Using linear regression for fitting terrain data

Eina B. Jørgensen, Anna Lina P. Sjur and Jan-Adrian H. Kallmyr

University of Oslo

October 7, 2019

Abstract

We evaluate the ordinary least squares (OLS), ridge, and lasso regression algorithms for fitting a 2D polynomial to Franke's function, as well as real terrain data over the inner Oslo fjord in Norway. Studying first Franke's function, we can identify the bias-variance trade-off when resampling, and find it consistent with theory. Comparing methods, we find that the ridge algorithm yields the smallest mean square error (MSE) of 0.014. As for the terrain data, we identify no bias-variance trade-off, possibly due to the complexity of the data. We find that OLS performs better with a MSE of around 2000. Furthermore, we observe that both the ridge and lasso algorithms likely converge to the OLS solution for smaller hyperparameters.

1 Introduction

The use of machine learning for problem solving has risen in popularity as large data sets have become available for analysis. There now exists many different methods in varying complexity for both supervised and unsupervised learning. All of these methods have advantages and drawbacks, as well as many similarities. This means that we can get familiar with some of the central themes in machine learning by studying simple algorithms, such as different linear regression schemes. A notable example is the bias-variance trade-off, where it is observed that the total mean-square-error (MSE) of a model starts off high, decreases until a minimum, and increases again as discussed by Hastie et al. (2009). This effect is known to be rather general, but there are also quirks related to each regression algorithm.

While using the Ordinary Least Squares (OLS) is straightforward, the Ridge and Lasso algorithms must be tuned using a hyperparameter, see Hoerl and Kennard (1970), and Tibshirani (1996) respectively. In particular, the algorithms may perform differently depending on the data we analyse, which will be a central theme in this report.

Starting with the Theory and methods section, we will describe three different algorithms for linear regression, as well as our resampling and cross-validation techniques and the bias-variance trade-off. In the Results section we will show our selected figures and data, with a focus on comparisons between the different methods. Moving on to the Discussion section we will consider the compared values and try to conclude which method seems to be most fit for fitting terrain data. We will also argue why that is the case. Finally, concluding in the Conclusion section, we will summarise the most important results as well as our thoughts around them.

2 Theory and methods

2.1 Linear Regression

We consider a dataset with n cases consisting of the response variable $\boldsymbol{y} = [y_0, y_1, \ldots, y_{n-1}]^T$ and some explanatory variables. The assumption is then made that $\boldsymbol{y}$ can be explained as a functional relationship on the form

$$\boldsymbol{y} = f(\cdot) + \boldsymbol{\epsilon} \tag{1}$$

Here, $\boldsymbol{\epsilon}$ is assumed to be normally distributed with mean 0 and variance σ^2.

A linear Regression model is built on the assumption that $f(\cdot)$ is a linear mapping from the explanatory variables x_{ij} to the response variable y_i, given by computing a weighted sum of the explanatory variables:

$$\tilde{y}_i = \beta_0 x_{i0} + \beta_1 x_{i1} + \ldots + \beta_{p-1} x_{ip-1}$$

Here, $i = 0, 1, \ldots, n-1$, $\tilde{y}_i$ is the prediction, $\{\beta_j\}_{j=0}^{p-1}$ are the regression parameters, while p is the number of explanatory variables.

In vectorized form, this can be written as

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

where $\tilde{\boldsymbol{y}} = [\tilde{y}_0, \tilde{y}_1, \ldots, \tilde{y}_{n-1}]^T$ are the predicted values, $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots . \beta_{n-1}]^T$ are the regression parameters, and $\boldsymbol{X}$ is the so called design matrix given by

$$\boldsymbol{X} = \begin{bmatrix} x_{00} & x_{01} & \ldots & x_{0p-1} \\ x_{10} & x_{11} & \ldots & x_{1p-1} \\ \vdots & \ddots & \ddots & \vdots \\ x_{n0} & \ldots & \ldots & x_{np-1} \end{bmatrix},$$

In this project, we are dealing with a two dimensional problem, where each row of the design matrix represents the variables of a m-th order polynomial, i.e. is on the form $[\{x^i y^j : i + j \leq m\}]$

In order to compute the regression parameters, a cost function $C(\boldsymbol{\beta})$ is introduced. The $\boldsymbol{\beta}$ is then defined as the minimization of the cost function. Different cost functions gives rise to different regression methods. Here we will look at Ordinary Least Squares, Ridge and Lasso regression.

