FYS-STK4155 Project 1

Bendik Steinsvåg Dalen & Gabriel Sigurd Cabrera

October 7, 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.

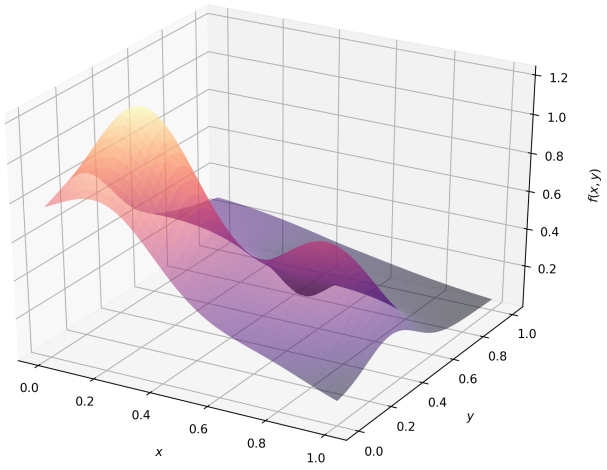

Figure 1: The *Franke function* for x and y values ranging from zero to one.

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.

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

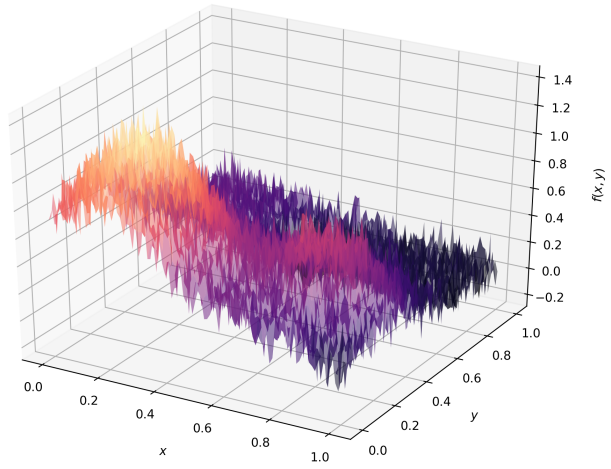

Figure 2: The *Franke function* for x and y values ranging from zero to one, with a Gaussian noise $N(0, 0.01)$

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.

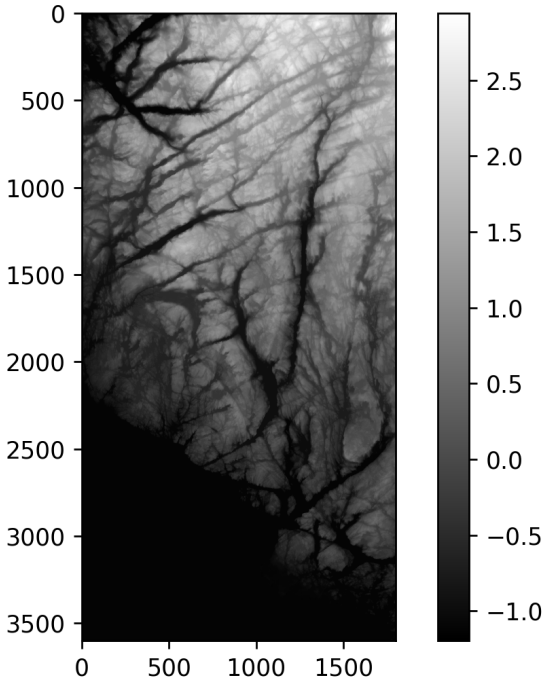

Figure 3: Altitude data from Møsvatn Austfjell, from the USGS website [1].

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[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

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

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:

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

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[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.

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

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:

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

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:

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

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

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

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.

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:

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

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

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

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.

[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.

