Draft version October 10, 2020
Preprint typeset using LaTeX style emulateapj v. 12/16/11

FYS-STK4155 - PROJECT 1

Dahl, Jon Kristian
Bekkevik, Marit
Reaz, Fardous

Draft version October 10, 2020

ABSTRACT

In this report we study regression algorithms and resampling techniques to determine how these different machine learning algorithms perform for different data sets. We analyse the performance using constructed data sets generated by Franke's function, and compare these results with the results from a real-life terrain data set, analysed with the same tools. We compare the regression algorithms ordinary least squares, ridge regression and lasso regression. In addition, we study the resampling techniques bootstrapping and cross validation. For comparing the algorithms we look at four different measurements of error, namely mean squared error, R^2, bias and variance. The 5-fold cross validation resampling technique performed superior to the bootstrap (B = 50) method for all three regression methods and for both the constructed data and for the real-life data. The lasso regression method showed no advantage over simple OLS for both manufactured and real terrain data, but we found that ridge regression had a decrease in mean squared error for the terrain data at $\lambda_{\text{ridge}} = 10^{-1}$. The best fit were found with the ridge regression method for both data types.

1. INTRODUCTION

One of todays hottest topics in, not only computational science, but in science as a whole, is the topic of machine learning. With an ever-increasing amount of data in virtually every branch of our lives, machine learning is an incredibly powerful tool to extrapolate and predict from the data we have at hand. Since various types of data have different inherent true distributions, it is necessary to apply the correct methods of machine learning to make the best predictions.

In this report, we start by exploring the theory of bias-variance tradeoff and its relation to overfitting. Then we go through three methods of linear regression, namely ordinary least squares, ridge regression, and lasso regression. We study the pros and cons of using the different methods for both constructed data and real-life data. The concept of scaling is introduced. We then look at the theory behind two of the most common resampling methods, which are bootstrapping and cross validation. The resampling methods are compared in regards of the mean squared error, variance and bias, and we see what impact resampling techniques have compared to no resampling. We end the theory section by looking at Franke's function which is a function commonly used for testing machine learning algorithms.

In the method section, we go through how the controlled data is generated. This data is generated by the Franke function and is used for analysis of the different methods of linear regression and the different resampling methods. We then explain the framework for how we find a fit to the training data, and how we decide the optimal parameters, $\boldsymbol{\beta}$. A description of the data analysis follows, where we talk about what parameters are used for the different algorithms, like the number of folds, bootstrap repetitions and data points, and what quantities of interest we study. Then, the real-life data management is briefly introduced, followed by a short description of the layout of the program code.

We present our data and discuss the implications in the results and discussion section. The report ends with a concluding section.

All code used to generate data for this report is available at the GitHub repository https://github.uio.no/jonkd/FYS-STK4155_H20/tree/master/projects/project1.

2. THEORY

The theory section is mainly based upon Morten Hjorth-Jensen's lecture notes in the Applied Data analysis and Machine learning course (FYS4155) at the University of Oslo Hjorth-Jensen (2020).

2.1. *Bias-variance trade off*

The mean square error (MSE) of an estimated quantity is the average of the square of error, where error refers to the difference between the observed and predicted values. It is a measure of how accurately a model can predict the observed values. The MSE is given by

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

Similar to MSE, R-squared (R^2) provides a quantitative measure of the efficiency of a predictive model to reproduce observed data. However, in contrast to the absolute measure of MSE, R^2 provides a relative measure. The R^2 is given by

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

By optimizing the cost function $C(\mathbf{X}, \beta)$, here the MSE, the optimal parameters β are found. The cost function is given by

$$C(\mathbf{X}, \beta) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \mathbb{E}[(y - \tilde{y})^2], \tag{3}$$

where the $\mathbf{X}$ and β dependence comes from the calculation of $\tilde{y}_i$. The MSE cost function can be expressed as

$$\mathbb{E}[(\mathbf{y}-\tilde{\mathbf{y}}^2] = \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{\mathbf{y}}])^2 + \sigma^2. \tag{4}$$

The data points y_i are given by an unknown function f, with function values f_i, with some noise ϵ added. That is $y_i = f_i + \epsilon_i$. The noise is assumed to be normally distributed with a 0 mean and a variance of σ^2. Thus $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Since the mean of the noise is 0, $\mathbb{E}[\epsilon] = 0$ and the variance $\text{var}[\epsilon] = \mathbb{E}[\epsilon^2 - \mu] = \mathbb{E}[\epsilon^2] = \sigma^2$.

$$\begin{aligned}
\mathbb{E}[(y-\tilde{y})^2] &= \mathbb{E}[(f+\epsilon-\tilde{y})^2] && (5)\\
&= \mathbb{E}[(f+\epsilon-\tilde{y}+\mathbb{E}[\tilde{y}]-\mathbb{E}[\tilde{y}])^2] && (6)\\
&= \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2 + (\tilde{y}-\mathbb{E}[\tilde{y}])^2 + \epsilon^2 && (7)\\
&\quad - 2(f-\mathbb{E}[\tilde{y}])(\tilde{y}-\mathbb{E}[\tilde{y}]) - 2(f-\mathbb{E}[\tilde{y}])\epsilon && (8)\\
&\quad - 2(\tilde{y}-\mathbb{E}[\tilde{y}])\epsilon] && (9)\\
&= \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[(\tilde{y}-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[\epsilon^2] && (10)\\
&\quad - 2\mathbb{E}[f-\mathbb{E}[\tilde{y}]]\underbrace{\mathbb{E}[[\epsilon]}_{=0} - 2\mathbb{E}[\tilde{y}-\mathbb{E}[\tilde{y}]]\underbrace{\mathbb{E}[\epsilon]}_{=0} && (11)\\
&\quad - 2\mathbb{E}[f-\mathbb{E}[\tilde{y}]]\underbrace{\mathbb{E}[\tilde{y}-\mathbb{E}[\tilde{y}]]}_{=0} && (12)\\
&= \mathbb{E}[(f-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[(\tilde{y}-\mathbb{E}[\tilde{y}])^2] + \mathbb{E}[\epsilon^2] && (13)\\
&= \frac{1}{n}\sum_i (f_i - \mathbb{E}[\tilde{y}])^2 + \frac{1}{n}\sum_i (\tilde{y}_i - \mathbb{E}[\tilde{y}])^2 + \sigma^2 && (14)
\end{aligned}$$

In equation (12) the term $\mathbb{E}[\tilde{y} - \mathbb{E}[\tilde{y}]] = 0$, because when the differences between the points and their average is not squared, they will cancel each other.

The first term in equation (14) is the bias squared. Bias is a measure of the ability (or inability) for a machine learning algorithm to capture the true relationship between the parameters. Unfortunately, the true relationship f_i is rarely known. Instead the values y_i are used in place of f_i

The second term in equation (14) is the variance. The variance is a measure on how much the predictions vary between training samples. A large variance means that our model may give a very good fit for some training data, but a very bad fit for some other training data. We want the variance to be low so to know that the predictions will be approximately equally good across different test data sets.

The last term of (14) is the noise variance, also called the irreducible error.

In data analysis, the goal is always to get the variance and bias as low as possible, but lowering the variance will often make the bias increase, and vice versa. In figure 1 the principle of the bias-variance trade-off is illustrated. In the figure, the two training data points are fitted with a line. This line interpolates the two training points perfectly and has therefore a variance of 0. But compared to the rest of the data points, it deviates a lot. This means that its ability to fit the true relationship between the parameters is poor, and thus the bias is high.

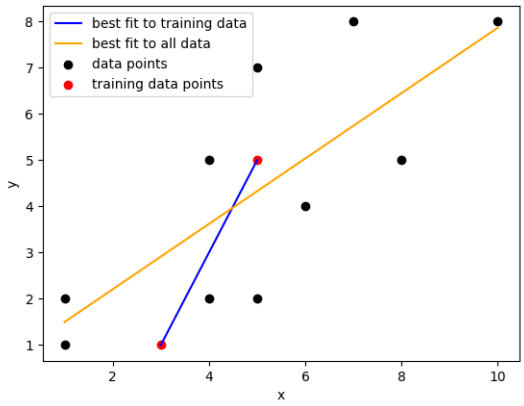

FIG. 1.— The blue line is the best line between to data points which represents an incredibly small set of training data. This line interpolates the two points perfectly, and therefore has 0 variance. But compared to the rest of the data points, it deviates a lot, which represents a high bias. In this case it would be better to have a larger variance and not intersect the training points perfectly, but get a lower bias and a better general fit to all the data.

