An introductory study of regression methods with machine learning and its applications

Elin Finstad & Anders Bråte
Institue for Physics
University of Oslo
Norway
October 10, 2020

Collecting data in order to understand a subject is a practice as old as time, but few, incomplete or inaccurate datapoints force us to a certain degree of approximation. This is where the concept of regression using machine learning techniques comes in. Resampling data using the bootstrap and cross validation methods, as well as regression using Ordinary Least Squares, Ridge regression and Lasso regression make us able to approximate function to fit data that is otherwise incomplete. We first look at the functionality of these resampling and regression methods on the infamous Franke's function, then we apply the methods on actual topographic data. We found that whilst being a conceptually more complicated method, the Cross validation resampling algorithm was significantly faster, and proved more reliable for complex systems. Ridge regression showed better results for noisy data, in accordance with theory. Cross validation and OLS found to produce the best model when applied to terrain data, giving a model with MSE of 0.0358.

I. INTRODUCTION

Little can be said to overemphasize the importance of machine learning in recent times. As data collection has become its own major business, interpreting said data is just as important, and this is where machine learning comes in. The essential element of machine learning is finding patterns in data which by the naked eye would be impossible to find, or where limited data makes finding underlying patterns equally difficult. As incomplete data very often is the case, we must rely on resampling techniques, which help us extract the most out of the data. The bootstrapping and cross validation methods both have advantages and disadvantages which we will explore. In addition, Ordinary Least Squares (OLS), Ridge regression and Lasso regression all have their own pros and cons which we will delve deeper into.

The regression methods we initially apply to a known function known as the Franke's function, which is popular for testing regression methods, with added noise. Looking at parameters such as Mean square error, R2 score and confidence interval, with and without resampling methods we explore the accuracy and idiosyncrasies of the different methods. In doing this, we can get a clear picture of just how the different methods react to variables that we often don't know anything about.
After exploring the different methods of regression and resampling we apply these methods to real life topographical data, to investigate the real life application of these methods.

II. THEORY

Train, test and validation data

The crux of machine learning is essentially applying regression methods on a dataset, training the method if you will, which gives us an approximation to the underlying functions, and then testing it to see how well it will apply to data that hasn't been used in the training. This naturally brings us to splitting the data in training and test data. It is of utmost importance that we don't get these two mixed up, such that information from the test dataset affects the train data! If we for example scale the whole dataset with respect to the train data, information about the train data will influence the test data, and we won't get an accurate result for the validity of our model.

If we tune our model to perfection on the dataset, so that it perfectly fits the training data, it would be too effected by the specifics of the training dataset it would no longer accurately predict a general case, and as one would find, it would perform very poorly on any test data. This is a central theme in Machine learning, and is called overfitting. To combat this, some methods use intermediate testing', to see if the model has become a victim of this. These datasets are usually referred to as a validation set.

Resampling methods

As all machine learning is based on algorithms learning from data it is always in our best interest to have the most amount of data to train, test and validate our algorithms on. Large datasets however is unfortunately often a luxury, and getting the most out of the data we have is paramount. By using resampling methods we are effectively able to extract more information out of our data than would initially be feasible.

Another advantage to certain resampling methods is the ability to determine the error in an algorithm. By drawing different samples from a dataset we can see how much the, for example, linear fit of a dataset varies across the samples. This we can use to gauge to what degree the fits are consistent.

Bootstrap method

Bootstrapping is in essence resampling of data with replacement, and looking at the characteristics of these new samples. This resampling with replacement maintains the data structure, but reshuffles values, meaning we can find values such as confidence intervals, R squared and variance in a dataset which initially couldn't tell us these things.

One danger in relying on this method is that certain parameters are inherently biased. This means in practice that certain extreme values that seldom occurs in your dataset has a tendency to be underrepresented during bootstrapping. This may lead to values such as the standard deviation and variance being underestimated.

It can be shown that the bootstrap method reduces the bias of $\tilde{\boldsymbol{z}}$. By definition, the bias of $\tilde{\boldsymbol{z}}$ relative to z is given by

$$bias(\tilde{\boldsymbol{z}}) = \boldsymbol{z} - \mathbb{E}[\tilde{\boldsymbol{z}}].$$

As the number of resamples in the bootstrap method increases, the expectation value of $\tilde{\boldsymbol{z}}$ gets closer to true value z. Hence, the bias is reduced.

Cross-validation

Cross validation differs from the bootstrap method in many aspects. The biggest difference however is the fact that it loops over different complexities in the model, and averages the error from this model applied on different validation data to find the degree which gives the best fit. In essence, if done correctly, cross validation will ensure minimal overfitting.
Since the bootstrap method is based on random samplings of data, it may lead to unwanted effects of certain values having a bigger influence on the resulting model, for example in the way mentioned above. To avoid this, cross-validation uses structured resampling, as explained below

Initially we split the dataset into train and test data, as with all machine learning. The training data is then split in k folds, where each fold takes its turn being the validation set to test the model while the remaining $k-1$ folds have the role as training set to build the model. When each fold has been the validation set exactly once, the k results can be average to produce a single estimate. This is then done multiple times, whilst increasing the complexity of the model in order to see which degree is best for fitting the data.
conclusively the whole training dataset is used to make a model with the polynomial complexity that was found to produce the smallest error. This model is then tested on the initial test data, which until now has not been used.
The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation once. dividing the dataset into n number of folds, each one takes its turn acting as the validation set, whilst the remaining $n-1$ folds form a training set.

