Maria Emine Nylund, October 2020

FYS-STK4155: Data Analysis and Machine Learning

Project 1
Regression Analysis and Resampling Method

Abstract

Here I study the best polynomial fit to Franke's function and terrain data over Norway. I use different regression methods, such as OLS, Lasso and Ridge. For the last two I visualise the effects of various penalty of alpha. To evaluate their performance I use MSE and R2. The best results on 40 points for Franke's function are achieved using four degree polynomial Ridge regression with 0.1 alpha penalty. For terrain data using 100 points, polynomial degree 7 using Lasso with 10 alpha penalty yielded best results.

Contents

1 Introduction

In this study, I use Franke's Function and terrain data over Norway. The goal of the analysis is to study the various methods such as Ordinary Least Squares (OLS), Ridge and Lasso regression. To validate the results sampling methods like bootstrap and cross validation are compared and discussed.

After introduction here I will discuss the implementation of the modules to run the tests. Then I will discuss the results I got on both Franke's function and the terrain data.

For the code and additional plots see here:
https://github.com/marianylund/fysstkprojects

2 Implementation

Here I will explain what data was used in this analysis and how the module is structured.

2.1 Model

First I studied how to fit polynomials to a two-dimensional function called Franke's function. It is a weighted sum of four exponentials which read as follows:

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

[2]

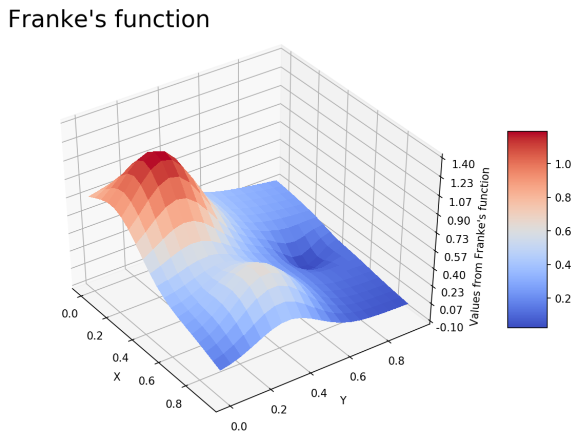


Figure 1: 3D visualisation of Franke's function

Terrain data of Norway is collected from https://earthexplorer.usgs.gov/. Here are 3D visualisations of it

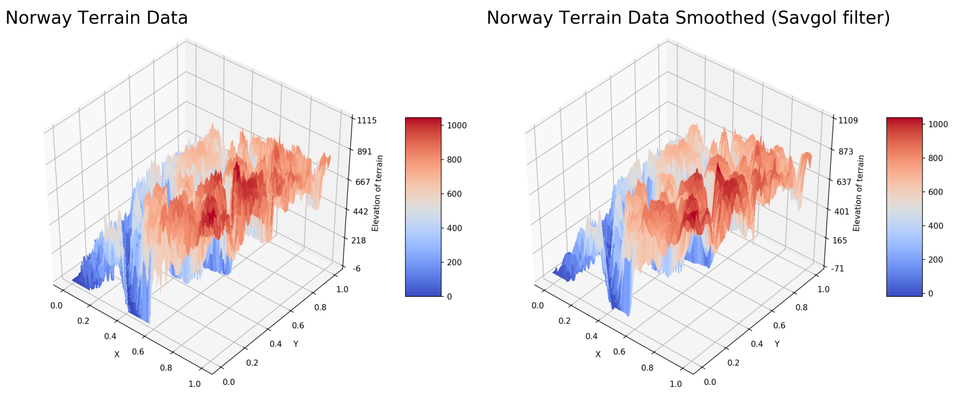


Figure 2: 3D visualisation of terrain data

2.2 Code

For the code source see github:
https://github.com/marianylund/fysstkprojects.

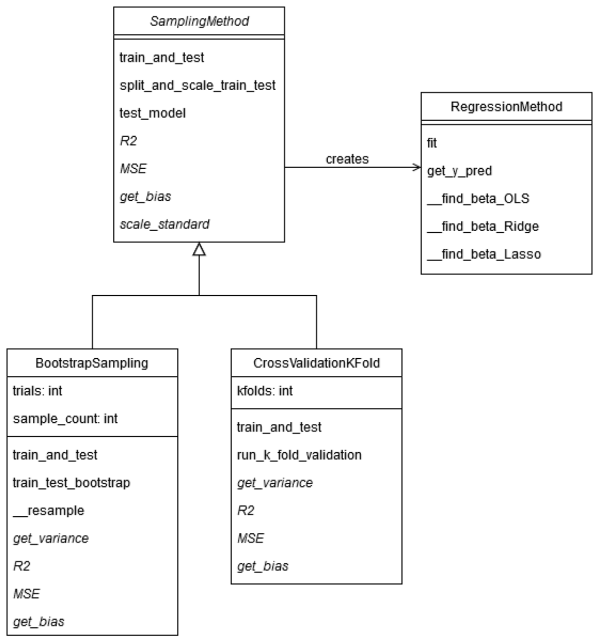


Figure 3: Diagram to show the relationship between regression classes

RegLib module consists of:

2.2.1 SamplingMethod

The SamplingMethod class is the solver that is responsible for splitting and scaling incoming feature matrix and output. It takes in a model type enum and creates a regression method depending on it. It also tests the model on the given prediction.

2.2.2 BootstrapMethod

The Bootstrap Method class is a child of SamplingMethod. It reuses split and scale function from the parent. When training and testing it repeats the same process as in SamplingMethod just for the number of trials.

When all predictions are calculated, it finds r2, MSE, bias and variance based on the predictions. The most notable function in the class is a private resample function which creates a list of indices. And uses random.choice function from numpy library to pick out indices with replacement. It then returns X and y train data using those sampled indices.

2.2.3 CrossValidationKFold

The Cross Validation KFold class is a child of SamplingMethod as well as Bootstrap Method class. Since splitting into test and train is not necessary in our case, it goes straight to running the method. It uses the same idea as bootstrap's resample where it creates a list of indices. Then it divides the indices into the number of kfolds. For every fold it takes one of the part with indices to use for testing and the rest for training. See visual representation of the proccess here:

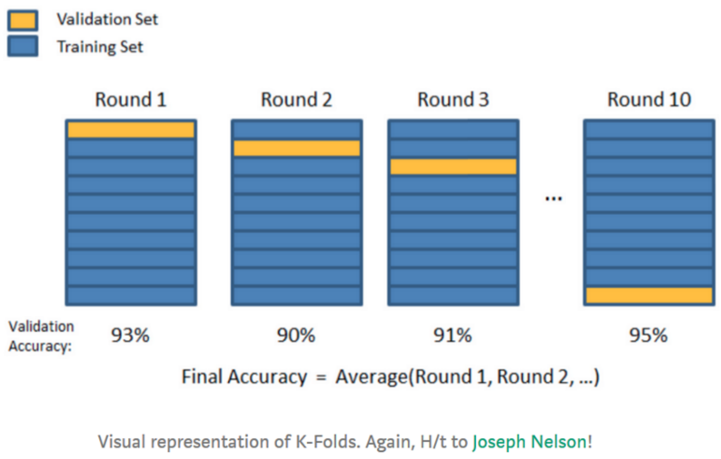

Validation Set
Training Set
Round 1
Round 2
Round 3
...
Round 10
Validation Accuracy:
93%
90%
91%
95%
Final Accuracy = Average(Round 1, Round 2, ...)
Visual representation of K-Folds. Again, H/t to Joseph Nelson!

Figure 4: Visual representation of cross validation proccess from Medium blog[1]

2.2.4 RegressionMethod

This class holds the functions for different methods to fit data, such as OLS, lasso and ridge. It saves found coefficients in the class.

2.2.5 HelperFunctions

This file is a container for different functions used in the project for creating data, plots and saving images.

2.3 Testing

To run tests of the program open terminal in the Project1 folder and run "python RegressionTesting.py". Majority of them compare the implemented results with the same functions in scklearn. In the file itself it is possible to change the error tolerance. For now it passes with error tolerance set 1e-10.