2.2 Ordinary least squares

One form of the cost function is given as

$$\begin{aligned} C(\boldsymbol{\beta}) &= \frac{1}{n}\sum_{n=0}^{n-1} (y_i - \tilde{y}_i)^2 \\ &= \frac{1}{n}\sum_{n=0}^{n-1} (y_i - \boldsymbol{x}_{i*}\boldsymbol{\beta})^2 \\ &= \frac{1}{n}\left((\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\right) \end{aligned}$$

where $\boldsymbol{x}_i$ is the ith row of the design matrix. By setting $\frac{\partial C(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0$, one can show that the model parameters that minimizes the cost function are given by

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{2}$$

These are the model parameters used in the Ordinary Least Squares (OLS) model.

2.3 Ridge regression

A problem that can occur, especially when dealing with a large design matrix $\boldsymbol{X}$, is that the columns of $\boldsymbol{X}$ are linearly dependent, causing $\boldsymbol{X}^T\boldsymbol{X}$ to be singular. Hoerl and Kennard (1970) proposed a solution to the singularity problem by introducing a tuning parameter λ. This tuning parameter is added to the diagonal elements of $\boldsymbol{X}^T\boldsymbol{X}$, and thus causing the matrix to be non-singular. In Ridge regression, the cost function then takes the form

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{n=0}^{n-1} (y_i - \boldsymbol{x}_{i*}\boldsymbol{\beta})^2 + \lambda\sum_{i=0}^{p-1}\beta_i^2 \tag{3}$$

where $\lambda \in [0, \infty)$. The last term of (3) is often called the penalty term. Minimizing this

results in model parameters on the form

$$\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{\lambda I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{4}$$

Here, $\boldsymbol{I}$ is the identity matrix.. The hyperparameter λ then has to be tuned, for example by the use of cross-validation. One can see that by setting $\lambda = 0$, (4) simplifies to the OLS solution in (2).

2.4 Lasso regression

The choice of the penalty term in (3) is somewhat arbitrary, and other penalty functions could be considered. Tibshirani (1996) introduced a penalty function such that the cost function takes the form

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{n=0}^{n-1}(y_i - \boldsymbol{x}_{i*}\boldsymbol{\beta})^2 + \lambda\sum_{i=0}^{p-1}|\beta| \tag{5}$$

This gives rise to the Lasso regression model. Unlike for OLS and Ridges regression, there is no explicit expression for the model parameters. Instead, methods like gradient descent is used to find the $\boldsymbol{\beta}$ that minimizes the cost function.

2.5 Confidence interval of model parameters

The variance-covariance matrix of the OLS parameters in (2) is given as

$$Var\left(\boldsymbol{\beta}\right) = \sigma^2\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}$$

Here, σ^2 is the variance of $\boldsymbol{\epsilon}$ in (1).

It can be shown that the confidence interval of β_i then is given as (Hastie et al., 2009)

$$(\beta_i - z^{1-\alpha}v_i^{\frac{1}{2}}\sigma, \quad \beta_i + z^{1-\alpha}v_i^{\frac{1}{2}}\sigma) \tag{6}$$

Here, v_i is the ith diagonal element of $\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}$ and $z^{1-\alpha}$ is the $1-\alpha$ percentile of the normal distribution. For a 95% confidence interval, $z^{1-\alpha}$ equals 1.96.

2.6 Evaluation scores

In this project, we will use two well known expressions to calculate the error in the predicted values. That is the Mean Square Error (MSE)

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{7}$$

and the R^2 score function

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \overline{y})^2} \tag{8}$$

Here, the mean value $\overline{y}$ is given as

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

2.7 k-fold cross validation

There are several methods for estimating the skill of a machine learning model. One such method is the k-fold cross-validation procedure, which can be used when working with a limited data sample. The idea is to divide the data sample into k groups or folds, and then retain one of the folds to use as a test set after fitting a model to the remaining data. This is done for all folds. Algorithm 1 outlines the different steps in the procedure.

This yields a statistical estimate for how well the model will preform on new data. The choice of k will however affect the bias and variance in the estimation of the evaluation scores. It has been shown empirically that $k = 5$ or $k = 10$ gives neither a high bias nor variance (James et al., 2013). In this project, a value of 5 was chosen for k.

2.8 Bootstrap method for resampling

Another procedure for estimating the skill of a model is the Bootstrap method for resampling.

Shuffle the dataset randomly;
Divide the dataset into k folds;
foreach k **do**
  Take the kth fold out to use as test data set;
  Set the remaining folds as training data set;
  Fit a model to the training set;
  Evaluate the model on the test set;
  Retain the evaluation score and discard the model;
Calculate the mean of the evaluation scores;

Algorithm 1: The k-fold cross-validation algorithm.