In this case it would be better to accept a bit larger variance and not intersect the training points perfectly, so to obtain a lower bias and get a better fit of all the data points. One often has a battle between lowering the variance and lowering the bias since we wish to minimize both values. In practice, it is not possible to lower one without increasing the other, hence we have a bias-variance tradeoff.

Normally the set of training data is much larger. If the degree of the interpolating polynomial is very high, it is possible to interpolate every point of the training data perfectly with zero variance. Take for example any set of data points in a coordinate system and select some of them for training. By adjusting the amplitude and frequency of a simple sine wave, we could make the wave pass through every point in the set of training data points. If we were to compare this crazy sine wave to the test points that we left out, the sine wave is highly unlikely to represent the true distribution of the total data set.

We can invoke the same behaviour by choosing a polynomial of a very large degree for the model. The model will then fit the training data almost perfectly, but will fail to fit the test data. This is called overfitting and is illustrated in figure 2. The model complexity, in our case the polynomial degree, is on the x-axis and the error on the y-axis. With increasing complexity, the error for the training set decreases, but for the test set it increases. The best model complexity is where the test data is at its lowest error. This is the model complexity where the bias-variance tradeoff is optimized.

2.2. *Linear regression methods*

Regression is a collection of statistical techniques to estimate the relationship among quantities and predict a future response for a predictor variable. When faced with data points that looks linearly distributed or there is reason to suspect that they are, a linear regression method is used. It's a very popular linear approach to model the relationship between input, or explanatory variables, x_i

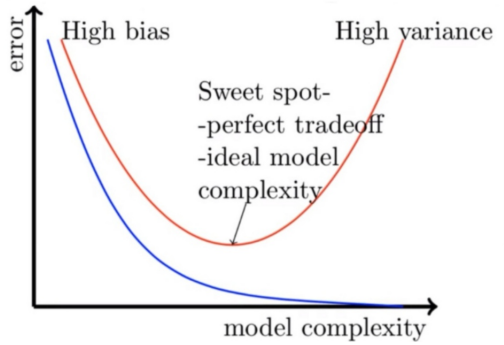

FIG. 2.— The blue line illustrates how the error of the training data decreases with increasing complexity. The variance can go to zero if, given n number of training data points, the polynomial degree $p \geq n-1$. But with increased polynomial degree (complexity), the model gets so adjusted to the training data that it fits the test data (red curve) less and less. The minimum of the red curve is the optimal complexity of the model. You can therefore say that the left side of the figure is the area of high bias, and the right side the area of high variance. The sweet spot is somewhere in the middle [a]

[a] https://mc.ai/bias%E2%80%8A-%E2%80%8Avariance-tradeoff/

and output, or response variables, y_i. The actual distribution model $f(x)$ is unknown. What is observed is $y_i = f_i + \epsilon_i$, where ϵ_i is noise assumed to be normally distributed with a zero mean and variance σ^2, $\mathcal{N}(0, \sigma^2)$. If the relationship between the output variable (y) and the input variable (x) is linear, for single dependent variable it can be described by the following equation,

$$y = \beta_0 + \beta_1 x + \epsilon. \tag{15}$$

where β_1 is the slope of the line. β_1 represents the strength of the relation between the quantities. If its value is close to 1 (or -1), there is a strong positive (or negative) relation between the quantities. In contrast, there is no significant relationship between the quantities if the value of β_1 is close to zero. For a multiple linear regression model with k dependent variables,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k + \epsilon. \tag{16}$$

A complex model can be constructed including higher power polynomials of one or more dependent variables,

$$y = \sum_{j=0}^{n-1} \beta_j x_i^j + \epsilon_i. \tag{17}$$

In linear regression, the idea is to use a line or curve (polynomial) and calculate how well it fits the data. There are multiple ways to measure this, but they all consider how far from the line/curve the data points are measured parallel with the response axis. This distance is called the residual. The residuals are, in other words, observed data minus fitted data.

2.2.1. *Ordinary least squares*

The simplest method to determine linear regression coefficients is the ordinary least squares (OLS). We begin with the matrix equation

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{18}$$

where $\boldsymbol{\beta}$ is the unknown vector we wish to solve for. Instead of simply neglecting the noise $\boldsymbol{\epsilon}$ and left-multiplying by $\mathbf{X}^{-1}$ to solve for $\boldsymbol{\beta}$, we link $\boldsymbol{\beta}$ to a cost function. For OLS, the cost function is the norm-2 of the mean squared error

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2, \tag{19}$$

which can be shown to equal [1]

$$C(\boldsymbol{\beta}) = \frac{1}{n}((\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})). \tag{20}$$

The purpose is to find the $\boldsymbol{\beta}$ which minimizes the cost function, since minimizing the cost function means minimizing the difference between the data we predict and the actual real data. We minimize the cost function by differentiating with respect to $\boldsymbol{\beta}$,

$$\frac{\partial C}{\partial \boldsymbol{\beta}} = 0 = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{21}$$

From (21) we get an expression for $\boldsymbol{\beta}$

$$\boldsymbol{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{22}$$

Now that we have the parameter values $\boldsymbol{\beta}$, we can make predictions by the equation

$$\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}. \tag{23}$$

There is a chance that the $\mathbf{X}^T\mathbf{X}$-matrix is non-invertible. In that case, equation (22) will not be exact solvable, but there are solutions to this problem. One way to circumvent the problem is to use the singular value decomposition. The problem can also be solved by adding a small extra term to $\mathbf{X}^T\mathbf{X}$ which, as we will see in section 2.2.2, is exactly what is happening when we apply ridge regression. In practice when the calculations are performed, we use a Moore Penrose pseudo-inversion of the matrix to avoid singularity issues [2].

How well the model fits the data is evaluated by the cost function, hence the motivation to minimize it. The OLS works well in many cases, but some conditions have to be fulfilled for it to function properly. It is important that the error is normally distributed as assumed, has a mean of 0, and a constant variance of σ^2. The error in one measurement must be uncorrelated to the error in the next, and the variables must be uncorrelated with the error. The variables may be correlated, but not perfectly correlated. E.g. like rolling a six on a dice is perfectly negative correlated to not getting a six. There may be terms that are dependent of more than one variable (e.g. a xy-term), but the model must be linear in the β_is.

2.2.2. *Ridge regression*

Ridge regression comes from the same basic idea as OLS, but has some more finesses. Some parameters, or features, are less important in predicting the outcome. To make these less significant in the model, we can give them a penalty. In ridge regression the penalty is $\lambda\boldsymbol{\beta}^2$, and what we are minimizing is the sum of the squared

[1] https://compphysics.github.io/MachineLearning/doc/pub/week35/html/week35.html

[2] https://numpy.org/doc/stable/reference/generated/numpy.linalg.pinv.html

residuals, plus $\lambda\boldsymbol{\beta}^2$, where λ is a penalty parameter. The cost function for ridge regression is given by

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 + \lambda\sum_{j=0}^{n-1}\beta_j^2. \tag{24}$$

It's not obvious which λ-value to use, so several values must be tested to see which produces the lowest error. We solve for $\boldsymbol{\beta}$ in the same manner as with OLS, namely differentiating the cost function and equaling it to zero. We then get

$$\boldsymbol{\beta}^{\text{ridge}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{25}$$

where $\mathbf{I}$ is the identity matrix. Note that if $\lambda = 0$ the expression for $\boldsymbol{\beta}^{\text{OLS}}$ is achieved.