Regression methods

Regression analysis let you examine the relationship between two or more random variables. The goal is to explain the response variable y_i in terms of a design matrix $\mathbf{X}$ through a functional relationship like $z_i = f(\mathbf{X}_{i,*})$. When there is no prior knowledge to this functional relationship, it is normal to assume linearity. The linear regression model is given by

$$\boldsymbol{z} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{1}$$

where $\boldsymbol{X}$ is a design matrix, $\boldsymbol{\beta}$ are regression parameters and $\boldsymbol{\epsilon}$ is the error of the model. The left hand side of the equation and the design matrix are known, while the parameter vector and the error are unknown. In our case, we are dealing with two variables, hence, the design matrix is given as follows

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_0^1 & y_0^1 & x_0^2 & x_0y_0 & y_0^2 & \dots & x_0^p \\ 1 & x_1^1 & y_1^1 & x_1^2 & x_1y_1 & y_1^2 & \dots & x_1^p \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1}^1 & y_{n-1}^1 & x_{n-1}^2 & x_{n-1}y_{n-1} & y_{n-1}^2 & \dots & x_{n-1}^p \end{bmatrix}$$

The total number of combinations of x and y is $m = \frac{(p+1)(p+2)}{2}$, where p is the model complexity. Hence, $\boldsymbol{X} \in \mathbb{R}^{n\times m}$.

The aim of the regression methods is to obtain the optimal β_i values. We define the approximation $\tilde{\boldsymbol{z}}$ via

the unknown quantity $\boldsymbol{\beta}$ as

$$\tilde{\boldsymbol{z}} = \boldsymbol{X}\boldsymbol{\beta} \tag{2}$$

Ordinary least square

One of the most used estimation models is the ordinary least squares (OLS). This model minimizes the sum of the squares of the difference between the dependent variables in the given dataset and the predicted values from the linear model (eq. 2). To find the optimal β_i, define a so called cost function based on the least squares and minimize this. The cost function for OLS is as follows

$$\begin{aligned} C(\boldsymbol{X}, \boldsymbol{\beta}) &= \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 \\ &= \frac{1}{n}\left\{(\boldsymbol{z} - \tilde{\boldsymbol{z}})^T(\boldsymbol{z} - \tilde{\boldsymbol{z}})\right\} \\ &= \frac{1}{n}\left\{(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})\right\} \end{aligned}$$

The minimum can be found by setting the derivative with respect to β equal to zero, and we get

$$\boldsymbol{\beta}^{OLS} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{z} \tag{3}$$

[1, p. 12]. Since $\boldsymbol{X} \in \mathbb{R}^{n\times m}$, $\boldsymbol{X^T X} \in \mathbb{R}^{m\times m}$. Normally $m << n$, resulting in inversion of a low dimensional matrix, a computational advantage.

Ridge regression

The ordinary least squares method requires inversion of $\boldsymbol{X^T X}$, but for high-dimensional $\boldsymbol{X}$, this is often singular or near singular, meaning it is non-invertible. A matrix is non-invertible if the determinant of the matrix equals zero. A 'cheap fix', yet a very common one is simply adding a very small element $\lambda \geq 0$, a ridge parameter, to the diagonal of $\boldsymbol{X^T X}$,

$$\boldsymbol{X}^T\boldsymbol{X} \to \boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}.$$

By adding a number to the diagonal, the determinant is forced to not equal zero. Thus the ridge estimator is given by

$$\boldsymbol{\beta}^{Ridge} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{z}. \tag{4}$$

λ is regularization parameter which shrinks the regression coefficients β_i. Larger λ gives larger shrinkage. This is more clear from the cost function, which can be defined by

$$C(\boldsymbol{X}, \boldsymbol{\beta}; \lambda) = \frac{1}{n}\left\{(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{z} - \boldsymbol{X}\boldsymbol{\beta})\right\} + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{5}$$

In practice, we find that Ridge Regression is simply a modified OLS, where the λ factor acts as a 'shrinking' factor, which reduces the coefficients of the polynomials and leads to less variance in the fitting. The reason this is important is in using data which has a lot of noice, or a large variance. The OLS method might have a tendency towards overfitting, compared to the modified Ridge method.

Lasso regression

Similar to the ridge method, lasso (least absolute shrinkage and selection operator) is also a shrinkage method, but here the L_2 ridge penalty $\boldsymbol{\beta}^T\boldsymbol{\beta}$ is replaced by the L_1 lasso penalty $\sqrt{\boldsymbol{\beta}^T\boldsymbol{\beta}}$. Hence, the cost function is given by

$$C(\boldsymbol{X}, \boldsymbol{\beta}; \lambda) = (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{z})^T(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{z}) + \lambda\sqrt{\boldsymbol{\beta}^T\boldsymbol{\beta}} \tag{6}$$

This minimization problem has no closed form solution, thus a numerical approach is required. [1, p. 68]

Bias-variance trade-off

The mean squared error can be rewritten as

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \sigma^2. \tag{7}$$

where the first term is the bias squared of the function values, the second term is the variance of the prediction values and the last term is the variance of the error $\boldsymbol{\epsilon}$. Complete calculations can be found in Appendix A.

The variance can to a certain degree be thought of as how much the data varies from the assumed underlying function. A large amount of noise will produce data that has a large variance.

