FYS-STK4155 Project 1

Bendik Steinsvåg Dalen & Gabriel Sigurd Cabrera

October 6, 2019

Abstract

dtcfvgbjhkgvfcdxszeasxrdtgyubhnijhugfvcdfguhij ftufykubhk nj jkdsi jkjk dsjknslj fd,nøj ka n,sdf jknm bnjdsajk nmnj mbfdjksa ,nfdsa mn n,asdf

Introduction

The purpose of this report is to analyze the performance of several regression methods – Ordinary Least Squares (OLS), Ridge Regression, and LASSO[1] Regression. In addition to implementing these regression methods, we will be validating (and cross-validating) our results by calculating the *mean squared error*, *bias*, and *variance*, giving us an understanding of how these models' predictive capabilities vary as functions of complexity.

Once we have a clear picture of how these regression models behave, we hope to determine which model best fits our input data. This will require analyzing our validation data and determining what complexity and hyperparameter minimizes our total error.

To actually implement all of this, we will use `python` and create a `class Regression`, initialized with a set of inputs $\mathbf{X} \in \mathbb{R}^{n \times p}$ and $\mathbf{y} \in \mathbb{R}^n$. Once an instance of the object is created, methods such as `Regression.poly(degree, alpha)` or `Regression.lasso(degree, alpha)` can be called in order to perform OLS/Ridge regression or LASSO regression, respectively. Other methods, such as `Regression.mse()` will give us our *mean squared error*.

We will be using two datasets – the first will be generated by the sum of the *Franke function* and some normally distributed noise, and the second will consist of real data taken from the *U.S. Geological Survey*.

Data

The Franke Function

The first dataset will be given by the *Franke function*, which is defined as follows:

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

We will be solving the Franke function for 100 x-values and 100 y-values in the range $[0, 1]$, leaving us with a grid containing a total of 10000 xy coordinate pairs. This leaves us with the values plotted in Figure 1.

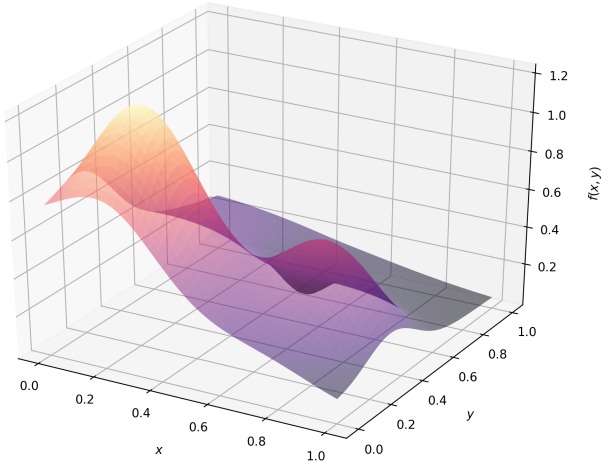

Figure 1: The *Franke function* for x and y values ranging from zero to one.

In addition, we will also be adding *Gaussian noise* to each value $f(x, y)$, such that we are left with values as seen in Figure 2.

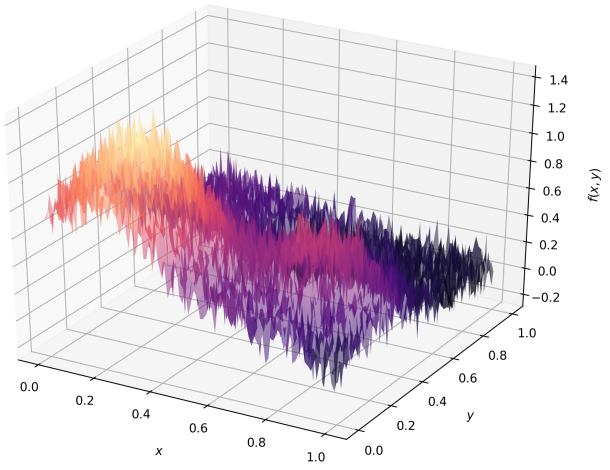

Figure 2: The *Franke function* for x and y values ranging from zero to one, with a Gaussian noise $N(0, 0.01)$

[1] Short for *least absolute shrinkage and selection operator*.

Møsvatn Austfjell

For our second dataset, we will be using real data taken from the *U.S. Geological Survey* [?] official website https://earthexplorer.usgs.gov/. More specifically, we will be using a `.tif` file containing altitude data for a rectangular region of Møsvatn Austfjell shown in Figure 3.