As mentioned previously, we may get a $\mathbf{X}^T\mathbf{X}$ matrix which is not invertible. This stems from linear dependence between the columns of $\mathbf{X}$. A wonderful advantage with ridge regression is that the possibility of linear dependence in the columns of $\mathbf{X}$ is circumvented by adding the $\lambda\mathbf{I}$-term to the $\mathbf{X}^T\mathbf{X}$ in the expression for $\boldsymbol{\beta}$. Another source of singularity in $\mathbf{X}^T\mathbf{X}$ is fitting a polynomial of degree n when there are fewer than $n+1$ data points. If the rank of $\mathbf{X}^T\mathbf{X}$ is smaller than n, then there are not enough linearly independent columns to solve the equation unambiguously. This problem too can be avoided by adding the ridge parameter λ.

By making λ progressively larger, the significance of the less relevant parameters is weakened and can be asymptotically shrunk towards 0. This can be shown by the singular value decomposition (SVD). In SVD, $\mathbf{X}$ being a $n\times p$-matrix, can be factorized of the form $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$. $\mathbf{U}$ is a $m\times m$ orthogonal matrix, $\boldsymbol{\Sigma}$ is an $n\times p$ rectangular and diagonal matrix with non-negative numbers σ's, called the singular values, on the diagonal, and $\mathbf{V}^T$, a $n\times n$ orthogonal matrix with the eigenvectors of $\mathbf{X}^T\mathbf{X}$ as rows.

$$\begin{aligned}
\tilde{\mathbf{y}}^{\text{ridge}} &= \mathbf{X}\boldsymbol{\beta}^{\text{ridge}} \\
&= (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \\
&= \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T(\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T + \lambda\mathbf{I})^{-1}\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T\mathbf{y} \\
&= \left(\sum_{j=0}^{p-1} U_j\frac{\sigma_j^2}{\sigma_j^2+\lambda}U_j^T\right)\mathbf{y}
\end{aligned} \tag{26}$$

So if $\lambda \geq 0$, then

$$\frac{\sigma_j^2}{\sigma_j^2+\lambda} \leq 1. \tag{27}$$

which means that the parenthesis in front of $\mathbf{y}$ will get smaller with increasing λ. So this regression method shrinks the eigenvalues of $\mathbf{X}^T\mathbf{X}$ (singular values), and therefor the coordinates of $\mathbf{y}$ by

$$\frac{\sigma_j^2}{\sigma_j^2+\lambda}. \tag{28}$$

We see that small eigenvalues is relatively shrunk more.

2.2.3. *Lasso regression*

Lasso (least absolute shrinkage and selection operator) is similar to ridge with respect to the penalty, but instead of λ times the parameter squared, it has λ times the absolute value of the parameter. It is the L_1-norm instead of the L_2-norm. The cost function is

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 + \lambda\sum_{j=0}^{n-1}|\beta_j|. \tag{29}$$

Unlike ridge regression there's not an analytic expression for the optimal $\boldsymbol{\beta}$. What is being minimized James et al. (2017) can be expressed in a different way as

$$\min_{\beta}\left\{\sum_{i=0}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \text{ subject to } \sum_{j=1}^{p}|\beta_j| \leq s. \tag{30}$$

In the ridge case this would be

$$\min_{\beta}\left\{\sum_{i=0}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \text{ subject to } \sum_{j=1}^{p}\beta_j^2 \leq s. \tag{31}$$

That is to minimize the first expression as long as the last constraint is fulfilled. This is illuminated geometrically in figure 3. This example is simplified by having only two β_is (the x- and y-axis), even though ridge and lasso should not be used when there are only two parameters. This is done because it is easier to interpret a two dimensional figure visually than the multidimensional case. The ridge constraint is a circle in this space since it's the L_2-norm. Since the lasso constraint is of type L_1-norm, this represents a diamond shape with the circumference equal to s in this space. The ellipses in the top right corner are representing the expression that we are minimizing. The impact of the ellipse and the circle/diamond is the solution we want. In ridge this is some point on the circumference. Since the constraint is diamond-shaped for lasso, it has sharp corners. Therefore it is likely that the minimal ellipse will be touching one of the corners. Since the corners are characterized by lying on one of the axes we get that one or more of the β_is are 0. So by choosing this solution, some of the parameters are excluded from the model, leaving it simpler and easier to interpret. This is called feature selection, and is quite handy. In other words, lasso can actually shrink the impact of the irrelevant parameters all the way to zero, while in ridge the model were only desensitized to the irrelevant variables. In cases with many significant features, ridge tends to perform a bit better.

2.3. *Scaling*

Machine learning algorithm should always include scaling. It fixes possible data outliers, and some functions and algorithms will in fact not work properly without normalization. The most common ways of scaling are the rescaling, or min-max normalization, mean normalization and standardization. In rescaling the modified parameter x' is found from

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}. \tag{32}$$

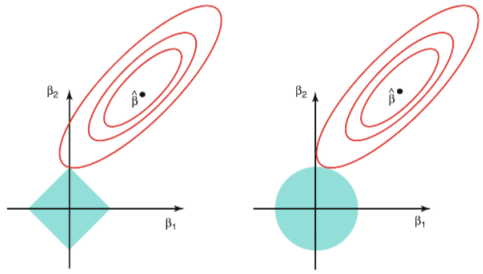

FIG. 3.— The ellipses are the functions to be minimize in respectively ridge (left) and lasso (right) regression. The light blue areas are the constraints, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2|\beta_2| \leq s$ respective James et al. (2017)

In mean normalization the formula is

$$x' = \frac{x - \overline{x}}{\max(x) - \min(x)}, \tag{33}$$

and for standardization

$$x' = \frac{x - \overline{x}}{\sigma}, \tag{34}$$

where $\overline{x}$ is the mean and σ is the standard deviation. In our algorithm we use standardization.

2.4. *Resampling techniques*

Resampling techniques are important tools for utilizing the available data as good as possible. They are particularly important tools when the amount of data is scarce, since few data points in general means bad predictions and the result of resampling is to improve predictability with the available data. As the name suggests, resampling techniques involve making resamples of the training data. The resamples are done several times, and are either done in a systematic fashion, or randomly. Two of the most common resampling techniques are bootstrapping and cross validation.

2.4.1. *Bootstrapping*

With bootstrapping one starts with randomly selecting N data points from the training set with replacement, with the test data left untouched. The same data point may be selected several times. After the selection, some value of interest is calculated, let's call it Θ. This may for example be the MSE. The value of Θ is calculated and stored. This constitutes a single bootstrap resample. A new bootstrap data set is drawn, and Θ is calculated and stored again. This process is repeated B times. After all bootstraps have been completed, the mean of Θ over all bootstraps is calculated.

The bootstrap method is not complicated and is beneficial to use when faced with a small sample set. Estimating something like the variance of the mean demands several independent samples, and bootstrapping enables us to calculate estimates from a single population. It does not require assumptions about the distribution (like normal, uniform etc.) Kulesa et al. (2015).

The name bootstrap comes partly from booting, which is what happens after starting a computer, and from the saying "to pull oneself up by one's bootstraps", meaning to try to improve by using already existing resources or capabilities. The straps is often found in the top back of the boot (like you find on Dr. Martens boots). This is a metaphor for how the machine is trying to improve its knowledge of the model on its own by using the already available data.

2.4.2. *Cross Validation*

Cross validation is a more ordered resampling technique, where the idea is to utilize all the training data for both training and for testing. The data set is divided into subsets of equal size, often called folds. We divide the data set into k subsets. $k-1$ subsets are then used for training, and the remaining subset is used for testing. The values of interest are calculated and stored. The process is repeated until all subsets have been used as both training and test sets. With cross validation you don't have to decide which of the subsets to use for testing, since all subsets are used. Cross validation with k subsets is called k-fold cross validation. A common form is 10-fold cross validation. In the extreme case where each subsets only contains one sample, it's called "leave one out" cross validation. Cross validation can for instance also be used to test which λ to use in ridge or lasso regression.

2.5. *Franke's function*

Franke's function is a function of two variables, x and y, and is a weighted sum of 4 exponentials. It has two characteristic Gaussian peaks of different amplitudes and a smaller dip. It is frequently used as a test case for testing interpolation algorithms. It is given by

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

and when plotted it has two Gaussian peaks, ones higher than the other, and a small dip. See figure 4.

