Project 1 - Regression, Resampling and Data analysis

FYS-STK3155 at University of Oslo

Simen Løken

October 2020

1 Abstract

In this project we'll discuss and analyze three different regression methods and two different examples of resampling. We do this so as to show how one can assess the different methods and how to consider which one is best suited for your particular purpose.
We find that for all methods, variables can be a tricky thing, and we have to be careful when picking 'free varaibles' such as our data points N or our parameters for bootstrap runs or Cross-Validation folds.

2 Introduction

At the heart of machine learning lies a few important statistical methods, namely regression and resampling. Regression is the method of comparing an dependent response variable to another (or several) independent variable(s) and examine their relationship(s). In it's most general form, for some set of variables $\mathbf{X}$, and a dependent variable y, it'd take the form:

$$y = \beta_0 + \beta_1 \mathbf{X} + \epsilon \tag{1}$$

where ϵ is the error
with possibility for expansion given multiple variables $\mathbf{X}$ as:

$$y = \beta_0 + \beta_1 \mathbf{X}_1 + \beta_2 \mathbf{X}_2 \cdots \beta_{n-1} \mathbf{X}_{n-1} + \beta_n \mathbf{X}_n + \epsilon$$

This is where resampling comes in. If we can seperate our dataset $\mathbf{X}$ into subsets, and use one of our subsets to better fit our dependent variable y, we can then use the remaining independent subsets to best possibly fit our data.
In this report we're going to be looking at two statistical methods for resampling, bootstrapping and cross-validation. We're going to be employing three regression analysis methods with each of these, OLS (Ordinary Least Squares), LASSO (Least Absolute Shrinkage and Selection Operator) and Ridge regression. We're going to employ these methods onto two seperate datasets, one we create on our own using the Franke Function, and one supplied by UiO containing geographical data of an area close to Stavanger, Norway.

3 Theory and Method

Like previously mentioned, we're going to be employing the Franke Function, which is defined as:

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

This will allow us, for a randomly generated array x and y, to create a dataset spanning the dimensions of the array x and y, in addition to letting us create a design matrix, henceforth referred to as $\mathbf{X}$. We're defining $\mathbf{X}$ to be:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^1 & y_1^1 & x_1^1 y_1^1 & \cdots & x_1^o y_1^o \\ 1 & x_2^1 & y_2^1 & x_2^1 y_2^1 & \cdots & x_2^o y_2^o \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n^1 & y_n^1 & x_n^1 y_n^1 & \cdots & x_n^o y_n^o \end{bmatrix}$$

where o is the order of our polynomial, in other words, it's complexity. Knowing this, let's take a look at the methods we're going to be using.

3.1 OLS

Given what we know from Equation [1], then an approximation of y $\tilde{y}$ can be expressed as:

$$\tilde{y} = \mathbf{X}\beta \tag{2}$$

We can then define a function over which measures the spread of values y_i and the parameterized $\tilde{y}$.

$$C(\beta) = \frac{1}{n}\left((y - \mathbf{X}\beta)^T(y - \mathbf{X}\beta)\right)$$

which can be simplified as:

$$C(\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{3}$$

Additionally, we have

$$\frac{\delta C(\beta)}{\delta \beta} = 0 = \mathbf{X}^T(y - \mathbf{X}\beta) \tag{4}$$

which then means:

$$\beta = \frac{\mathbf{X}^T y}{\mathbf{X}^T\mathbf{X}} \tag{5}$$

We can then use these values to perform our OLS operation. If we find β through Equation [5], then we can use that to find $\tilde{y}$ and it's accompanying predictor-array through the python code:

```
B = np.linalg.inv(X_train.T @ X_train) @ X_train.T @ y_train
ytilde = X_train @ B
ypred = X_test  @ B
```

which can then be used to finishing with one of our resampling methods.

3.2 Ridge Regression

We modify our existing equation to include such that:

$$\mathbf{X}^T\mathbf{X} \rightarrow \mathbf{X}^T\mathbf{X} + \lambda I$$

As such we modify our other existing equations and get

$$\beta_{Ridge} = \frac{\mathbf{X}^T y + \lambda I}{\mathbf{X}^T y} \tag{6}$$

Our new variables, I and λ are the identity matrix and a parameter, respectively. λ is a parameter that represents the amount of shrinkage in the coefficients β.

