FYS-STK4155 Project 1
Regression Analysis and Resampling Methods

Teah Kaasa McLean

October 7, 2019

Abstract

The project deals with predictions of two types of data using regression methods and resampling techniques. We look at a simple two-dimensional function, the Franke function, and real terrain data. The predictions are performed with Ordinary Least Squares, Ridge regression and Lasso regression. The Ordinary Least Squares method fitting a fifth order polynomial proved most satisfactory for both data sets.

Contents

1 Introduction

Machine learning is now a very attractive and fashionable field. It combines ideas from neuroscience, biology, statistics, mathematics and physics to make computers learn. A subset of this field is that of linear regression, a well-established tool for prediction. Linear regression makes it easier to understand other methods like Neural Networks, and concepts such as the bias-variance tradeoff. In this report we shall dive into the world of linear regression and resampling.

It is common to test methods and models on relatively simple functions or data first, before applying the methods to more interesting data. We shall develop methods for Ordinary Least Squares (OLS), Ridge regression and Lasso regression, and use the Franke function (defined in appendix .1) for testing. In order to reduce the likelihood of overfitting, we will implement two resampling techniques: k-fold Cross Validation (CV) and the Boostrap method.

With a well-functioning set of algorithms, we can now study some data from the real world. In particular, terrain data from an area close to Stavanger, Norway. We shall compare the different regression methods using the *Mean Squared Error* (MSE) to determine which method best fits the data.

We will start by introducing the various regression and resampling methods and giving a brief description of the implementation. Further, we will examine the MSEs and R^2 scores for all the methods. Finally, we will determine which method best reproduces the data.

2 Methods and theory

In this report, data will be denoted as (x_i, y_i) where x_i is the *independent* variable and $y_i = f(x_i)$ is the *response* variable or *outcome*. If we have m cases, then the index $i = 1, \ldots, m$. Estimates will be denoted as $\tilde{y}_i$ or $\tilde{f}$. We let $\mathbf{y}$ denote the vector of true values y_i, and $\tilde{\mathbf{y}}$ denote the vector of estimated values $\tilde{y}_i$. We will use n to describe the complexity of the model, i.e. the polynomial degree.

2.1 Linear regression

Linear regression is a machine learning method based on supervised learning. It is a method for modeling the relationship between two variables. A linear function is fitted to some observed data, and the performance of the fit is determined by evaluating the so-called *cost-function* $C(\beta)$. The cost-function can tell us how far off the predicted values the model gives are from the true values.

The linear regression model has the form

$$f(\mathbf{x}) = \mathbf{y} = \tilde{\mathbf{y}} + \boldsymbol{\epsilon} = \beta_0 + \sum_{j=1}^{n} x_j \beta_j + \boldsymbol{\epsilon}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_m)$ is the input values and $\boldsymbol{\epsilon}$ is the error in the approximation. This error should not be confused with any noise or error in the data.

We can rewrite this into a linear algebra problem:

$$\mathbf{y} = f(\mathbf{x}) = X\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where X is the *design matrix* and $\boldsymbol{\beta}$ is a vector consisting of the coefficients β_i. We want to find a $\tilde{\boldsymbol{\beta}}$ which minimizes the cost-function.

2.1.1 Ordinary Least Squares

The most popular method of linear regression is the *Ordinary Least Squares* (OLS) method. It models the relationship of the variables by minimizing the sum of the squares in the difference between the observed y_i values and predicted $\tilde{y}_i$ values. That is, the cost function is the MSE (described in more detail in section 2.2).

As mentioned, we want to find the $\boldsymbol{\beta}$ that minimizes the cost-function. We will find the derivative of the cost-function, set it equal to zero and solve for $\boldsymbol{\beta}$ to find an analytical expression.

We leave out the fraction $\frac{1}{m}$ in our calculations. Then we have that

$$C(\boldsymbol{\beta}) = \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = (\mathbf{y} - \tilde{\mathbf{y}})^T (\mathbf{y} - \tilde{\mathbf{y}})$$

Inserting $\tilde{\mathbf{y}} = X\boldsymbol{\beta}$, then the expression can be written as

$$C(\boldsymbol{\beta}) = (\mathbf{y} - X\boldsymbol{\beta})^T (\mathbf{y} - X\boldsymbol{\beta}).$$

The derivative is given by

$$\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = X^T (\mathbf{y} - X\boldsymbol{\beta}).$$

We set this equal to 0, and solve for $\boldsymbol{\beta}$ by multiplying from the left by $(X^T X)^{-1}$. Then we get that

$$\boldsymbol{\beta} = (X^T X)^{-1} (X^T) \mathbf{y}.$$

2.1.2 Ridge Regression

For matrices with high dimensionality, the likelihood of linearly dependent columns increases. If our design matrix has linearly dependent columns, then we will not be able to invert $X^T X$. An *ad-hoc* way of solving this issue is to add a small diagonal component λ, usually called a *hyperparameter*. This is what we do in Ridge regression.

We define a new cost function

$$C(\boldsymbol{\beta}) = \frac{1}{m}||\mathbf{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_2^2,$$

where the 2-norm is defined as

$$||\mathbf{x}||_2 = \sqrt{\sum_i x_i^2}.$$

Taking the derivatives of $C(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$, we get that

$$\boldsymbol{\beta}^{Ridge} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}.$$

We note that $\boldsymbol{\beta}^{Ridge} = (1+\lambda)^{-1}\boldsymbol{\beta}^{OLS}$, that is, the Ridge estimator scales the OLS estimator by a factor $(1+\lambda)$. If $\lambda = 0$, the Ridge and OLS estimators are equal.

As $\lambda \to \infty$, the $\boldsymbol{\beta}$ will tend towards 0 (Hjorth-Jensen 2019).