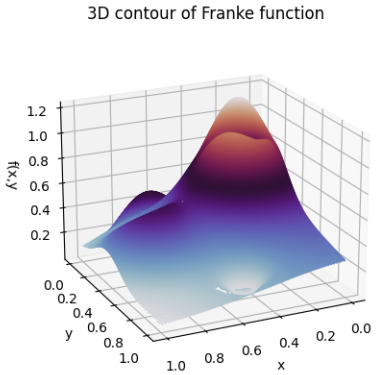

FIG. 4.— 3D contour plot of Franke's function.

3. METHOD

3.1. *Generating controlled data*

The data is generated by drawing two sets of n random numbers, $\mathbf{x}_1, \mathbf{x}_2$, from the uniform distribution in the interval $[0, 1)$. These two sets of data are used as arguments for the Franke function, giving us a data set, $\mathbf{y}_{\text{observed}}$. Stochastic noise is then added by drawing n numbers from the standard normal distribution and adding it to $\mathbf{y}_{\text{observed}}$.

A design matrix $\mathbf{X}$ is created with the $\mathbf{x}_1, \mathbf{x}_2$ data sets for a given polynomial degree. The design matrix has n rows and the number of columns according to the number of features which is given by

$$\text{features} = \frac{(p+1)(p+2)}{2}, \tag{35}$$

where p is the polynomial degree. This way of calculating the number of features is particular for the case of a function of two variables, like the Franke function. Different polynomial degrees are used in this report, ranging from 0 to 20. With the observed data and design matrix in place, we split the data into test and training sets using `sklearn.model_selection.train_test_split`, with a test size of 20%. The data is also shuffled in the splitting process.

3.2. *Fitting*

With the data in place, we find the optimal parameters $\hat{\boldsymbol{\beta}}$ by ordinary least squares, ridge regression or lasso regression, which are described in the previous sections. The resampling techniques bootstrapping and cross validation are also used, as well as no resampling. The experiments are repeated and averaged over several times to wash out any large statistical deviations. Typically 100 repetitions are used for the data presented in this report.

A set of predicted values, $\tilde{\mathbf{y}}$, are calculated with the obtained $\boldsymbol{\beta}$. These values are compared to the test data and evaluated by the means of MSE and the R^2-score. These measurements were plotted for different polynomial degrees to see how this affects the training and test data. The confidence intervals of β_is are also studied for a polynomial fit of degree 5.

3.3. *Data analysis*

We use the resampling technique bootstrapping, with 50 bootstrap resamples, to calculate the MSE, variance and bias for different polynomial degrees. With bootstrapping, we do an analysis of the bias-variance trade-off to see how the bias and variance changes as the polynomial degree is increased.

We do a similar procedure, once again with 50 bootstraps, but now also a 5-fold cross validation for comparison. The MSE from the two resampling techniques are plotted together as a function of the polynomial degree to study how the two different resampling techniques improve our predictions.

The two techniques lasso regression and ridge regression are studied. We look at how changing the lasso and ridge parameters λ affect the MSE, bias and variance of the predictions, and we determine where to find the optimal values of the parameters, if possible.

3.4. *Terrain data*

After model tuning with the constructed data set, we use real-life terrain data to see how our models fare in real life. We get the data from the `SRTM_data_Norway_1.tif` data file.

The data is loaded with `imageio.imread`, storing the data as a 2D array. For simplicity, the terrain data is sliced so that we are left with an equal number of rows and columns in the array. The terrain data is again sliced to include only every N'th data point. The $\mathbf{x}_1, \mathbf{x}_2$ data for the design matrix are orderly drawn in the interval [0, 1], and the design matrix is constructed. The data is split in the same manner as previously and scaled by the standardization procedure. The same techniques are applied to this real-life data set as with the constructed data set.

3.5. *The program*

All the experiments are coded in Python. The code is available at https://github.uio.no/jonkd/FYS-STK4155_H20/tree/master/projects/project1. The code consists of a main file `common.py`, which contains all the functions common for most of the experiments. A class `Regression` is used, since certain procedures like creating the random data, generating the design matrix, and scaling the data, is common for the different types of regression. Each different experiment and its needed parameters for generating the data and plots are separated in individual files, named `part_x.py` for x = a, b, ..., g.

The resampling techniques bootstrapping and cross validation are coded from scratch in `common.py`, as well as the algorithms for ordinary least squares and ridge regression, with the help of common functionality from `numpy`, including matrix pseudo-inversion `numpy.linalg.pinv`. The code for lasso regression is imported from `sklearn` and not coded from scratch.

4. RESULTS AND DISCUSSION

4.1. *Mean squared error*

MSE and R2 are two essential statistical parameters to evaluate the accuracy of a regression model. We have calculated both of these two quantities for different polynomial degrees. MSE and R2 have been determined for training and test data individually.

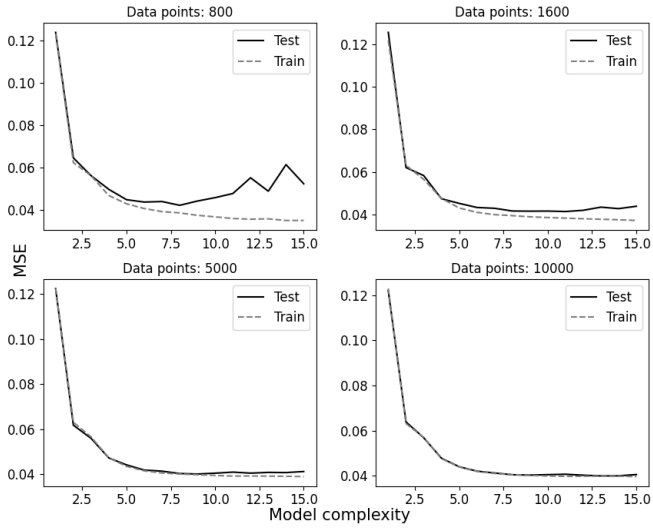

FIG. 5.— The mean squared error as a function of polynomial degree. From top left, we have 800, 1600, 5000, and 10000 data points of which 80% are used for training. From the plots it is clear that many data points are needed to create a model which predicts well with a high degree polynomial. This is constructed data, created with Frankes function.

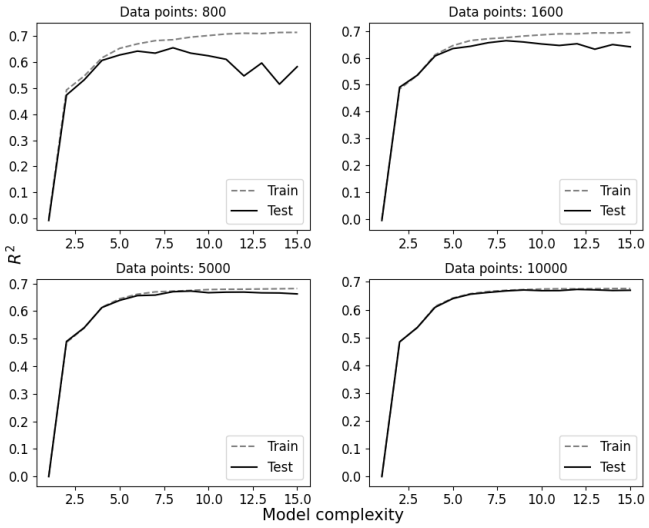

FIG. 6.— R^2 calculated for training and test data sets. From top left, we have 800, 1600, 5000, and 10000 data points of which 80% are used for training. The data shows clearly that a high degree polynomial fit requires a sufficient amount of data points to function properly.

As shown in figure 5, MSE for training and testing data varies with model complexity. There is an explicit dependency on the number of data points that we can identify from the plots. We have calculated MSE for 800, 1600, 5000 and 10000 data points, where 80 percent of data was used to train the model. For all the different amounts of data points, MSE of the training data gets lower as the polynomial degree gets higher. The test data follows suit, but only to a certain polynomial degree, dependent on the number of data points. All polynomials above the point where the MSE is lowest for the test data is subject to overfitting.

