FYS-STK4155 Project 1

Bendik Steinsvåg Dalen & Gabriel Sigurd Cabrera

October 22, 2019

Abstract

In this report, we analyze the relative effectiveness of three regression methods: Ordinary Least-Squares, Ridge Regression, and LASSO Regression. Using data generated using the *Franke function*, as well as satellite-generated terrain altitude data from Møsvatn Austfjell, we implement each of these methods by creating a `python` program suited to this purpose; in addition, we also utilize validation and *k-fold* cross-validation techniques to quantify the accuracy of each model.

Overall, our results have mostly matched our expectations – the stability of k-fold cross-validation superseded that of simple train-test validation, our MSE values tended to decline as a function of complexity as our R^2-scores increased, and the validation scores were significantly better for the Franke function data ($MSE_{\text{min}} \approx 0.0003$) than for our real data ($MSE_{\text{min}} \approx 0.231$); note that each of these scores were found using Ridge regression for $\lambda = 10^{-16}$. Finally, we come to the conclusion that Ridge regression is optimal for *both* datasets, as long as $\lambda = 10^{-16}$ for a 9^{th}-degree polynomial regression. OLS also works well, and is absolutely competitive when compared to Ridge. On the other hand, we found that LASSO scores[1] badly compared to Ridge and OLS in all our test cases, with MSE values of 0.005 and 0.350 for the Franke function and our real data, respectively.

Introduction

The purpose of this report is to analyze the performance of several regression methods – Ordinary Least Squares (OLS), Ridge Regression, and LASSO[2] 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*.

Our `python` code and dataset is accessible online for the purpose of reproducibility:

https://github.com/GabrielSCabrera/MachineLearning/tree/master/FYS-STK4155/Project_1.

Running the script `Project_1.py` with `python 3.6` or greater will generate all the data used in this report.

[1] With a maximum of 10,000 iterations.

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

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.

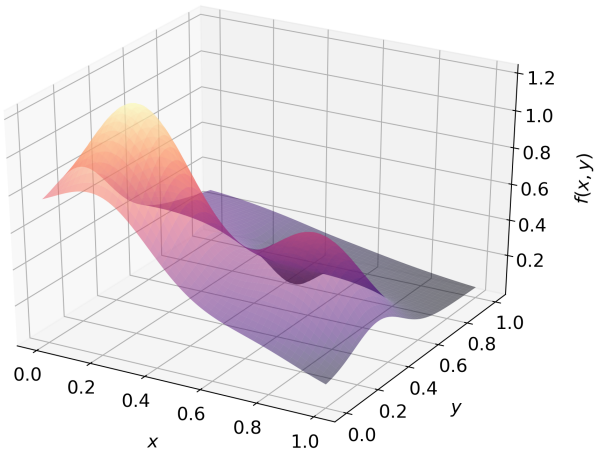

Figure 1: The *Franke function* for x and y values ranging from zero to one.

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.

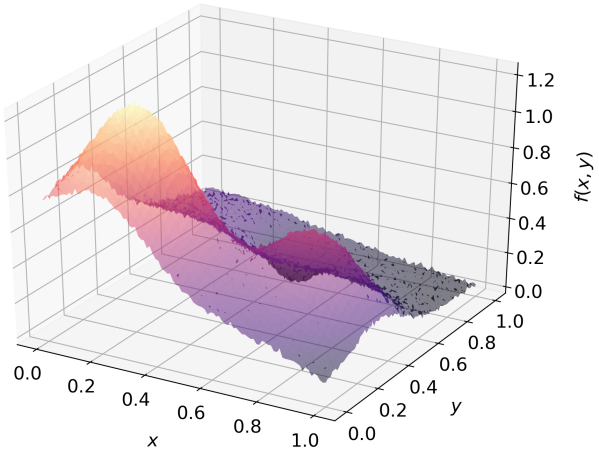

Figure 2: The *Franke function* for x and y values ranging from zero to one, with a Gaussian noise $N(0, 0.01)$

Møsvatn Austfjell

For our second dataset, we will be using real data taken from the *U.S. Geological Survey* [1] 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.