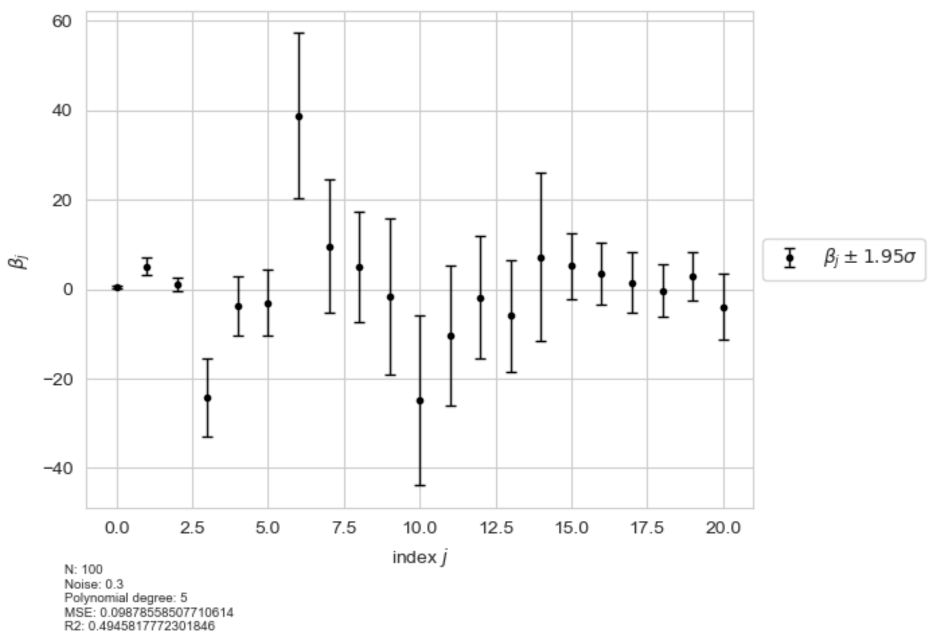
3 Analysis

Here I will present the results of the analysis and shortly discuss them

3.1 Franke Function

First I will discuss the results based on Franke Function. For more detail see (1 Franke's equation).

3.1.1 Confidence intervals



N: 100
Noise: 0.3
Polynomial degree: 5
MSE: 0.09878558507710614
R2: 0.4945817772301846

Figure 5: Confidence Intervals of beta for Franke Function

3.1.2 Bias-Variance Trade-off

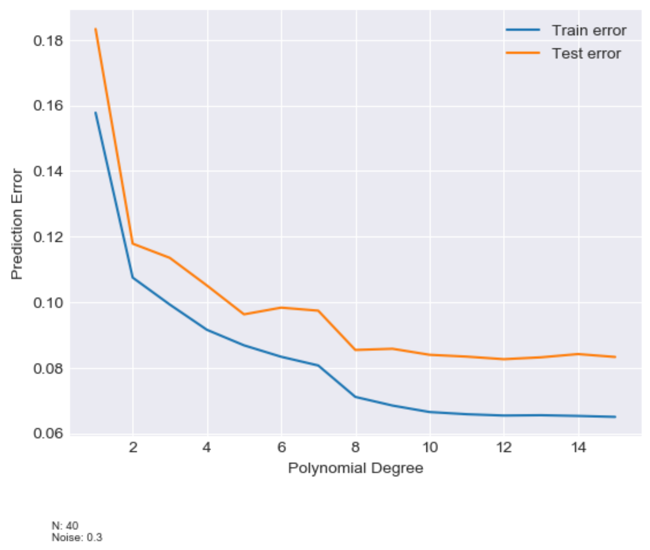


Figure 6: MSE errors of train and test using OLS on Franke data

Since it is quite small sample of 40 points, it is possible to see that overfitting starts already before polynomial degree 10.

3.1.3 Bootstrap/Cross-Validation

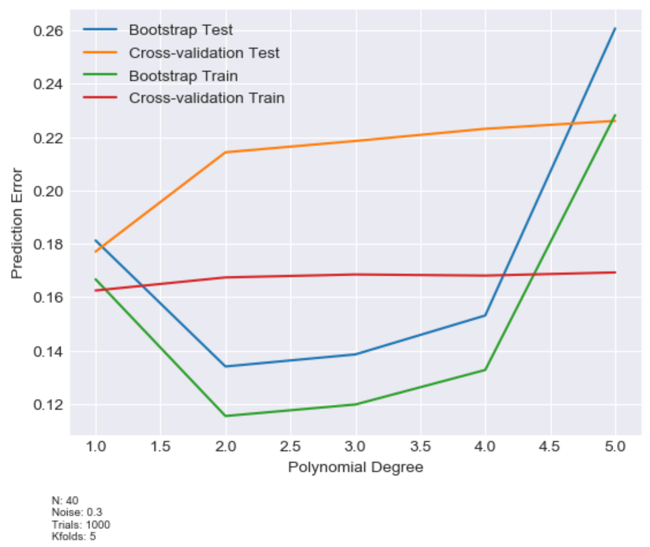


Figure 7: Comparison of Bootstrap and Cross-Validation

As cross-validation method trains on almost all data equally we see that it performs evenly through all polynomial degrees.