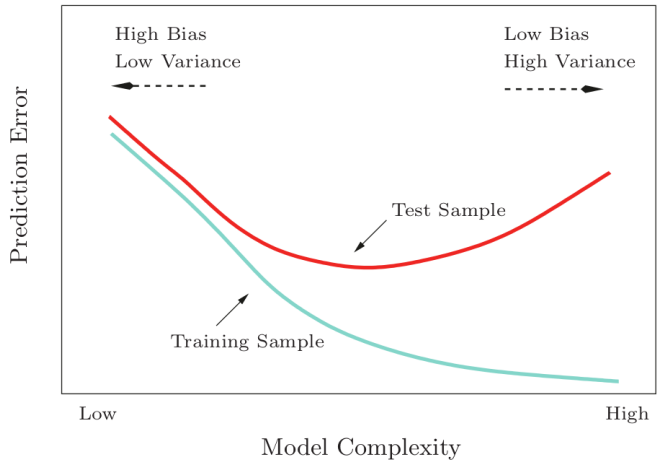

Figure 1. Illustration of overfitting. [1, p. 38, fig 2.11]. As the complexity of the model increases, the model is in an increasing capacity being too well fitted to the training data, mening the variance increases and bias decreases. The optimal degree of complexity is then the lowest error for the test sample, and it is this error the Cross validation method looks for.

As our model gets more complex, the training data tends to adapt to more complicated structures in the data. This causes the model to get less bias, but the variance increases for the test data. Hence, the test error will reach a minimum after a certain amount of complexity before it increases again. When the test error starts increasing, we have reached what we call overfitting. The training error decreases as the model is more and more fitted to the complicated structures in the trainingset, but the test error increases as test data does not have the same complicated structures. This is illustrated in Figure 1.

The bias of an estimator is the difference between the expected value of the estimator and the true value, i.e. it tells how good the estimator is at estimating the true value. If the estimator has zero bias it is called unbiased.

Scaling

Scaling data is vital to consider in all machine learning. When scaling one removes the danger of having a few extreme values skew the whole model, as well as making the model more general in use on other datasets.

when working with datasets that have a large amount of noise, which is often unavoidable when working with practical data derived from instruments with inherent inaccuracies and interference, it might be necessary and indeed vital to scale the data.

In addition, thought not relevant for the methods applied in this paper, certain machine learning methods require that datasets look like standard gaussian distributed data. In addition certain methods assume that the data is centered around 0, and that it has a variance in a similar magnitude.

Error analysis

Mean squared error

Mean square error is a statistical measurement which takes the average of the squares of the error. This is fairly trivial to compute, yet it is a very important tool in exploring different models. we define it as such;

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(z_i - \hat{z}_i)^2$$

where z are the actual values, and $\hat{z}$ are the approximated values for the same point.

R^2 score

The R^2 score normally lies between 0 and 1, but can be negative if the fit is very bad. A R^2 score equal to 1 means there is no variance. Mathematically R^2 score is defined as

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \hat{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z})^2},$$

where $\bar{z}$ is the mean of the actual values.

III. METHOD

Be aware that all code is shown in the github repository https://github.com/elinfi/FYS-STK4155/tree/master/Project1.

scaling

In our implementation we took the liberty of using scikit learn's sklearn.preprocessing.StandardScaler, which subtracts the mean and divides by the standard deviation.
We made sure to not scale any training or validation data so that is was affected by the training data.

Resampling

As a general rule we use 100 bootstraps to ensure a good result. For bootstrap it is especially important, due to the random nature of the method.

Franke's function

Franke's function is a two dimentional function primarlly used in testing interpolation methods numerically. This is precisely what we will be using this for aswell. The function itself is as follows

$$
\begin{aligned}
f(x,y) = &\frac{3}{4}exp\Big(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\Big) \\
&+\frac{3}{4}exp\Big(-\frac{(9x+1)^2}{48}-\frac{(9y+1)}{10}\Big) \\
&+\frac{1}{2}exp\Big(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\Big) \\
&-\frac{1}{5}exp(-(9x-4)^2-(9y-7)^2)
\end{aligned} \tag{8}
$$

$$
\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y} \tag{9}
$$

Cross-validation

We split our data into train and test before dividing in k folds. Thus, the cross-validation is performed on the training set and tested on the test set. Due to the fact that we evenly divide the training data into K number of folds, we had to work with the limitations this gave us. The number of datapoints that were used as training data thus had to be dividable by the number of folds.

Test-train

we use 30 % of the dataset as test data and the remaining 70 % as training data.

IV. RESULT

The MSE and R^2 score depends on the relationship between the number of data points and the order of the polynomial.

Confidence interval

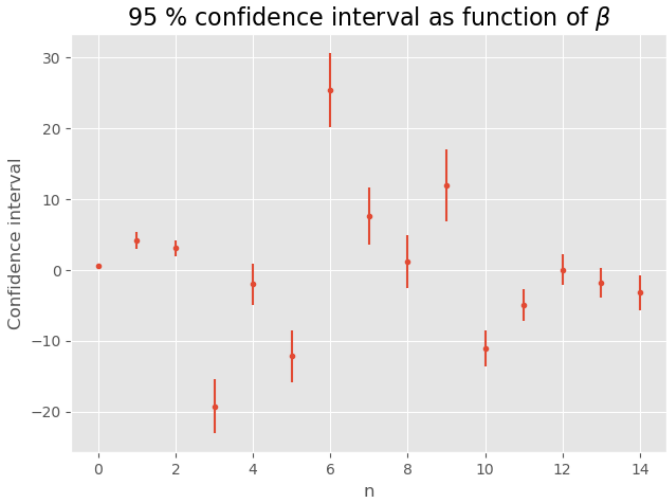

Figure 2. Confidence interval as function of β for polynomial of fifth order with 100×100 points and $\sigma^2 = 0.1$. The x-axis represent the index of the terms in the polynomial, where $n = 0$ corresponds to the constant term. The confidence interval of β_0 is very small compared to the β's for the higher order terms.

