FYS-STK H22: Project 1

Lasse Toltand, Domentas Sakalys, Andraes Lie Massey, Synne Mo Sandnes
(Dated: September 26, 2022)

abstracctttttttttttttttttttttt

I. INTRODUCTION

intro

II. THEORY

Some of the formulae presented in this report have been borrowed from the task descriptions' original LaTeX file. [1]

A. Franke Function

The Franke function, which is a weighted sum of four exponentials reads as follows:

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

B. Mean Square Error (MSE)

The Mean Square Error (MSE) of an approximated model $\tilde{y}$, given the inital data y can be expressed:

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

where n is the number of values in $\tilde{y}$.

C. Score Function, R^2

The score function can be used to determine the quality of our model. If $\tilde{y}_i$ is the predicted value of the $i-th$ sample and y_i is the corresponding true value, then the score R^2 is defined as

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

where we have defined the mean value of y as

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i.$$

The score function can have values in the range $R^2 \in [0,1]$ where $R^2 = 1$ would mean that $\tilde{y}$ was a perfect fit of y.

D. Ordinary Least Squares (OLS) method

Given a dataset with n values in the form $y = f(x)+\epsilon$, where $\epsilon \sim N(0,\ \sigma^2)$ is normally distributed noise, $f(x)$ can be approximated via the development of a model of the form:

$$\tilde{y} = X\beta.$$

$X \in \mathbb{R}^{n\times p}$ is a design matrix with columns representing the different features p of the model. For a simple second degree polynomial, the resulting model may look like this:

$$\tilde{y} = \beta_0 + x\beta_1 + x^2\beta_2.$$

β is thus a column vector with an amount of scalar values β_i ($i \in [0,p]$) representing specific features. The OLS method defines it as:

$$\beta^{OLS} = (X^TX)^{-1}X^Ty.$$

E. Ridge method

Ridge regression uses a modified definition of β^{OLS}:

$$\beta^{Ridge} = (X^TX - I\lambda)^{-1}X^Ty.$$

Here $I^{p\times p}$ is the identity matrix, and $\lambda \geq 0$ is a regularization parameter (...)

F. Lasso method

Lasso regression ('Least Absolute Shrinkage and Selection Operator') uses a modified definition of β:

$$\beta_i^{Lasso} = \begin{cases} (y_i - \frac{\lambda}{2}), & \text{if } y_i > \frac{\lambda}{2} \\ (y_i + \frac{\lambda}{2}), & \text{if } y_i < -\frac{\lambda}{2} \\ 0, & \text{if } |y_i| \leq \frac{\lambda}{2} \end{cases},$$

where λ is a regularization parameter as in the Ridge method.

G. Variance of β

H. The Cost function

Given a model $\tilde{y}$, approximating data $y = f(x) + \epsilon$, its corresponding parameters β are in turn found by optimizing the mean squared error via the so-called cost function, which we have rewritten as:

$$\mathbb{E}\left[(y - \tilde{y})^2\right] = (\text{Bias}[\tilde{y}])^2 + \text{var}[\tilde{f}] + \sigma^2,$$

(forklaring av alle leddene)

The derivation of this rewrite is presented in Appendix A.

III. METHOD

We generate the data that will be analyzed by inserting uniformly generated points $x, y \in [0, 1]$, inserting into Franke function $f(x, y)$ and then adding stochastic, normal-distributed noise $\epsilon \sim N(0,\ \sigma^2)$. We define the dataset as:

$$\mathbf{z} = f(\mathbf{x}, \mathbf{y}) + \epsilon,$$

assuming a variance of $\sigma^2 = 0.1$. See Figure 1 for an example of how the generated data may look.