3.1.4 Alpha values

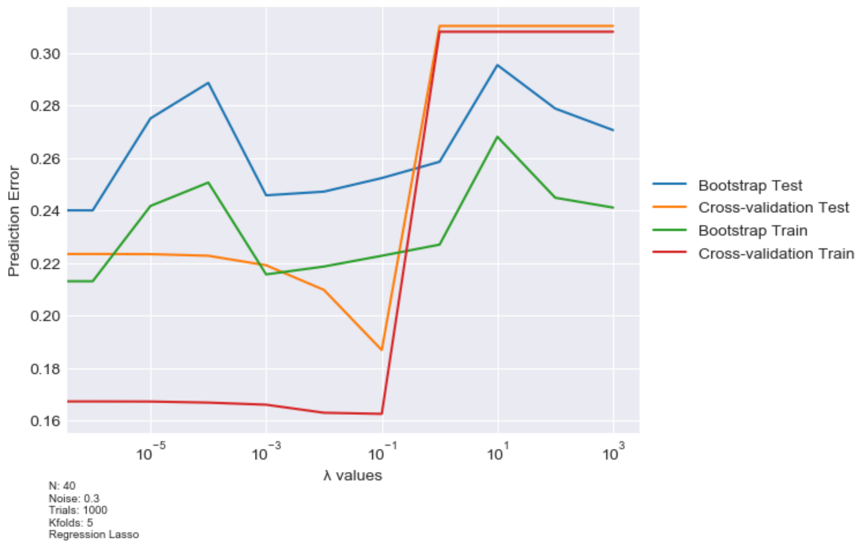


Figure 8: Comparison of Bootstrap and Cross-Validation on different alpha values using Lasso

Lasso in contradiction to ridge sets unnecessary features to zero. In the graph we can see that after 0.1 alpha it quickly jumps to a higher MSE and flattens out.

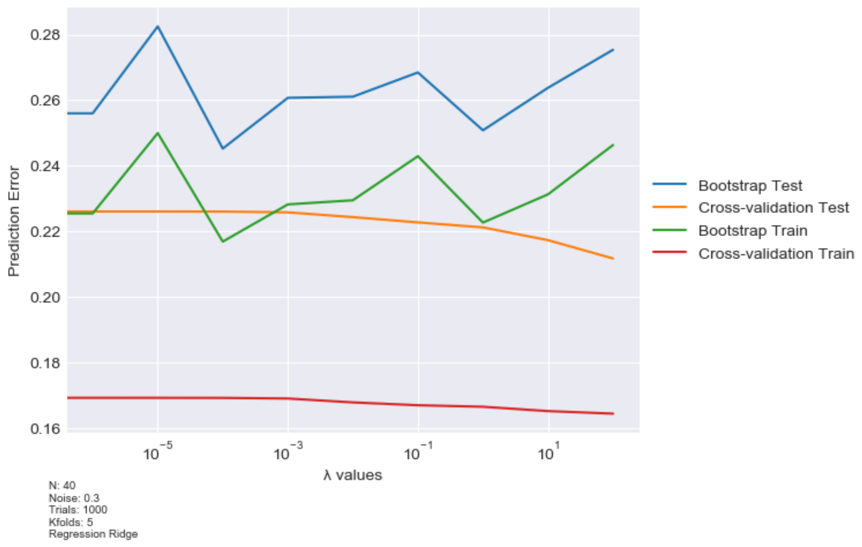


Figure 9: Comparison of Bootstrap and Cross-Validation on different alpha values using Ridge

