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

A REVIEW OF VARIOUS LINEAR REGRESSION METHODS AND RESAMPLING TECHNIQUES

Ø. Strand, I. Rivå

Draft version October 10, 2020

ABSTRACT

We consider three types of linear regression, ordinary least squares method (OLS), Ridge regression and Lasso regression and look at generated data with Franke's function and a terrain data set. With python, using our own implementations as well as functionality from sci-kit Learn, we minimised the cost function for these three methods. The Franke dataset consists of $n = 400$ datapoints. Terrain originally had $n = 6485401$ but it was reduced to $n = 6486$ datapoints. The best method for franke data was Lasso as it yielded the lowest MSE. For terrain data the best method turned out to be OLS since for terrain data overfitting is not an issue, due to the large amount of data points in terrain data. We were unable to plot and analyse the methods for high complexities because of computation time. The methods we examined work well with a limited dataset, but other methods might work better on terrain data.

Subject headings: Linear regression, Ordinary Least squares, Ridge, LASSO, Bias-variance tradeoff, K-Fold Cross Validation, Bootstrap resampling

1. INTRODUCTION

Machine learning has, in recent years, become an increasingly powerful tool which can be used to solve both research and industrial challenges. Linear regression models are known for their simplicity and low data requirement, which makes them valuable for a plethora of different problems. To form a better view of how linear regression models work, we examined three linear regression methods and saw how they worked on Franke's function as well as the effects resampling methods have on the models. We then applied the same methods to terrain data to examine how the models work on real life data and tried to assess whether or not this is a decent strategy for exploring terrain data. This paper presents three different linear regression methods, ordinary least squares, Ridge and Lasso, as well as resampling methods, bootstrap and K-Fold. It also discusses the bias variance trade-off, overfitting, and the benefits and disadvantages of using the different methods. This paper will first present the theory and algorithms we used to implement the models. It will then delve into how the methods were implemented. The results will then be presented followed by the discussion were we will evaluate the different models, set them up against each other and discuss the applicability of these linear regression methods. Then follows the conclusion where we will take a look at how our methods can be improved upon and summarize the discussion.

2. ALGORITHMS

We now present the data sets, regression and resampling methods and the methods used for their analysis in the following sections.

2.1. *Data*

orjanstr@student.matnat.uio.no ,
ingunriv@student.matnat.uio.no
[1] Institute of Theoretical Astrophysics, University of Oslo, P.O. Box 1029 Blindern, N-0315 Oslo, Norway

In our analysis, we worked on two different datasets. We produced one set ourselves, and then moved on to terrain data taken from eastern Norway. The dataset we produced consists of the Franke function for random uniformly distributed $\boldsymbol{x}, \boldsymbol{y} \in [0, 1]$ plus some noise $\boldsymbol{\epsilon} \in N(1, \sigma^2)$.

$$\boldsymbol{z} = f(\boldsymbol{x}) + \boldsymbol{\epsilon} \tag{1}$$

We set $\boldsymbol{x}$ and $\boldsymbol{y}$ to be one dimensional such that $\boldsymbol{z}$ ended up being one dimensional as well. The terrain data (see figure 1) is a substantially larger set than the Franke data. The set consists of $3601 \times 1801 = 6485401$ data points. The resolution of the data set is 30m per data point.

We used a polynomial fit to generate our model. All terms in the polynomial can be represented by combinations of x^i times all degrees of y up to y^{deg-i} (eq. 2).

$$x^i \cdot [y^0, y^1, ..., y^{deg-i}] \tag{2}$$

Our design matrix, $\boldsymbol{X}$, takes all these combinations, which are our features, as columns starting with $i = 0$ and ending up with $i = deg$. With $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^1$ of length n, we get a design matrix of shape $n \times p$ where p represents amount of features.

We also scaled and standardized our data because the range of the data might vary and we do not want our methods to interpret some features as more important just based on their value. We scaled our data by subtracting the mean and dividing by the standard deviation:

$$\boldsymbol{x}_{scaled} = \frac{1}{\sigma_x}(\boldsymbol{x} - \bar{\boldsymbol{x}}) \tag{3}$$

We scaled our data before splitting, but we realise that this was a mistake. By doing it this way the mean is calculated based on all values which means that the testing set and the training set are not kept totally separate. Our data was split it into training and test sets with a 4:1 relationship. Whenever we study effects such as bias and variance, it is always on the test set unless otherwise specified.

2.2. *Minimisation of the Cost function*

We define the so-called cost function, which is the function we wish to minimize in order to find the best fit for our model.