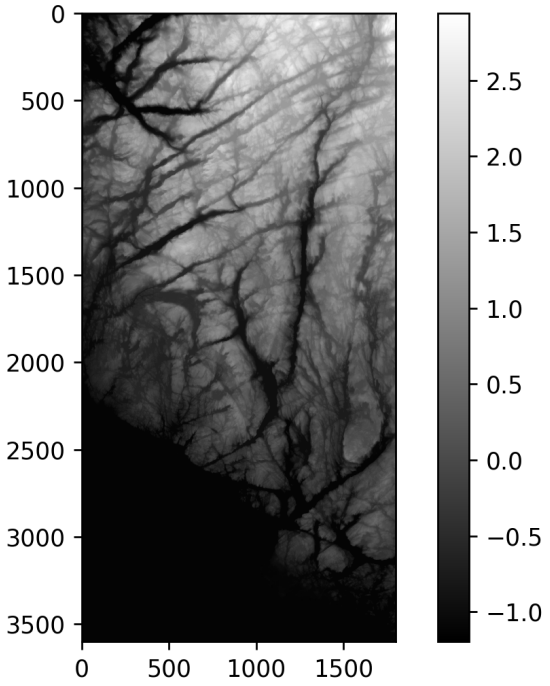

Figure 3: Altitude data from Møsvatn Austfjell, from the USGS website [?].

Method

Generalization of Multidimensional Polynomials

If we wish to construct a p-dimensional polynomial of degree d, we need to know what terms need to be included to give us a completely generalized polynomial. For a 1-D polynomial of second degree, we would have three terms:

$$f(x) = \beta_1 + \beta_2 x + \beta_3 x^2$$

Where β is a vector containing each coefficient. For a 2-D polynomial of second degree, we would have 6 terms:

$$f(x, y) = \beta_1 + \beta_2 x + \beta_3 y + \beta_4 xy + \beta_5 x^2 + \beta_6 y^2$$

And for a 3-D polynomial of second degree, we would have 10 terms:

$$\begin{aligned} f(x, y, z) = \beta_1 &+ \beta_2 x + \beta_3 y + \beta_4 z + \beta_5 xy \\ &+ \beta_6 xz + \beta_7 yz + \beta_8 x^2 + \beta_9 y^2 + \beta_{10} z^2 \end{aligned}$$

There are many possible combinations of p and d, and the number of terms blows up significantly as these values increase. We can, however, create a general expression [?] for any p and d using summation notation:

$$f(\mathbf{x}) = \sum_{\sum_{j=1}^{d} i_j \leq p} \left(\beta_{i_1, i_2, \cdots, i_d} \prod_{k=1}^{d} x_k^{i_k} \right) \tag{1}$$

Alternatively, a simple `python` script can be used to find all the terms' exponents by calculating all permutations of the natural numbers from zero to d in sets of length p, then removing all results whose sum is greater than d.

```
powers = np.arange(0, degree + 1, 1)
powers = np.repeat(powers, p)
exponents = list(permutations(powers, p))
exponents = np.unique(exponents, axis = 0)

if p != 1:
expo_sum = np.sum(exponents, axis = 1)
valid_idx = np.where(np.less_equal(expo_sum, degree))
    ↪ [0]
exponents = np.array(exponents, dtype = np.int64)
exponents = exponents[valid_idx]
else:
exponents = np.array(exponents, dtype = np.int64)
```

Ordinary Least-Squares (OLS) Regression

We are given a $p+1$-dimensional dataset[2] consisting of N datapoints per feature such that:

$$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,p} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N,1} & X_{N,2} & \cdots & X_{N,p} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{2}$$

Where the $N \times p$ matrix $\mathbf{X}$ contains the dataset's *input data*, and the N-vector $\mathbf{y}$ contains its *output data*, such that each row in $\mathbf{X}$ corresponds to a single output in $\mathbf{y}$.

Now, we wish to find a p-dimensional polynomial of degree d which most closely matches our dataset. We will need a design matrix $\mathbf{A}$; this will require using the knowledge presented in (1), since a design matrix should contain each polynomial term as an individual column.

Next, we will be using the method of *least squares* [?], whereby we attempt to minimize the *residual sum of squares*:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - \mathbf{A}_i^{\mathrm{T}} \beta)^2$$

Where $\mathbf{A}_i$ represents the i^{th} row in $\mathbf{A}$, and *beta* is the set of coefficents in the aforementioned polynomial; in matrix form, this can be written more concisely:

$$RSS(\beta) = (\mathbf{y} - \mathbf{A}\beta)^{\mathrm{T}} (\mathbf{y} - \mathbf{A}\beta)$$

To minimize the RSS, we can differentiate it with respect to β and set the right-hand side equal to zero – this allows us to solve for β, which will give us the coefficients to the polynomial that best matches our dataset:

$$\mathbf{A}^{\mathrm{T}}(\mathbf{y} - \mathbf{A}\beta) = 0 \iff \mathbf{A}^{\mathrm{T}} y = \mathbf{A}^{\mathrm{T}} \mathbf{A} \beta$$

Solving for β then gives us our desired result:

$$\beta = (\mathbf{A}^{\mathrm{T}} \mathbf{A})^{-1} \mathbf{A}^{\mathrm{T}} \mathbf{y} \tag{3}$$

Using the set of coefficients given by (3), we can then match the dataset from (2) as effectively as possible.

[2] Meaning a set of p input features and 1 output.

Ridge Regression

The solution for β given in (3) can be used without issue in many cases, but if the matrix $\mathbf{A}$ is singular[3], we run into an issue – namely, we cannot take the inverse of a singular matrix! As a result, we must look to more robust methods; one such method is called *ridge regression*.