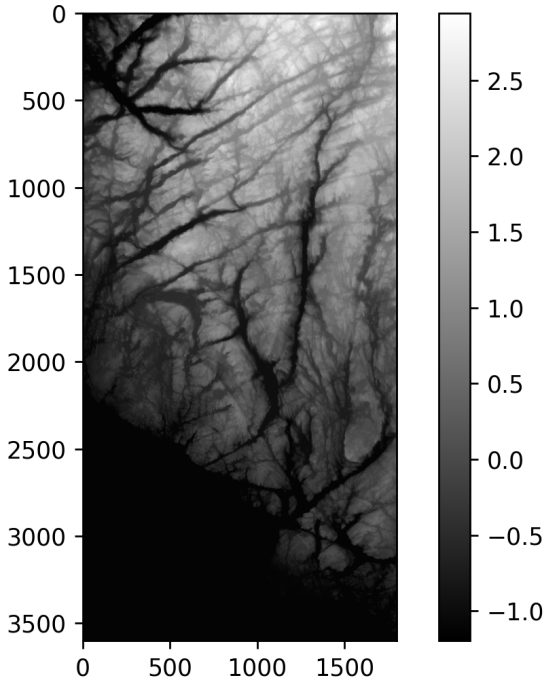

Figure 3: Altitude data from Møsvatn Austfjell, from the USGS website [1].

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 [2] 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[3] 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* [3], 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:

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

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

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

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

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[4], 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):

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

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:

$$RSS(\beta) = \sum_{i=1}^{N}(y_i - \mathbf{A}_i^{\mathrm{T}}\beta)^2 + \lambda\sum_{i=1}^{N}|\beta| \tag{4}$$

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 (4).

It is important to note that, although the above algorithm functions correctly, we will be using `sklearn` to implement LASSO, due to its vastly superior performance.

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 $\hat{y}_i$:

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

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.

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:

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

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:

$$\mathbf{y}(\mathbf{X}) = f(\mathbf{X}) + N(0, \sigma) \tag{5}$$

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 and mean 0.

As a regression model increases in complexity[5], it so happens that the *variance* of a prediction increases. Variance is defined as follows:

$$\mathrm{Var}(\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbb{E}[\mathbf{y}])^2 \tag{6}$$

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

$$\mathrm{Bias}(\mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\mathbf{y}]) \tag{7}$$

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 (6) and the square of (7) as well as σ^2 will yield the *mean squared error*. We will show this to be the case in the following subsection.

Derivation

We wish to show that:

$$\mathbb{E}\left[(\mathbf{y} - \hat{\mathbf{y}})^2\right] = \frac{1}{n}\sum_{i=1}^{n}(f_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - \mathbb{E}[\hat{\mathbf{y}}])^2 + \sigma^2 \tag{8}$$

We begin by rewriting the MSE into summation notation, decomposing the terms as defined in (5), 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:

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

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

Finally, we see that our original assumption given by (8) 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[6] 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[7]. 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

Using our `class Regression`, we implemented each aforementioned regression methods; using cross-validation, we tested for a variety of conditions so as to optimize our potential models.

The Franke Function

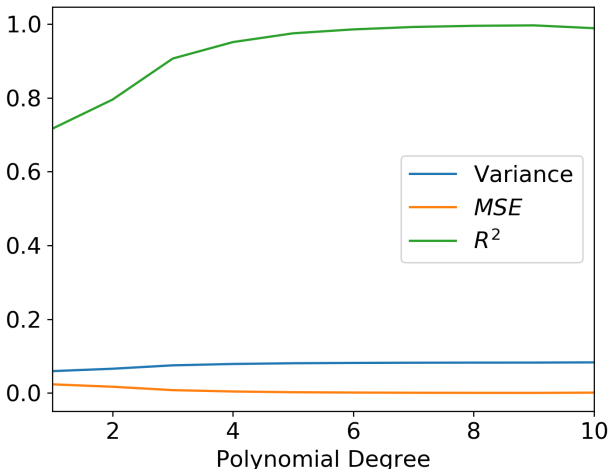

Figure 4: The MSE, the R^2-score, and the variance σ in the vector of coefficients β, as functions of the polynomial degree after performing OLS on the Franke function