In Bootstrap, a sample is drawn, with replacement, from the data, and the model is fitted to the sample. This is done multiple times, and for each cycle, the fitted model makes a prediction on a test set. Finally, all the predictions are evaluated. Algorithm 2 gives an overview of the bootstrap method used in this project.

Split the data into training and test sets;
Set a number of bootstrap samples n;
Set a sample size N;
foreach n **do**
  Draw a N-sized sample with replacement from the training set;
  Fit a model to the data sample;
  Apply the model on the test set, and store the prediction;
Evaluate all the model predictions;
Calculate the mean of the evaluation scores;

Algorithm 2: The bootstrap algorithm.

The sample size N was set to the same size as the train set in this project.

2.9 The bias-variance trade-off

Taking a look at the MSE again, (7) can be expressed as the expectation value of $(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2$, which can be decomposed as follows. For the full derivation, see (A.1).

$$\begin{aligned}\mathbb{E}\left[(\boldsymbol{y}-\tilde{\boldsymbol{y}})^2\right] =& \frac{1}{n}\sum_{i=0}^{n-1}(f_i-\mathbb{E}\left[\tilde{y}_i\right])^2 \\ &+\frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i-\mathbb{E}\left[\tilde{y}_i\right])^2+\sigma^2 \\ =& Bias^2+Variance+\sigma^2\end{aligned} \tag{9}$$

Here, f_i comes from (1) written out element wise as $y_i = f_i + \epsilon_i$. The first term in (9) is the squared bias, while the second term is the variance of the model. The last term, σ^2, comes from the assumption of a normal distributed noise in (1), and is the so called irreducible error. This error is beyond our control, even if the true value of f_i is known.

When the complexity of the model, i. e. the order of the polynomial, increases, the squared bias tends to decrease. For the variance, the opposite tends to happen (Hastie et al., 2009).

2.10 Franke's function and digital terrain data

In this project, the different regression methods were applied on both constructed and real data. The first was in the form of a sum of weighted exponentials, known as Franke's function:

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

In addition to the above terms, a normal distributed noise term with $\mu = 0$ and $\sigma = 0.1$ was added.

After testing the code on the simpler Franke's function, the same regression methods were used and evaluated on real digital terrain data downloaded from `https://earthexplorer.usgs.gov/`. The data used in this project is of the inner Oslo fjord region. Due to the large amount of data points, the set was reduced to 1% of its original size, by selecting every 10th point of every 10th row. The main features of the terrain was still preserved, as can be seen in Figure 10 in the Appendix.

2.11 Implementation

Since both OLS and Ridge gives an explicit expression for $\boldsymbol{\beta}$, the model parameters can be calculated directly. In this project, expression (2) and (4) was calculated using the linear algebra functionality in the Python package `numpy`. To prevent problems with matrix inversions, the function `numpy.linalg.pinv`, which calculates the inverse using singular-value decomposition, was used.

Unlike OLS and Ridge regression, there is no general, explicit expression for the model parameters in Lasso regression. Therefore, functionalities from the `scikit-learn` package was used to calculate $\boldsymbol{\beta}$ in the Lasso regression case.

All plots were made using the Python package `matplotlib`.

The full code can be found in a github repository [1].

3 Results

3.1 Regression on Franke's function

The "terrain" data z in this part was produced by applying Franke's function to a $n \times n$ evenly spaced xy grid, with $x_i, y_i \in [0, 1]$, and for each point added a normally distributed noise with $\mu = 0$ and $\sigma = 0.1$. For a visualization of the franke functions data points and the prediction, see figure **??** in the appendix.

3.1.1 β-values confidence interval

Before applying any resampling methods we used the regular *ordinary least squares* method (2) to our data and had a look at the confidence interval of the β values when approximating the data to a polynomial of degree $m = 5$. The confidence intervals are shown in Figure 1. They range between 0.23 and 35 in width, while the calculated β_i values range between approximately -50 and 50.

3.1.2 MSE and R^2 of OLS, with and without resampling

To study the mean squared error (7) as a function of the model complexity, we applied the OLS-method to the data set for various values of polynomial degree m, both when using the entire data set as both test and training data, and by using the bootstrap method (Algorithm 2) for resampling (Figure 2).

Applying the same analisys to the R^2-score (8) of the two cases, we get the result as shown in Figure 3. For the R^2 score, we found the results easier to obtain when using the *k-fold cross validation* method (Algorithm 1).