A. Applying the Ordinary Least Squares (OLS) analysis

We wish to approximate the initial function $f(\mathbf{x}, \mathbf{y})$ with our model $\tilde{\mathbf{z}}$ following from the OLS-method:

$$\tilde{z} = X\beta,$$

where X is the design matrix determined by the polynomial degree of our model. The form of our design matrix will correspond to a multiple polynomial regression model (?), meaning each feature of our β will be a product of various degrees of x^i, y^i for $i \in [1, N]$ where N is the polynomial degree of our regression. For $N = 1$ for example, our model will have three features and may look like this (including intercept):

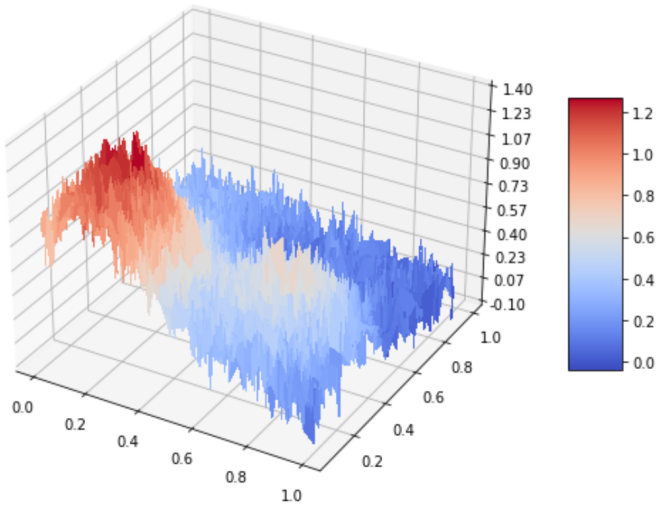

Figure 1. An example of generated data $\mathbf{z}$ as described in the method section.

$$\tilde{\mathbf{z}} = \beta_0 + x\beta_1 + y\beta_2 + xy\beta_3.$$

The python function for constructing the design matrices is borrowed from the lecture notes [2], and can be found in the attached jupyter notebook with our code.

With a defined design matrix, we can can find β^{OLS} as defined in the theory section, and then we should have a complete model $\tilde{z} = X\beta$.

We'll use several strategies to evaluate the quality of our model. Firstly we'll split the data into 'training' and 'test' sets. More specifically, we are going to assign 80% of the data z and the same percentage of the rows in X to the training set, and use it to develop our model (calculate β), then use the remaining 20% to see if the finished model is in correspondence with the remaining data.

To quantitatively evaluate this we'll calculate the Mean Square Error (MSE) and the Score Function (R^2) for the training and test data sets and compare. We'll use the definitions presented in the theory section.

We'll also take into account the polynomial degree N of our OLS regression. A higher degree N will give the model more features, and more parameters to adjust the model to the data, but we'll have to evaluate if the added model complexity gives meaningful improvement or if it leads to overfitting (the model tries to fit to the noise and not the data 'underneath').

To do this we'll compare MSE and R^2 as functions of model complexity N. In addition, we will compare $var[\beta]$ as function of N, which represents the spread of elements in the β column-vector, and may also show signs of overfitting.

B. Bias-Variance trade-off analysis

We assume a dataset $\mathcal{L}$ consisting of the data $\mathbf{X}_\mathcal{L} = \{(y_j, \boldsymbol{x}_j), j = 0 \dots n-1\}$.

As earlier, we assume that the true data is generated from a noisy model

$$y = f(\boldsymbol{x}) + \epsilon.$$

Here ϵ is normally distributed with mean zero and standard deviation σ^2. (...)

IV. RESULTS

A. Results from Ordinary Least Squares analysis

To understand the comparisons of model quality as function of complexity, it may be useful to see what the resulting OLS regression models look like for different polynomial degrees. In Figure 2 we demonstrate how increasing the degree from 2 to 5 gives a gradually more complex terrain, and that the higher degrees seem like more accurate recreations of the initial noisy terrain.