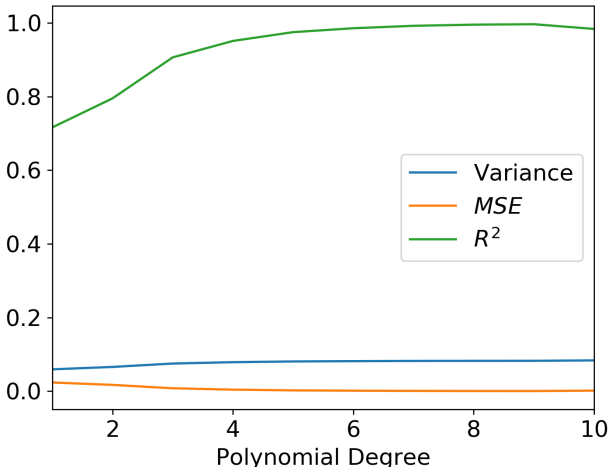

Figure 5: The MSE, the R^2-score, and the variance σ in the vector of coefficients β as functions of the polynomial degree after performing *OLS* on the Franke function. Using 12-fold cross validation.

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

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

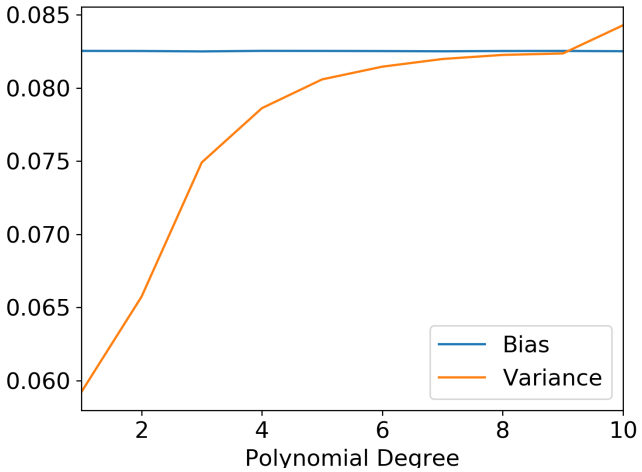

Figure 6: The *bias* and *variance* as functions of the polynomial degree after performing *OLS* on the Franke function

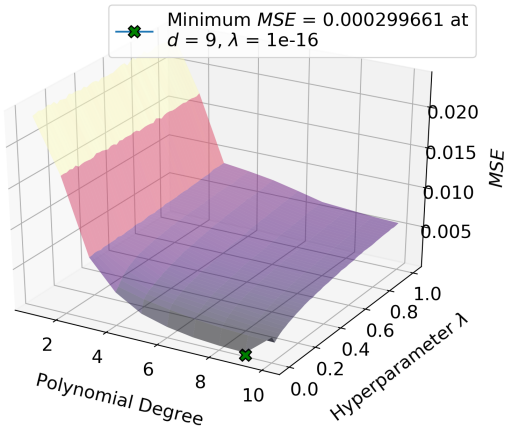

Figure 9: The MSE as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

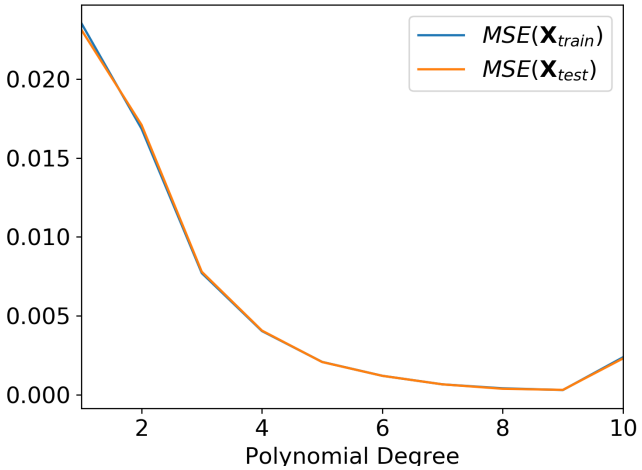

Figure 7: The MSE for the *training data* and the *testing data*, as a function of the polynomial degree after performing *OLS* on the Franke function