[1] `https://github.com/janadr/FYS-STK4155/tree/master/project1`

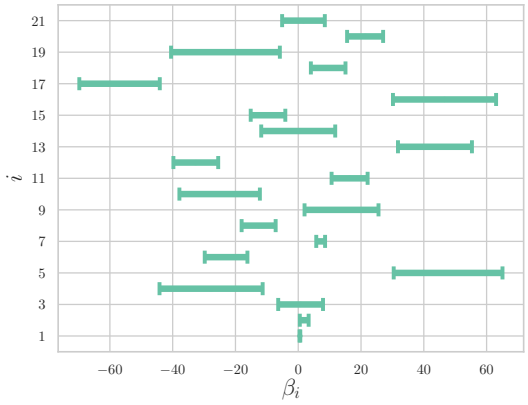


Figure 1: The β-values and their 95% confidence interval with $m = 5$ and $\sigma = 0.1$ for OLS used on data generated with Franke's function.

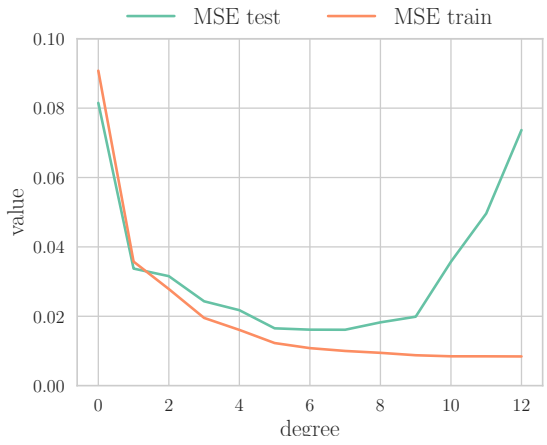


Figure 2: MSE of the model as a function of degree m. The model is trained on data generated with Franke's function, with n=20, noise $\sigma = 0.1$. In the case where no resampling is done (MSE-train), the MSE flattens out at a low value. When applying the bootstrap method (MSE-test) for resampling however, the MSE begins to rise after reaching a certain model complexity, creating a minimum point where the error is the lowest.

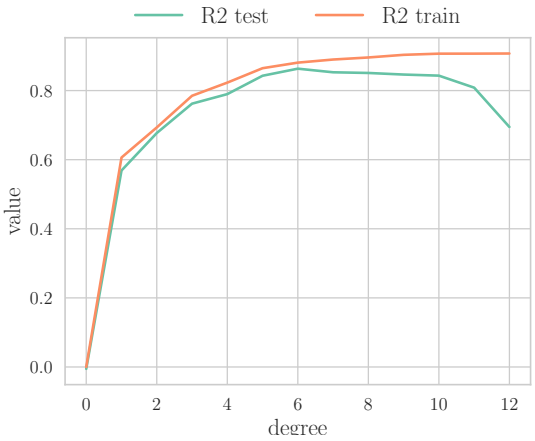


Figure 3: R^2-score of the model as a function of degree m. The model is trained on data generated with Franke's function, with n=20, noise $\sigma = 0.1$. In the case where no resampling is done (R2-train), the R^2 increases and then flattens out. When applying the k-fold cross validation method (R2-test) for resampling however, the R^2 begins to decrease after reaching a certain model complexity, creating a maximum point where the R^2-score is closest to 1 (the optimal value).

3.1.3 Bias-Variance-tradeoff

Sticking to the bootstrap method for resampling, and using $n = 20$ with $\sigma = 0.1$ we also compute the bias and variance of the model as discussed in the theory. Plotting the variance, bias and MSE together results in Figure 4.

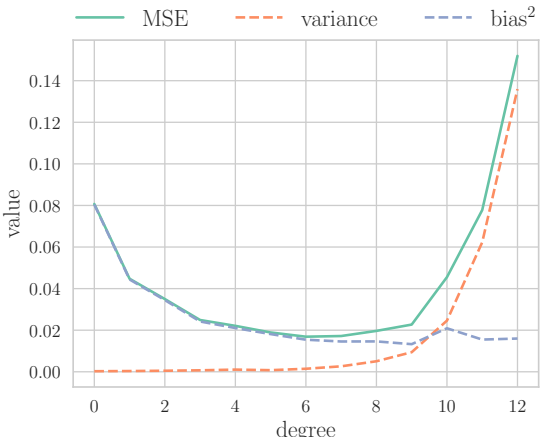


Figure 4: Bias, variance and MSE for the ordinary least square method on the Franke's function data set with $n = 20$ and noise $\sigma = 0.1$, resampled with the bootstrap method, as a function of model complexity/polynomial degree. The bias starts of high and decreases as the model complexity increases, whilst the variance grows with the model complexity.