3.3 LASSO Regression

LASSO is very similar in form to the Ridge Regression, but it instead yields nonlinear solutions and as such has no closed form expression. We express β as:

$$\beta = argmin \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j+1}^{p} x_{ij}\beta_j \right)^2$$

restricted by

$$\sum_{j=1}^{p} |\beta_j| \leq t$$

which is practically identical to the non-vector form of Equation [6], the only difference being that Equation [6] is resticted by

$$\sum_{j=1}^{p} \beta_j^2 \leq t$$

3.4 Resampling

3.4.1 Bootstrap

The bootstrap method perhaps the most 'basic' of the resampling methods we're going to be using. In a short, non-code, we're looking to:

- Draw numbers for observed variables (training data, in our case), replace afterwards
- Create an array with the drawn values
- Apply these drawn values onto our model
- Return the mean of our calculations
- repeat for N-times
- This is done for each polynomial order, going from 0 up to some polynomial degree o

When we're done with this, we can use our predictive values and retrieve the Mean-Squared-Error of our method for each given polynomial order o

3.4.2 Cross Validation

Cross-validation is, in theory, a bit more "stable" and perhaps more useful than the bootstrap method.
In short, we're looking to

- Shuffle dataset
- Split dataset into k groups

For each unique group:

- Dedicate a group as test data
- Use remaining groups as training data
- Fit training data onto a selected model(OLS, LASSO etc..), evaluate this onto the test data
- Discard model, retain evaluation score.

After doing this we can extract from our evaluation score the Mean-Squared-Error of our method using shrinkage parameters λ

3.5 Estimating Errors

We've talked a fair bit about the Mean-Squared-Error, but we've yet to see what it actually looks like. As given by our assignment [[2]], MSE is defined as:

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

and the R2 score is defined as:

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

These both have in common that they tell you something about how good a fit is in relation to it's actual analytical value. MSE (obviously) tells you the mean-squared error of your fit, or rather. We can then use this in addendum with a regression and resampling model, to best find out which is the best fit for our particular situation. We can also use MSE to find out which polynomial order is the best fit for our situation. If we calculate MSE for each polynomial order, then the one with the lowest value should in theory be the best fit for our particular situation, atleast within area we are checking.
The R2 score measures how much of our variance in our dependent variable is explained by an independent variable. So let's say we have and R2 of .99: That means that .99 of our observed variations can be explain by our model. That means that the closer we are to 1, the better.

4 Results

Firstly, a three-dimensional plot of the Franke Function over a meshgrid is:

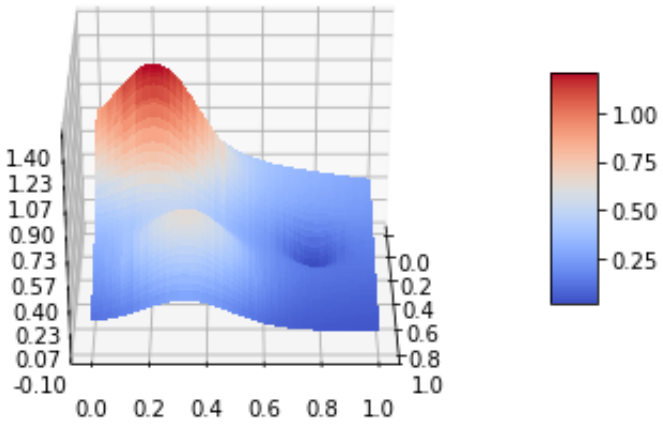
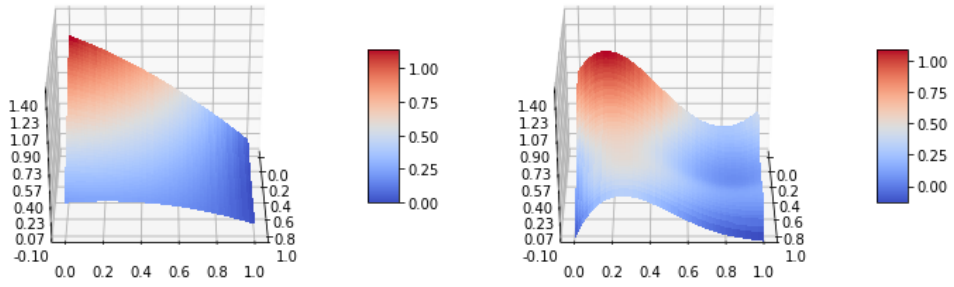