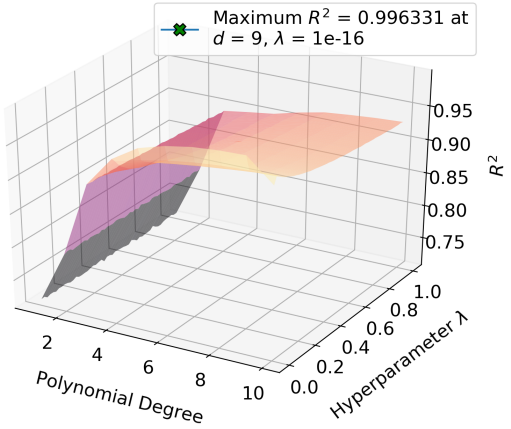

Figure 10: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

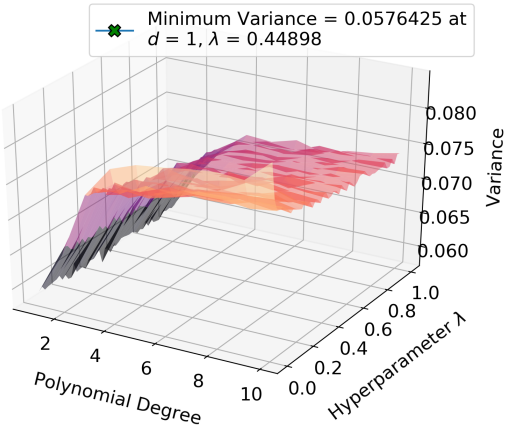

Figure 8: The *variance* as a function of polynomial degree and hyperparameter λ, after performing *Ridge regression* on the Franke function

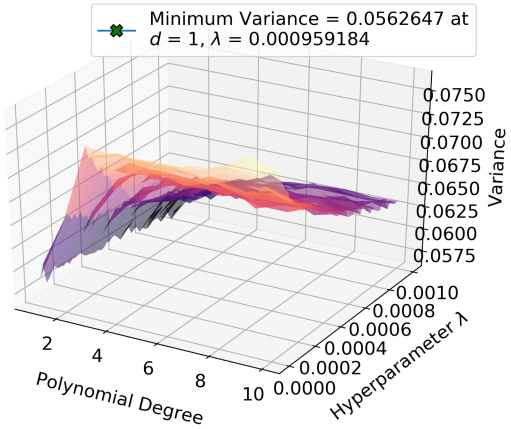

Figure 11: The *variance* as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on the Franke function

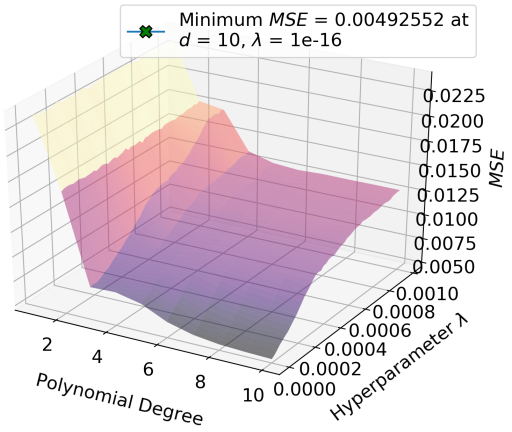

Figure 12: The MSE as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on the Franke function