3.1.4 Resampling with Ridge-regression

In order to find the parameters that will give the least MSE for the Ridge regression method, we need to tune both the degree m of the polynomial that we fit, and the optimal λ-value. This is obtained by doing Ridge regression on the data set with bootstrap, for both different values of m and different values of λ as shown in Figure 5.

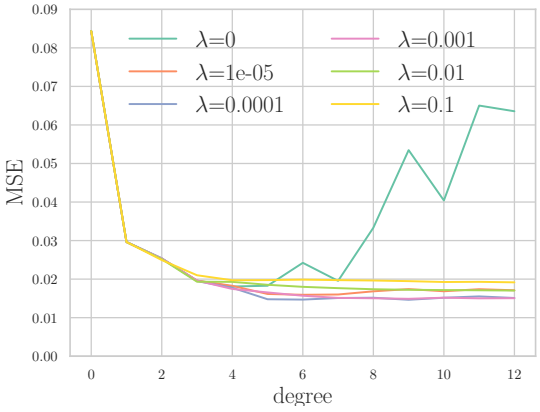


Figure 5: MSE for the Ridge regression method on the Franke's function data set with $n = 20$ and noise $\sigma = 0.1$, resampled with the bootstrap method, as a function of model complexity/polynomial degree. The different lines in the plot represent different values of the hyperparameter λ. When $\lambda = 0$ it is equivalent to the OLS-method.

3.1.5 Resampling with Lasso-regression

The same analisys, with various λ and m-values is done for Lasso-regression (Figure 6).

3.1.6 Comparing MSE of the different regression methods

A different, and less visual way of finding the method that gives the best approximation, that is, the smallest MSE, is to store all the MSE data for all hyperparameters and degrees, make the computer find the smallest value, and the corresponding parameters which is presented in table 1.

3.1.7 Terrain data

Some of the same figures were produced for real terrain data as for data generated with

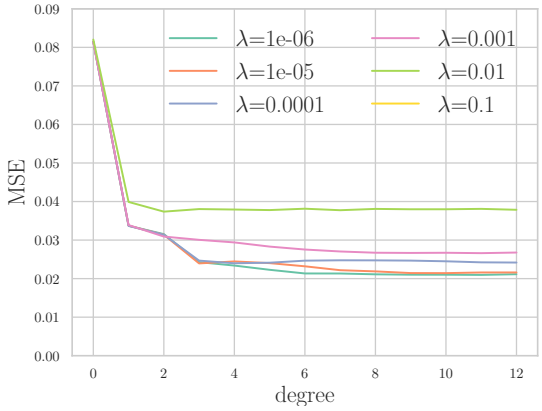


Figure 6: MSE for the Lasso regression method on the Franke's function data set with $n = 20$ and noise $\sigma = 0.1$, resampled with the bootstrap method, as a function of model complexity/polynomial degree. The different lines in the plot represent different values of the hyperparameter λ

Table 1: The smalles MSE for the different regressions methods used on the Franke's function dataset, and the hyperparameter λ and the degree m that results the MSE in question. Produced using the bootstrap resampling method.

Reg	**min. MSE**	m	λ
OLS	0.017	6	-
Ridge	0.015	9	1e-04
Lasso	0.021	11	1e-06

Franke's function. MSE as a function of degree for the OLS regression method can be seen in Figure 7. The MSE starts off at about 14500 for $m = 0$, and decreases to about 2000 as the degree approaches 35.

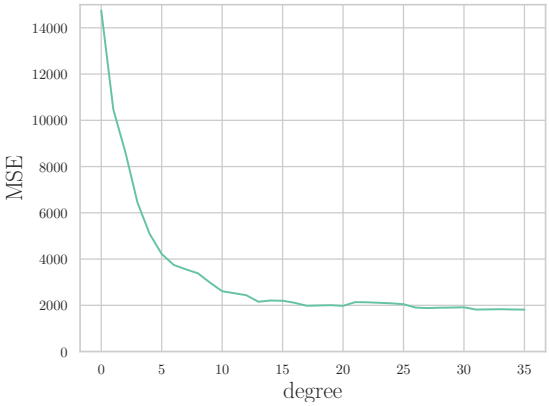


Figure 7: MSE for the OLS regression method on the reduced terrain data set. The MSE decreases as the degree increases, and flattens out for degrees higher than about 15

Figure 9 and 8 shows the same plot, but for Ridge and Lasso regression. Note that the degree here only goes up to 20 for Ridge and 15 for Lasso. Similar as for the results for Franke's function, each figure shows MSE for multiple λ.