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 = \mathbb{E}\left[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2\right]. \qquad (4)$$

When we split our data into test and training sets, we try to minimise the cost function on the training data. For higher complexities of the polynomial fit we will then see our model fit the training data perfectly, but not the test data. This is the problem of overfitting. For too low complexities we will not see any good fit in either of the sets. Our goal is then to find the sweet spot of complexities such that we reduce both under and overfitting.

2.3. *Bias and Variance*

We can rewrite the cost function (eq. 4) to

$$\mathbb{E}\left[(\boldsymbol{z} - \tilde{\boldsymbol{z}})^2\right] = \frac{1}{n}\sum_{i=0}^{n-1}(f_i - \mathbb{E}\left[\tilde{\boldsymbol{z}}\right])^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{z}_i - \mathbb{E}\left[\tilde{\boldsymbol{z}}\right])^2 + \sigma^2 \qquad (5)$$

.

(See section 7.2 for derivation of equation 5). Equation 5 consists of three terms; The bias^2, variance and variance of noise ($var(\boldsymbol{\epsilon}) = \sigma^2$).

The bias describes the difference between the average of the data and our model. A high bias tells us that our model is not very well fit to our training data. Low bias indicates overfitting. Variance describes the difference between a set of models and a mean model, and tells us something about how the model varies when resampling. In our analysis, we calculate the model variance by finding the variance of each element in a set of resampled models using bootstrap resampling. Typically, a high variance will indicate overfitting to the training data, and result in a high error in the test data. The sum of the bias and the variance should equal the MSE between our model and our data.

When fitting a model, the goal is to find a good balance between the two. Something that results in a somewhat low bias and a somewhat low variance. This tradeoff is called a Bias-Variance Tradeoff. Since we are working with fitting a polynomial, we try to find the complexity that results in the best bias-variance tradeoff.

In our work we included the noise term in the bias term such that the bias term becomes $\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2$. This does not affect the behavior of the bias, so we can safely perform a tradeoff analysis like this.

2.4. *Bootstrap resampling*

The bootstrap resampling method is probably one of the simplest resampling methods out there. Having split our sample into training and test data, we keep the test data untouched, and simply resample the training data. When resampling, we see how much our model varies from sample to sample, giving us a nice and intuitive way to study the bias and variance. This resampling method thus plays an important role our analysis of the bias-variance tradeoff, and was the only resampling method used for exactly this.

2.5. *K-Fold cross validation*

K-fold is a cross validation technique used to evaluate the accuracy of machine learning models. The data is split into k different folds. One of the folds is conserved as testing data while the rest of the folds are used as training data and this is repeated until all the folds have had a change to be used as testing data. For each set we compute the mean squared error. We then take the mean of all of the errors to obtain our estimate for the model. Since all of the data points will have the chance to be used as testing data it will result in less bias. K-Fold is also useful because it allows us to reuse our data, artificially inflating out dataset, making it valuable if we have limited data.

The correct way to perform K-Fold cross validation is to first split the data into training and test data, then perform the resampling and lastly test on the test set that has been kept separate from the beginning, but for simplicity's sake we will use the slightly less complex method of not splitting the data beforehand.

2.6. *Ordinary Least Squares Method*

The first regression method up for review is the well known Ordinary Least Squares method, or OLS for short. The goal of the OLS is simply to minimize the sum of squared residuals between our model $\tilde{\boldsymbol{z}}$ and our data $\boldsymbol{z}$.

Our model $\tilde{\boldsymbol{z}}$ is an approximation of the continuous term $\boldsymbol{f(z)}$ from equation 1. We define a set of coefficients,$\boldsymbol{\beta}$, for our polynomial such that

$$\tilde{\boldsymbol{z}} = \beta_0 + \beta_1 \boldsymbol{y} + ... + \beta_i \boldsymbol{x^2 y} + ... = \boldsymbol{X\beta}. \tag{6}$$

(see section 2.1 for design matrix specifics)

Our goal then becomes to find the optimal set of coefficients $\hat{\boldsymbol{\beta}}$ that minimises the cost funcion (eq.4). Following Hastie et al. 2009, section 3.2(2), we find that this optimal $\hat{\boldsymbol{\beta}}$ is given by

$$\hat{\boldsymbol{\beta}} = \boldsymbol{(X^T X)^{-1} X^T z} \tag{7}$$

We split our noisy Franke dataset (see equation 1) as well as the calculated design matrix into training and test sets (see section 2.1). We then calculated our optimal coefficients $\hat{\boldsymbol{\beta}}$ on the training set, and generated a split model following equation 6:

$$\tilde{\boldsymbol{z}}_{train} = \boldsymbol{X}_{train}\hat{\boldsymbol{\beta}} \tag{8}$$

$$\tilde{\boldsymbol{z}}_{test} = \boldsymbol{X}_{test}\hat{\boldsymbol{\beta}} \tag{9}$$

From this we calculate the mean squared error (MSE) to see how well our model fits the dataset.

$$MSE(\boldsymbol{z}, \tilde{\boldsymbol{z}}) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 \tag{10}$$

We then studied the effects of overfitting on our training and test data as well as the bias-variance tradeoff with increasing complexity (see section 3.1). We then looked at how bootstrap and k-fold cross validation affect the MSE in the test data.