2.1.3 Lasso Regression

If we replace the term $||\boldsymbol{\beta}||_2^2$ in the cost-function for Ridge regression with $||\boldsymbol{\beta}||_1$, we get the cost-function for Lasso regression,

$$C(\boldsymbol{\beta}) = \frac{1}{m}||\mathbf{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_1.$$

The 1-norm is defined here as

$$||\mathbf{x}||_1 = \sum_i |x_i|.$$

Lasso reaches good solutions for low λ-values, but will decline when λ becomes too large. Ridge regression is more stable for a larger range of λ-values, but will as mentioned, also eventually tend toward 0 (Hjorth-Jensen 2019).

For Lasso regression we have no analytical expression for $\boldsymbol{\beta}$, and must use a Gradient Descent method to find the βs. A lot of factors come into play in gradient descent methods, such as the so-called *learning rate*, the number of iterations and the value of λ. These factors must be tuned to ensure that the gradient descent method converges and does not "get stuck" in a local minimum.

2.2 Mean Squared Error

The *Mean Squared Error* (MSE) is defined as

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

where $\mathbf{y}$ contains the true values and $\tilde{\mathbf{y}}$ contains the predicted target values.

The MSE measures the average of the squares of the errors. That is, it finds the distance between the true values and the estimated values, squares these distances, and then takes the average of all the squares. The MSE is non-negative, and the ideal value is 0.

In this report, the MSE will be used as the scorer for the model quality. We want to find the model that minimizes this scorer, i.e. the model which gives the smallest MSE value.

2.3 R^2 Score

Another measure of the quality of a model is the R^2 score, which is defined as

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

where we have defined the mean value of $\mathbf{y}$ as

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

The R^2 score measures how close the data are to the fitted regression line. It finds the ratio between the explained variation and the total variation, i.e. how well our model explains the variation of the target variable. The ideal R^2 score value is 1. This would mean our model explains 100% of the variation.

2.4 The Bias-Variance Decomposition

It can be shown that our chosen cost function, the MSE,

$$C(\mathbf{X}, \boldsymbol{\beta}) = \frac{1}{m} \sum_{i=0}^{m-1} (y_i - \tilde{y}_i)^2 = \mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2]$$

can be written as

$$\mathbb{E}[(\mathbf{y} - \tilde{\mathbf{y}})^2] = \sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2$$

where the true data is generated from a noisy model $\mathbf{y} = f(\mathbf{x}) + \boldsymbol{\epsilon}$ and $\boldsymbol{\epsilon} \sim N(0, \sigma^2)$.

Proof.

$$\begin{aligned}
\mathbb{E}[(\mathbf{y}-\tilde{\mathbf{y}})^2] &= \mathbb{E}[(f+\boldsymbol{\epsilon}-\tilde{\mathbf{y}})^2] \\
&= \mathbb{E}[(f-\tilde{\mathbf{y}})^2+\boldsymbol{\epsilon}^2+2(\boldsymbol{\epsilon}(f-\tilde{\mathbf{y}}))] && \text{(multiplying out expression)} \\
&= \mathbb{E}[\boldsymbol{\epsilon}^2]+\mathbb{E}[(f-\tilde{\mathbf{y}})^2]+2\mathbb{E}[f\boldsymbol{\epsilon}]-2\mathbb{E}[\boldsymbol{\epsilon}\tilde{\mathbf{y}}] \\
& && \text{(linearity of the expectation)} \\
&= \mathbb{E}[(\boldsymbol{\epsilon}-0)^2]+\mathbb{E}[(f-\mathbb{E}[\tilde{\mathbf{y}}]+\mathbb{E}[\tilde{\mathbf{y}}]-\tilde{\mathbf{y}})^2]+0-0 \\
& && \text{(adding 0 and } \mathbb{E}[\boldsymbol{\epsilon}]=0) \\
&= \mathbb{E}[(\boldsymbol{\epsilon}-\mathbb{E}[\boldsymbol{\epsilon}])^2]+\mathbb{E}[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])^2]+\mathbb{E}[(\mathbb{E}[\tilde{\mathbf{y}}]-f)^2]+2\mathbb{E}[(\tilde{\mathbf{y}}-\mathbb{E}[\tilde{\mathbf{y}}])(\mathbb{E}[\tilde{\mathbf{y}}]-f)] \\
& && \text{(multiplying out and linearity)} \\
&= \mathbb{E}[(\boldsymbol{\epsilon}-\mathbb{E}[\boldsymbol{\epsilon}])^2]+\mathbb{E}[(\mathbb{E}[\tilde{\mathbf{y}}]-f)^2]+(f-\mathbb{E}[\tilde{\mathbf{y}}])^2 \\
& && \text{(last term in line above cancels out to zero)} \\
&= Var(\boldsymbol{\epsilon})+\sum_i(\tilde{y}_i-\mathbb{E}[\tilde{\mathbf{y}}])^2+\sum_i(f_i-\mathbb{E}[\tilde{\mathbf{y}}])^2 \\
&= \sigma^2+Var(\tilde{\mathbf{y}})+Bias(\tilde{\mathbf{y}})^2
\end{aligned}$$

□

This equation represents the relationship between the bias and the variance of an estimator. The MSE is equal to sum of the variance, the square of the bias and the noise.

The bias of an estimator describes the tendency the estimator has to over- or underestimate the $\mathbf{y}$-value, i.e. the difference between the data and the expected value of its estimator. The bias is represented by the expression

$$Bias(\tilde{\mathbf{y}}) = f_i - \mathbb{E}[\tilde{\mathbf{y}}] = \sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])$$

The variance of an estimator describes how far the collection of estimates are from the expected values of the estimates. In other words, how close together or far apart the estimated values are. The expression for the variance is

$$Var(\tilde{\mathbf{y}}) = \mathbb{E}[(\tilde{\mathbf{y}} - \mathbb{E}[\tilde{\mathbf{y}}])^2] = \sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2.$$

The bias and variance are the sources of error of an estimator. In order for the MSE to stay the same, if the bias decreases, then the variance must increase and vice versa. The bias and variance are complementary to each other. If an estimator has high bias, it will vary less and thus have low variance. If an estimator has high variance, meaning it can vary more to fit the data points better, it must have less bias.

