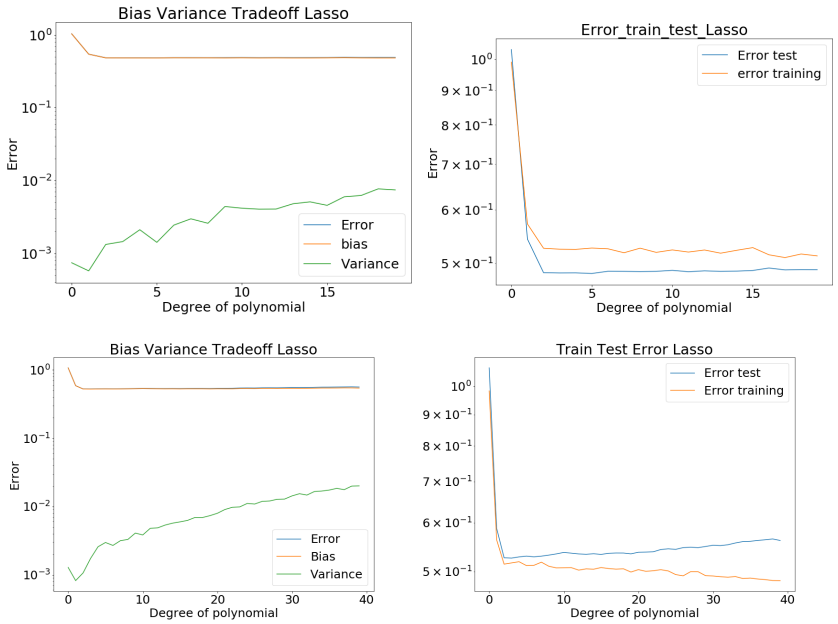

Figure 7: Left column: Bias variance tradeoff dependence on degree of polynomial for Lasso method. Right column: Mean squared error dependence on degree of polynomial for Lasso training and test data set. Both graphs show terrain data. In first row, graphs obtained for 10 bootstrap loops can be seen, in second one that for 30 bootstrap loops.

[2] Morten Hjorth-Jensen, (2019), Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis., `https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/Regression.html`

[3] Hastie, Trevor, et al. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, 2017.

[4] Digital terrain data `https://github.com/CompPhysics/MachineLearning/blob/master/doc/Projects/2019/Project1/DataFiles/SRTM_data_Norway_1.tif`

[5] Project1 directory with used code accessible from `https://github.com/storejar/Data_analysis_and_machine_learning/tree/master/Project1`

A Appendix

Component of $\vec{\beta}$	Value	Relative error in %
β_0	0.283208 ± 0.037686	13.3
β_1	8.533527 ± 0.393004	4.6
β_2	4.671530 ± 0.371744	7.9
β_3	-35.964077 ± 1.771625	4.9
β_4	-18.447689 ± 1.414057	7.6
β_5	-10.915477 ± 1.747184	16.0
β_6	50.168522 ± 3.877114	7.7
β_7	49.680363 ± 2.956665	5.9
β_8	26.258939 ± 2.905819	11.0
β_9	-6.024485 ± 3.948321	65.5
β_{10}	-24.507189 ± 4.041404	16.4
β_{11}	-57.834235 ± 3.186016	5.5
β_{12}	-11.163261 ± 2.904767	26.0
β_{13}	-34.476319 ± 3.110732	9.0
β_{14}	29.244102 ± 4.203074	14.3
β_{15}	1.519052 ± 1.601295	105.4
β_{16}	20.760483 ± 1.464535	7.0
β_{17}	11.181155 ± 1.395630	12.4
β_{18}	-4.018381 ± 1.396448	34.7
β_{19}	18.199890 ± 1.419539	7.7
β_{20}	-17.078097 ± 1.685583	9.8

Table 8: Summary of $\vec{\beta}$ components with their confidence intervals for OLS on Franke function sample.