The process of obtaining our vector of coefficients β via ridge regression is functionally very similar to that of OLS. The main difference is that we include an extra term in the residual sum of squares:

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - \mathbf{A}_i^{\mathrm{T}}\beta)^2 + \lambda\sum_{i=1}^{N}\beta^2$$

In matrix form, this can be rewritten:

$$RSS(\beta) = (\mathbf{y} - \mathbf{A}\beta)^{\mathrm{T}}(\mathbf{y} - \mathbf{A}\beta) + \lambda\beta^{\mathrm{T}}\beta$$

Where λ, known as the *hyperparameter*, is a scalar value. Performing the same process as in the previous subsection, we are left with a solution similar to that in (3):

$$\beta = (\mathbf{A}^{\mathrm{T}}\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{y} \tag{4}$$

In cases where $\mathbf{A}$ is singular, it is therefore possible to make very few changes to the OLS algorithm and still get a good result, one must simply optimize the hyperparameter and find a λ that minimizes the *mean squared error* (yet to be introduced) of our polynomial approximation.

LASSO Regression

Algorithm 1 The LASSO algorithm, over the course of 500 iterations.

$z = \sum_i A_i^2$
$i = 0$
while $i \leq 500$ **do**
  $i = i + 1$
  $j = 0$
  while j ¡ p **do**
    $\hat{y} = \sum_{k \neq j} \beta A_{*,k}$
    $\rho = \sum_k A_{*,k}(\mathbf{y} - \hat{\mathbf{y}})$
    if $\rho < -\lambda/2$ **then**
      $\beta_j = (\rho + \lambda/2)/z_j$
    else if $\rho > \lambda/2$ **then**
      $\beta_j = (\rho - \lambda/2)/z_j$
    else
      $\beta_j = 0$
    end if
  end while
end while

The *least absolute shrinkage and selection operator*, commonly abbreviated as *LASSO*, is a method that implements the **L1** norm in place of the **L2** (or Euclidian) norm used in ridge regression. The residual sum of squares is therefore given by:

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - \mathbf{A}_i^{\mathrm{T}}\beta)^2 + \lambda\sum_{i=1}^{N}|\beta| \tag{5}$$

[3]Meaning that $\det(\mathbf{A}) = 0$

Unfortunately, differentiating the above with respect to β will not work as intended, since we cannot take the matrix-form derivative of $\lambda\sum_{i=1}^{N}|\beta|$. As a result, we must use an iterative *gradient descent* method to minimize the right-hand side of (5).

Mean Squared Error

To get a measure of success with respect to the implemented method and parameters, we can calculate the mean difference in the squares of each measured output y_i and their respective predicted outputs $\hat{y}_i$:

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] \tag{6}$$

The lower the MSE, the closer the polynomial approximation is to the original dataset. If it is too low, however, we run the risk of overfitting our dataset, which is not desireable either – fortunately, this not an issue within the scope of this report.

$\mathbf{R^2}$ Score

Another measure of success is the *coefficient of determination*, colloquially known as the R^2 score, is given by the following expression:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \tag{7}$$

The closer R^2 is to one, the closer the polynomial approximation is to the input/output dataset, although a perfect score can once again arise due to overfitting just as in the case of the MSE.

Bias-Variance Tradeoff

Before we continue, we can decompose the range of outputs $\mathbf{y}$ as follows:

$$\mathbf{y}(\mathbf{X}) = f(\mathbf{X}) + N(0, \sigma) \tag{8}$$

Where $f(\mathbf{X})$ represents the *actual* function used to generate the dataset, and $N(0, \sigma)$ is a Gaussian noise with a standard deviation of σ.

As a regression model increases in complexity[4], it so happens that the *variance* of a prediction increases. Variance is defined as follows:

$$\mathrm{Var}(\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbb{E}[\mathbf{y}]) \tag{9}$$

On the other hand, we have that the *bias* of the prediction decreases as the complexity increases. We define the bias as:

$$\mathrm{Bias}(\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\mathbf{y}]) \tag{10}$$

Note that in order to calculate the bias, we need to know the original function f used to generate $\mathbf{y}$.

Interestingly enough, taking the sum of (9) and (10) as well as σ^2 will yield the *mean squared error*. We will show this to be the case in the following subsection.

[4]For a polynomial regression, this would refer to its *degree*.

Derivation

We wish to show that:

$$\mathbb{E}\left[(\mathbf{y}-\hat{\mathbf{y}})^2\right] = \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \sigma^2 \tag{11}$$

We begin by rewriting the MSE into summation notation, decomposing the terms as defined in (8), and adding/-subtracting a term $\mathbb{E}[\hat{\mathbf{y}}]$:

$$\begin{aligned}\mathbb{E}\left[(\mathbf{y}-\hat{\mathbf{y}})^2\right] &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(f_i + \varepsilon - \hat{y}_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(f_i + \varepsilon - \hat{y}_i + \mathbb{E}[\hat{\mathbf{y}}] - \mathbb{E}[\hat{\mathbf{y}}])^2\end{aligned}$$