The Franke Function

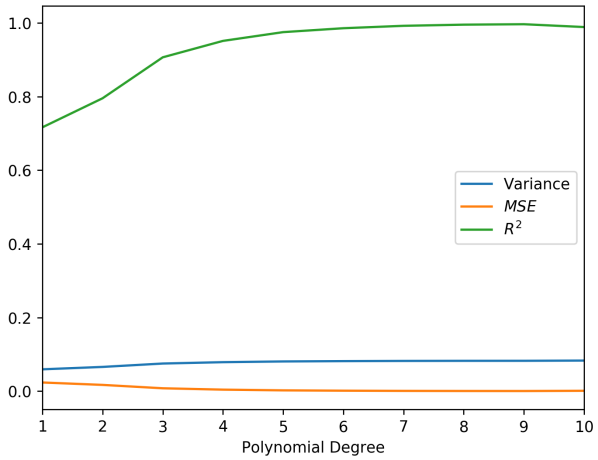

Figure 4: The MSE, the R^2-score, and the variance σ in the vector of coefficients β, as functions of the polynomial degree after performing OLS on the Franke function

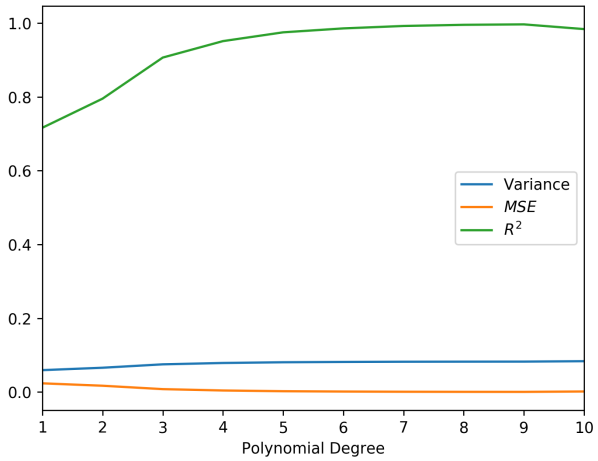

Figure 5: The MSE, the R^2-score, and the variance σ in the vector of coefficients β as functions of the polynomial degree after performing *OLS* on the Franke function. Using 12-fold cross validation.

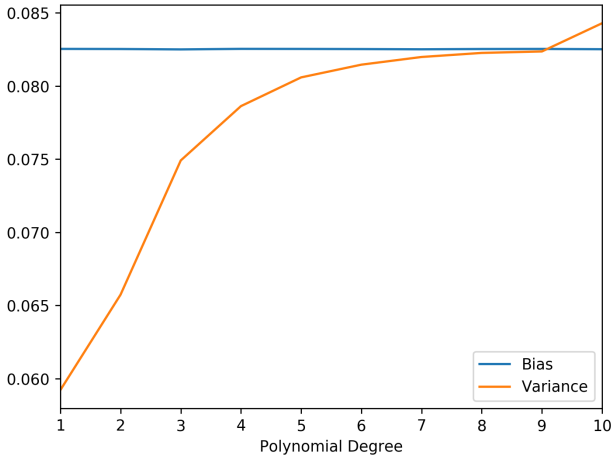

Figure 6: The *bias* and *variance* as functions of the polynomial degree after performing *OLS* on the Franke function

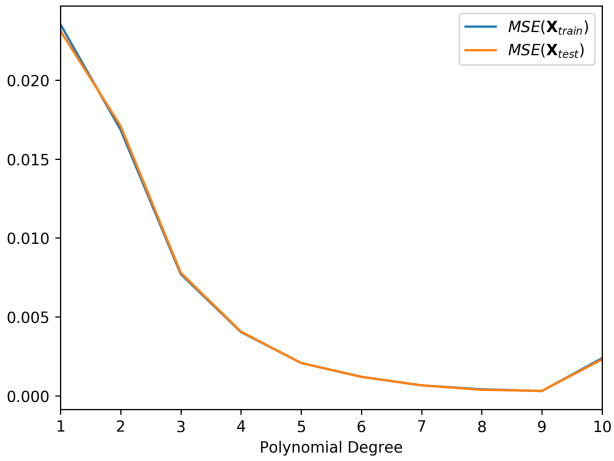

Figure 7: The MSE for the *training data* and the *testing data*, as a function of the polynomial degree after performing *OLS* on the Franke function