Resampling

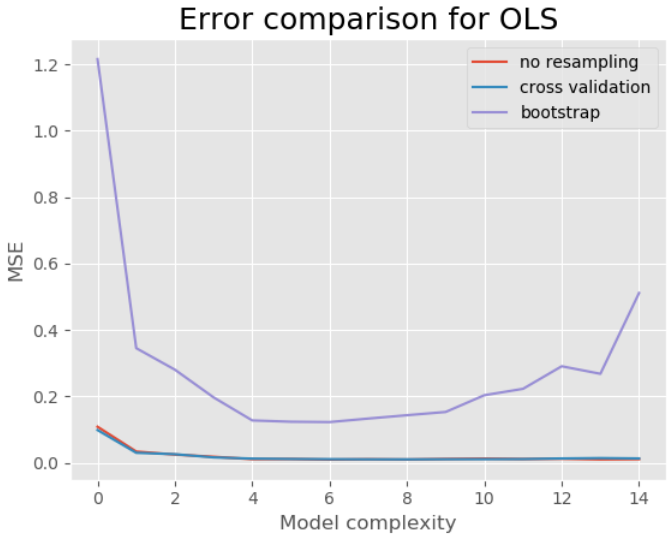

Figure 3. Comparison of MSE produces without resampling, with bootstrap and with cross-validation for OLS. The MSE is clearly lower without resampling than with resampling. The plot is produced with 30×30 points, $\sigma^2 = 0.1$, $n_{folds} = 10$ and $n_{bootstrap} = 100$.

Ordinary Least Squares

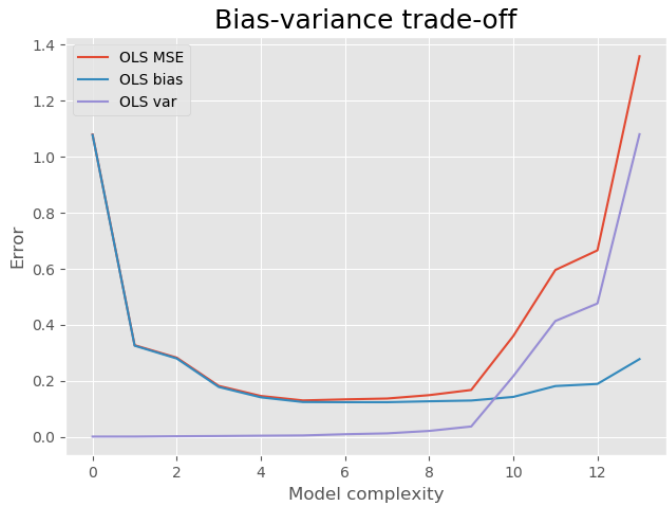

Figure 4. Bias-variance trade-off for OLS with bootstrap. The model is run with 30×30 points, $\sigma^2 = 0.1$ and 100 resamples. The bias decreases as the variance increases and they cross each other at degree eight where the model start to overfit.

Lasso regression

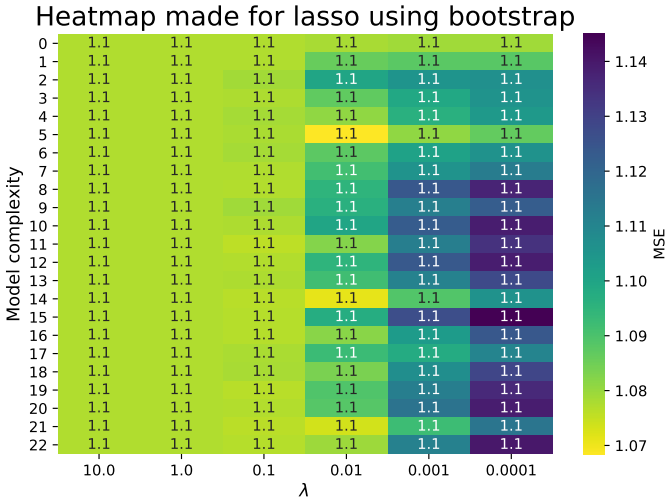

Figure 6. Heatmap for lasso produced wiht 30×30 points, $\sigma^2 = 0.1$ and 50 resamples in bootstrap. As can also be seen in figure 13, the lasso method seems to be behaving badly, as the MSE if fairly large, and practically the same for all complexities and values of λ.

Ridge regression

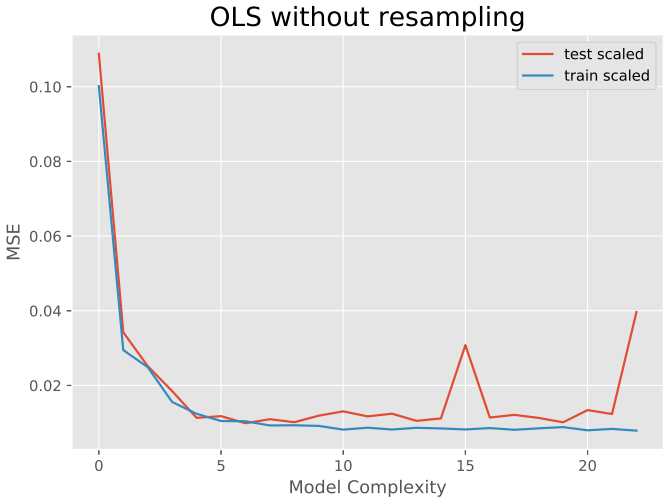

Figure 5. To show overfitting for OLS, the model is applied on the test data, and the train data. Since the model increasingly fits better and better for the test data (overfitting), the MSE for test simply goes lower and lower, while in reality overfitting ensures no generality is left in the model. The plot is produced with 30×30 points and $\sigma^1 = 0.1$.