2.7. *Ridge Regression*

Ridge is a linear regression method that uses regularization and has the ability to handle multicolliniarity. Here the $\hat{\beta}$ values are found by minimising the Ridge cost function. The cost function for Ridge regression can be written as

$$\sum_{i=1}^{n}(z_i - \beta_0 - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_{j-1}^{p} \beta_j^2$$

according to Hastie et al 3.41 (2). For our purposes we have chosen to penalize the intercept for simplicity's sake. The cost function we want to minimize is

$$(\boldsymbol{X^T X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{X^T z}$$

,

We can see that Ridge is, in essence, OLS with an added penalty $\lambda \boldsymbol{I}$, where $\lambda \geq 0$. That means that if we gradually decrease λ, our model will get closer and closer to OLS, and Ridge with $\lambda = 0$ corresponds to OLS. Increasing λ will increase the penalty on the β values and push them towards zero, but due to the nature of the penalty term they will never fully reach zero. If the penalty is too small Ridge will face the same issues with overfitting as OLS does, but a penalty that is too large will result in underfitting. Therefore it is important to chose good λ values. If the matrix $\boldsymbol{X^T X}$ is linearly dependant i.e not invertible the extra term in Ridge will resolve the singularity problem we face when we try to use OLS regression, which means that we can use Ridge to calculate a model even if the matrix is non invertable. This will also help the model be a little bit more stable because it reduces some of the numerical error you expect when trying to take the inverse of a nearly non invertable matrix. The bias Ridge introduces makes our model less inclined to change, and less likely to get fit to the unpredictable noise of the datapoints and thereby get overfit. This helps reduce the variance and makes Ridge more resistant to overfitting.

To implement Ridge regression we added the penalty term to our implementation of OLS. We then plotted MSE for different λ values and different complexities with bootstrap and K-Fold. We also plotted bias and variance for different lambda values for one given complexity. These plots were made for both Franke's function and terrain data, see section 3.

2.8. *Lasso Regression*

Lasso regression is a type of linear regression that uses feature selection and regularization. We find the cost function for Lasso regression in Hastie et al 3.52 (2):

$$\sum_{i=1}^{n}(z_i - \beta_0 - \sum_j x_{ij}\beta_j)^2 + \lambda \sum_{j-1}^{p} |\beta_j|, \lambda \geq 0$$

.

Lasso is quite similar to Ridge but with a different penalty term. $\sum_1^p |\beta_j|$. When $\lambda = 0$ no features are eliminated and when it increases more and more features are pushed to zero. Lasso works well in the case where we have a lot of features that are not important for our data because it sets the corresponding β values to zero. This is because in addition to minimising the cost function we are putting a constraint on the size of the beta values. This constraint is $\boldsymbol{\beta}|| \leq t$ for ridge. If we draw out this shape in 2 dimensions, we will get something that looks like a diamond. We want our β values to be inside our on the edge of this diamond, as this corresponds to our restriction. This restriction is to hinder the beta values getting too big and creating a lot of variance. Since the constraint is in the shape of a diamond the point where the cost function first hits the constraint is likely to be on a corner. That means that, in 2 dimensions, either β_1 or β_2 is going to zero. the Lasso cost function is not strictly convex and there is no explicit solution for it, so it cannot be computed in the same way we did with OLS and Ridge. we are going to use scikit learn's library for linear regression (4), which uses coordinate descent to find a minimum. We plotted the MSE using and K-Fold for Lasso in addition to the variance. We produced these plots for both franke's function and terrain data, see plots in section 3

2.9. *Terrain data*

Having tested our methods on the Franke function we applied them to a real life dataset in the from of a terrain map. From earthexplorer.usgs.gov (1), we chose an appropriate data set from eastern Norway. The data is of a Norwegian fjord featuring quite erratic terrain, especially so around the fjord.