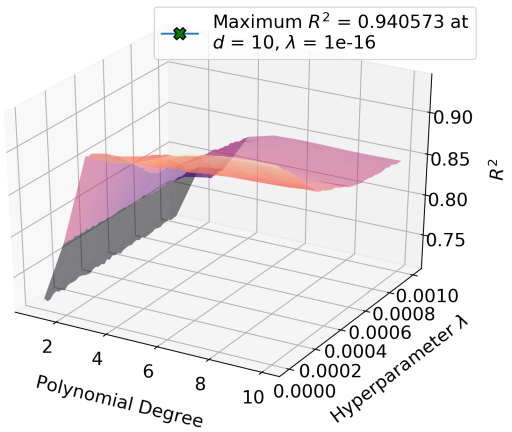

Figure 13: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on the Franke function

Møsvatn Austfjell

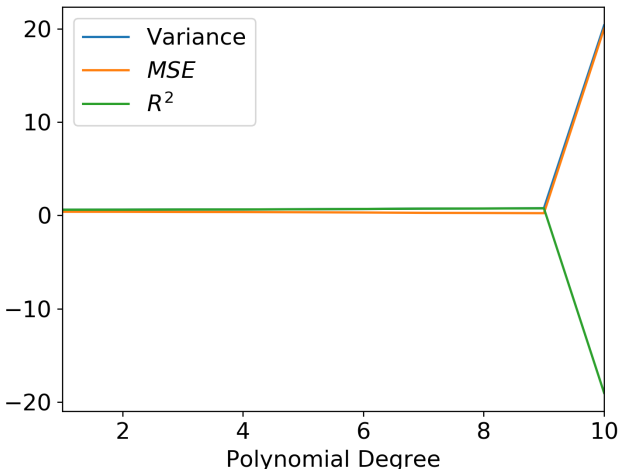

Figure 14: The MSE, R^2-score and variance σ of the vector of coefficients β as a function of the polynomial degree after performing OLS on real terrain data from *Møsvatn Austfjell*

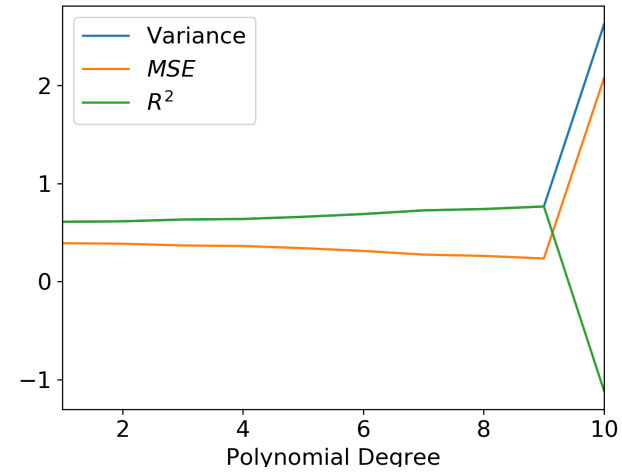

Figure 15: The MSE, R^2-score and variance σ of the vector of coefficients β as a function of the polynomial degree after performing OLS on real terrain data from *Møsvatn Austfjell*. Using 12-fold cross validation.

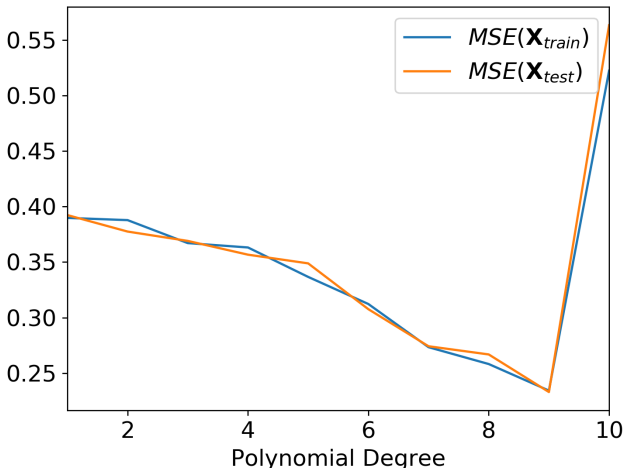

Figure 16: The MSE for the *training data* and the *testing data*, as a function of the polynomial degree after performing OLS on real terrain data from *Møsvatn Austfjell*