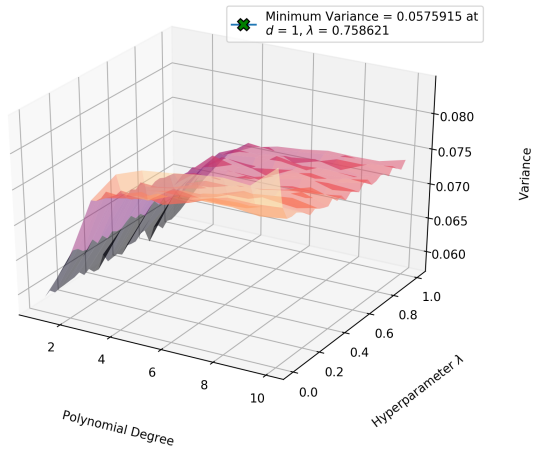

Figure 8: The *variance* as a function of polynomial degree and hyperparameter λ, after performing *Ridge regression* on the Franke function

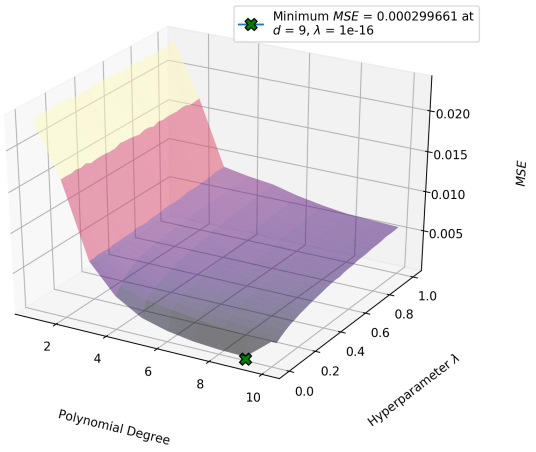

Figure 9: The MSE as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

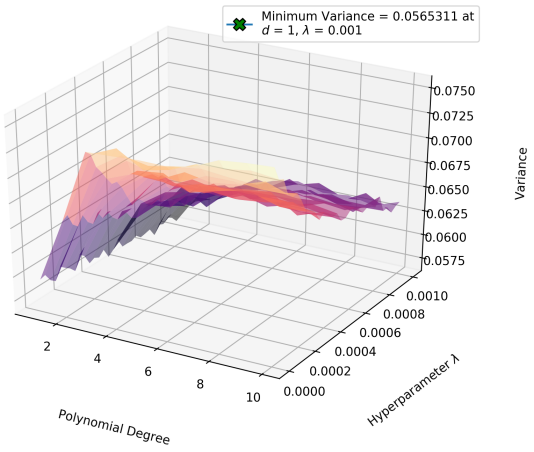

Figure 12: The *variance* as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on the Franke function