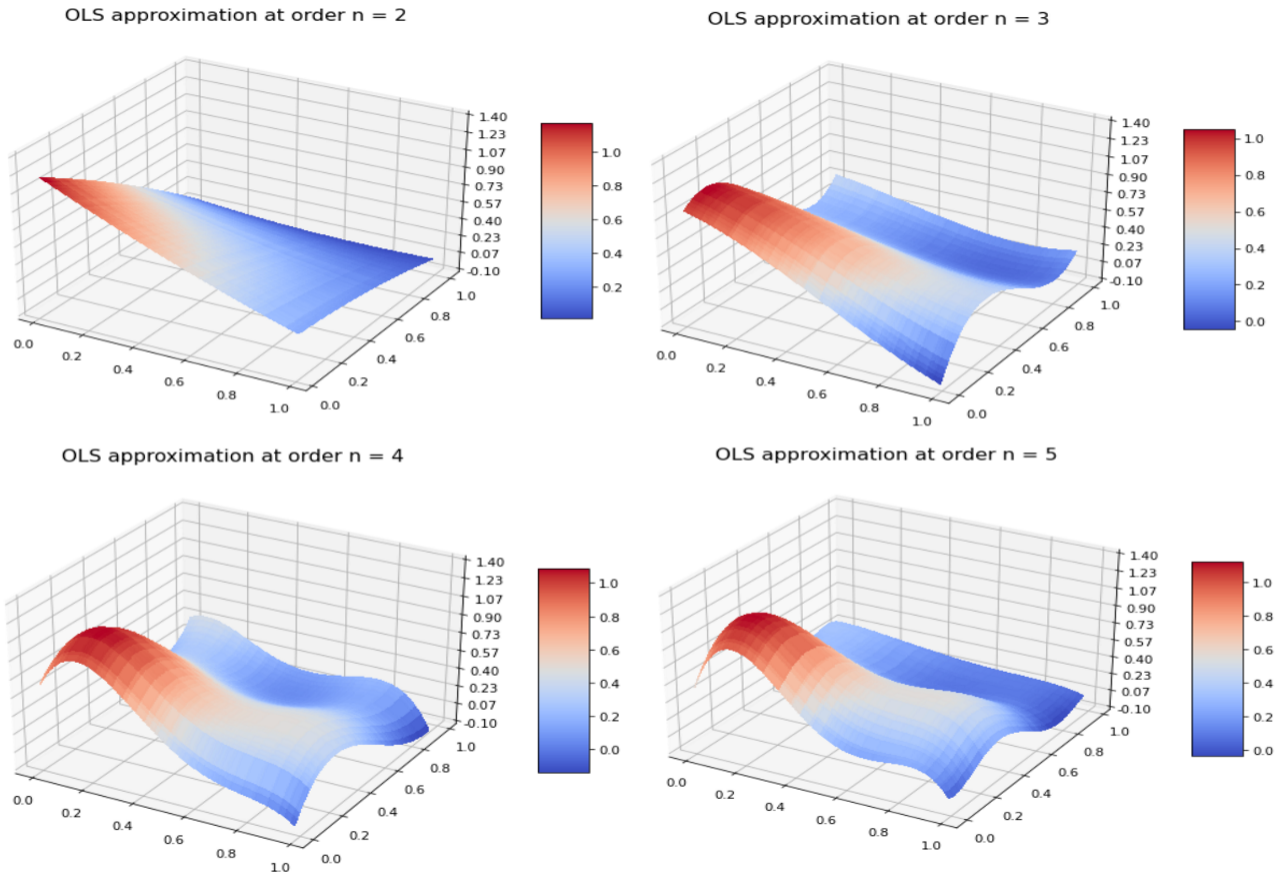

Figure 2. OLS regression models using different polynomial degrees to approximate a generated Franke function terrain data set.

Figure 3 and 4 present the MSE and R^2 from a run of the script, respectively, as functions of model complexity. Figure 5 shows the variance of β as function of model complexity from the same run, as a way to compare the parameters.

V. DISCUSSION

A. Comparison of models

(mention elements of randomness; the outcome of one run isn't necessarily fact)

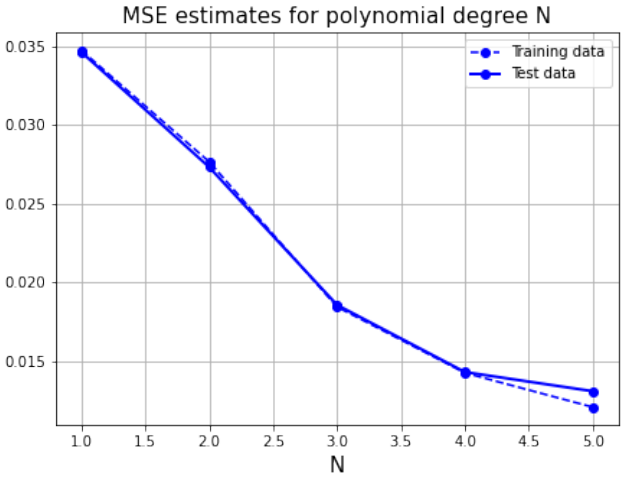

Figure 3. Mean Square Error of the OLS regression on the Franke function training and test sets as function of polynomial degree N.