From this we see that a complex model can formulate an accurate relation between the quantities for the training data. At the lower polynomial degree, there is a strong agreement of MSE obtained using test and train data for all different numbers of data points. As the number of data points increase, the agreement between training and testing data improves for all polynomial degrees, until at 10000 data points, the agreement is almost perfect relative to the other amounts of data points. In contrast, for a small data set, MSE for testing and training data quickly differs when the polynomial degree increases. Thus, a complex model based on a small data set can not predict test set outcomes with significant accuracy. Lower polynomial models are suitable for a limited data set, and an efficient complex model can be developed when a sufficient amount of data are available.

4.2. *R squared*

R-squared (R^2, R score) is a statistical measure of the accuracy of a prediction model, and is displayed in figure 6. Since R^2 is a scaled value, as seen in equation (2), it can easily be compared for different prediction models. From the value of R^2, we get a quantitative measure of a models ability to replicate the observed data. R^2 is calculated for the same sets of data as we used for MSE in figure 5. In figure 6, we can see more or less the same trend as with the MSE. For fewer data points, overfitting is reflected by the decrease in R^2 for the test data when the polynomial degree rises. This trend is less prominent as we increase the number of data points, and at 10000 data points there is almost no visible deviation. This is in good agreement with the trend of the MSE value.

4.3. *Confidence interval*

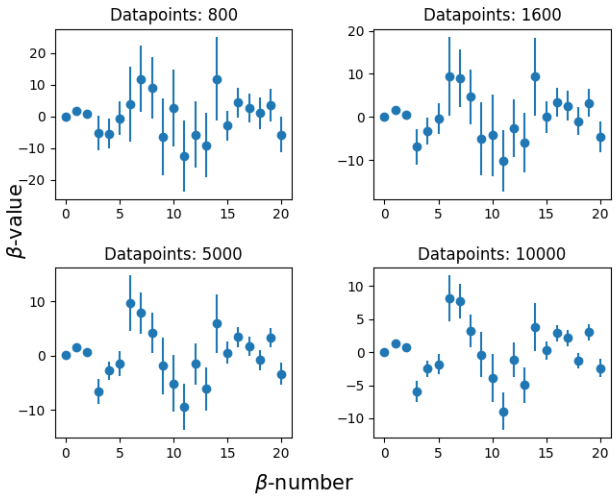

FIG. 7.— Confidence interval for the different parameter values β_i for a 5th degree polynomial using ordinary least squares with no resampling. The plots show 800, 1600, 5000, and 10000 data points. A clear trend is seen, that the uncertainty for each parameter value decreases as the number of data points increases.

We have determined the confidence interval for each β_i for several different number of data points, as given in figure 7. The intervals for different β_is are not equal; however, all of these intervals decrease with increasing data points. We can improve the precision of model parameters significantly by having more data. The inconsistent precision of β_is is an open question to us.

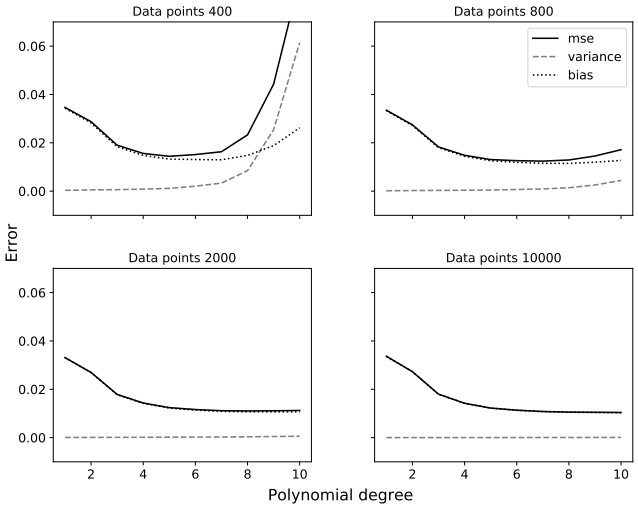

FIG. 8.— The mean squared error (MSE), variance and bias for the test data plotted as a function of polynomial degree for different numbers of data points. For low complexity the MSE and bias is high and the variance is low. For high complexity the variance rises. For high complexity the effect of overfitting is apparent for the lower amounts of data points.

4.4. *Bias-variance tradeoff*

From the implementation of the bootstrap resampling method, with ordinary least squares, we have analysed the bias-variance trade-off. In figure 8 the mean squared error, variance and bias is plotted as a function of polynomial degree for 400, 800, 2000 and 10 000 data points respectively. The polynomial degree where the MSE starts to rise after its minimum deviates further to the right for increasing number of data points. For 10 000 data points, we see no rise in MSE, but we expect to see the rise if we were to use even higher order polynomials. For all numbers of data points the variance is low for low complexity and starts to rise for higher complexity. The bias follows the MSE rather closely, but rises less than MSE for higher degrees, except for 10 000 data point where the MSE and bias can't be distinguished. The complexity sweet spot is where the variance is still quite low and the bias has fallen to an acceptable level. From the plots we would suggest somewhere around polynomial degree 5 or 6 for the lower amounts of data points, while it is safer to choose a larger polynomial degree when there are more data points.

4.5. *Resampling techniques and regression methods*

4.5.1. *OLS*

We compare the two resampling techniques bootstrap and cross validation in figure 9. The mean squared error from both resampling techniques are shown as a function of polynomial degree, with different number of data points and k-folds as indicated in the title of the plots. The number of folds had surprisingly little effect on the result. Increasing the number of folds from 5 to 10 does little to improve the MSE. We see that bootstrapping consistently performs worse than cross validation in all cases, but bootstrap MSE approaches cross validation MSE when the number of data points is increased. The difference between the two resampling methods is most apparent for high polynomial degree, where it seems that cross validation helps to prevent overfitting. For 400 data

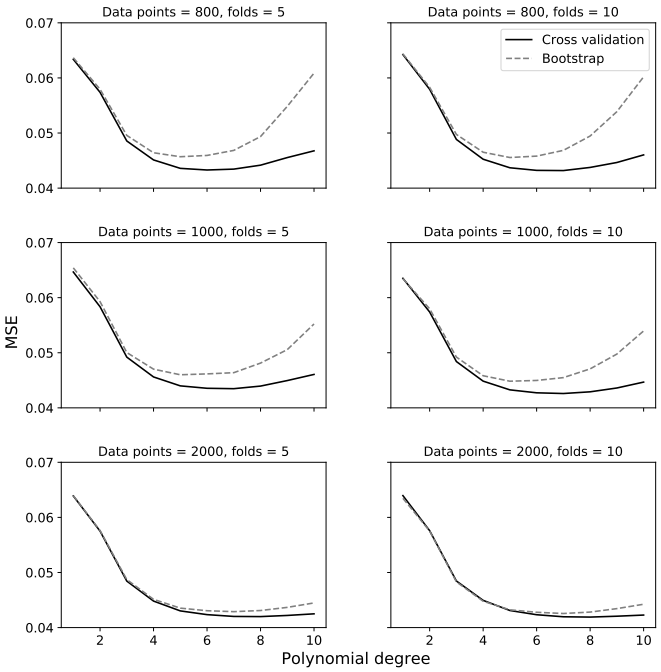

FIG. 9.— Comparing the two resampling techniques bootstrapping and cross validation. The plots show cross validation and bootstrapping MSE for an increasing number of data points for 5 and 10 cross validation folds. Bootstrapping consistently performs worse than cross validation, but the two coincide for low polynomial degrees. As the polynomial degree gets larger, bootstrap MSE approaches that of cross validation. Increasing the number of folds from 5 to 10 has very little impact on the results.

points and cross validation the lowest MSE = 0.04568 for degree 5 and with bootstrap the lowest MSE = 0.04990 also for degree 5.

4.5.2. *Ridge regression*

In ridge regression, a quadratic penalty term is added to the OLS cost function to modify the model's parameters. Correlated predictors and an insufficient number of data impose a limitation on OLS method. The inclusion of this extra term provides us an opportunity to overcome these limitations. The penalty parameters (λs) can have any value from zero to infinity. In general, it remains within 0 to 1. If λ is large, corresponding β approaches zero. Ridge regression allows us to penalize the coefficient at different magnitude. As a result of the quadratic penalty term, larger βs shrink more than smaller ones.