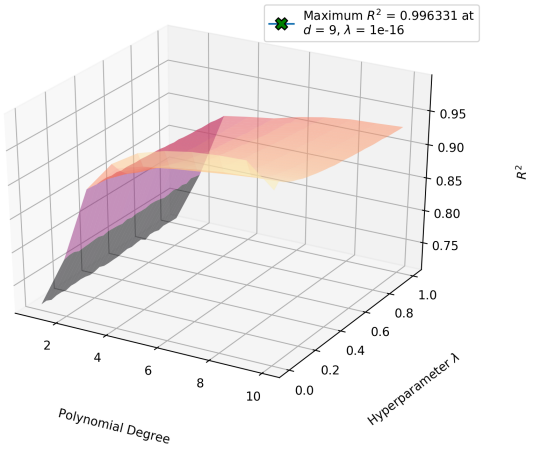

Figure 10: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

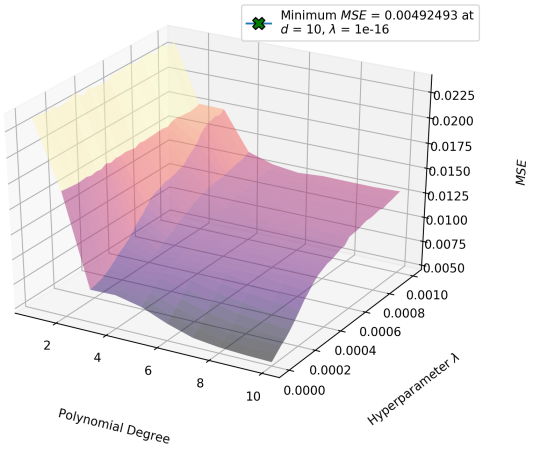

Figure 13: The MSE as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on the Franke function

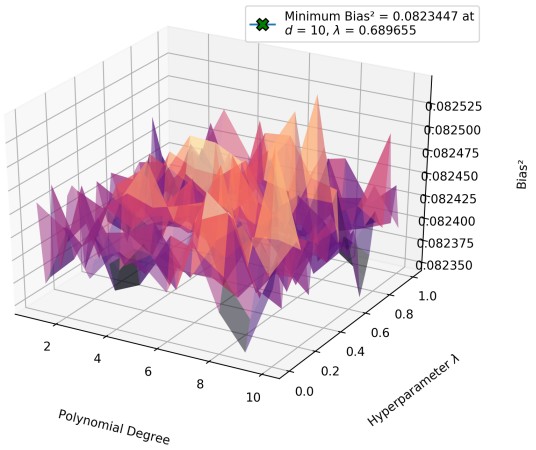

Figure 11: The *squared bias* as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on the Franke function

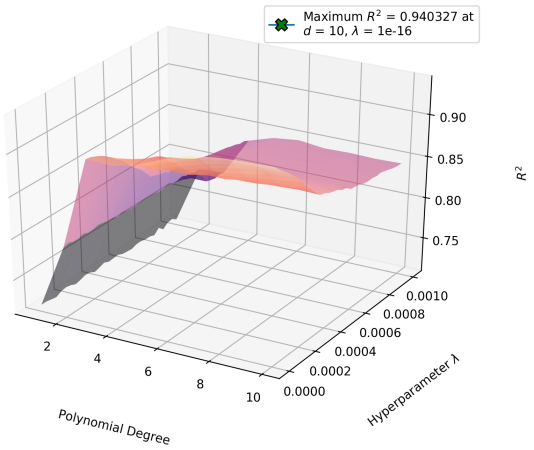

Figure 14: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on the Franke function

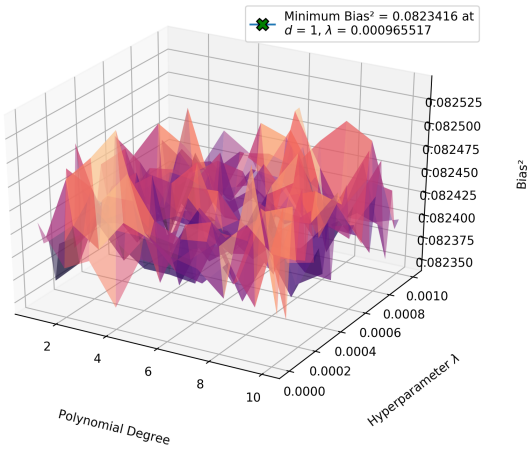

Figure 15: The *squared bias* as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on the Franke function