For a visualization of what the prediction made by regression methods can look like, see figure **??** in the appendix

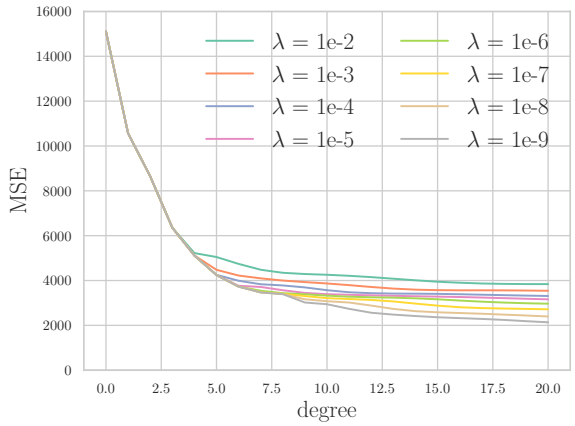


Figure 8: MSE for the Ridge regression method on the reduced terrain data set. Different colours represents different hyperparameters λ. The MSE decreases as the degree increases. The minimum MSE decreases as λ decreases.

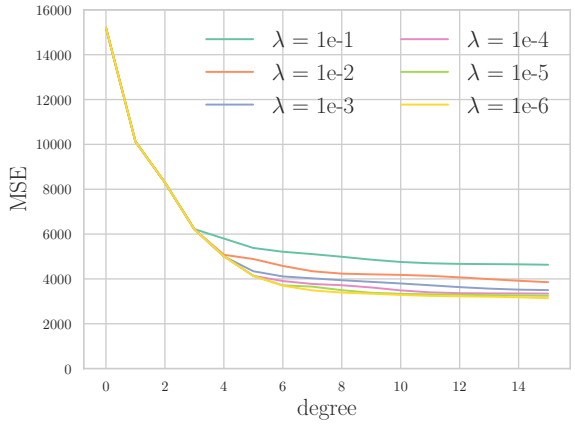


Figure 9: MSE for the Lasso regression method on the reduced terrain data set. Different colours represents different hyperparameters λ. The MSE decreases as the degree increases. The minimum MSE decreases as λ decreases.

4 Discussion

Behaviour of the different regression methods applied to Franke's function

Let's first look at the $\boldsymbol{\beta}$ confidence interval in Figure 4. As we can see, the range of values that has a 95% probability of containing β_i is quite wide, for some in the same order of magnitude as the β-value itself. Considering the expression for the interval, stated in (6), we see that the width of the interval strongly depends on the spread of $\boldsymbol{\epsilon}$ in (1). As the noise term $\boldsymbol{\epsilon}$ goes to zero, so does the width of the confidence interval.

Looking at the the mean squared error and the R^2-value of the approximation to Franke's function using ordinary least squares (Figure 2 and 3), we see the difference in the results when using the entire training set as test data, and when separating the data into test and training data using resampling. Comparing the two results, they both display the same trend: When train and test data is not separated, the approximation seems to unambiguously become better with increased model complexity, though, converging to what looks to be the minimum error. When using designated test data, however, it becomes apparent that using the training data for testing gives a false sense of security, as the mean squared error now starts increasing and the R^2-value decreasing when a certain model complexity is reached. It appears that a complexity given by a polynomial of degree $m \in [5, 7]$ gives the best result for both the R^2-value and the MSE when doing ordinary least squares regression in this case.

From equation (9) we expect the total error to be the sum of the bias squared, the variance of model, plus some irreducible error. Looking at Figure 4, showing the MSE, variance and bias for the ordinary least squared method on Franke's function, we see that the results displayed correspond nicely with these expecta-

tions. Thus we observe how bias and variance gives rise to the shape of the MSE curve, both in Figure 4 and 2. As discussed in lectures, and explained in Hastie et al. (2009), a higher variance for a more complex polynomial is a consequence of over fitting, allowing the polynomials to fit to the noise in the data. When only using the training data as test data, this over fitting goes unnoticed, which is the reason for the behaviour of the training data curve in Figure 2.

Understanding the importance of proper use of test data and resampling, we move on to compare the now discussed ordinary least squares method to our two other regression methods, ridge regression and lasso regression. We have here, as can be seen in Figure 5 and 6, chosen to focus on comparing the MSE of the methods, as opposed to the R^2-score simply because we found it more intuitive.

Moving into the territory of regression methods with hyperparameters, we study the behaviour of the mean squared error of the approximations for various values of λ and varying complexity. Common for both Lasso and Ridge is the almost constant behaviour of the MSE after reaching a certain model complexity, even when a designated test set is used. This stands in contrast to the convex shape of the OLS-MSE in Figure 2. By adding the penalty term in (3) and (5), we introduce a bias to the model. However, the penalty term also prevents over-fitting, and thus reduces the model variance.