The bias-variance tradeoff can be seen as a function of the complexity of the chosen model. We will get back to this in section 4.

2.5 k-fold Cross Validation

The *Cross Validation* (CV) method is a resampling technique often used to evaluate a model when we have limited data. We set a side a segment of the data and fit the model on the rest of the data. The segment that was set aside is later used to test the model.

There are many versions of the CV method. A popular one is the k-fold Cross Validation, the algorithm is described as follows:

Algorithm 1 k-fold Cross Validation

```
Shuffle the dataset randomly
Split the dataset into k folds
for each fold do
    take the fold as a hold out test set
    take the rest of the folds as training data set
    fit the model on the training set
    evaluate the model on the test set
    compute and store the evaluation score
Take the average of the evaluation scores for all the folds
```

The evaluation score mentioned in Algorithm 1 could be the MSE, R^2 score or some other measure. In this report we will use the MSE.

The k-fold CV method ensures that every data point appears at some point in time as training and test data. This can prevent overfitting from occuring, and is a reason why the k-fold CV is often preferred to the bootstrap method (section 2.6).

2.6 The Bootstrap method

The *bootstrap* method is a resampling technique used to estimate values on a dataset by sampling the dataset with replacement. The bootstrap method can be beneficial when we have a small data set or a poorly behaved data set, because it does not require any assumptions about distribution. Another advantage of the bootstrap method is that it is quite simple to implement, and simple to apply to complex data sets.

Unlike the k-fold CV method, all of the data is used as training data in the bootstrap method. If we are using the bootstrapping method to evaluate the MSE for a regression method, this means we would fit the model and predict new values using the same data. This could likely result in overfitting. Because the bootstrapping method is easier to implement than the k-fold CV, it is wise to be familiar with it. However, in practice, the k-fold CV is more commonly used.

Algorithm 2 summarizes the bootstrap procedure.

The evaluation score mentioned in algorithm 2 could be the MSE, R^2 score or some other measure.

Algorithm 2 Bootstrapping

let k be the number of bootstrap iterations
for $i = 1, \ldots, k$ **do**
Draw with replacement m numbers from the observed variables $\mathbf{x} = (x_1, x_2, \ldots, x_m)$
Define a vector $\mathbf{x}^*$ that contains the values which were drawn from $\mathbf{x}$
Using the vector $\mathbf{x}^*$, compute the evaluation score
Take the average of the evaluation scores for all the bootstrap iterations

3 Implementation

The full implementation, including tests and benchmark calculations, can be found at my GitHub repository (McLean 2019).

The OLS and Ridge regression methods have been implemented, using the Singular Value Decomposition (SVD) to find the inverse in the β-expressions. For the Lasso regression the scikit-learn functionality has been used (Pedregosa et al. 2011). All the implementations are function-based, however scikit-learn has opted for object-oriented implementations of the regression methods. In retrospect, it would have been better to implement the methods this way, in order to have better compatibility with scikit-learn.

The following code shows how the β coefficients were found for the OLS regression using SVD:

```
inv = SVDinv(X.T.dot(X))
beta = inv.dot(X.T).dot(y)
```

To evaluate the MSE and R^2 score, the functions `mean_squared_error` and `r2_score` from scikit-learn have been utilized.

The k-fold CV method has been implemented as well. For the shuffling of the data, an index set was constructed using a random permutation of the input data:

```
index = np.random.permutation(np.arange(len(z)))
X_shuffled = X[index]
z_shuffled = z[index]
```

The results using the k-fold CV function in the GitHub repository (McLean 2019) for the bias-variance tradeoff did not behave as expected. The MSE was not larger than or equal to the sum of the bias^2 and variance, as it should be. Therefore, the bootstrap method has been implemented, and will be used to demonstrate the bias-variance tradeoff.

All the implementations have been tested by comparing the MSEs resulting from these functions with the MSEs resulting from the scikit-learn functions. The test functions compare the MSEs using a tolerance. The following code is an example:

```
tol = 1E-14
success = abs(MSE_own - MSE_skl) < tol
assert success
```

In the functions that test the k-fold CV code, the tolerance had to be set to a relatively large number, for example 10^-3. Because of the randomness in the splitting of the data, it is unlikely to get an MSE that is nearly identical to that of scikit-learn.

The k-fold CV code does not pass the test when using Lasso regression. Therefore the Lasso results have been analyzed with caution.

When implementing the real terrain data, the data was downsized and normalized because otherwise the program would run very slowly.

The following lines perform the downsizing and normalizing:

```
terrain_downsized = imresize(terrain1, 0.05)
terrain_downsized = terrain_downsized/terrain_downsized.max()
```

All figures have been generated using the matplotlib and seaborn libraries.

4 Results

4.1 Franke Function

The following subsections include the results when analyzing our methods on the Franke function (defined in appendix .1). Some stochastic noise from the normal distribution with $\mu = 0$ and $\sigma^2 = 1$ has been added to perform the analysis. The MSEs and R^2 scores for each regression method have been calculated. The MSE values from the k-fold CV have been compared to choose the best model fit.

Figure 1 is the Franke function evaluated on $x = 0, 0.05, 0.1, \ldots 1$ and $y = 0, 0.05, 0.1, \ldots 1$. Figure 2 is the Franke function evaluated on the same x and y, but with the added noise.

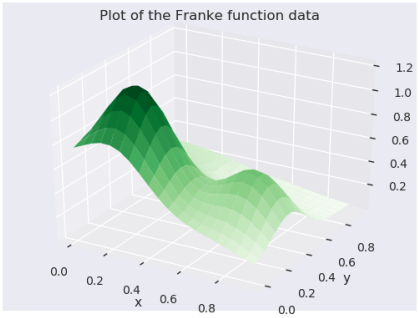

Figure 1: Surface plot of the Franke function evaluated on $x = 0, 0.05, 0.1, \ldots, 1$ and $y = 0, 0.05, 0.1, \ldots, 1$ without added noise.