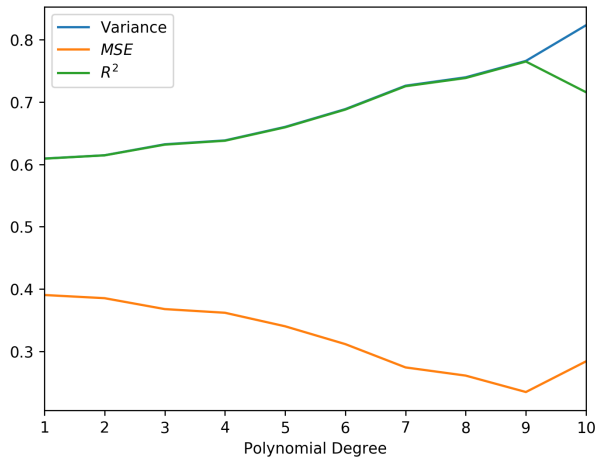

Figure 17: The MSE, R^2-score and variance σ of the vector of coefficients β as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*. Using 12-fold cross validation.

Møsvatn Austfjell

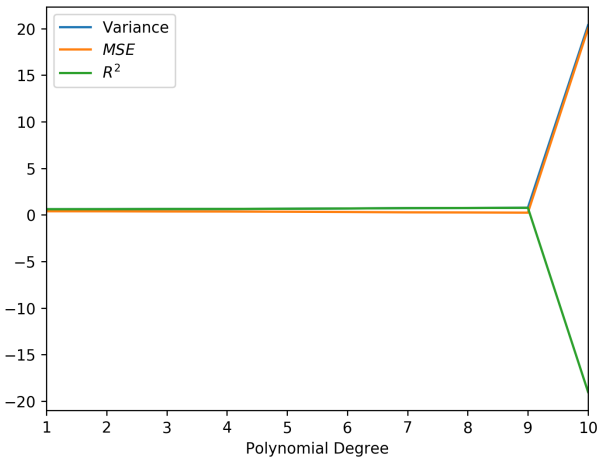

Figure 16: The MSE, R^2-score and variance σ of the vector of coefficients β as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*

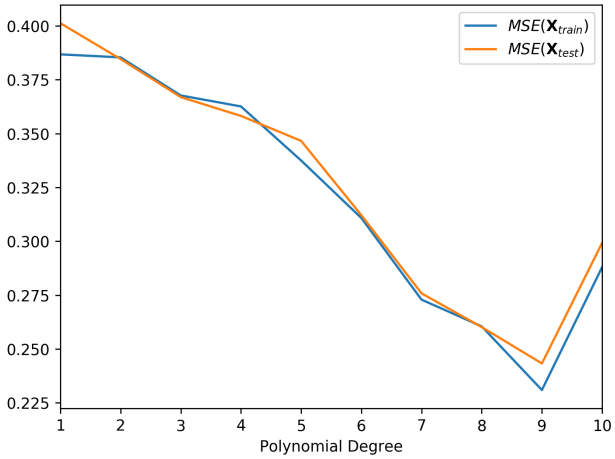

Figure 18: The MSE for the *training data* and the *testing data*, as a function of the polynomial degree after performing *OLS* on real terrain data from *Møsvatn Austfjell*

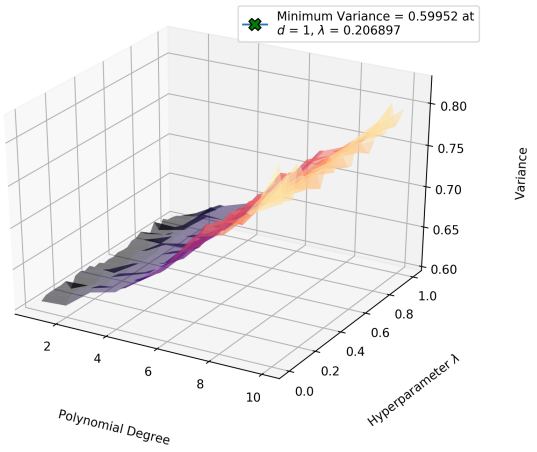

Figure 19: The *variance* as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