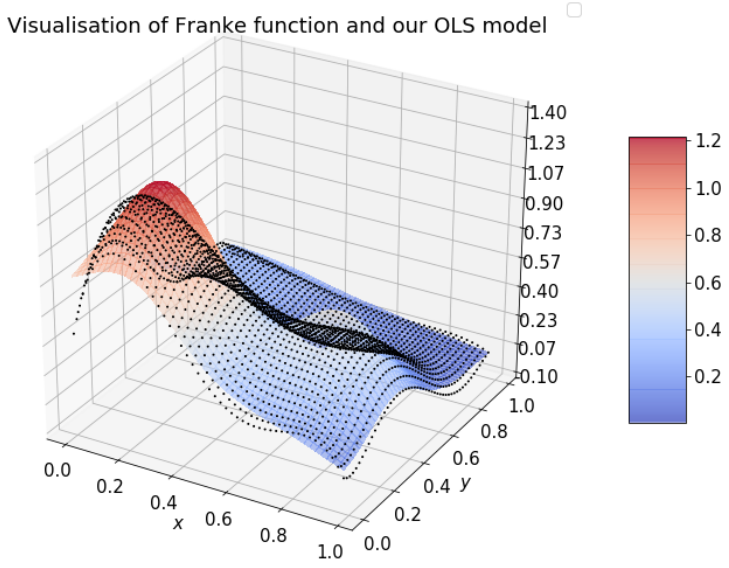

Figure 8: Visualisation of Franke function (colored contour) and our OLS model (black dots). To generate x and y, $N = 50$ points has been used.