In figure 10 we demonstrate the influence of complexity and penalty parameter on bias and variance using the data set obtained from Franke's function. As the model complexity increases, the bias gets larger for all λs, because the training data can be fitted accurately with a higher degree polynomial. In contrast, the opposite trend is observed for the variance at all penalty parameters. We do not observe a prominent influence of the penalty parameters on the bias and the variance, particularly at lower polynomial degrees. A very small change in bias and variance occurred with the variation of λ up to the fourth degree polynomial model, and at polynomial degrees higher than four, we notice a more significant change in variance with the penalty parameter. Within the range of polynomial degrees between

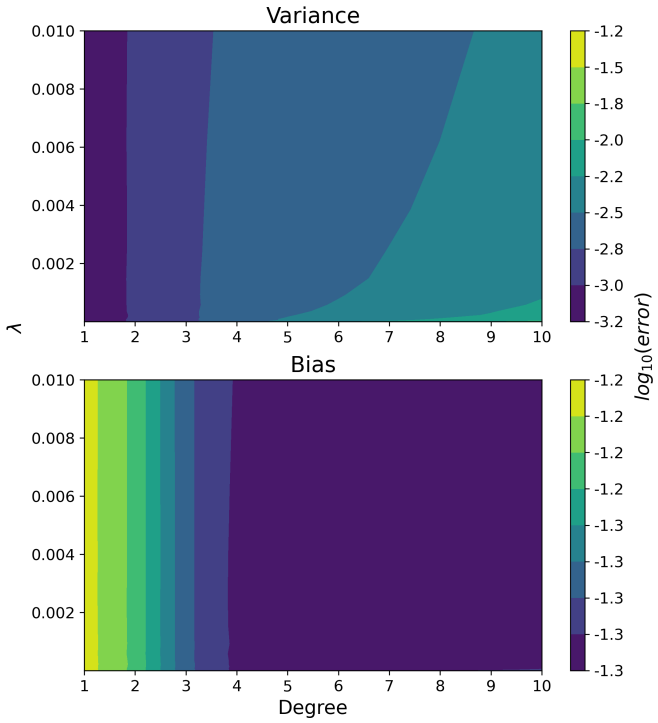

FIG. 10.— The variance and bias plotted with different polynomial degrees on the x-axis and different λ-values on the y-axis. The color scale is logarithmic. The trend is that the variance is increasing for increasing complexity and lower λs. While the bias is decreasing by increasing complexity and for lower λs. The dataset is of 800 data points.

5 to 10, we observe a downshift in the variance with λ. The variation in the bias between 4 to 10 degrees of the polynomial is not significant. A larger λ causes increment of the bias; however, the simultaneous reduction in the variance makes the model accurate. Overall, we can not observe any advantage of using a larger penalty parameter at polynomial degrees below 4, since bias and variance remaines stable. This trend continues for larger values of λ outside the region of figure 10, but is not included in the plots in this report. As λ approaches zero, ridge regression provides the same values for the parameters as OLS since $\lambda = 0$ corresponds to OLS.

The benefit of using ridge regression over OLS is not present in this specific data set for polynomial degrees below 4. We do observe smaller variances for larger values of λ at polynomial degrees larger than 4, but see no corresponding increase in bias at the same parameter values. For the variance to be lowered and the bias to remain unchanged, MSE must decrease. An optimum value of λ can be determined using MSE values. We identify similarities between the bias-variance tradeoff obtained from ridge regression i figure 10 and OLS in figure 8. In ridge regression, variance increases with the model's complexity, like what we see with OLS. We observe a drop in bias at polynomial degrees up to 4, similar to what we observe for OLS. For polynomial degrees larger than 4, we se no change in the bias, just like what we observe for OLS for 800, 2000, and 10000 data points.

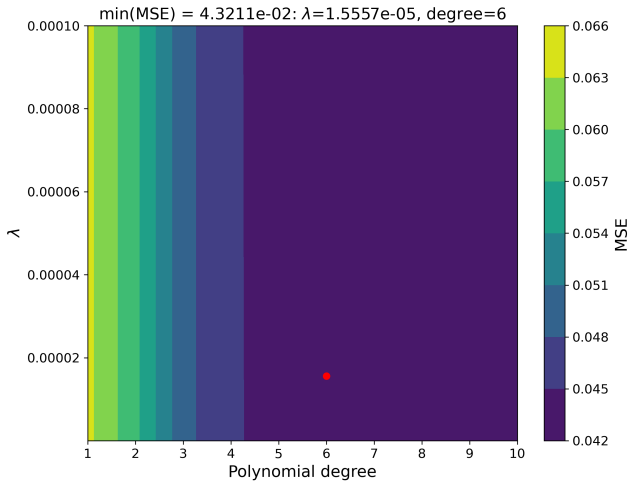

FIG. 11.— With the ridge regression method and cross validation resampling: The mean squared error plotted for different polynomial degrees on the x-axis and different λs on the y-axis. Red dot is the least MSE.

We analyze the MSE error in ridge regression, and the result is compiled in figure 11. It demonstrates a firm agreement with the bias-variance tradeoff data for OLS in figure 9 with 800 data points. In both OLS and ridge regression, a polynomial of degree around 6 produces the least MSE. We noticed a change in MSE with the penalty parameter. For the polynomial of degree 5 to 7, the least MSE is produced at lower value of λ. The value of lambda corresponding to the lowest value of MSE of $4.321 \cdot 10^{-02}$ is $1.556 \cdot 10^{-05}$. The red dot in figure 11 represents this point. It was produced for 6th degree polynomial. Bias-variance tradeoff analysis using CV and Bootstrap method (figure 9) for OLS also found the minimum MSE for a 6-degree polynomial model. Using suitable non-zero λ, we achieved a smaller MSE with ridge regression. This non-zero value of lambda indicates the effectiveness of ridge regression over OLS in model prediction. See also table 1 for a comparison of the MSE for the different regression techniques.

4.5.3. *Lasso regression*

We fit the data with lasso regression and evaluate the MSE with cross validation and bootstrap. The results from cross validation are shown in figure 12 where we use 400 data points. The main tendency is that lower λ and higher degree gives lower MSE. The least error (MSE=0.047206) is obtained with a polynomial degree of 6 and a small $\lambda = 1.00 \cdot 10^{-12}$ which is the lowest λ we tried, and is very close to zero. We suspect that if we ran the program with even smaller λ or $\lambda = 0$, that would give an even lower MSE. This means we don't achieve any reduction in MSE with this method compared to simple OLS. Some care must be shown in interpreting these results since the implemented lasso method, `sklearn.linear_model.Lasso` is numerically unstable at $\lambda = 0$ and possibly for values close to zero [3].

To get a better view of how the MSE varies with λ, we plotted the graph for a good polynomial degree of 5

[3] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html

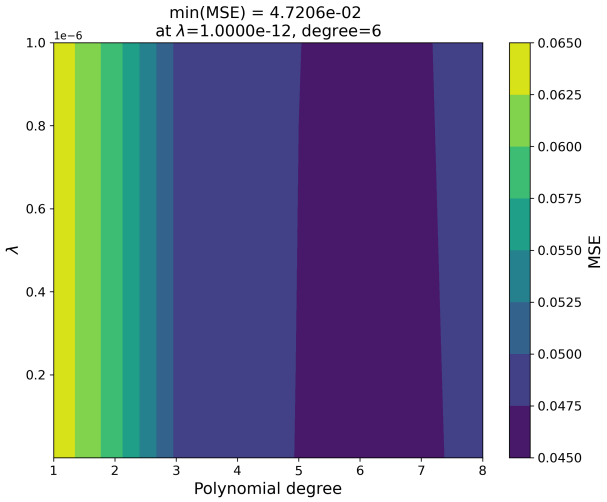

FIG. 12.— With the lasso regression method and 5-fold cross validation resampling, 400 data points: Mean squared error plotted for different polynomial degrees and λ-values.