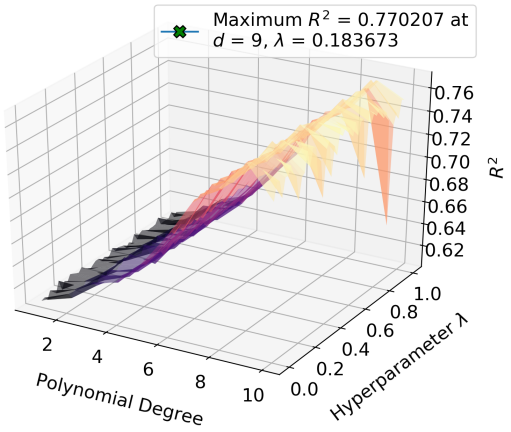

Figure 19: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

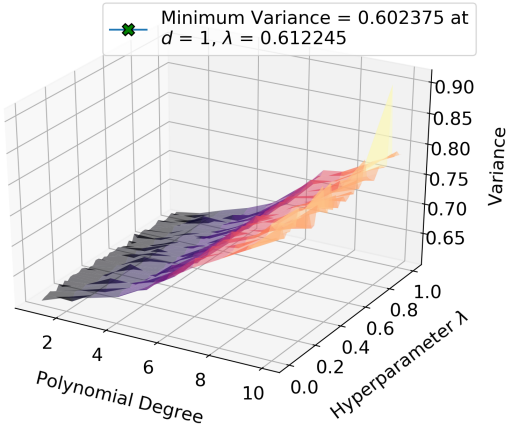

Figure 17: The *variance* as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

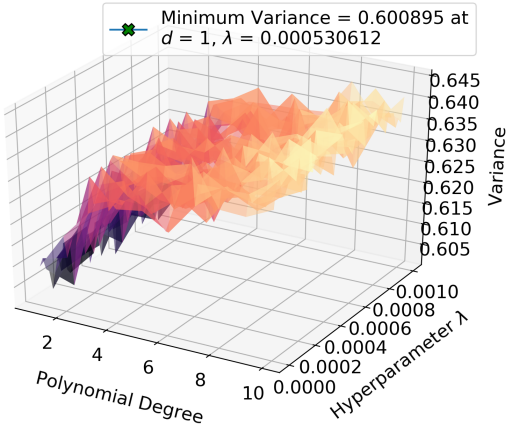

Figure 20: The *variance* as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on real terrain data from *Møsvatn Austfjell*

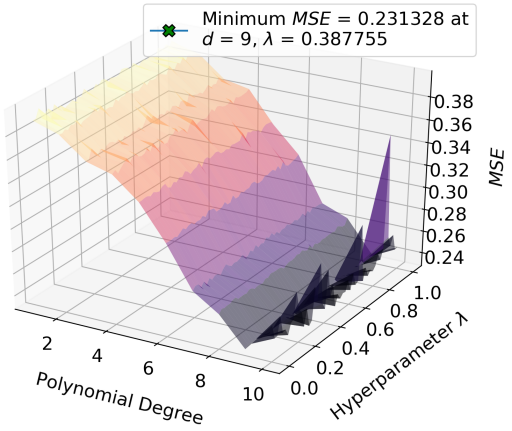

Figure 18: The MSE as a function of the polynomial degree and the hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

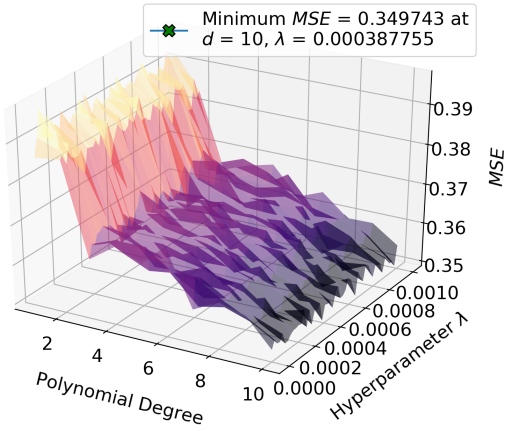

Figure 21: Plots of the MSE as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on real terrain data from *Møsvatn Austfjell*