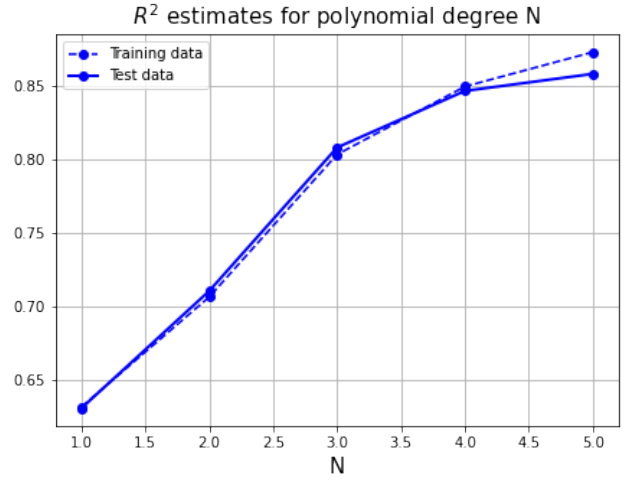

Figure 4. Score Function R^2 of the OLS regression on the Franke function training and test sets as function of polynomial degree N.

VI. CONCLUSION

VII. APPENDIX A: DERIVATIONS

A. The Cost function

The given cost function was defined as

$$C(X,\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(y-\tilde{y})^2\right].$$

Here the expected value $\mathbb{E}$ is the sample value. Given $y = f + \epsilon$ we can equate:

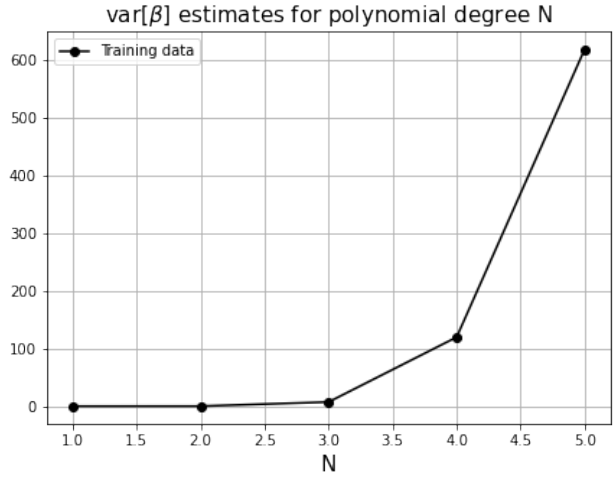

Figure 5. The variance $var[\beta]$ of the OLS regression on the Franke function training and test sets as function of polynomial degree N.

$$\mathbb{E}\left[(y-\tilde{y})^2\right] = \mathbb{E}\left[(f+\epsilon-\tilde{y})^2\right]$$

$$\mathbb{E}\left[(y-\tilde{y})^2\right] = \mathbb{E}\left[(f+\epsilon-\tilde{y}+\mathbb{E}[\tilde{y}]-\mathbb{E}[\tilde{y}])^2\right].$$

This multinomial expansion becomes long and tedious, but can be simplified if we keep in mind that all terms with a factor ϵ or $\mathbb{E}[\epsilon]$ become zero (the expectation value of normal distributed noise with mean value zero gives zero). The remaining terms are:

$$= \mathbb{E}[(y-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[\epsilon^2] + \mathbb{E}[(\mathbb{E}[\tilde{y}]-\tilde{y})^2]$$

$$= (\text{Bias}[\tilde{y}])^2 + \text{var}[\tilde{f}] + \sigma^2. \qquad \blacksquare$$

Here we also used that $(Bias[\tilde{y}])^2 = (y - \mathbb{E}[\tilde{y}])^2$ and that $var[\tilde{f}] = \frac{1}{n}\sum_i(\tilde{y}_i - \mathbb{E}[\tilde{y}])^2$.

[1] FYS-STK4155 Project 1 H22, Morten Hjorth-Jensen.

[2] Applied Data Analysis and Machine Learning lecture notes H22 subsection 4.8, Morten Hjorth-Jensen.