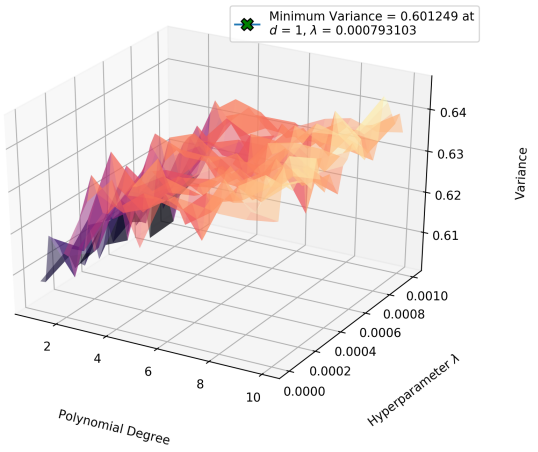

Figure 22: The *variance* as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on real terrain data from *Møsvatn Austfjell*

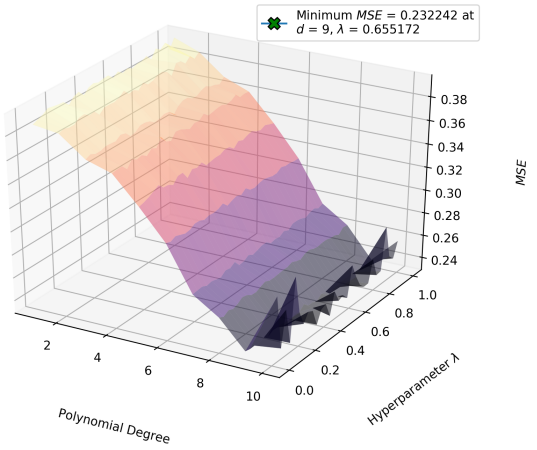

Figure 20: The MSE as a function of the polynomial degree and the hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

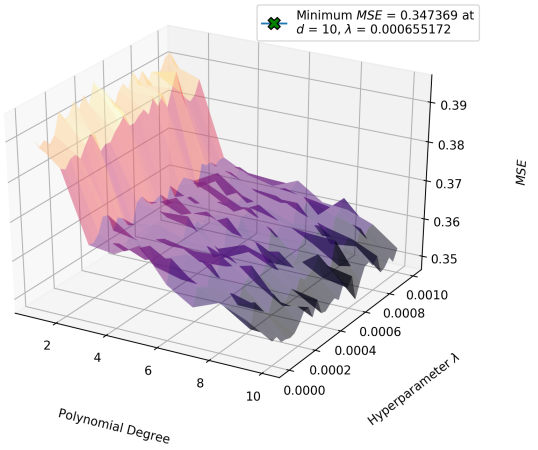

Figure 23: Plots of the MSE as a function of the polynomial degree and hyperparameter λ after performing LASSO *regression* on real terrain data from *Møsvatn Austfjell*

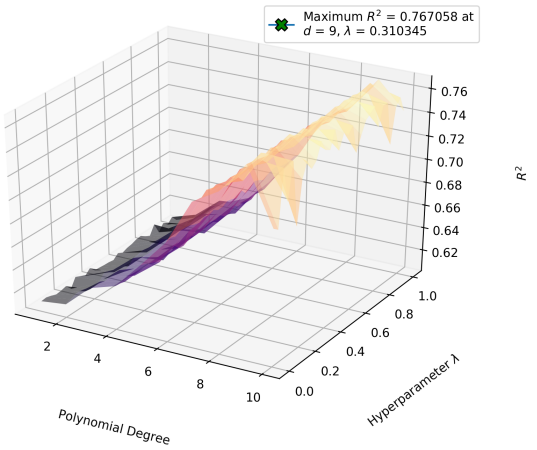

Figure 21: The R^2-score as a function of the polynomial degree and hyperparameter λ after performing *Ridge regression* on real terrain data from *Møsvatn Austfjell*