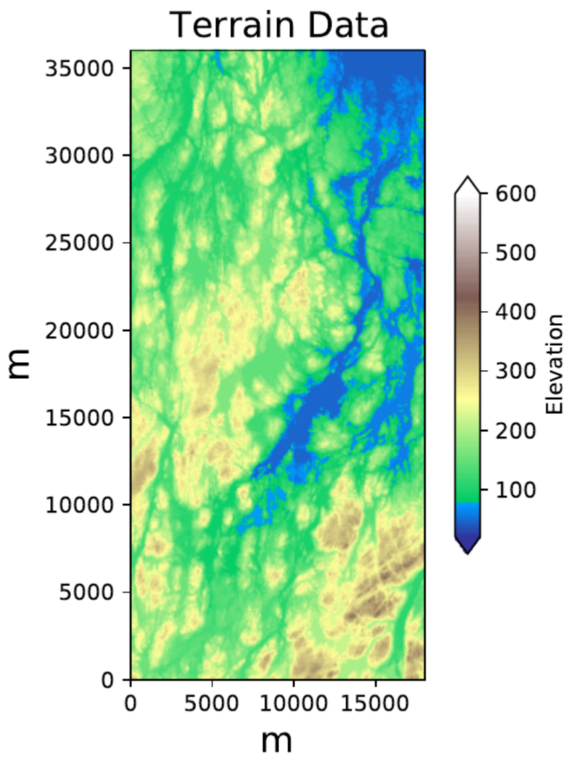

Fig. 1.— Terrain data plotted before doing any fitting.

We first had to process the data so we would be able to feed it into our Regression class. We used a combination of ravel and meshgrid to get the data on a form where our class would be able to process it. To reduce computation time and be able to run our program on the terrain data we only worked on every 1000th element, resulting in $n = 6486$ data points. Then we went on to plot our data in the same manner as we did with Franke's function. While plotting we tried varying the λ values and the maximum complexity to get more insight into how our different models behave.

3. RESULTS

We now present the results found in our analysis of the various regression and resampling methods. See section 4 for further discussion of these results.

3.1. *OLS for Franke data*

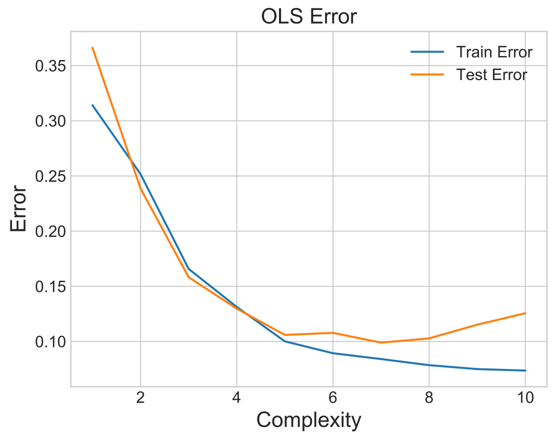

Fig. 2.— MSE for training and test set of OLS model as function of complexity. $n = 400$, $\sigma^2 = 0.1$, dataset: Franke

In figure 2 We see how the training and test error (MSE) behaves as a function of complexity. The training and test error behave similarly at lower complexities, but split at polynomial degree 5. The training error converges towards 0 with higher complexities, but the test error increases dramatically. There is a clear similarity between figure 2 and figure 2.11 in Hastie et al. 2009, section 2.9 (2).

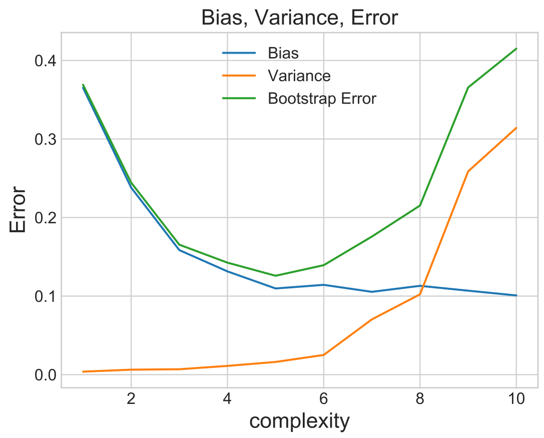

Fig. 3.— OLS resampled using bootstrap method. Figure shows bias, variance and MSE in test set as function of complexity. $n = 400$, $\sigma^2 = 0.1$, $n_{\text{bootstraps}} = 100$, dataset: Franke.

In figure 3 We see low variance and relatively high bias for lower complexities. With increased complexity we get a steep decrease in bias, and a shallow increase in variance. The bias decreases up til polynomial degree 7, where it starts increasing slightly. The variance starts increasing dramatically at polynomial degree 6. The error clearly equals the sum of the bias and the variance of our model.