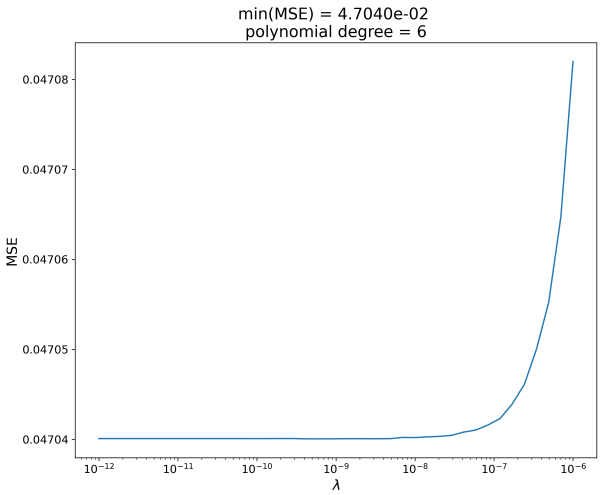

FIG. 13.— Lasso regression with 5-fold cross validation with 400 data points and degree 6. MSE plotted against λ-values (logarithmic scale). Degree 6 gave the lowest MSE with this method.

for 400 points, figure 13. This shows the MSE is lower for smaller λ. This plots states that the lowest MSE = 0.04704, somewhat larger than that from figure 12. This may be due to the different number of data points and statistical randomness.

We also evaluate the MSE with bootstrap resampling and found the same main trend, that MSE decreases with higher polynomial degrees and lower λ. This can best be seen in figure 14 where we ran the program with a wider range of λ-values, but also in figure 15, where λ ranges from 10^{-12} to 10^{-6}. The lowest MSE = 0.0503 is achieved with a degree 6 and $\lambda = 3.2 \cdot 10^{-8}$. This is the same degree we found from ridge regression, but the λ is a tiny bit higher then the lowest value we used. The lowest MSE is a bit higher (0.0503 versus 0.0472) than what were achieved by cross validation.

Considering the figures 16 and 17 the variance seems to be lowest for lower complexity and higher λ, while with the bias it's the opposite: Lower bias for higher complexity and lower λ. So considering both variance and bias the complexity and the λ should be chosen to be somewhere in the middle. This fits well with the optimal degree of 6 that we found. It's not easy to see from these plots, but when we ran with a wider range of λs, we saw a tendency of increasing bias and decreasing variance with increasing λ.

TABLE 1
LEAST MSE, POLYNOMIAL DEGREE FOR 400 DATA POINTS

	OLS	Ridge	Lasso
CV	0.04568, 5	0.04321, 6	0.04721, 6
Bootstrap	0.04990, 5	0.04816, 5	0.05039, 6

When we study table 1 (MSE, polynomial degree) we see that cross validation performs superior to bootstrap in all cases. All in all, when comparing the three regression methods OLS, ridge, and lasso regression, the lowest MSE is obtained with ridge regression and with cross validation resampling technique for a 6th degree fit.

4.6. *Terrain data*

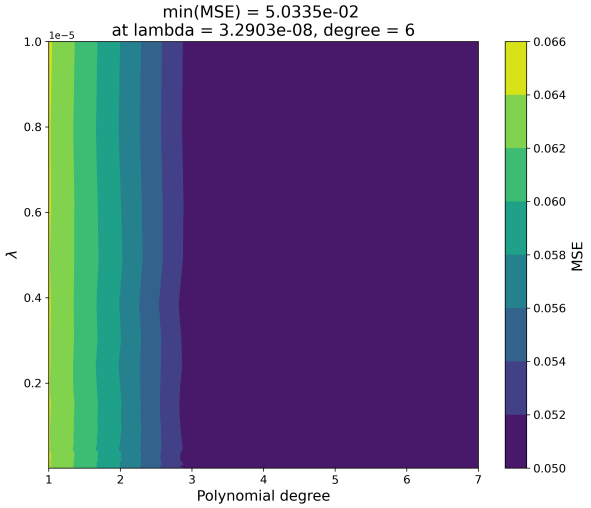

FIG. 14.— Lasso regression with 400 data points and bootstrap resampling method (B=50). The contour plot shows MSE for different values of polynomial degree on the x-axis and λ-values on the y-axis. $\lambda \in [10^{-12}, 10^{-6}]$.

The same analysis as we did for the constructed data is now performed on the terrain data. In figure 18 we see the MSE as a function of polynomial degree. We clearly see the same trend here as with the constructed data set, that a higher polynomial degree always gives a better training fit, and that a higher polynomial degree only gives a better test fit if there are enough data points. The MSE values have a large magnitude, but cannot be interpreted as good or bad without knowing the scale of the numbers being fitted. For that we can consider R^2 which is displayed in figure 19. Due to the value being scaled, see equation (2), it is easily comparable to the constructed data set we presented in section 4.2.

As expected, we see the same trend for R^2 as we see for the MSE. More data gives better results with larger polynomial degrees. The value of R^2 though, is lower for the terrain data set than for the constructed data set in

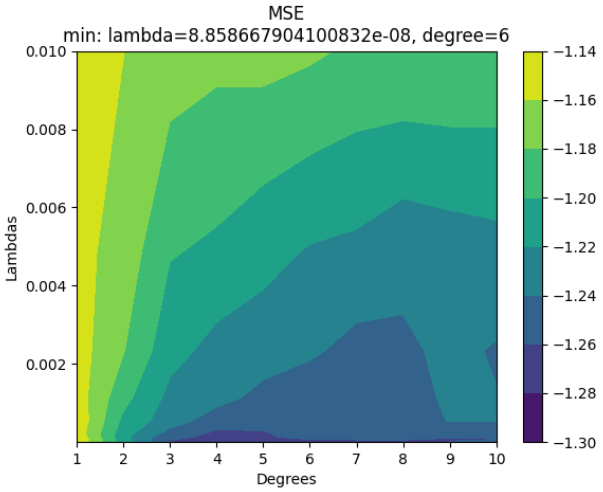

FIG. 15.— Lasso regression with 400 data points and bootstrap resampling method (B=50). The contour plot shows MSE for different values of polynomial degree on the x-axis and λ-values on the y-axis. $\lambda \in [10^{-8}, 10^{-2}]$.

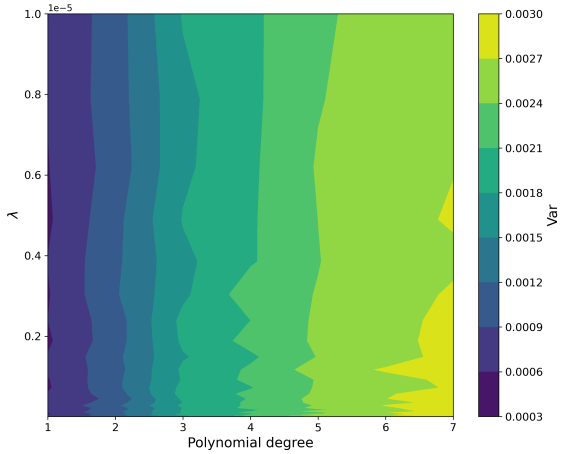

FIG. 16.— Lasso regression with 400 data points and bootstrap resampling method (B=50). The contour plot shows variance plottet with different polynomial degree on the x-axis and λs on the y-axis.

figure 6. It seems that it is more difficult to find a fit to real-life data than constructed data.

Figure 20 shows the MSE for the terrain data as a function of λ_{ridge}, the ridge regression parameter, explained in section 2.2.2. Note that the x-axis is logarithmic. Values λ_{ridge} from 10^{-16} to 10^{-7} has little impact, but in fact, we see a lowering of the MSE for values λ_{ridge} in the vicinity 10^{-2} to 10^{1}. The training MSE is consistently lower than the test MSE as before, but they both join and shoot to the stars at λ_{ridge} above 10^{1}.