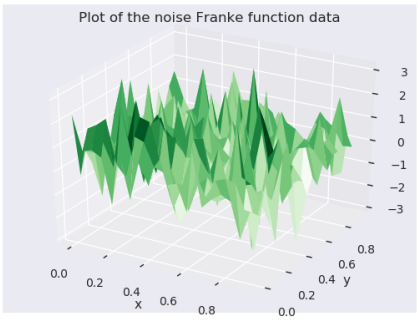

Figure 2: Surface plot of the Franke function evaluated on $x = 0, 0.05, 0.1, \ldots, 1$ and $y = 0, 0.05, 0.1, \ldots, 1$ with added noise from the normal distribution $N(0, 1)$.

4.1.1 Ordinary Least Squares

In table 1 we see that generally, the CV has a higher MSE than when we have no resampling. When de do not resample the data, we tend to overfit, resulting in less error. We also note that the MSE tends to get smaller when we increase the model complexity. However, due to the randomness in the CV, the MSEs aren't strictly following this trend.

Degree	No resampling	`train_test_split`	5-fold CV
1	1.09176	0.97884	1.10551
2	1.06856	1.04647	1.09041
3	1.06511	1.19498	1.13441
4	1.06214	1.23368	1.16380
5	1.02929	0.97475	1.12000

Table 1: The MSEs using the OLS model on data from the Franke function with stochastic noise from the normal distribution $N(0, 1)$. For the resampling, the test size was 0.33.

Table 2 shows the R^2 scores when we do not resample the data and when we use train and test data, where the test data is $\frac{1}{3}$ of the entire data. Again, we see that the fit seems to be better when the model complexity increases. If a fit yields a negative R^2 score, then the fit should be discarded in favor of a horizontal line.

Degree	No resampling	`train_test_split`
1	0.02982	0.01650
2	0.05043	-0.08154
3	0.05350	-0.02506
4	0.05614	0.04073
5	0.08534	0.01535

Table 2: The R^2 scores from the OLS model for data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. For the resampling, the test size is 0.33.

Based on the results in table 1, polynomial degree 5 gives the lowest MSE scores. Therefore we will look at the confidence intervals for the β's when the polynomial degree is 5, see figure 3.

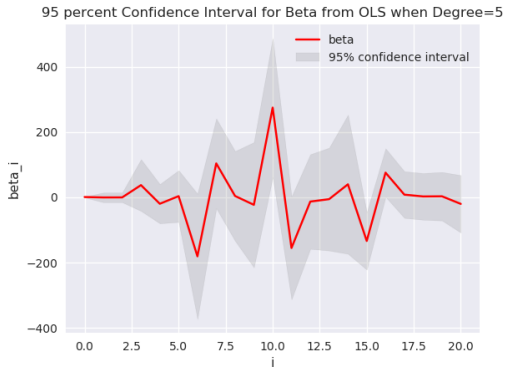

Figure 3: Plot of the 95% confidence interval for the β-values using the OLS method and polynomial degree 5.

The scores in table 1 were tested against scikit-learn's functions and found to be satisfactory.

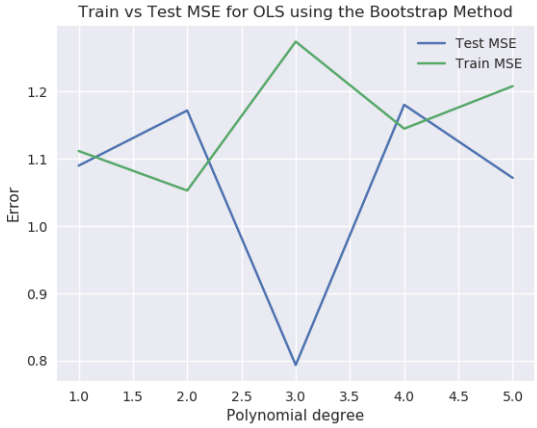

Figure 4: Plot of the test vs train MSE for OLS with polynomial degree up to 5. Performed on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The MSEs were found using the bootstrap method.

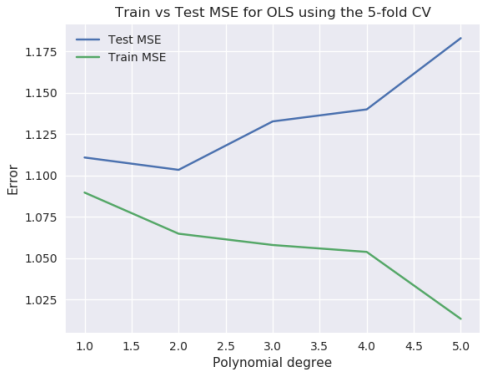

Figure 5: Plot of the test vs train MSE for OLS with polynomial degree up to 5. Performed on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The MSEs were found using 5-fold CV.

The MSEs in figure 4 are not behaving quite as expected. In figure 2.11 in Hastie, Tibshirani, and Friedman (2009, p. 38) we see that as the complexity (polynomial degree) increases, the train error should decrease and the test error should decrease to a certain point and then start to increase. The reason is that we experience overfitting when we use the same data for fitting the model and predicting the values. In figure 4, we can see that the test error does decrease to a certain point, and then starts to increase. The train error, however, is not showing the expected the behavior.

Because the bootstrapping method has a higher risk of overfitting than the k-fold CV, we look at the train and test MSEs from the k-fold CV as well. In figure 5, the errors are behaving as expected. We see that around polynomial degree $n = 2$, we start to overfit the data when using the training data.

As mentioned in section 2.4, the bias-variance tradeoff can be viewed as a function of the complexity. In figure 6, we see that for low complexity, the variance is low and the bias is high. As the complexity increases, the variance increases and the bias decreases. Having low bias and high variance might indicate that the model has overfitted the data. We see that the shift happens around polynomial degree $n = 2$ or $n = 3$, which we saw in figure 4 is where the overfitting started to happen. The ideal model would favor neither the bias nor the variance, but find some kind of compromise.