Focusing on the behaviour of the MSE for different tuning parameters λ, it appears that the model preforms better for smaller λ.

Looking at the minimum MSE values of both Ridge and Lasso, the first preforms somewhat better than the latter on Franke's function. Both OLS and Ridge reaches a minimum MSE of about 0.015, with Ridge preforming slightly better. Lasso, on the other hand, has a minimum MSE around 0.021. Why Lasso preforms noticeably worse is not obvious, but could be due to us not finding the right value of convergence tolerance or some other factor. The exact numbers of the MSE will also vary with the random noise added to Franke's function. We have chosen a specific seed to evaluate, but the trend was the same regarding what seed was used.

Performance on real terrain data

Looking at the performance of the different regression methods for the reduced terrain data, a challenge arises. For ridge and lasso regression, the computational power required for polynomials of degree $m > 20$ is so great that the time needed to compute them is unreasonably high for a regular laptop. A consequence of this is that we don't have the data that is necessary in order to directly compare OLS to ridge and lasso for degrees like m around 30. From the behaviour of ridge and lasso on Franke's function, and our expectations from theoretical knowledge, we can however, predict that the behaviour of the MSE will continue the trend that we see for the degrees of which we have data.

For OLS, we don't get the same convex shape in the plot of the MSE as for Franke's function. As we have seen, the increase in MSE for Franke's function is due to an increasing model variance, i.e. overfitting the model to the training data. In the case of the real terrain data, we believe that the data is so complex, and demanding such a high order of polynomial to be represented, that the data does not become overfitted, even for polynomials up to 35th order. Taking a look at Ridge and Lasso, we see that the minimum MSE decreases as λ decreases, as we also found for Franke's function. However, since we did not experience overfitting with OLS, introducing a penalty term does not decrease the variance, and we do not get a better result with the Ridge or

Lasso regression.

5 Conclusion

In summary, we looked at the OLS, ridge, and lasso methods of linear regression, and compared them with each other for two different sets of data: a analytical function, the Franke function, and real terrain data of the inner Oslo fjord. The Franke function looks like idealised terrain data, and so it was natural to compare the methods for each data set. For OLS, we could observe a bias-variance trade-off when resampling using the Bootstrap algorithm. For the Franke function, we found that ridge gave the smallest MSE of 0.014, outperforming the other two methods, especially lasso, which yielded a MSE of 0.043. As a bias-variance trade-off was observed, it was not surprising that the ridge algorithm outperformed OLS, as ridge reduces variance at the cost of a penalty to bias. The lasso algorithm was outperformed possibly because of too high a tolerance (lack of computing power).

Performing the same analysis on the real terrain data, we observed no bias-variance trade-off, implying that the data set was even more complex than a 35 degree polynomial. As such it was never overfitted, and so the variance did not decrease. Comparing our methods on this data set, we found that OLS performed better than both ridge and lasso, with a MSE of around 2000. This was expected as we did not observe a bias-variance trade-off. The complexity of the data made ridge and lasso less feasible, as the variance never increased, i.e. the data did not become overfitted, even for OLS. The complexity of our terrain data may be due to a sharp trough at the beachline, and so more careful handling of the fjord topography could have resulted in a simpler data set.

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: prediction, inference and data mining. *Springer-Verlag, New York*, 2009.

Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer New York, 2013. ISBN 9781461471387.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Appendix

Decomposition of MSE in bias and variance

$$
\begin{aligned}
\mathbb{E}[(y-\tilde{y})^2] =& \mathbb{E}[(f+\epsilon-\tilde{y})^2] \\
=& \mathbb{E}[(f+\epsilon-\tilde{y}+\mathbb{E}[\tilde{y}]-\mathbb{E}[\tilde{y}])^2] \\
=& \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2]+\mathbb{E}[\epsilon^2]+\mathbb{E}[(\mathbb{E}[\tilde{y}]-\tilde{y})^2]+2\mathbb{E}[(f-\mathbb{E}[\tilde{y}])\epsilon] \\
& +2\mathbb{E}[\epsilon(\mathbb{E}[\tilde{y}]-\tilde{y})]+2\mathbb{E}[(\mathbb{E}[\tilde{y}]-\tilde{y})(f-\mathbb{E}[\tilde{y}])] \\
=& (f-\mathbb{E}[\tilde{y}])^2+\mathbb{E}[\epsilon^2]+\mathbb{E}[(\mathbb{E}[\tilde{y}]-f)^2]+2(f-\mathbb{E}[\tilde{y}])\mathbb{E}[\epsilon] \\
& +2\mathbb{E}[\epsilon]\mathbb{E}[\mathbb{E}[\tilde{y}]-\tilde{y}]+2\mathbb{E}[\mathbb{E}[\tilde{y}]-\tilde{y}](f-\mathbb{E}[\tilde{y}]) \\
=& (f-\mathbb{E}[\tilde{y}])^2+\mathbb{E}[\epsilon^2]+\mathbb{E}[(\mathbb{E}[\tilde{y}]-\tilde{y})^2] \\
=& Bias[\tilde{y}]^2+Var[y]+Var[\tilde{y}] \\
=& Bias[\tilde{y}]^2+\sigma^2+Var[\tilde{y}]
\end{aligned}
\tag{A.1}
$$