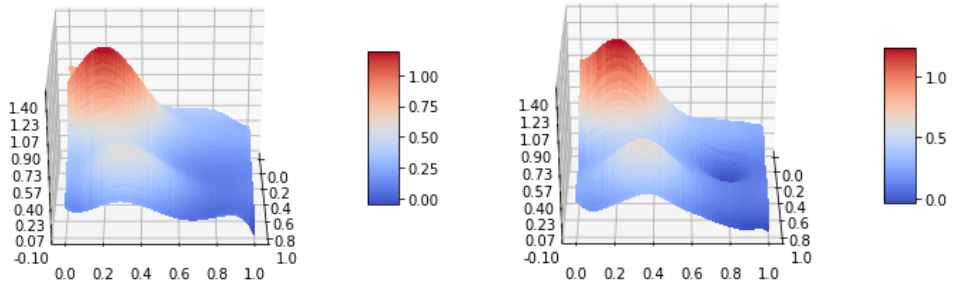
Figure 1: A standard, analytical solution to the Franke Function over a two dimensional meshgrid.

Now let's look at a statistical solution using the OLS method, using different polynomial orders to see which is a better fit. Now let's look at some data given by our OLS-fit for the polynomial order 7:



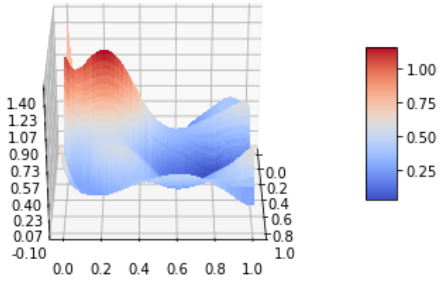
(a) Polynomial order: 2

(b) Polynomial order: 4



(c) Polynomial order: 7

(d) Polynomial order: 20



(e) Polynomial order: 27

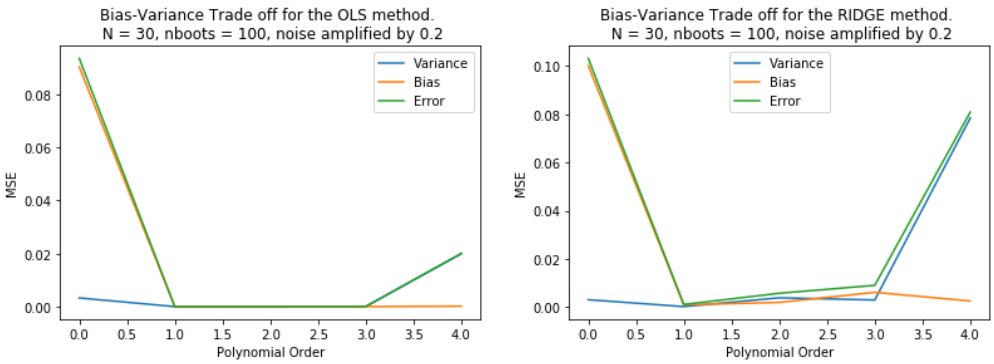
Figure 2: Here are five different statistical solutions using OLS to plot the Franke Function. Notice how as we approach a break-point for the best possible fit (7 in a set with these values) it starts to look more and more like the analytical solution shown in figure [1]. Once we're past that point, we move into overfitting. Notice how for (e) it's features are greatly exaggerated.
These are all without noise, naturally.

We find, using our error estimates:

	Training	Testing
MSE	0.15340805	0.15295584
R2	-0.99540759	-0.99210637

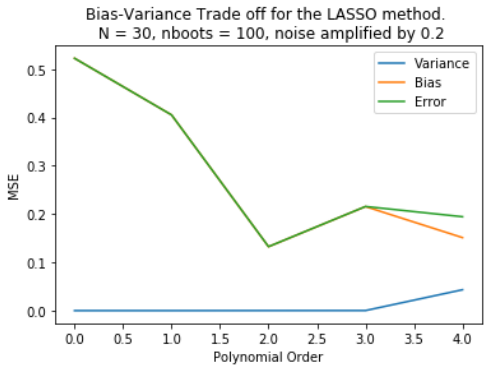
Table 1: Error estimates and R2 scores for polynomial order = 7
I can't explain why R2 is negative. Maybe I was supposed to square it?