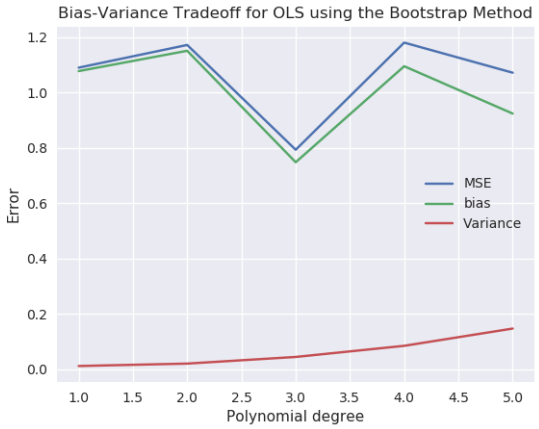

Figure 6: Plot of the bias-variance tradeoff for OLS with polynomial degree up to 5. Performed on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The values have been found using the bootstrap method.

4.1.2 Ridge Regression

Table 3 and table 4 show the MSE and R^2 scores when $\lambda = 0.1$. This λ-value was chosen based on figure 7, as it seemed to be the value that worked best in average for all the polynomial degrees.

Degree	No resampling	`train_test_split`	k-fold CV
1	1.09176	1.23330	1.10058
2	1.06863	1.25628	1.08679
3	1.06617	0.98473	1.09149
4	1.06592	1.09433	1.12546
5	1.06557	1.09295	1.11068

Table 3: The MSEs using the Ridge regression model on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The hyperparameter $\lambda = 0.1$.

Degree	No resampling	`train_test_split`
1	0.02982	-0.03928
2	0.05037	0.04611
3	0.05256	-0.07913
4	0.05278	-0.06364
5	0.05310	0.05515

Table 4: The R^2 scores from the Ridge regression model for data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The hyperparameter $\lambda = 0.1$.

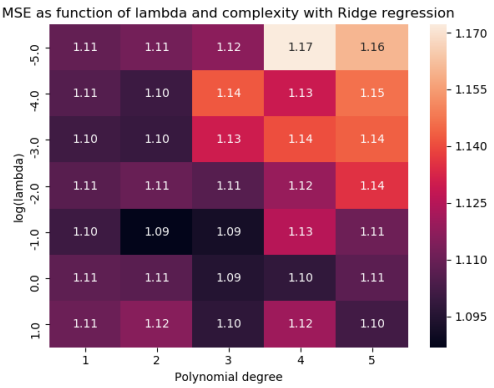

Figure 7: Plot of the MSE for Ridge regression as a function of complexity and λ. Regression on the Franke function with added stochastic noise from the normal distribution $N(0, 1)$.

From figure 7, we can see that the optimal Ridge regression fit is a polynomial of second degree where $\lambda = 10^{-1} = 0.1$. The MSE for this model is 1.08679.

One can also infer from figure 7 that there is no "magical" λ. Which λ-value to chose depends on the complexity of the model. There doesn't seem to be a general trend for the relationship between MSE and λ for any given degree, either. However, for higher complexity, it appears that the MSE is decreases as λ-values increase. This can be seen especially for polynomial degree $n = 5$.

Similar to the OLS case, we see in figure 8 that we have some overfitting around $n = 2$.

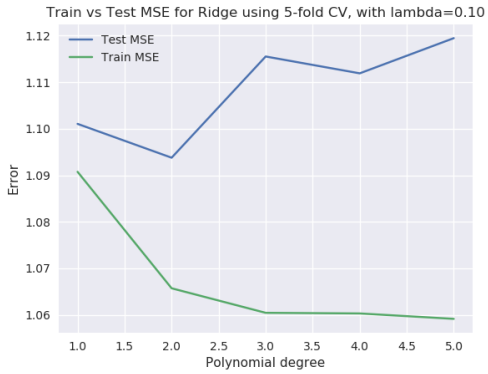

Figure 8: Plot of the test vs train MSE for Ridge regression with polynomial degree up to 5. The regression was performed on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The MSEs were found using 5-fold CV.

The overfitting can be viewed in the plot of the bias-variance tradeoff as well. Although figure 9 isn't quite as illustrative as figure 6 for OLS, we can see the same trends. The bias is initially high and starts to decrease around $n = 2$, whereas the variance is initially low and starts to increase around $n = 2$.

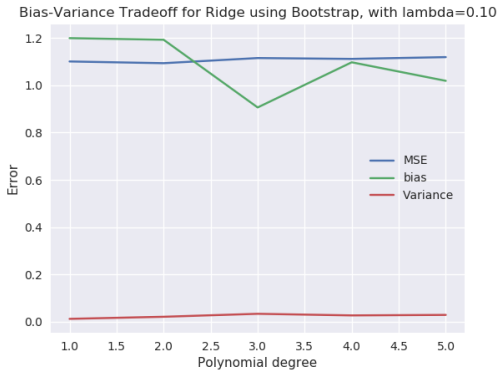

Figure 9: Plot of the bias-variance tradeoff for Ridge with polynomial degree up to 5. Performed on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$.

4.1.3 Lasso Regression

Since the implemented Lasso methods did not seem to fully agree with those of scikit-learn, the following results may not be completely trustworthy.

For table 5 and table 6, $\lambda = 1$ was chosen, because the machine gave warnings about the method not convering when using a lower λ-value.

Degree	No resampling	`train_test_split`	k-fold CV
1	1.12532	1.32632	1.13025
2	1.12532	0.85222	1.12966
3	1.12532	1.12957	1.12824
4	1.12532	1.35374	1.13248
5	1.12532	1.09352	1.13037

Table 5: The mean squared errors using the Lasso regression model on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The hyperparameter is $\lambda = 1$. The test size is 0.33.

Degree	No resampling	`train_test_split`
1	0.0	-0.00852
2	0.0	-0.00017
3	0.0	-0.00185
4	0.0	-0.00122
5	0.0	-0.01279

Table 6: The R^2 the Lasso regression model on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The hyperparameter is $\lambda = 1$. The test size is 0.33.