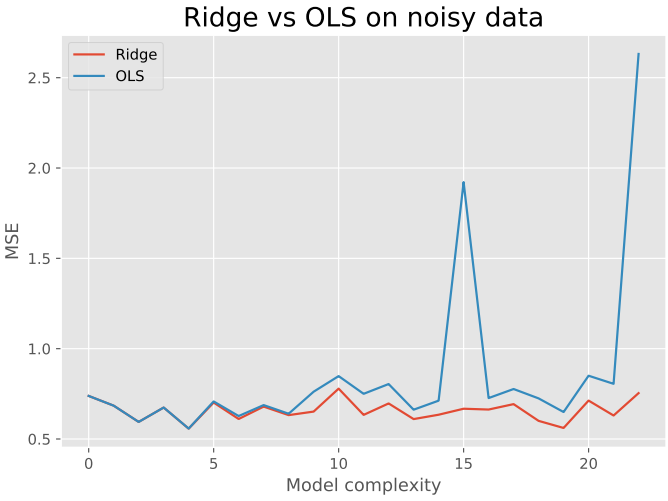

Figure 7. A dataset with a large variance has a large potential to lead to overfitting in our regression model. As shown above, the regularization parameter λ in the Ridge method decreases this effect. Here we have used 30×30 points, $\sigma^2 = 0.8$, $\lambda = 0.001$ and no resampling.

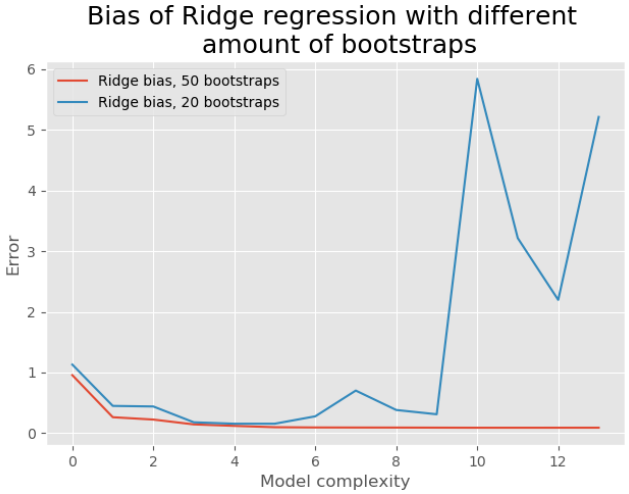

Figure 8. The bias for two different amounts of bootstraps. The model with only 20 bootstraps shows a quite massive amount of Error (MSE), for relatively low model complexity.

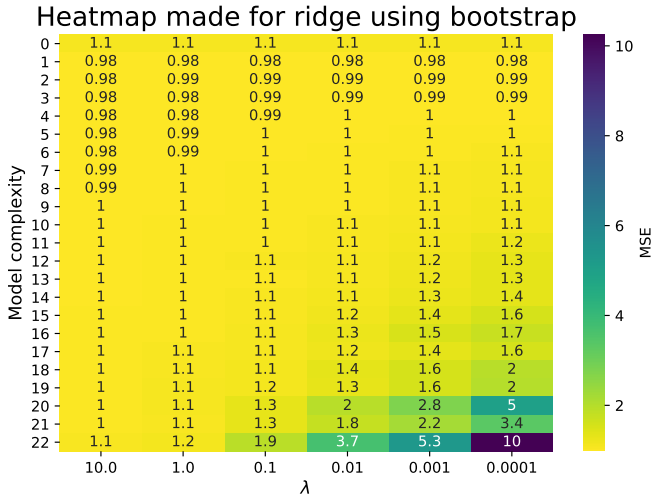

Figure 10. As the complexity increases we see a diagonal band of similar MSE values as λ increases. This is due to the fact that a higher lambda mitigates the large varaince brought on by overfitting as the complexity increases. As expected the λ variable has no effect on the lower degrees of complexity, as these are simply linear or near linear, and thus have few coefficients, and little variance to regularize. The same models are tested on a dataset with more noise in figure 6Heatmap produced with 30×30 points, $\sigma^2 = 0.8$ and 50 resamples in bootstrap.

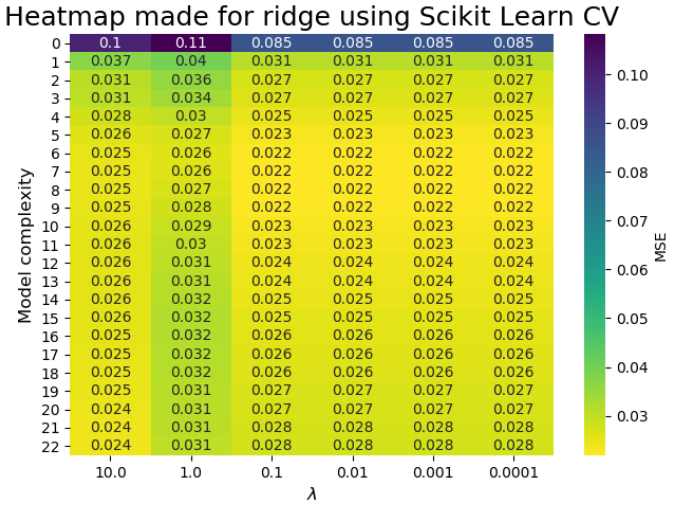

Figure 9. The heatmap is produced with 35×35 points, $\sigma^2 = 0.1$ and 5 folds for Scikit Learn's cross validation. In comparing this heatmap with figure 12, we see that for both models very little changes for the second decimal place. Scikit learns method does provide far lower MSE, but the lack of overfitting that might be expected holds true for both methods.