Here, we have used that

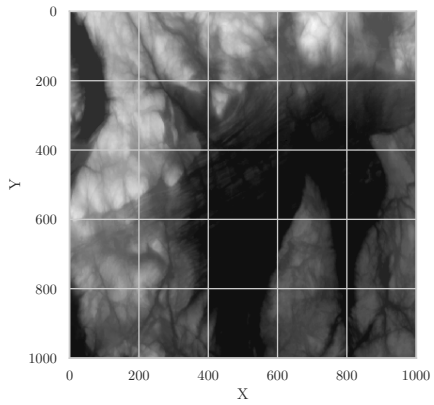
$$
\mathbb{E}[X^2] = Var[X]+(\mathbb{E}[X])^2
$$

$$
\mathbb{E}[f] = f \rightarrow \mathbb{E}[y] = \mathbb{E}[f+\epsilon] = f
$$

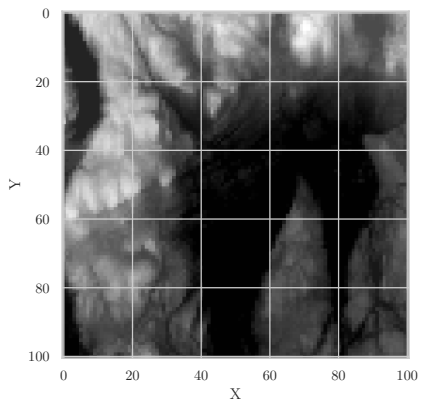
$$
Var[\epsilon] = \sigma^2 \rightarrow Var[y] = \mathbb{E}[(f+\epsilon-f)^2] = \sigma^2
$$

Visualization of terrain data

Figure 10a visualizes the real terrain data before reduction. Figure 10b displays the data after reduction, for comparison.



(a) Full terrain data over the Inner Oslo fjord with 1000×1000 *data points. Here black represents the ocean level.*



(b) Reduced terrain data over the Inner Oslo fjord in grey scale with 100×100 *data points, i.e. every tenth point of every tenth row of the full data set.*

Figure 10: Comparison of full and reduced terrain data. As geographical reference, Nesoddtangen can be seen to the lower right, with Bunnefjorden to the right and the Inner Oslo fjord in the centre. Parts of Drammensfjorden can be seen to the far left.

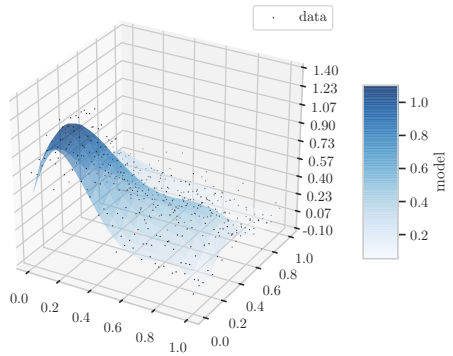


Figure 11: The prediction z *produced by OLS on Frankes function with a polynomial of degree* $m = 5$*. Det scatter plot is the data, and the surface plot is the predicted model.*

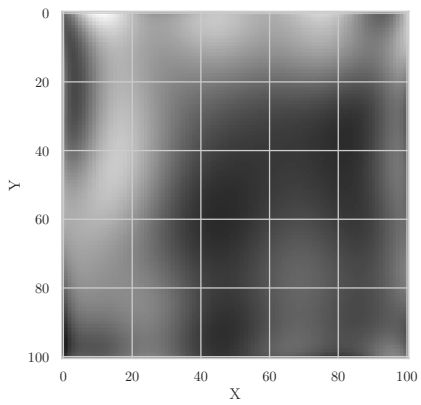


Figure 12: The prediction z produced by ridge regression with polynomial of degree $m = 10$.