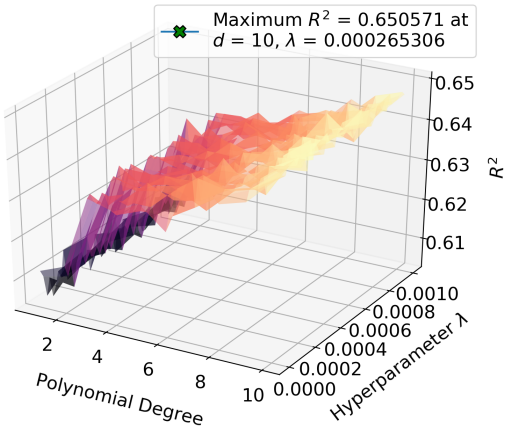

Figure 22: The R^2-score as a function of the polynomial degre and hyperparameter λ after performing LASSO *regression* on real terrain data from *Møsvatn Austfjell*

Discussion

The Franke Function

In Figure 4, we see that the *variance* increases as a function of complexity (as expected). It seems that the R^2-score also increases, specifically until the polynomial degree[8] reaches 9. In Figure 5, it appears that implementing cross-validation leaves our results mostly unchanged from those in Figure 4. Given that we know the *Franke function*, we are also able to calculate the *squared bias* (with 12-fold cross-validation) in our model. In Figure 6 we can see the *bias-variance* tradeoff in action, though the squared bias does not seems to vary much at all relative to the variance.

In Figure 7, we see that the MSE, on average, decreases up until we perform a 9^{th}-degree polynomial regression with 12-fold cross-validation[9], and that this holds for both the *training set* and the *testing set*. Once we begin to implement *Ridge regression* for hyperparameters $\lambda \in [10^{-16}, 1]$, we see that the variance in Figure 8 tends to increase at the same rate for most λ, with divergence occuring at low λ. In Figure 9, we see that the MSE decreases as a function of the polynomial degree up until it reaches 9, after which it begins to increase. The maximum R^2-score of 0.996 shown in Figure 10 *also* occurs at $d = 9$ with $\lambda = 10^{-16}$. Overall, this implies that the best model for our system is likely a 9^{th}-degree polynomial Ridge regression with $\lambda = 10^{-16}$; the MSE is minimized at that point with a value of 2×10^{-4}.

Our LASSO results are somewhat different from our OLS/Ridge results; here, we see that the variance in Figure 11 depends much more on λ than it does in Figure 8. The MSE for LASSO also differs from that of Ridge regression – in Figure 12, we see that the optimal polynomial degree[10] is 10. This time, however, the MSE is much larger, clocking it at 0.005, nearly 20 times greater than the minimum for Ridge regression shown in Figure 9. Additionally, we see that the R^2-score in Figure 13 *also* performs worse than that in Figure 10, with an optimal value of 0.94, compared to the optimal value of 0.99 in Figure 10. The ideal polynomial degree per R^2 remains unchanged, with 10 once again being our optimal value.

Møsvatn Austfjell

In Figure 14, we see that the MSE, R^2-score, and variance each diverge when evaluated for a 10^{th}-degree polynomial; fortunately, 12-fold cross-validation gives us a *slightly* clearer picture of what is happening, as seen in Figure 15, where our variance, MSE, and R^2-score are much more modest than those we saw for the Franke function. A closer look at the MSE (for both the *training set* and the *testing set*) in Figure 16 shows that a polynomial of degree 9 is optimal when performing OLS.