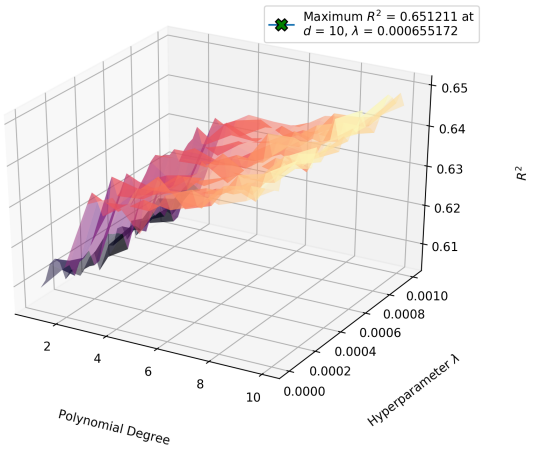

Figure 24: The R^2-score as a function of the polynomial degre and hyperparameter λ after performing LASSO *regression* on real terrain data from *Møsvatn Austfjell*

Discussion

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 up until the polynomial degree reaches 10, where it begins to decrease. Finally, we see that the MSE decreases, though we will see this more clearly in Figure 7.

In Figure 5, it appears that implementing cross-validation leaves our results mostly unchanged from those in Figure 4, though the decrease in the R^2-score is somewhat less pronounced.

Given that we know the *Franke function*, we are able to calculate the *bias* (with 12-fold cross-validation) in our model. In Figure 6 We can see the *bias-variance* tradeoff in action.

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[7], and that this holds for both the *training set* and the *testing set*.

Once we begin to implement *Ridge regression* for hyperparameters $\lambda \in [10^{-12}, 10^{-3}]$, we see that the variance in Figure 8 tends to increase at the same rate for most λ, with all diverging at degree 10.

In Figure 9, we see that the MSE decreases as a function of the polynomial degree up until it reaches 10, where it diverges at varying rates over λ. In addition to this, we see that the best model for our system may be a 9^{th}-degree polynomial Ridge regression with $\lambda = 10^{-16}$; our MSE is minimized at that point with a value of 2×10^{-4}.

The maximum R^2-score shown in Figure 10 also occurs at $d = 9$ with $\lambda = 10^{-16}$, which further confirm our previous assertions.

The plot of the squared bias in Figure 11, is essentially a flat plane with a value close to 0.0824.

Our LASSO results are somewhat different. Here, we see that the variance in Figure 12 depends much more on λ than it does in Figure 8.

The MSE for LASSO also differs from that of Ridge regression – in Figure 13, we see that the optimal polynomial degree is 10, while the ideal λ is still 10^{-16}. However, this time the MSE is much larger: 0.005, nearly 20 times that found in Figure 9.

We see that the R^2-score in Figure 14 also performs worse than that in Figure 10, with an optimal value of 0.94, whereas we have an optimal value of 0.99 in Figure 10. However, the ideal polynomial degree per R^2 is 9, rather than 10.

The bias for LASSO in Figure 15 behaves similarly to that of Ridge regression; as such it is not very important in our evaluation of methods.

Møsvatn Austfjell

In Figure 16, we see that the MSE, R^2-score, and variance each diverge when evaluated for a 10^{th}-degree polynomial regression.

12-fold cross-validation gives us a clearer picture of what is happening, as seen in Figure 17, 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 (of both the *training set* and the *testing set*) in Figure 18 show that a polynomial of degree 9 is optimal when performing OLS.

Figure 19, which shows the variance as a function of polynomial degree and hyperparameter λ, behaves differently than it's Franke function counterpart[8]; instead of flattening out, the variance consistently grows as the polynomial degree is increased[9]

Interestingly enough, we see that the optimal polynomial degree in Figure 20 is 9, just as for the Franke function [10]. However, the minimum MSE is much larger than that of the Franke function, at 0.23, and the optimal hyperparameter is now $\lambda = 0.655$.