3.1.5 Ridge vs Lasso

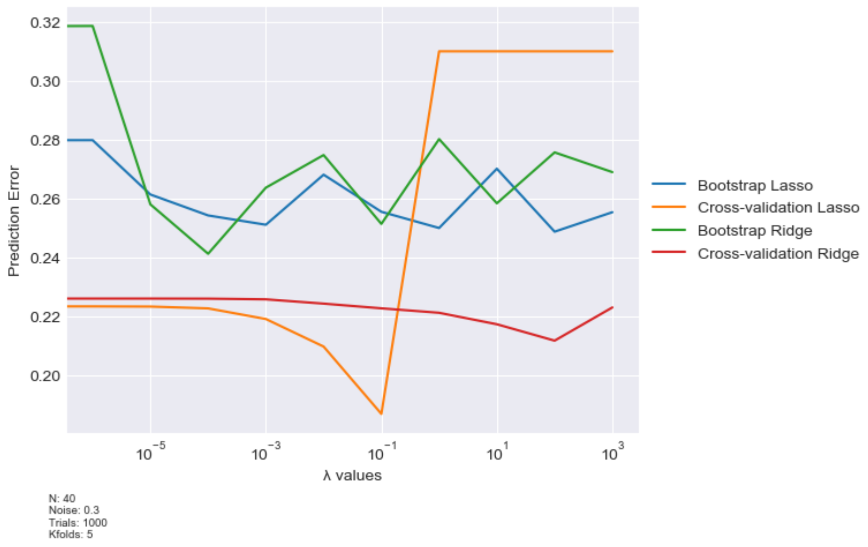


Figure 10: Comparison of Lasso and Ridge alpha

3.1.6 OLS vs Ridge vs Lasso

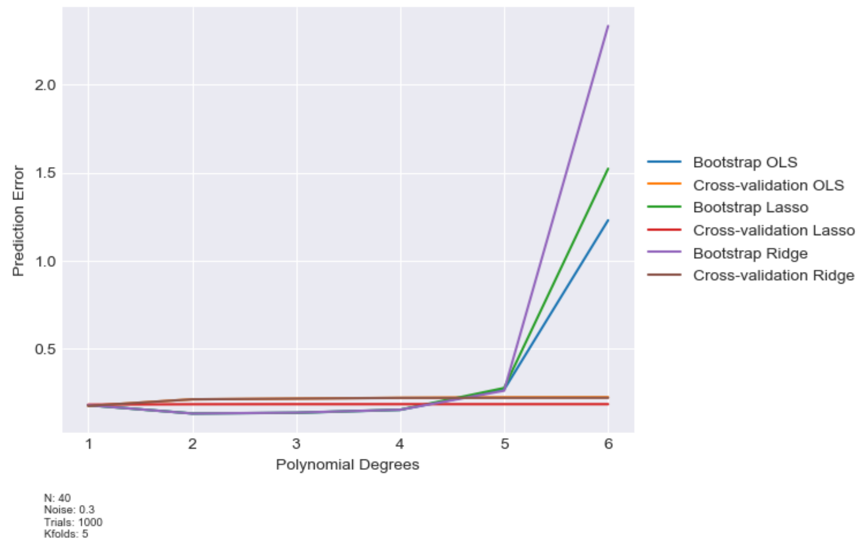
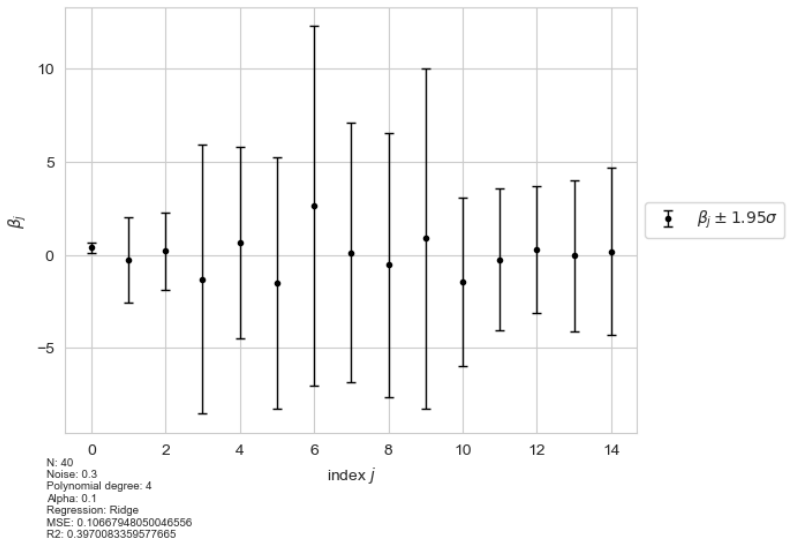


Figure 11: Comparison of Lasso and Ridge alpha

Here we can see a clear result of how Lasso and Ridge manage to force smoother results even with higher polynomials in comparison to OLS.

3.1.7 Best model

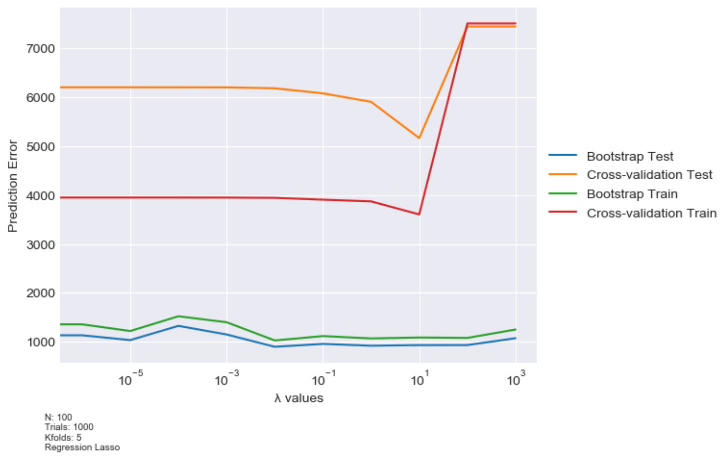


N: 40
Noise: 0.3
Polynomial degree: 4
Alpha: 0.1
Regression: Ridge
MSE: 0.10667948050046556
R2: 0.3970083359577665