For the lasso regression parameter, we are not so lucky. Figure 21 shows the MSE as a function of λ_{lasso} for $\lambda_{\text{lasso}} \in [10^{-16}, 10^{1}]$. We only see statistical noise, and no improvement of the MSE by varying λ_{lasso}. At the end, we see the same trend as with ridge regression, that the MSE increases rapidly above $\lambda_{\text{lasso}} = 10^{1}$.

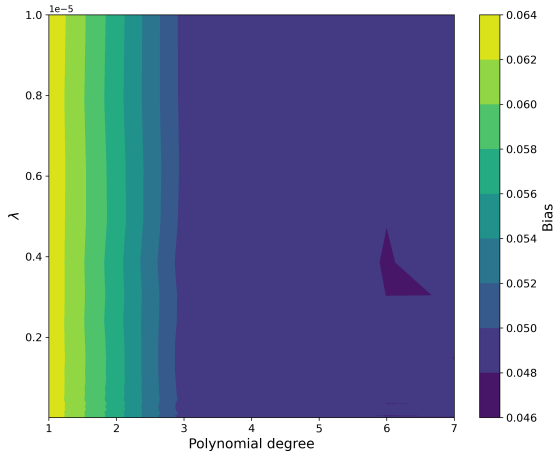

FIG. 17.— Lasso regression with 400 data points and bootstrap resampling method. The contour plot shows bias plotted with different polynomial degree on the x-axis and λs on the y-axis. The variance and bias shows an opposite tendency. The variance increases with lower λ, while bias decreases. How they vary

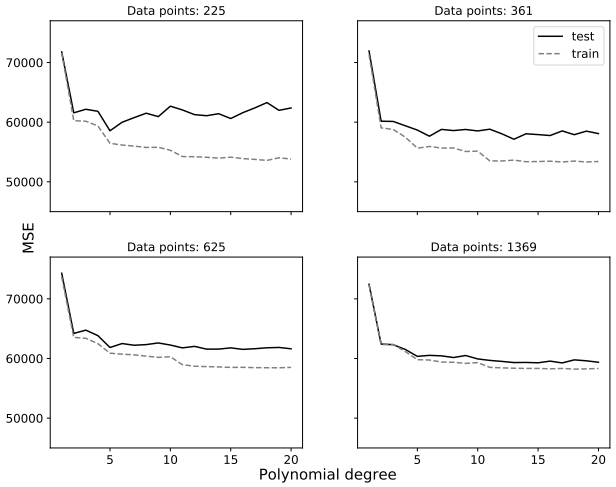

FIG. 18.— Model trained on the terrain data using ordinary least squares and no resampling. Four different amounts of data points have been chosen. We can see the same trend as with the constructed data, namely that the model shows overfitting for larger polynomial degrees, but that this trend is less prominent when there are lots of data.

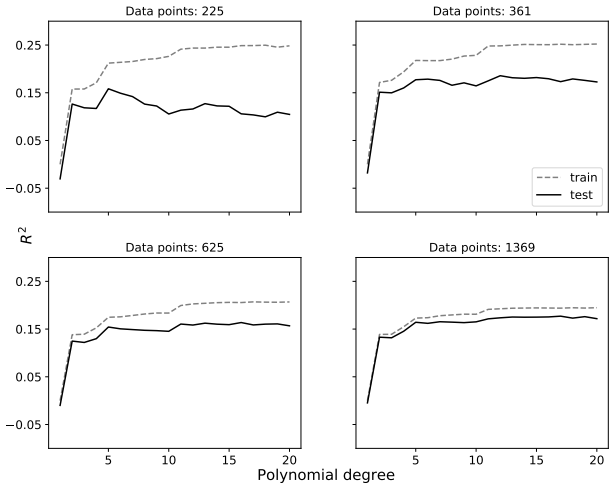

FIG. 19.— R^2 as a function of polynomial degree. Model trained on the terrain data using ordinary least squares and no resampling. Four different amounts of data points have been chosen.

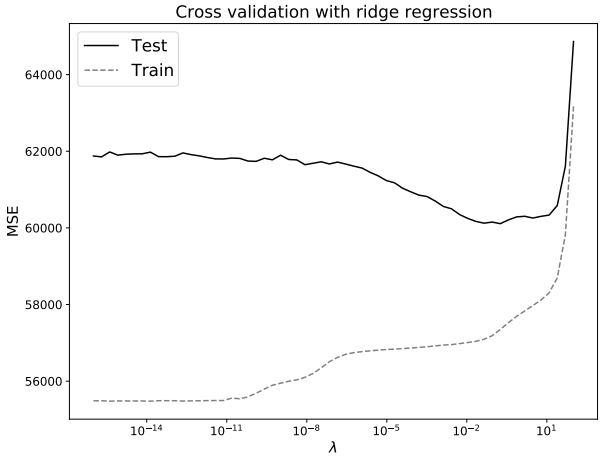

FIG. 20.— MSE as a function of λ_{ridge}, the ridge regression parameter. Resampling with cross validation. The MSE is unchanging for low λ_{ridge} until approximately $\lambda_{\text{ridge}} = 10^{-5}$ where the MSE starts to lower. A minimum MSE is found at approximately $\lambda_{\text{ridge}} = 10^{-1}$.

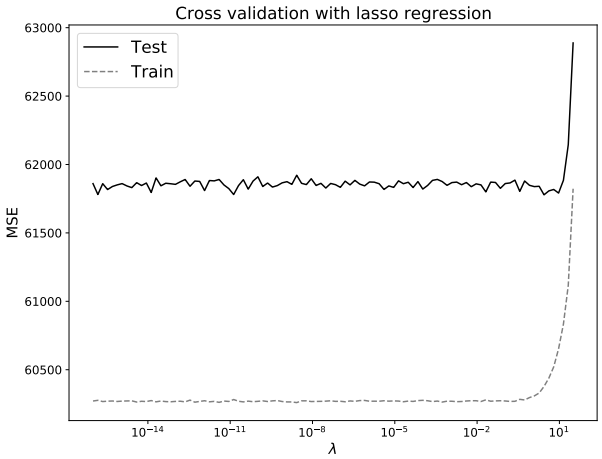

FIG. 21.— MSE as a function of λ_{lasso}, the lasso regression parameter. No noticeable difference in MSE is seen, save for statistical changes. No lowest λ_{lasso} can be determined.

5. CONCLUSION

With both manufactured data from the Franke's function and real terrain data we studied the regression methods OLS, ridge and lasso and the two common resampling techniques bootstrap and cross validation with the statistical evaluation tools mean squared error, R^2-score, variance and bias. We studied the bias-variance tradeoff with all models and learned that increasing the complexity lowers the bias and variance for the training data, but increases the variance for the test data. This phenomenon is known as overfitting. The best polynomial degree was found to be around 5-6. Increasing the numbers of data points had a significant positive effect on all error-estimates. For the Franke function data we found an overall advantage in using cross validation over bootstrap resampling. Adding a lasso penalty to the model did not improve the error, but the ridge method had a small superiority over the OLS. The least MSE of 0.043211 were obtained with ridge regression with a polynomial degree of 6 and small penalty parameter of $1.5557 \cdot 10^{-05}$. The effect of raising the fold in cross validation from 5 to 10 had very little effect. A closer look at how the number of folds in cross validation changes the error is something to consider for a later study. The real terrain data displayed the same behavior. Cross validation was superior to the bootstrap method in all cases. The ridge method has a clear advantage on the error over simple OLS, while the lasso method had no gain. The optimal ridge penalty parameter was in the order of 0.01 to 10.

All code used to generate data for this report is available at the GitHub repository https://github.uio.no/jonkd/FYS-STK4155_H20/tree/master/projects/project1.

REFERENCES

Hjorth-Jensen, M. 2020, Overview of course material: Data Analysis and Machine Learning (weekly schedule may be revised). https://compphysics.github.io/MachineLearning/doc/web/course.html

James, G., Witten, D., Hastie, T., & Tibshirani, R. 2017, An Introduction to Statistical Learning: With Applications in R, 1st edn. (Springer New York)

Kulesa, A., Krzywinski, M., Blainey, P., & Altman, N. 2015, Nature Methods, 12, 477. https://www.nature.com/articles/nmeth.3414#citeas