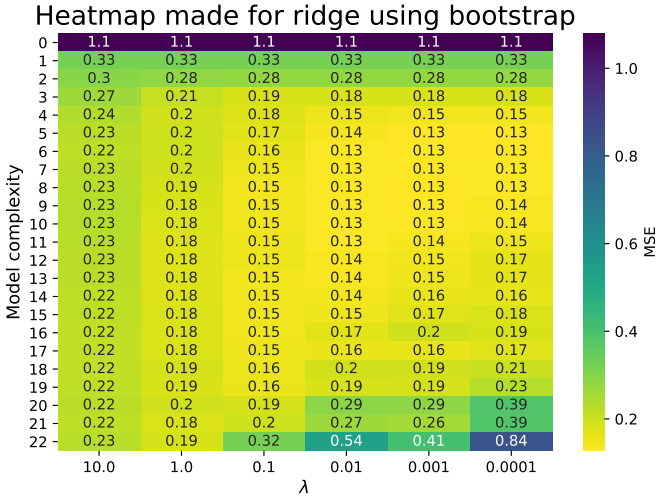

Figure 11. Heatmap produced with 30×30 points, $\sigma^2 = 0.1$ and 50 resamples in bootstrap. As expected, this result is vary similar to figure 10.

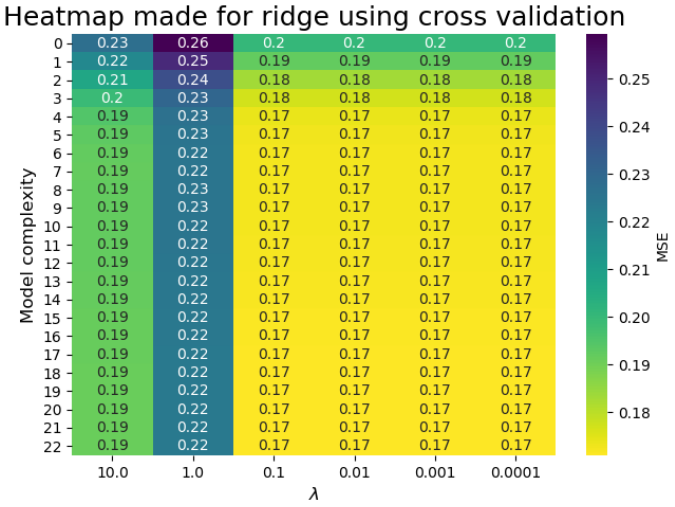

Figure 12. The heatmap is produced with 35×35 points, $\sigma^2 = 0.1$ and 5 folds for cross validation. Due to only having two decimal significance this model seems completley stable for degrees higher than 4. However an underlying behaviour such as in figure 9 is most likely the case.

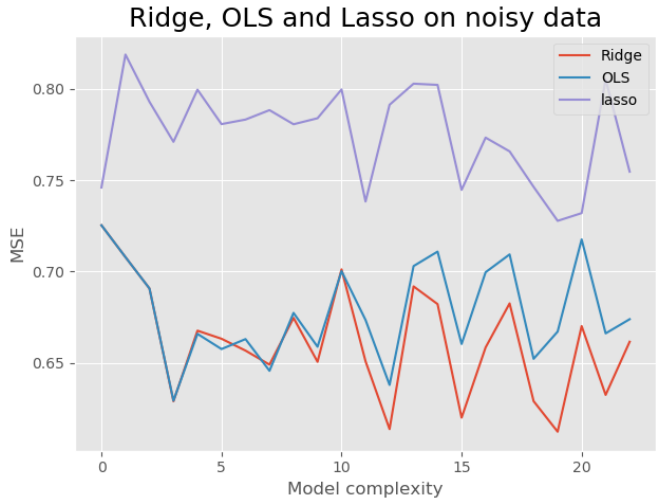

Figure 13. Ridge, lasso and OLS compared using the same dataset, and no resampling. $\lambda = 0.1$, noise $= 0.8$. As can be seen the lasso method produces a method which in large part deviates massivly from the other two. Despite using a relativley high lambda, we see that Lasso beahves badly

Terrain data

V. DISCUSSION

Train test validatiaon

As mentioned in II on train test and validation data, a point was made about ensuring unaffected test data. in figure 5 an example of this is shown. As we apply the model to the train data which it was fitted to it naturally