Next, we set $a \equiv f_i - \mathbb{E}[\hat{\mathbf{y}}]$ and $b \equiv \hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]$ and expand:

$$\frac{1}{n}\sum_{i=1}^{n}(a-b+\varepsilon)^2 = \frac{1}{n}\sum_{i=1}^{n}(a^2 - 2ab + b^2 - 2b\varepsilon + \varepsilon^2 + 2a\varepsilon)$$

The next few steps are messy, and require lots of algebraic manipulation:

$$\begin{aligned}&= \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\varepsilon^2) + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 \\ &- \frac{2}{n}\sum_{i=1}^{n}\varepsilon(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]) + \frac{2}{n}\sum_{i=1}^{n}\varepsilon(f_i - \mathbb{E}[\hat{\mathbf{y}}]) \\ &- \frac{2}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]) = \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 \\ &+ \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \sigma^2 - \mathbb{E}[\varepsilon]\frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}]) \\ &+ \mathbb{E}[\varepsilon]\frac{2}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}]) - \frac{2}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])\end{aligned}$$

Finally, we see that our original assumption given by (11) is correct.

$$= \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \sigma^2 \qquad \square$$

Where $\frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])$ is the *bias* and $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2$ is the *variance*.

Cross-Validation

So far, all our methods involving validation[5] may have involved the separation of our data into a training and testing set, whereby the vector of coefficients is calculated using the *training* set, and the validation is performed on the *testing* set that remains. There is however one more way to obtain a clearer picture of how well a model works: *k-fold* cross validation.

In short, once the training and testing set have been separated, we can choose a value for k. Next, we divide the training set into k equally sized parts[6]. The next step is to be performed k times; here we take $k-1$ of the parts and combine them into a temporary training set, and leave the last part as our testing set, and we perform validation on the testing set, and save the values. During each iteration, we must shuffle our parts such that each step has a unique training-testing split.

Finally, we can take all the calculated MSE values, among others, and take their average. This leaves us with a well-rounded result *without* the need for more input data!

Results

Franke function

Following bellow ia several plots we made of our result after studying regression on the Franke function. All the plots shows the results for both 1 to 5 polynomil degrees and 1 to 10 polynomial degrees.

Figure 4 to 7 details the ordinary least square method. Figure 4 shows the variance, mean square error and the R^2-score. Figure 5 shows the mean square error and the R^2-score after performing k-fold cross validation. Figure 6 shows the bias and the variance. Figure 7 compares the MSE of the training set and the testing set.

Figure 8 to **??** detail the Ridge method. osv...

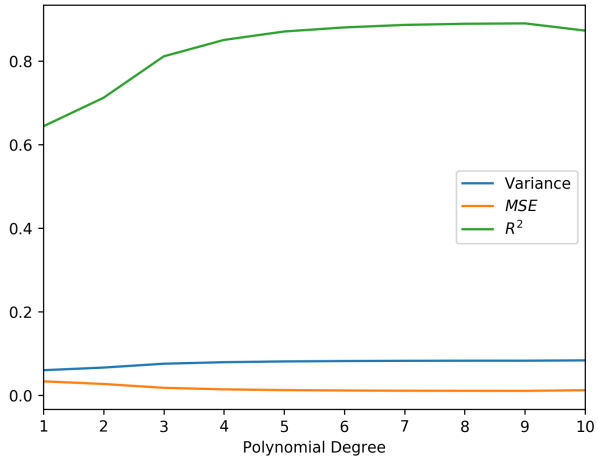

Figure 4: The MSE, the R^2-score, and the variance σ in the vector of coefficients β, as functions of the polynomial degree after performing OLS on the Franke function

[5] This refers to calculating the MSE, variance, bias, and so on.

[6] We can have slightly unequal-sized parts without it being an issue, if the size of the dataset doesn't divide perfectly into k.

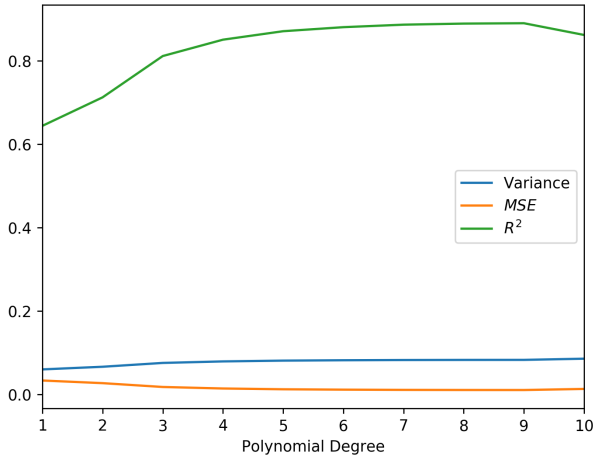

Figure 5: The MSE, the R^2-score, and the variance σ in the vector of coefficients β as functions of the polynomial degree after performing OLS on the Franke function. Using 12-fold cross validation.