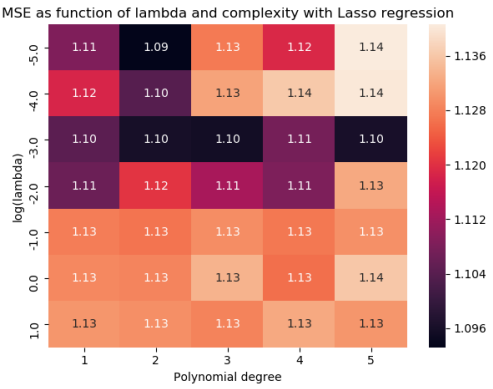

Figure 10: Plot of the MSE for Lasso regression as a function of complexity and λ. Regression on the Franke function with added stochastic noise from the normal distribution $N(0, 1)$.

From Figure 10, we see that the best Lasso fit is with a polynomial of second order and $\lambda = 10^{-5}$. The MSE for this model is 1.09318.

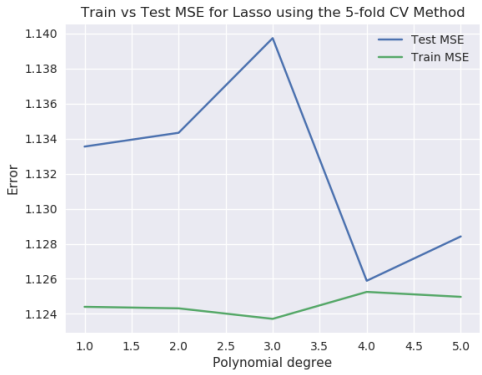

Figure 11: Plot of the test vs train MSE for Lasso regression with polynomial degree up to 5. Performed on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The MSEs were found using 5-fold CV.

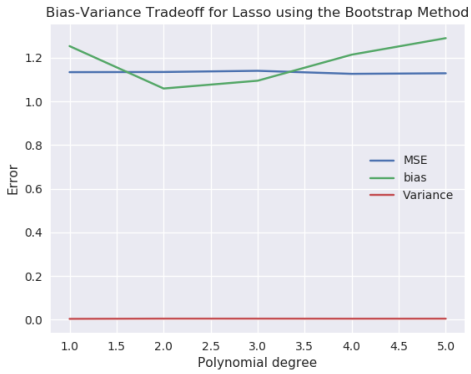

Figure 12: Plot of the bias-variance tradeoff for Lasso with polynomial degree up to 5. Performed on data from the Franke function with added stochastic noise from the normal distribution $N(0, 1)$. The values were found using the bootstrap method.

4.1.4 The Best Fit

Summarizing the results from the three previous sections, we get table 7.

Regression	Minimum MSE	Polynomial Degree	λ value
OLS	1.09041	2	N/A
Ridge	1.08679	2	10^{-1}
Lasso	1.09318	2	10^{-5}

Table 7: The minimal MSE values from 5-fold CV with OLS, Ridge and Lasso regression on data from the Franke function with noise from the normal distribution $N(0, 1)$.

From table 1, we can see that the lowest MSE from k-fold CV for OLS is when we use a second order polynomial. However, in general for all three methods of calculating the MSE that are posted in table 1, a fifth order polynomial is better when using OLS. And when looking at the R^2 scores for OLS in table 2, a fifth order polynomial seems best as well.

We see from table 3, that though the MSE from the k-fold CV is lower for Ridge than OLS, in general Ridge regression with a second order polynomial and $\lambda = 0.1$ gives higher MSE values than an OLS polynomial of the fifth order.

Therefore we can deduce that the best fit for the Franke function (appendix .1) with some stochastic noise is a fifth order polynomial using OLS.

Figure 13 shows the fifth order OLS polynomial fitted to the noisy data in figure 2. Figure 14 shows the polynomial fitted to the non-noisy data in figure 1. The model seems to reproduce the data well.

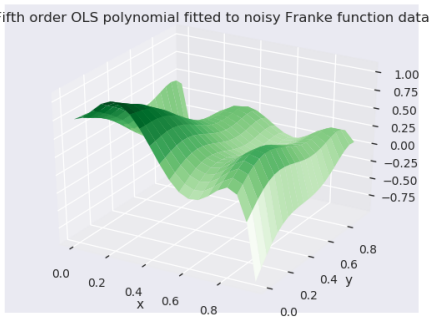

Figure 13: Plot of the fifth order polynomial fit from OLS of the noisy Franke function data in figure 2.

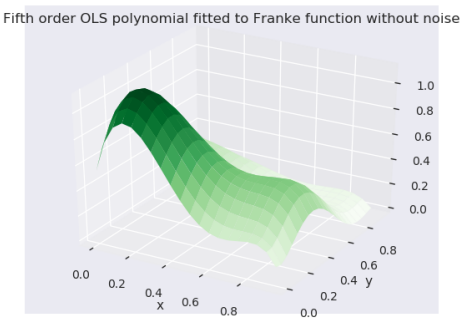

Figure 14: Plot of the fifth order polynomial fit from OLS of the Franke function data without noise in figure 1.

4.2 Terrain Data

The terrain data we will parametrize is from a region close to Stavanger in Norway. The data has been scaled by a factor of 0.05, to make the programs run faster. In addition to being downsized, the dataset has been normalized. We see that by taking the minimum and maximum of the terrain data, we get that the response values in the data set are between 50 and 1865. With this large range, we can get MSE scores in the order of 10^2 or even higher, depending on the model. In order to get MSE scores closer to 0, we normalize the data by dividing each entry by the maximum value. Now our values will lie in the interval $(0, 1]$. Because we are working with real data, we do not need to add any noise. The data contains natural noise.