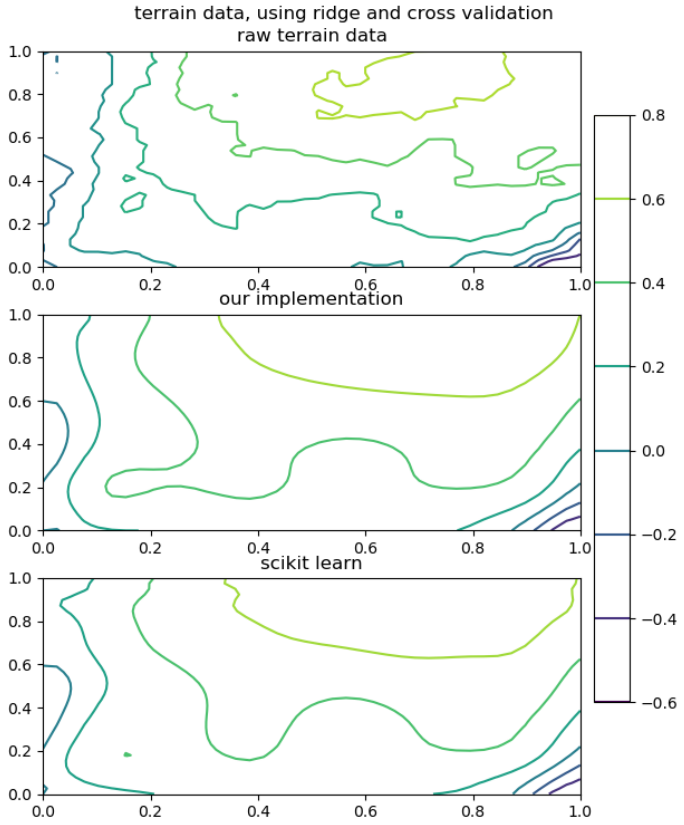

Figure 14. Terrain data from mineapolis, compared to our own ridge and cross validation implementation, and scikit learn. Here $\lambda = 0$ and we use five folds on a total of 40 datapoints. From the figure and from figure 15 we see that the two methods arrive at different complexities. Our uses degree 13, whilst scikit learn uses degree 22. For different numbers of datapoints the optimal polynomial degree would differ slightly between the two methods, but this is consistently less than two or three degrees, which by figure 9 we can see has very little effect on the MSE

fits better and better for higher degrees of complexity. If true independent test data is not used, we might fall into the trap of thinking our model is a lot better for the general case than is the reality.

Resampling

Figure 3 shows how cross validation is more robust to overfitting than bootstrap. Cross validation gives better results than boostrap, even for only five folds. The runtime is also way better for cross validation. It takes 15 times longer to run cross validation with five folds compared to bootstrap with 100 resamples, and the result is still better for cross validation.
As for the behaviour of no resampling it surealy isn't what we expected, and the reason why is unknown.

As can be seen in figure 12, and figure 3 our cross

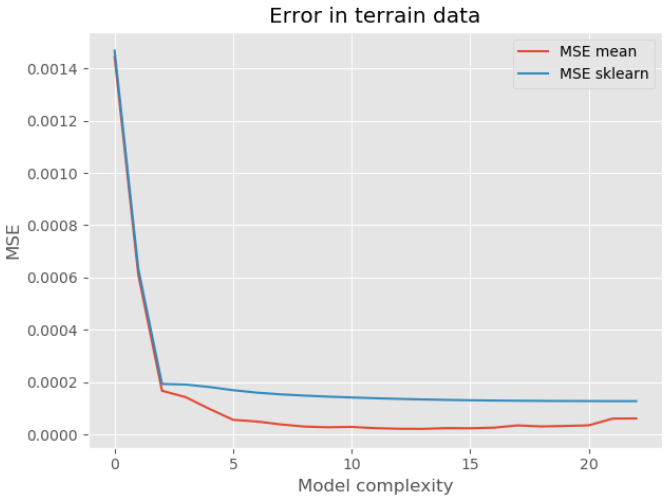

Figure 15. The MSE for sklearn and our own implementation, using 5 k-folds and OLS. Our method found degree 13 to be optimal with an MSE of 0.0358, while scikit learn arrived at 22

validation method behaves strangely. After reaching a certain polynomial degree the MSE for the cross validation method sits stable without reaching overfitting. This despite the fact that the regression method used does not have a regularization parameter λ that counters overfitting. In order to compare and investigate this seemingly strange behaviour we apply scikit learn's cross validation and ridge regression methods, as a form of control. The findings in figure 9 seems to confirm the behaviour found in figure 12. After reaching a fairly stable model around complexity 5, nearly no values change at the second decimal place, which is similar to our own method. In addition to this, no tendency towards overfitting is found, which also aligns with our findings.

Confidence-interval

Figure 2 shows that the confidence interval is way smaller for the constant term β_0 than for the coefficients of the higher order terms. The constant term moves the polynomial up and down the z-axis, so a small change results here increases the variance, hence the confidence interval is small. For the higher order terms, the change in coefficients does not affect the variance as much.
Another thing to observe is that as the variance increases, so will the difference in the coefficients. We found that for polynomials with less variance and higher bias, the spread in coefficients were closer centered around zero. The uncertainty however remained more or less the same.

Ordinary Least Squares

Overfitting is shown in Figure 4, a reproduction of Figure 1. As the model complexity increases, the fit gets better for the train data, but as stated in the theory, this causes the MSE to increase for the test data due to overfitting. In our case the overfitting starts at about degree 7 for a dataset of 30×30 points. The desired degree complexity is naturally found in the area around degree 6.

Numerical inaccuracies

The unstable tendencies seen in plots as degree increases, often far above the number of datapoints, can be explained by the fact that $(X^T X)^{-1}$ being very numerically unstable. As increasing errors are to be expected, a sudden leap that shows almost no mean square error for a high degree is most certainly due to a numerical calculation error.

ridge regression

As theory states, ridge regression should handle datasets with a large variance better than a simple OLS method. As can be seen in figure 7 this is indeed the case for random noise of size 0.8. Despite the fact that we used a fairly small λ we see a consistent difference in the two methods for degree 8 and 9.