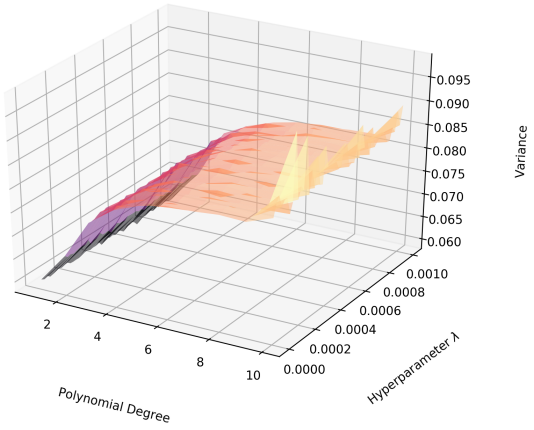

Figure 8: The *variance* as a function of polynomial degree and hyperparameter λ, after performing *Ridge regression* on the Franke function

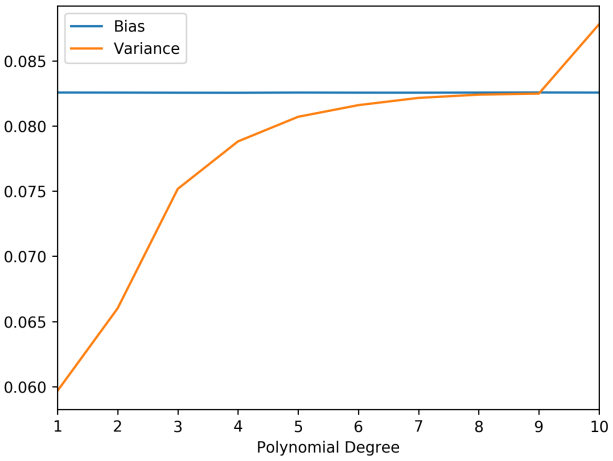

Figure 6: The *bias* and *variance* as functions of the polynomial degree after performing OLS on the Franke function

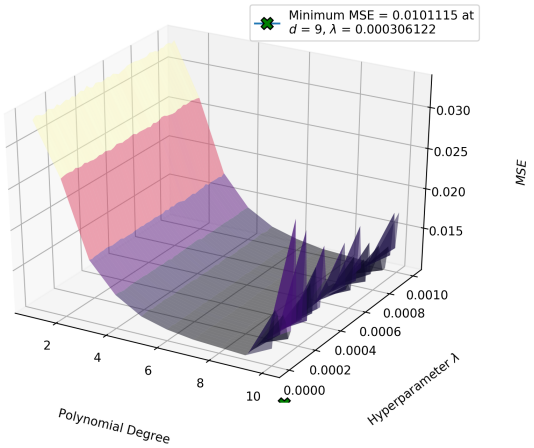

Figure 9: The MSE as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

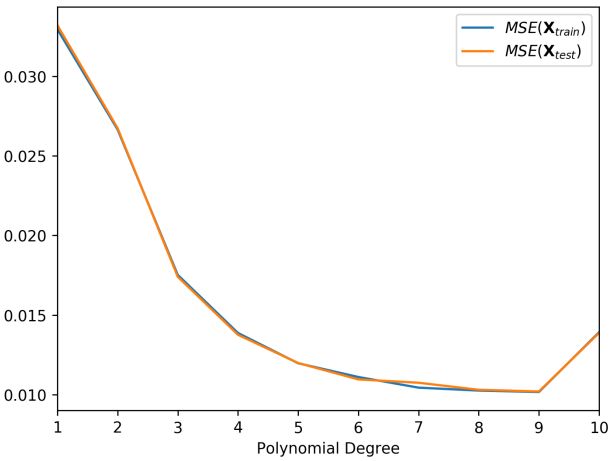

Figure 7: The MSE for the *training data* and the *testing data*, as a function of the polynomial degree after performing OLS on the Franke function

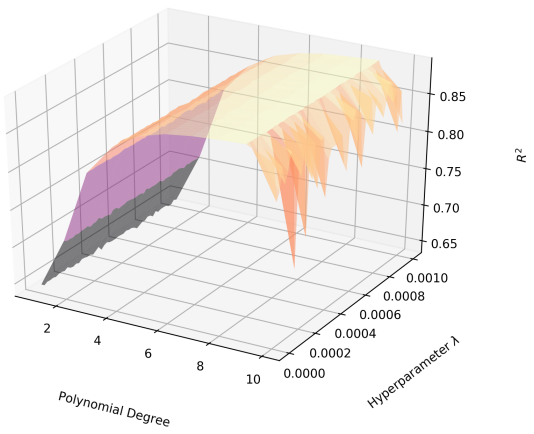

Figure 10: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

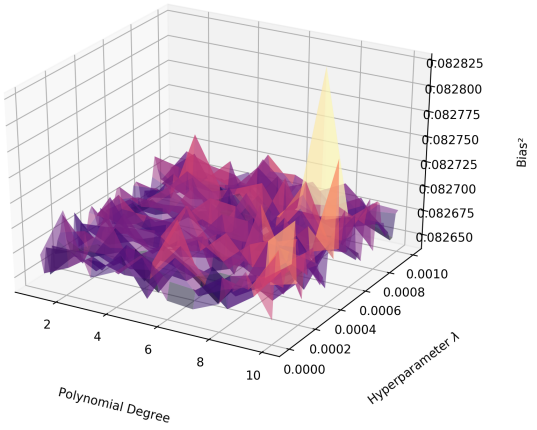

Figure 11: The *squared bias* as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