Figure 21 suggests a different hyperparameter of $\lambda = 0.31$, but continues to select 9 as the optimal polynomial degree.

Our implementation of LASSO yields very different results now when dealing with this real life data. We see in Figure 22 that there is much less stability in this variance, than that of the corresponding Figure 19.

As for the optimal MSE of our LASSO regression, we see in Figure 23 that a polynomial of 10^{th} degree is optimal, with a minimum MSE of 0.348.

Finally, we see that our R^2-score in Figure 24 has an optimal value of 0.651, while still suggesting that a polynomial of degree 10 would give us the best results.

Conclusion

This is likely due to the fact that we implemented matrix inversion without implementing the SVD. Perhaps a more robust algorithm would lead to higher degree polynomials giving us better values in the MSE, but the variance would certainly continue to increase.

K-fold cross-validation is known to produce more stable results, as we discussed in our method section.

Interestingly enough, it seems as though our bias does not decrease as much as expected.

We assume the reason that the MSE begins to diverge once we perform a 10^{th}-degree polynomial regression is due to floating-point errors.

It is likely numerical instability that leads to diverging at varying rates, and this can likely not be avoided, though implementing the SVD might help.

It also seems that the our MSE increases as a function of the hyperparameter λ, and that 10^{-16} is our best choice since smaller values will lead to floating point errors.

Given that the R^2-score in Figure 10 agrees as well, this could very well be the case.

We can assume that the variation in Figure 11 is simply a product of random number generation.

Perhaps if we had greater numerical precision, such as 128-bit precision, then we might be able to further decrease λ

[7] It can be assumed henceforth that all our data is obtained via 12-fold cross-validation, unless stated otherwise.

[8] Seen in Figure 8

[9] Though it is mostly constant with respect to λ.

[10] Seen in Figure 9.

and get even better results.

It is very likely that allowing the LASSO function to converge at a larger number of iterations would improve the MSE.

Given the inconsistencies in LASSO, as well as the fact that it performs worse on both the MSE and R^2, it seems apparent that Ridge regression with an optimized λ may be our ideal model.

It appears that cross-validation is necessary when dealing with a real life dataset, especially if given a dataset that is particularly sparse.

Once again, the fact that we are dealing with real life data leads to worse performance with regards to the MSE and other validation metrics.

With a minimum MSE of 0.25, we have to accept a good deal of imprecision in our regression, but that is simply a consequence of dealing with real data. However, we can be confident that our model performs reliably, as the MSE curves for the training and testing sets in Figure 18 differ little.

Since the geological features of our dataset do not match any polynomial particularly well, increasing the polynomial degree in Ridge regression will always lead to a larger variance.

The fact that two vastly different datasets [11] both have optimal polynomial regression degrees of 9, is suspicious. We suspect that implementing another method, such as the SVD, might lead to more stability in our model; if this is to be the case, perhaps a higher degree polynomial regression would be successful, and reduce the MSE further.

Perhaps the most important conclusion we can draw from our experience here is that when dealing with real life data, implementing Ridge regression over OLS is likely unnecessary, as we see that our results are all mostly independent of the hyperparameter λ.

It is also worth mentioning that LASSO performs with far less stability than OLS or Ridge regression, and with a noticeably larger MSE and variance, and smaller R^2-score. It is likely that increasing the iteration limit for LASSO would help improve this, but at an extreme performance penalty.

We should also note that the MSE, variance, and R^2 are all likely independent of the chosen hyperparameter λ for LASSO, just as they were for Ridge regression.

If we can draw any conclusion, it is that OLS and Ridge regression are equals when it comes to their performance, and that LASSO gives poorer results. This may not be the case if we allow for the LASSO algorithm to iterate a larger number of times, but this is simply not possible given the constraints of our equipment.

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.

[11] One a mathematical function, the other a scan of a landscape.