One effect of using ridge regression that can be seen in fig 10 is that as complexity increases, and overfitting becomes a major factor in the models, the λ factor keeps the models from overfitting, despite a fairly large degree of complexity. This can be seen in the diagonal bands of similar MSE values for increasing values of λ. For degree 22, a huge lambda of 10 produces the best result of all the different λ values. One can argue that from a computation point of view fitting the model to a unnecessarily large complexity, and then scaling it back with a massive λ is a waste of computations, and that a lower degree of complexity is preferable to this.

For lower degrees of complexity, it can be seen that λ doesn't have much effect on the result. This can be explained by the fact that there is little overfitting for λ to mitigate. Another factor that affects this is the fact that the bootstrap method indeed does reduce the bias in the dataset. This can be shown in fig 8 where Ridge regression with 20 bootstraps has a relatively high error very early. The same plot with 50 bootstraps has nearly no error, even for a high model complexity.

Lasso regression

In figure 13 we see some strange behaviours from the Lasso method, where OLS and Ridge behaves as expected. The same is true for figure 6. This unexpected strange behaviour for a method which has no analytical solution makes it very hard to work with. For this reason we have not spent much time exploring the benefits of this method, seeing as how there seems to be none.

terrain data

When applying the aforementioned methods and algorithms to real life topographical data, we choose what methods we found to be best, and proceed with these. As mentioned above, the Cross validation method was faster, and proved to be more stable for higher polynomial degrees, as compared to bootstrap (see fig 3). Ridge regression has been shown to handle noisy data better than OLS, meaning for terrain data with unknown noise it might prove to be better. Despite this we found that $\lambda = 0$ gave the best MSE, which was 0.0358 for our method. As can be seen in the figure 15, scikit learn and our own implementation arrived at different values for the MSE. This is also reflected in the terrain plot, as our had arrived at a slightly different shape for the contours. Our method arrived at degree 13 being best with an MSE of 0.0358, while sklearn chose degree 22.

VI. CONCLUSION

The behaviours found for bootstrap is shown in fig 4 and is as expected. Having explored the different characteristics of the bootstrap method, we found that it delivered as advertised.

Since cross validation is many times quicker, handles the dangers of overfitting, and is more stable for higher degrees of complexity, it makes choosing this method as a favorite quite easy. The bootstrap method might be perceived as a simpler concept to understand, but the added risks of randomness, and the other elements listed above, it is by far outclassed by cross validation.
To what degree resampling is necessary, and for how large a dataset one can do without, seeing as it is relatively demanding computationally, is something we did not consider in this paper.

Since Ordinary Least Squares is essentially a special case of Ridge, where $\lambda = 0$, we have a situation where the results of OLS can be found using ridge with a single variable adjustment. In addition, we have shown that ridge in certain cases does handle noisy data, as theory stated. The issue of matrix singularities, as mentioned as a motivation for the Ridge method in the theory is also an advantage of Ridge regression.

Applying the terrain data we found that OLS indeed arrived at the best possible MSE for our data. To what degree this method scales to larger datasets is a possible topic for a future paper, as we were operating with a fairly limited 40 datapoints when applying our method to the terrain data.

Appendix A: Rewriting of MSE

Let the true data y be generated by a noisy model

$$\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon},$$

where the noise ϵ is normally distributed with mean zero and variance σ^2. Furthermore, we have defined an approximation to the function f as $\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}$. The parameters β are found by optimizing the mean squared error via the so-called cost function

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

The expectation value of the squared difference between the true data and the approximation can be rewritten as follows

$$\begin{aligned}
\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^2] \\
&= \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon} - \mathbb{E}[\tilde{\boldsymbol{y}}] + \mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^2] \\
&= \mathbb{E}[((\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}]) + \boldsymbol{\epsilon} + (\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}))^2]
\end{aligned}$$

Use $(a+b+c)^2 = a^2 + b^2 + c^2 + 2ab + 2bc + 2ca$

$$\begin{aligned}
&= \mathbb{E}[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \boldsymbol{\epsilon}^2 + (\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^2 \\
&\quad + 2(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])\boldsymbol{\epsilon} + 2\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}) \\
&\quad + 2(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])] \\
&= \mathbb{E}[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2] + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^2] \\
&\quad + \mathbb{E}[2(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])\boldsymbol{\epsilon}] + \mathbb{E}[2\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})] \\
&\quad + \mathbb{E}[2(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])]
\end{aligned}$$

Since the mean of ϵ is zero, the expectation value of ϵ^2 equals the variance of ϵ, σ^2. For the fourth and fifth term, the noise is independent of the bias of $\tilde{\boldsymbol{y}}$, $\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}]$, and the deviation of $\tilde{\boldsymbol{y}}$, $\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}$. Hence, we can take the expectation value of these independently. We know the expectation value of ϵ is zero, thus these terms becomes zero. For the last term we know that the bias is always the same constant value for each sample, and can be taken out of the expectation value. The expectation value of the deviation is zero by $\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})] = \mathbb{E}[\mathbb{E}[\tilde{\boldsymbol{y}}]] - \mathbb{E}[\tilde{y}] = \mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}] = 0$, hence the last term also equals zero. We are then left with

$$\begin{aligned}
\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] &= \mathbb{E}[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^2] + \sigma^2 \\
&= \frac{1}{n}\sum_{i=0}^{n-1}(f_i - \tilde{y}_i)^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \sigma^2.
\end{aligned}$$

Appendix B: Code for reproduction

For code used in this project see https://github.com/elinfi/FYS-STK4155/tree/master/Project1.

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2 edition, 2009.