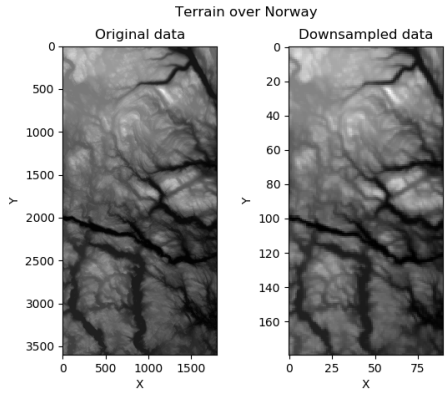

Figure 15: Plot of the terrain data. The plot on the left shows the data in its original size. In the plot on the right, the data is scaled by a factor of 0.05 and then normalized.

4.2.1 Ordinary Least Squares

Degree	No resampling	`train_test_split`	k-fold CV
1	0.01795	0.01817	0.01795
2	0.01632	0.01556	0.01634
3	0.01516	0.01530	0.01519
4	0.01472	0.01466	0.01474
5	0.01444	0.01454	0.01448

Table 8: The MSEs using the OLS model on the terrain data.

Degree	No resampling	`train_test_split`
1	0.44638	0.44503
2	0.49669	0.50970
3	0.53222	0.51991
4	0.54594	0.54914
5	0.55470	0.55287

Table 9: The R^2 scores using the OLS model on the terrain data.

From table 8, we gather that the best OLS model is a fifth order polynomial.

4.2.2 Ridge Regression

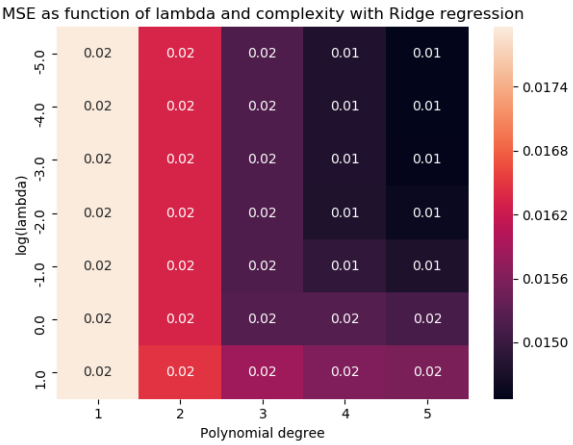

Figure 16: Plot of the MSE as a function of complexity and λ for Ridge regression of the terrain data.

From figure 16 we see that the MSE tends to decrease, as the λ-value decreases. As the complexity increases, the MSE tends to decrease. Therefore, we gather that the best Ridge model should be a fifth order polynomial with $\lambda = 10^{-5}$. In this particular run, we see that the lowest MSE indeed comes from a polynomial of degree $n = 5$ with $\lambda = 10^{-5}$. The MSE from this model is 0.01448.

We note that when $\lambda = 0$, performing Ridge regression is the same as performing OLS. Therefore, it is not altogether surprising that the MSE from Ridge regression with $\lambda = 10^{-5}$, which is close to 0, is very close to that of OLS.

Degree	No resampling	`train_test_split`	k-fold CV
1	0.01795	0.01737	0.01796
2	0.01632	0.01606	0.01632
3	0.01516	0.01484	0.01518
4	0.01472	0.01468	0.01477
5	0.01444	0.01426	0.01448

Table 10: The MSEs using the Ridge model with $\lambda = 10^{-5}$ on the terrain data.

Degree	No resampling	`train_test_split`
1	0.44638	0.46376
2	0.49669	0.49045
3	0.53222	0.53409
4	0.54594	0.54937
5	0.55470	0.56364

Table 11: The R^2 scores using the Ridge model with $\lambda = 10^{-5}$ on the terrain data.

4.2.3 Lasso Regression

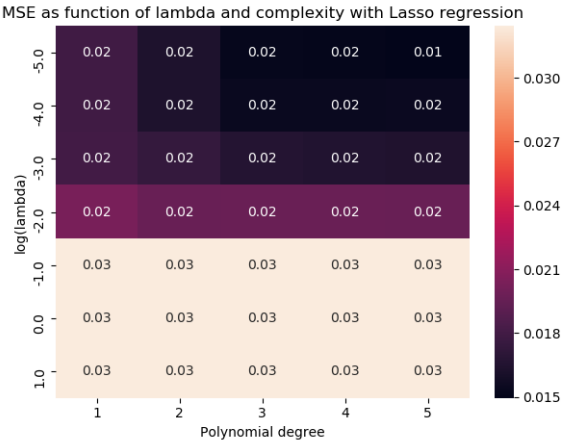

Figure 17: Plot of the MSE as a function of complexity and λ for Lasso regression of the terrain data.

From figure 17, we see that the lowest MSE comes from a fifth order polynomial with $\lambda = 10^{-5}$. For this model, the MSE is 0.01496.

4.2.4 The Best Fit

Table 12 shows that Ridge and OLS give the same MSE. Out of simplicity, we choose a fifth order OLS polynomial as the optimal model.

Regression	Minimum MSE	Polynomial Degree	λ value
OLS	0.01448	5	N/A
Ridge	0.01448	5	10^{-5}
Lasso	0.01496	5	10^{-5}

Table 12: The minimal MSE values from 5-fold CV with OLS, Ridge and Lasso regression on the downsampled terrain data.

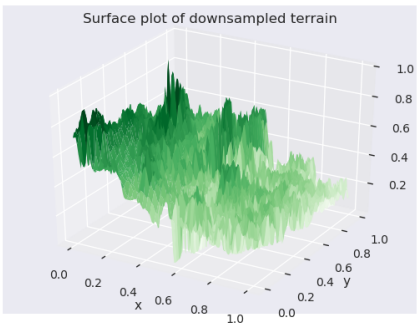

Figure 18: Surface plot of the downsampled terrain data from figure 15

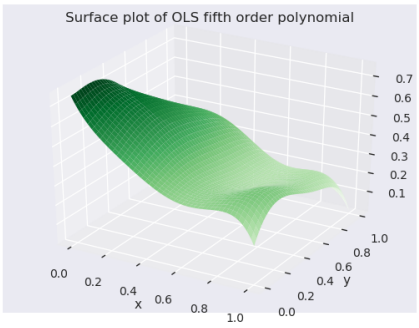

Figure 19: Surface plot of the fifth order OLS polynomial fit of figure 18.