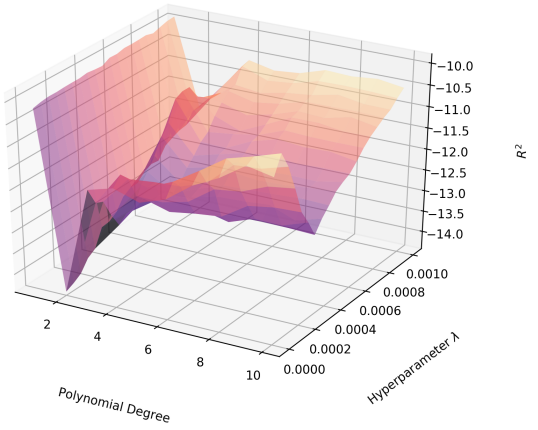

Figure 14: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing *LASSO regression* on the Franke function

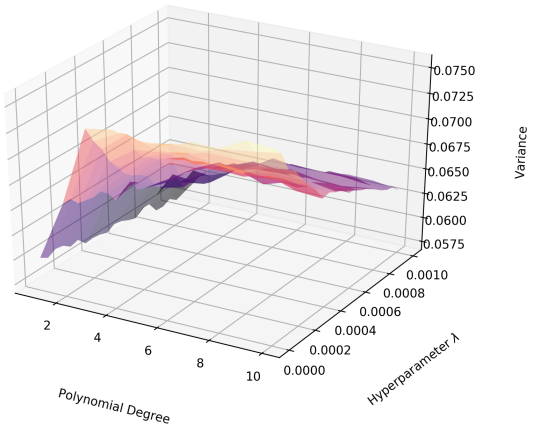

Figure 12: The *variance* as a function of the polynomial degree and hyperparameter λ after performing *LASSO regression* on the Franke function

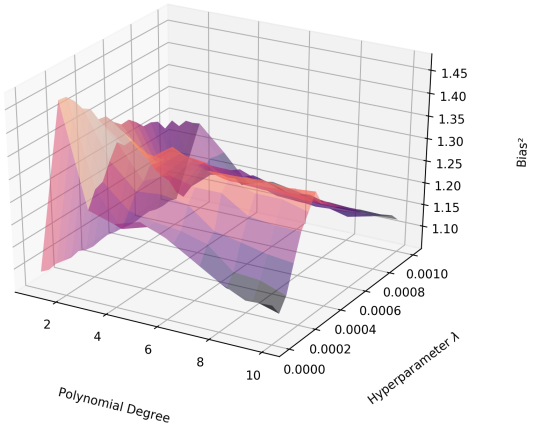

Figure 15: The *squared bias* as a function of the polynomial degree and hyperparameter λ after performing *LASSO regression* on the Franke function

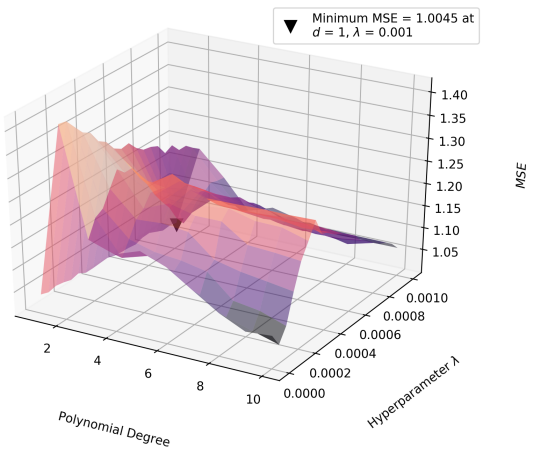

Figure 13: The MSE as a function of the polynomial degree and hyperparameter λ after performing *LASSO regression* on the Franke function

Real Data

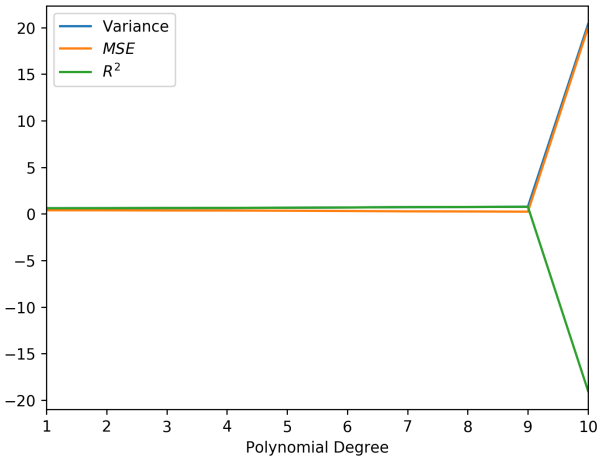

Figure 16: The MSE, R^2-score and variance σ of the vector of coefficients β as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*

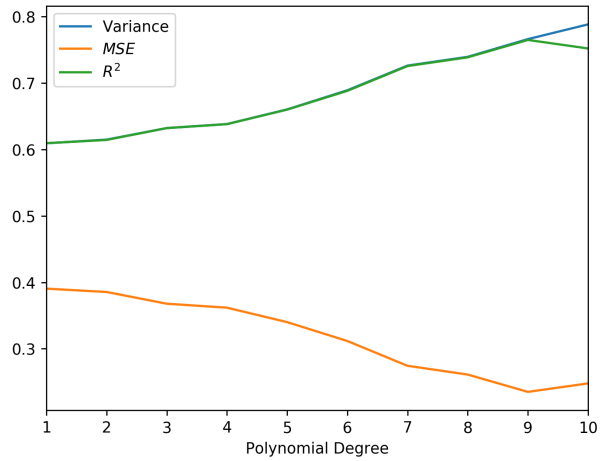

Figure 17: The MSE, R^2-score and variance σ of the vector of coefficients β as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*. Using 12-fold cross validation.