Figure 12: Franke's function

3.2 Terrain Data

When the methods got tested on a simpler case, I used digital terrain data and tried to reproduce similar results.



N: 100
Trials: 1000
Kfolds: 5
Regression Lasso

Figure 13: Alpha values for Lasso Regression

Here we can see that it performed quite well when alpha was equal 10. So we continue to use it for bias variance tradeoff:

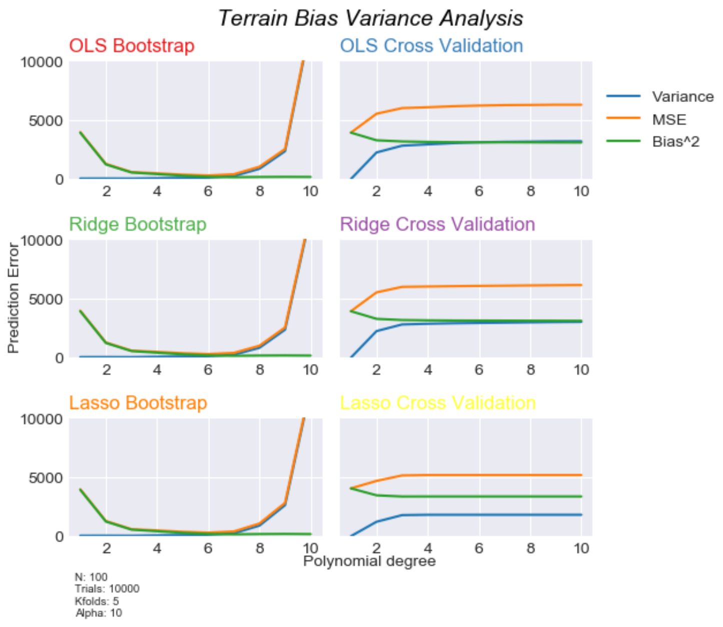


Figure 14: Bias variance trade-off using Ridge, OLS and Lasso

This resulting in the best model on 100 points using Lasso regression with alpha 10 and degree 7. MSE in that case is 6269, R2 is -3199.

4 Conclusion and further work

In this section I will present my suggestions for future work. I will conclude this research with short comments on the process itself and last words about the analysis.

There are plenty of limitations of the newly implemented RegLib library that were mostly discovered during the analysis. The first and foremost is the fact that the CrossValidationKfold method fails when the number of samples are not divisible by the number of kfolds. This happened mostly due to lack of testing, as the majority of tests were run with 100 number of samples and 5 kfolds. To fix this issue one would need to substitute np.reshape to np.array_split function. Reshape is an in-place algorithm, that is very efficient, but it only changes the way the array is viewed. Whilst array_split actually creates a new object with uneven shape which allows for any number of splits of data, even when it is not divisible. Moreover the models have not been tested on data bigger than 100 because of the limited computational power. So for future work would be to test more extensively the effects of having more data points and noise in Franke's function. It would be exciting to see the effects various smoothing kernels have on the results for the terrain data. As this was not the main focus of the project, I chose smoothing algorithm quite hastily.

As this is a project for a university subject, I want to add a few words about the process of working with it and what pitfalls to avoid.

I did not comprehend in the beginning how big of a module we will have to implement, so for the next project or for a next student I would recommend drawing out a code structure from the beginning. Testing played the most important role and I wish I wrote the tests before I started writing my code. It would have saved a lot of debugging and would help me to ask more precise question at the lab.

References

[1] A. Bronshtein. Train/test split and cross validation in python. https://towardsdatascience.com /train-test-split-and-cross-validation-in-python-80b61beca4b6, 2020. Accessed on 02-October-2020.

[2] M. Hjorth-Jensen. Project 1 description. https://compphysics.github.io/MachineLearning/doc /Projects/2020/Project1/html/Project1.html, 2020. Accessed on 02-October-2020.