Figure 19 shows the fifth order OLS fit of figure 18. The model seems to reproduce the data well.

5 Concluding remarks

We have now studied the various regression methods and resampling techniques in detail. We have seen how fitting and predicting on the same data can result in overfitting, and have seen the link between overfitting and the bias-variance tradeoff. After applying OLS, Ridge and Lasso regression to both the Franke function and the terrain data, we concluded that a fifth order polynomial from OLS best reproduces the data in both cases. However, for the terrain data, Ridge regression with polynomial degree $n = 5$ and hyperparameter $\lambda = 10^{-5}$ could have been a good fit as well.

In the future, it could be interesting to examine more than just the MSE to determine the best model fit. Perhaps taking the R^2 score into account as well, or defining some kind of score function that combines the two measures. It would also have been of interest to perform the same analysis with different amounts of noise. Perhaps then Ridge and Lasso regression would outperform OLS. Looking at the bias-variance tradeoff plots to determine the best fit could also be of interest.

I would have liked to spend more time developing my code, in order to ensure that the results are reliable. The MSE from the k-fold algorithm for Lasso regression did not compare well to the MSE that scikit-learn's k-fold function produces. Making an informed decision about which model best fits the data would be easier if the results are more reliable.

Furthermore, with more time, I would have liked to immerse myself more in the theory. Having a greater understanding of the methods, the differences and similarities would help me understand the results better and perhaps understand why a certain model might be better than another.

Appendices

.1 The Franke Function

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

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

The function will be defined for $x, y \in [0, 1]$.

.2 Extra Figures and Tables for Terrain Analysis

.2.1 Ridge Regression

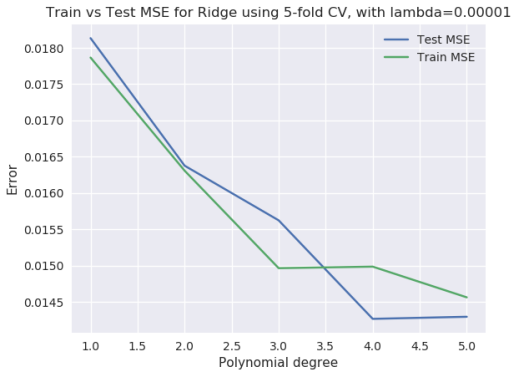

Figure 20: Plot of the train vs test MSE from Ridge regression with $\lambda = 10^{-5}$ on the terrain data.

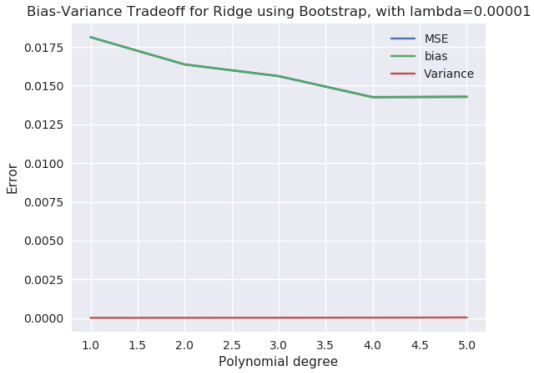

Figure 21: Plot of the bias-variance tradeoff from Ridge regression with $\lambda = 10^{-5}$ on the terrain data.

.2.2 Lasso Regression

Degree	No resampling	`train_test_split`	k-fold CV
1	0.03242	0.03251	0.03243
2	0.03242	0.03208	0.03242
3	0.03242	0.03234	0.03242
4	0.03242	0.03210	0.03242
5	0.03242	0.03221	0.03243

Table 13: The MSEs using the Lasso model with $\lambda = 1$ on the terrain data.

Degree	No resampling	`train_test_split`
1	0.0	-0.00049
2	0.0	-0.00026
3	0.0	-0.00076
4	0.0	-0.00017
5	0.0	-0.00014

Table 14: The R^2 scores using the Lasso model with $\lambda = 1$ on the terrain data.

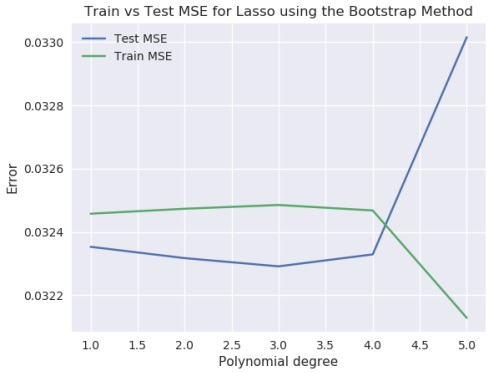

Figure 22: Plot of the train vs test MSE from Lasso regression with $\lambda = 1$ on the terrain data.

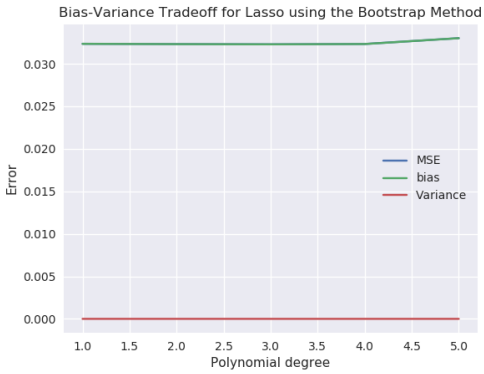

Figure 23: Plot of the bias-variance tradeoff from Lasso regression with $\lambda = 1$ on the terrain data.

References

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. 2nd ed. Springer.

Hjorth-Jensen, Morten (2019). *Lectures Notes in FYS-STK4155. Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis*. URL: `https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/Regression.html`.

McLean, Teah Kaasa (2019). *FYS-STK4155: Project 1 GitHub Repository*. URL: `https://github.com/teahkm/FYS-STK4155`.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.