Let's now examine the Bias-variance trade-off. We get the following plots:



(a) OLS

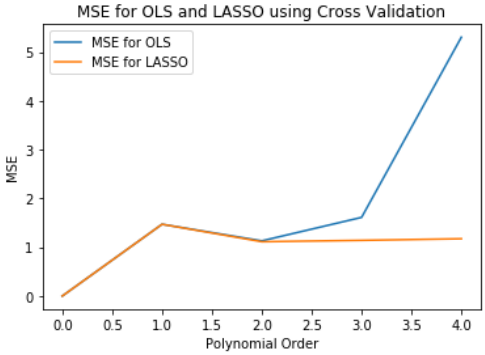
(b) Ridge



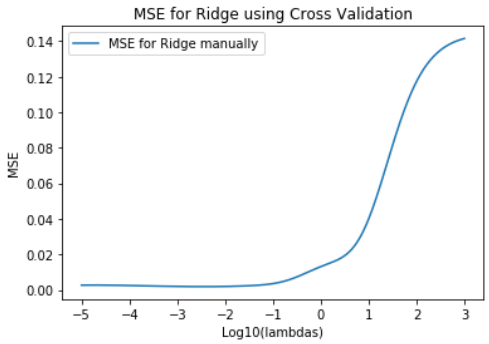
(c) Lasso

Figure 3: Here are three bias-variance trade-off plots for our three different statistical methods using the bootstrap method for resampling.
Based on what we're seeing here, we can assume that for fittings to polynomials of small orders, the OLS and Ridge methods are preferred, while the LASSO method looks to converge somewhere further along our x-axis.

Now, let's take a look at the cross validation:



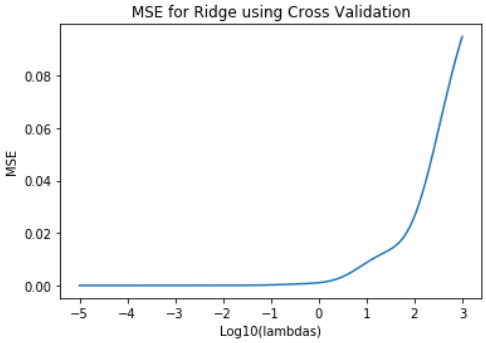
(a) OLS and LASSO



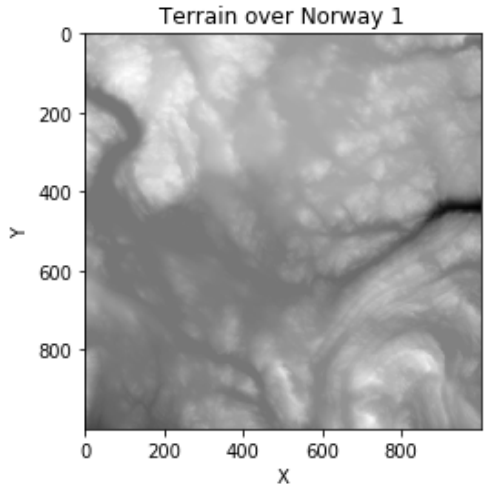
(b) Ridge

Figure 4: Here we can see the MSE given a shrinkage parameter λ. OLS and LASSO are in their own plot using polynomial order because I couldn't get lambdas to work for those methods.

Let's now apply our Cross-Validation method onto our terrain data.



(a) CV-Result for Ridge Regression



(b) The area of the map used

Figure 5: Here we see the area of the map we're collecting data from and the MSE for a given lambda shrinkage parameter.
As explained in the caption for Figure [??], I can't get it to work for OLS and LASSO, and as such we'll be using only Ridge.

5 Discussion

Discussing results

For our Franke Function we can see that it's behaving as expected. We start off initially with a very underfit 3D plot, but as we approach the the ideal polynomial order we see that our statistical model more and more resembles the analytical solution. We also see that as we leave this 'Goldilocks Zone', for lack of a better term, we start having to deal with overfitting where the polynomial order becomes too high, and we get a very "wobbly" and exaggerated plot. This is especially prevalent in the $o = 27$ example, where the features of the analytical solution are greatly exaggerated.
Further, lets look at the Bias-Variance Trade Off. While our results do mirror reality and look like "the real thing", I had some issues getting it to look nice. The Bias-Variance Trade-off plot is a pretty delicate process, and it can be hard to get nice results. As such, I don't think we should be too upset with what we got, but it does seem a bit "sharp". It feels as though variance should be a bit more "aggressive", but that might just be me not having much experience in this subject. That being said, ideally a Bias-Variance trade-off plot should look something like:

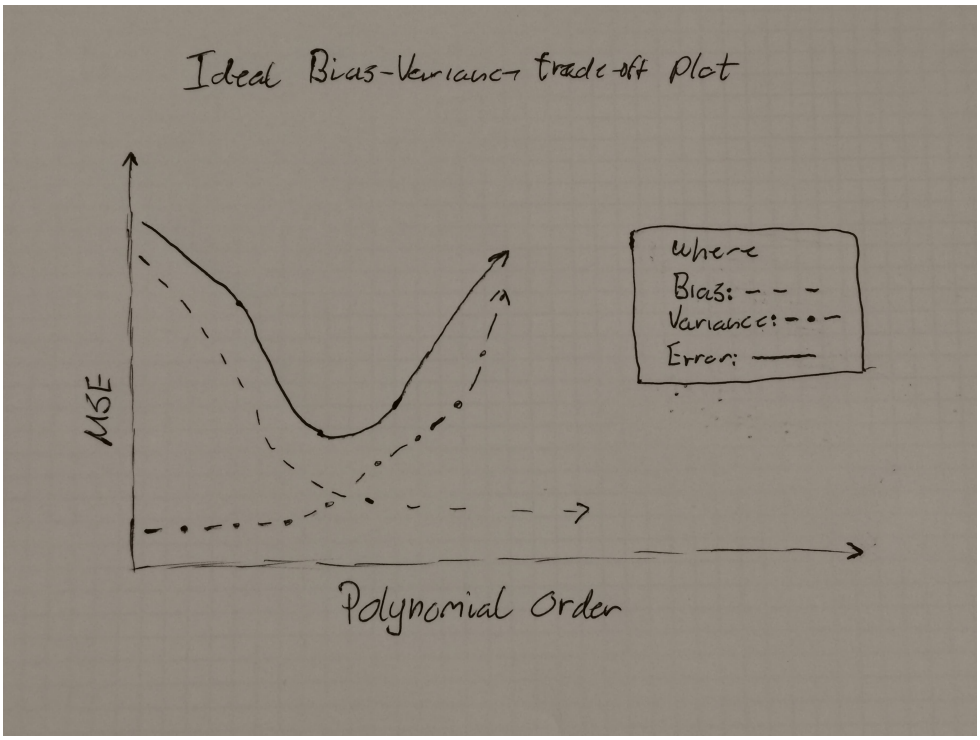


Figure 6: An ideal Bias-Variance Trade-off plot
As we increase our polynomial order, the bias, or the deviation from our data points. However, if we increase too much, our variance begins to grow and takes over. Ideally you'd try to model using a polynomial order within the "Goldilocks Zone" where the Error is the lowest.

Next lets look at our Cross-Validation Plots. It's pretty hard for me to honestly and sincerely discuss these seeing as I'm pretty sure the leftmost plots are entirely wrong. However, for our rightmost plot we can see that for which shrinkage values our model gives us the least amount of error, in other words, our $lambda_{min}$. We see much of the same for our terrain model's shrinkage plot.

Code and reflection

Like I said I had a lot of trouble getting my code to work. This is my first time ever working with statistics and any kind of machine learning/fitting outside of basic best-fit lines. Because of this I kind of realized a bit too late that I was a bit out of my depth in this subject, which probably comes off pretty clearly in this report.
All in all for the latter part of the project I had some rather lackluster results, but I do hope it's not too bad, and at least passable.

6 Conclusion

In conclusion we've seen that fitting data to a model can be a tricky thing, and that different models have different strengths (at least in theory, not so much in execution in this case). Balancing our models can be a delicate process, where-in we need to carefully pick our variables like polynomial order, kfold, data points etc.

References

[1] Morten Hjorth-Jensen. *Lecture Notes 2020, FYS-STK3155.* UiO, 2020.

[2] Morten Hjorth-Jensen. *Project 1 on Machine Learning, FYS-STK3155.* UiO, 2020.

Appendix A - a few comments regarding the project

- I had a serious problem with pretty long run-times for fairly small polynomial orders (a big jump as $o = 7$). I was wondering if this was perhaps because I was using inefficient methods. I do recall hearing something about using pandas to handle our data sets. Could that improve my run-times (for next time)?