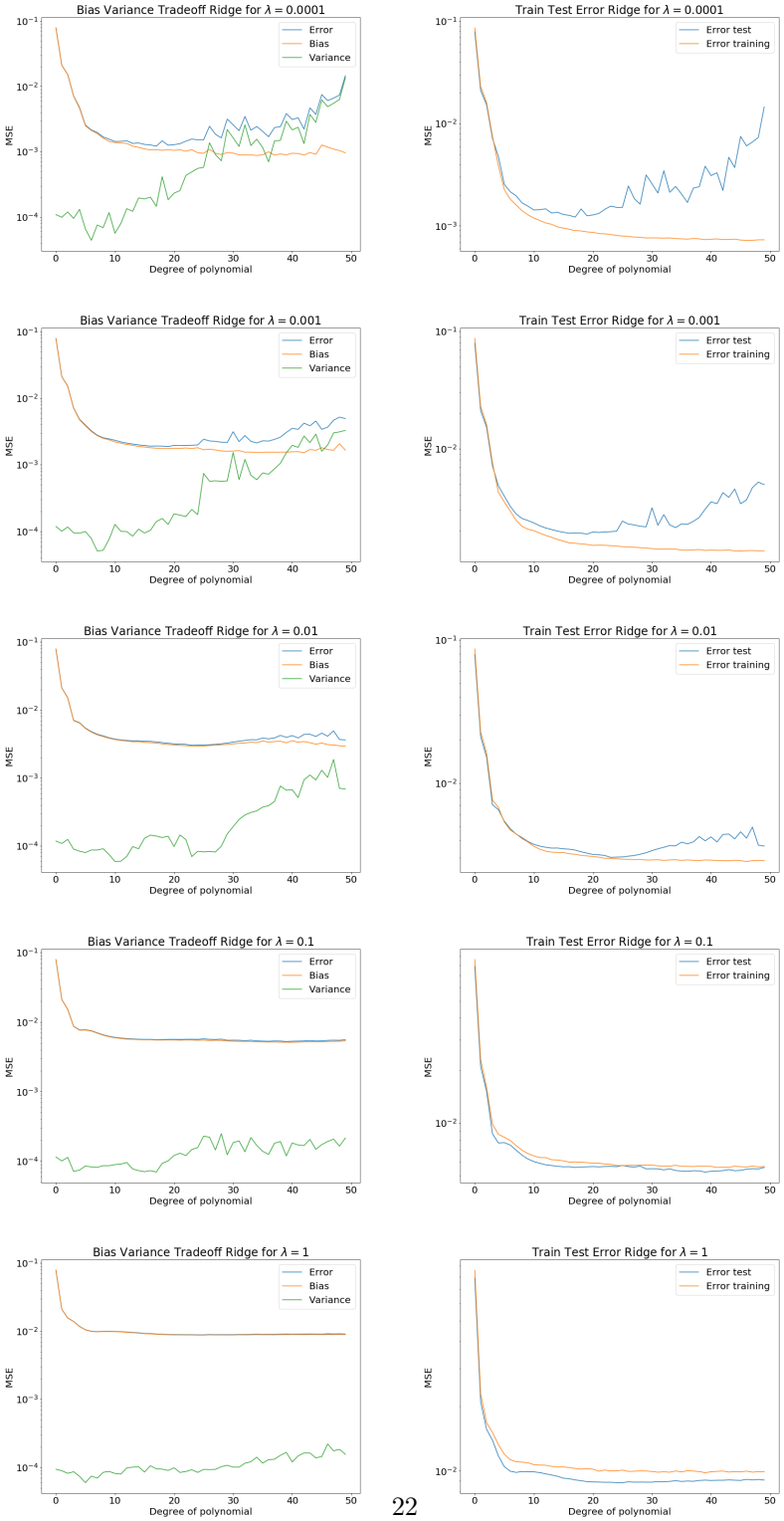

Figure 9: Left column: Bias variance tradeoff for Ridge method. Right column: Mean squared error dependence on degree of polynomial for Ridge training and test dataset. Every line corresponds to different lambda. Graphs show Franke function sample.

Component of $\vec{\beta}$	Value	Relative error in %
β_0	$-7.070063\text{e-}01 \pm 1.530489\text{e-}04$	0.021
β_1	$7.603597\text{e-}04 \pm 2.588526\text{e-}07$	0.034
β_2	$1.375801\text{e-}04 \pm 1.262494\text{e-}07$	0.091
β_3	$-6.105597\text{e-}07 \pm 1.276334\text{e-}10$	0.020
β_4	$-1.316708\text{e-}07 \pm 5.568163\text{e-}11$	0.042
β_5	$1.507610\text{e-}07 \pm 3.118131\text{e-}11$	0.020
β_6	$-7.737750\text{e-}12 \pm 3.468213\text{e-}14$	0.448
β_7	$2.419449\text{e-}12 \pm 3.108245\text{e-}14$	1.284
β_8	$-2.651334\text{e-}11 \pm 2.837347\text{e-}14$	0.107
β_9	$1.275379\text{e-}11 \pm 2.729267\text{e-}14$	0.213
β_{10}	$-5.760169\text{e-}13 \pm 2.571570\text{e-}14$	4.464
β_{11}	$-1.790928\text{e-}12 \pm 2.477031\text{e-}14$	1.383
β_{12}	$7.485540\text{e-}12 \pm 2.444225\text{e-}14$	0.326
β_{13}	$2.163269\text{e-}12 \pm 2.460036\text{e-}14$	1.137
β_{14}	$-2.121640\text{e-}12 \pm 2.426684\text{e-}14$	1.143
β_{15}	$4.704573\text{e-}12 \pm 2.573869\text{e-}14$	0.547
β_{16}	$-1.970074\text{e-}12 \pm 2.443665\text{e-}14$	1.240
β_{17}	$1.882480\text{e-}12 \pm 2.439019\text{e-}14$	1.295
β_{18}	$-3.672005\text{e-}13 \pm 2.438199\text{e-}14$	6.639
β_{19}	$-2.019426\text{e-}12 \pm 2.435146\text{e-}14$	1.205
β_{20}	$1.488272\text{e-}11 \pm 2.502765\text{e-}14$	0.168

Table 9: Summary of $\vec{\beta}$ components with their confidence intervals for OLS on terrain sample.