Figure 17, which shows the variance as a function of polynomial degree and hyperparameter λ, behaves notably different from its Franke function counterpart[11]; instead of flattening out, the variance consistently grows as the polynomial degree is increased[12] Interestingly enough, we see that the optimal polynomial degree in Figure 18 is 9, just as that of the Franke function[13]. However, the minimum MSE is much larger than that of the Franke function, at 0.231, and the optimal hyperparameter is now $\lambda = 0.388$. It is also worth noting that Figure 19 suggests a different hyperparameter of $\lambda = 0.184$ based on its R^2-score, but continues to select 9 as the optimal polynomial degree.

Our implementation of LASSO yields very different results now that we are dealing with real life data. We see in Figure 20 that there is much more instability in the variance, than that of the corresponding Figure 17; one thing to note is that the variance appears to reach a near-planar trend for polynomial degrees of four and greater. As for the optimal MSE of our LASSO regression, Figure 21 suggests that a polynomial of 10^{th} degree is optimal, with a minimum MSE of 0.350. Finally, we see that our R^2-score in Figure 22 suggests a different optimal λ of 0.651, while still implying that a polynomial of degree 10 would give us the best LASSO results.

Conclusion

When dealing with statistical models, it is important to understand how different models and validation methods perform relative to each other, for varying parameters and initial conditions – we chose to evaluate these methods using two datasets: a mathematical function with added Gaussian noise, and real-life altitude data from Møsvatn Austfjell.

Initially, we had certain expectations as to how our implementation might turn out – we assumed that implementing k-fold cross-validation would stabilize our results, and this turned out to be absolutely true (especially for our real-life data!) On the other hand, our bias-variance decomposition did not yield particularly meaningful results, as we were un-

[8] The score decreases for polynomials of degree ≥ 10.
[9] It can be assumed henceforth that all our data is obtained via 12-fold cross-validation, unless stated otherwise.
[10] The ideal λ is still 10^{-16}.
[11] Seen in Figure 8
[12] Though it is mostly constant with respect to λ.
[13] Seen in Figure 9.

able to successfully plot a set of curves demonstrating the decomposition of the MSE. We also expected the real-life data to perform worse than the Franke function for all forms of validation, and this was shown to be demonstrably true. It is worth noting that we can be quite confident that our models perform reliably, as the MSE curves for the *training* and *testing* sets in Figure 16 differ little from each other.

We also noticed certain major differences between our datasets – firstly, since the geological features of our real dataset do not match any polynomial particularly well, increasing the polynomial degree in Ridge regression will always lead to a larger variance; this is not the case for the Franke function dataset. In addition, the fact that our two *vastly* different datasets both have optimal polynomial regression degrees of 9, is suspicious. We suspect that implementing another method, such as the *singular-value decomposition*, might lead to more stability in our model; if this is the case, perhaps a higher degree polynomial regression would be successful, and reduce the MSE further. Furthermore, we also found that the MSE, variance, and R^2 are *all* unlikely to depend on of the chosen hyperparameter λ for both LASSO and Ridge regression when it comes to real data.

Overall, it appears that our evaluations of OLS, Ridge regression, and LASSO regression have successfully given us an idea of which methods perform best, and in which contexts – it appears that Ridge regression for $\lambda = 10^{-16}$ performs best for *both* our datasets, closely followed by OLS. It is worth noting that there is nearly no difference in performance between the two, though larger datasets or higher degree polynomial regressions using more stable methods[14] such as the aforementioned SVD might cause us to revisit this conclusion. In either case, we managed to determine that the implementation of LASSO with our choice of iteration limit[15] performed far worse that Ridge/OLS by *all* metrics.

References

[1] "Earthexplorer."

[2] M. (https://math.stackexchange.com/users/58320/macavity), "What is the general form of a polynomial of degree n and with m variables?." Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2482654 (version: 2017-10-21).

[3] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2013.

[14] Using 128-bit floating-point values might also help improve our results, and possibly allow for smaller *stable* hyperparameters than 10^{-16}

[15] We set the loop to break after 10,000 iterations; a larger maximum might lead to better results.

2