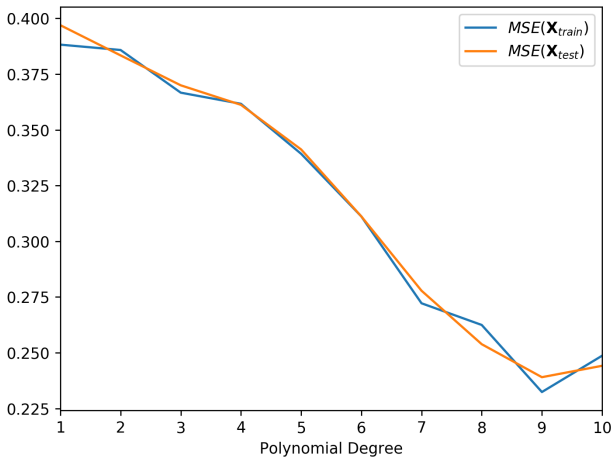

Figure 18: The MSE for the *training data* and the *testing data*, as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*

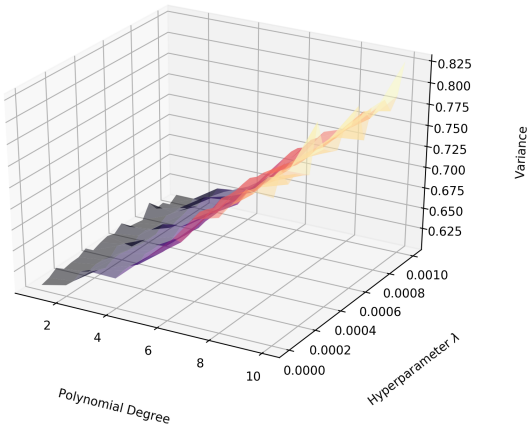

Figure 19: The *variance* as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

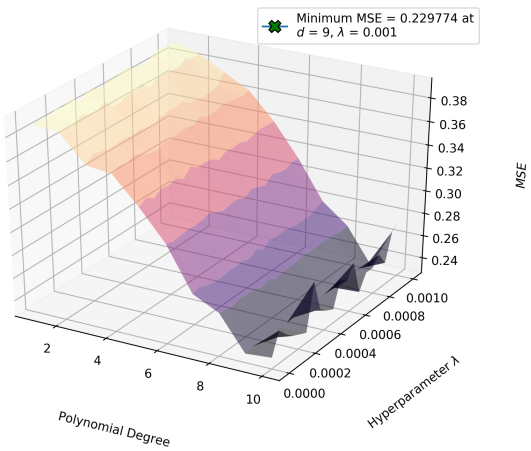

Figure 20: The MSE as a function of the polynomial degree and the hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

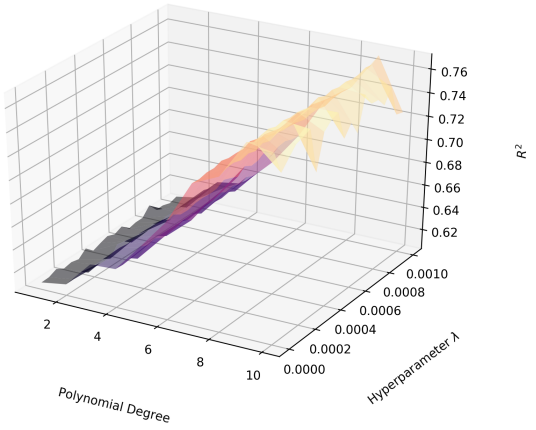

Figure 21: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

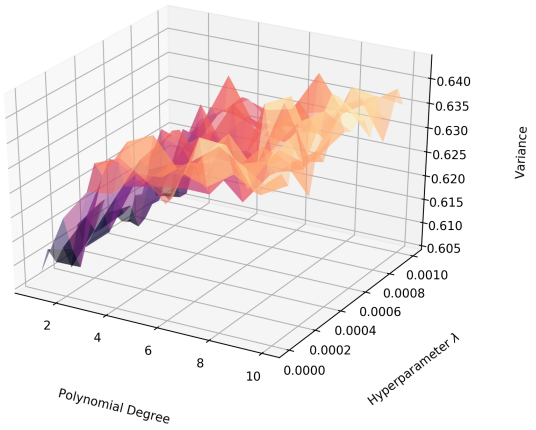

Figure 22: The *variance* as a function of the polynomial degree and hyperparameter λ after performing *LASSO regression* on real terrain data from *Møsvatn Austfjell*