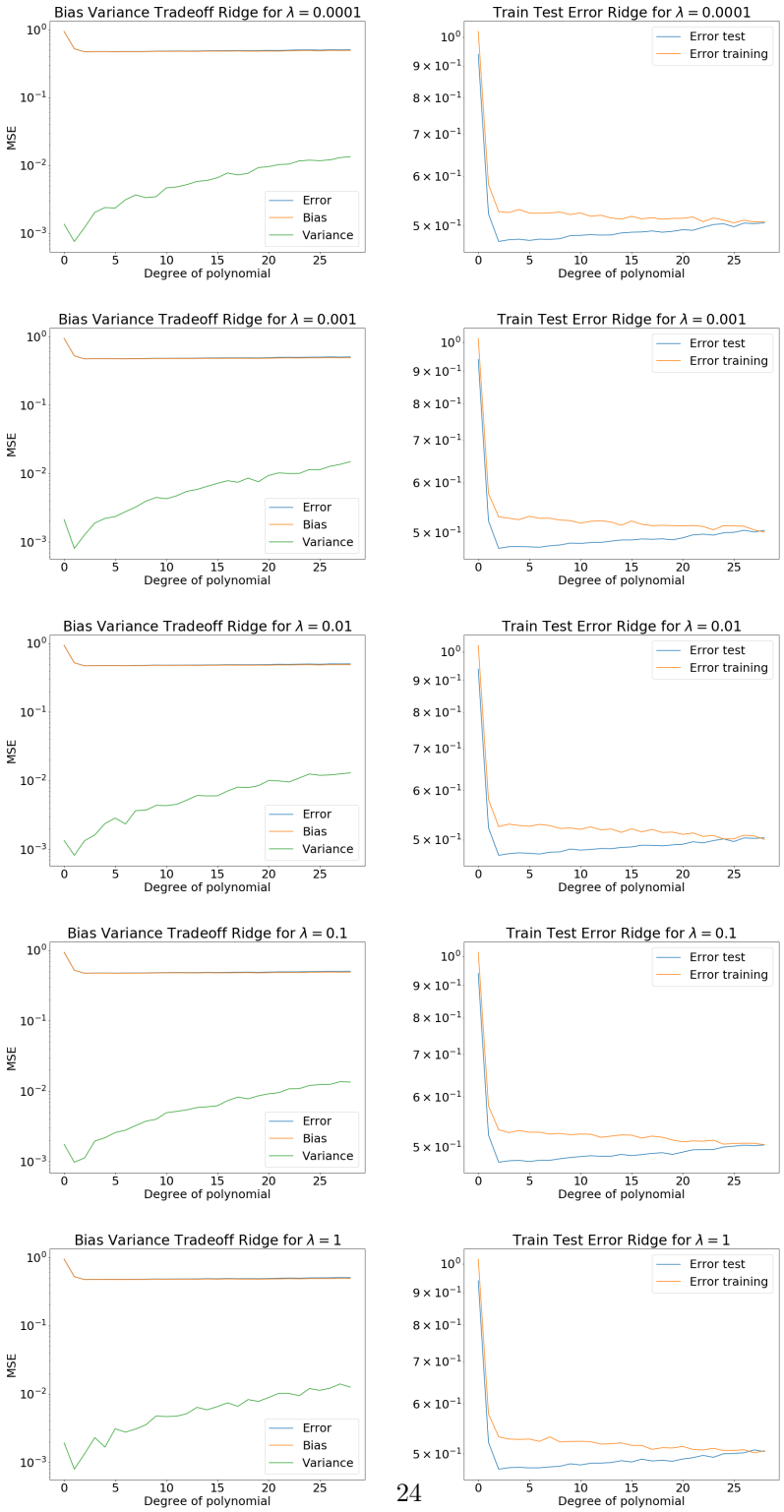

Figure 10: Left column: Bias variance tradeoff for Ridge method. Right column: Mean squared error dependence on degree of polynomial for Ridge training and test data set. Every line corresponds to different lambda. Graphs show terrain data sample and 30 bootstrap loops have been used.