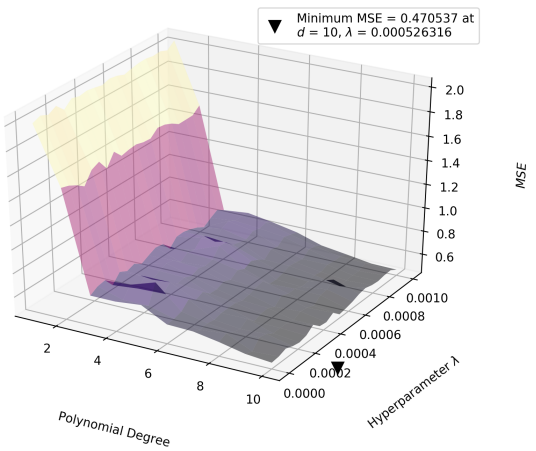

Figure 23: Plots of the MSE as a function of the polynomial degree and hyperparameter λ after performing *LASSO regression* on real terrain data from *Møsvatn Austfjell*

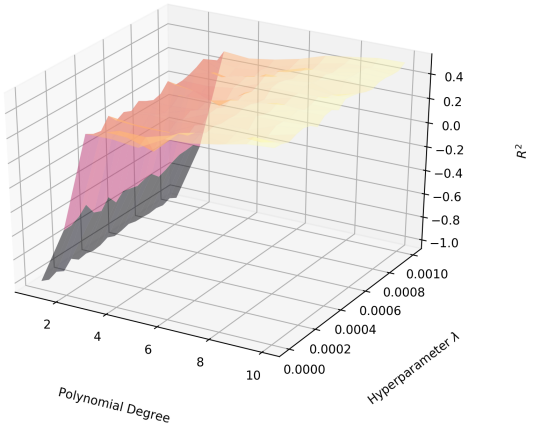

Figure 24: The R^2-score as a function of the polynomial degre and hyperparameter λ after performing *LASSO regression* on real terrain data from *Møsvatn Austfjell*

Discussion

franke

In Figure 4, we see that the *variance* increases as a function of complexity (as expected) – in addition, it sharply increases around a degree of 10. It seems that the MSE also begins to increase significantly, while the R^2 decreases at an equal but negative rate.

In Figure 5, it appears that implementing cross-validation stabilizes our results (albeit only by a slight amount)such that the sharp increase is less pronounced than previously in Figure 4.

Given that we know the *Franke function*, we are able to calculate the *bias* in our model. In Figure 6 We can see the *bias-variance* tradeoff in action.

In Figure 7, we see that the bias, on average, tends to decrease.

Figure 7: Pretty similar results for test and training, which I think is good, though train is slightly lower. Both have a minimum at $n = 9$ and start increasing after that, as in figure 5.

Figure 8: Not moch to say here, the variance seems to flatten out as n increases, suggesting diminishing returns. Much difference in result based on λ.

Figure 9: Much the same as in figure 8. Minimum MSE happens at $n = 9$, which is before the overflow, suggesting that this is the highest n can be before overflow. Minimum is also at lowest λ, suggesting that an even lower value could give even better results.

Figure 10: Difference in results flattens out as n increases. Not much difference in result based on λ, but best result happens at lower values, with a seeming spike at the lowest, coinsiding with the MSE.

Figure 11: Can't see any particular pattern here.

Figure 12: Reults flattens out as λ suggesting diminising results, but are highest for low λ. n dosen't seem to have a large impact on variance.

Figure 13: n has a small impact on results. MSE drops quickly for lower λ. Best value at highest n and lowest λ, suggesting a better result could be found for higher n and lower λ.

Figure 14: n has a small impact on results. Best value at lower λ, coninciding with MSE.

Figure 15: Same as figure 11.

OLS>Ridge>Lasso for MSE of Franke data. OLS and ridge best at $n = 9$, Lasso at $n = 10$. Best values at lowest λ as well.

real

Figure 16: Pretty much what was expected, variance and MSE increases and R^2 decrases as n increases. Total values are lower than for Franke, meaning higher uncertainty/data that is harder to interpret/fit to polynomial, makes sense as it's real life data. More or less exact overlap between variance and R^2.

Figure 17: Results are slightly better than without cross-validation.

Figure 18: MSE of train and test is pretty close, though more variance in test.

Figure 19: Unlike Franke, the data dosen't flatten out for any particular variable

Figure 20: Best result not at a extreme for λ, suggesting either that the best result is within tested λ-values or relativley low impact from λ. Best result is at highest tested n, but there's unlikely to be a beter result as $n = 10$ gave overflow. Minimum MSE is smaller than for Franke, as figure 16.

Figure 21: Same as figure 19.

Figure 22: More spikes/variance than Ridge.

Figure 23: Best MSE is neither for lowest λ or highest n.

Figure 24:

Ridge>OLS>Lasso for real data. OLS and Ridge at $n = 9$ and Lasso at $n = 8$.

we should maybe have a plot of made from the betas at the smalest MSE

Appendix

References