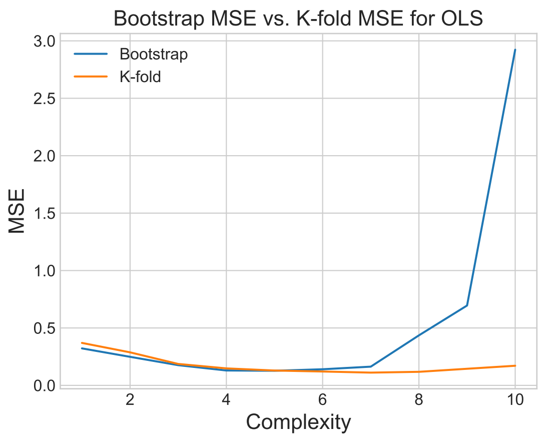

FIG. 4.— MSE of OLS model resampled using bootstrap method and k-fold cross validation. $n = 400$, $\sigma^2 = 0.1$, $k = 5$, dataset: Franke

In figure 4 we see the MSE of our OLS method for our new dataset resampled using both bootstrap and k-fold. At higher complexities, we still get an increase in test error when resampling using bootstrap. When resampling using k-fold cross validation, we still get a slight increase, but it is significantly smaller and less aggressive than for bootstrap.

3.2. *Ridge for Franke data*

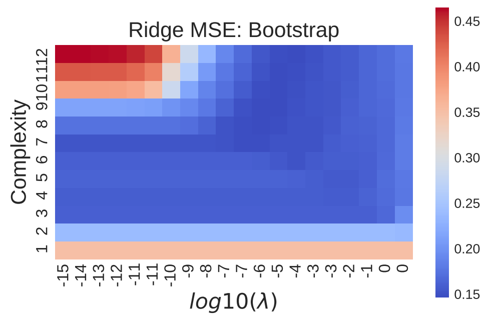

FIG. 5.— MSE of Ridge regression model as function of complexity and penalty, λ. Resampled using bootstrap method. $n = 400$, $\sigma^2 = 0.1$, $n_{bootstraps} = 100$, dataset: Franke

Figure 5 shows the MSE for our Ridge regression model resampled using the bootstrap method for the Franke data. For low penalties $\lambda < 10^{-11}$ we see our model behave much like the OLS, with higher MSE at high complexities, and a minimum around polynomial degree 7. With penalty $\lambda > 10^{-10}$ we see a decrease in the MSE for higher complexities, with a minimum at around $\lambda \sim 10^{-4}$. The MSE starts increasing again for $\lambda > 10^{-3}$.

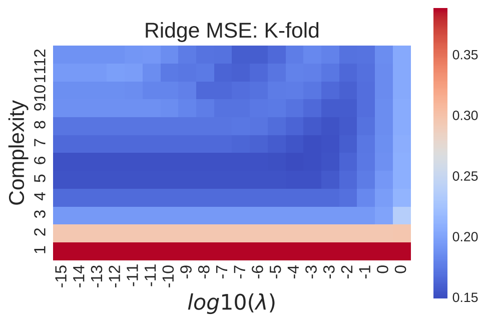

FIG. 6.— MSE of Ridge regression model as function of complexity and penalty, λ. Resampled using k-fold cross validation. $n = 400$, $\sigma^2 = 0.1$, $k = 5$, dataset: Franke

Figure 6 shows the MSE for our Ridge regression model resampled using k-fold cross validation for the Franke data. We still see an increase in MSE for higher complexities with low penalty, but not as significant as in figure 5. We see two minima for MSE for penalty around $\lambda \sim 10^{-7}$ and $\lambda \sim 10^{-1}$, with increasing MSE for $\lambda > 10^{-1}$.

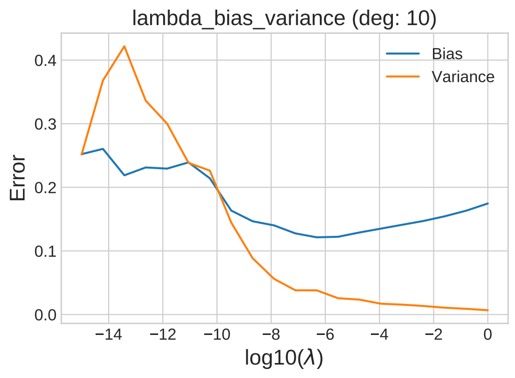

FIG. 7.— Penalty effect on bias and variance for Ridge calculated on Franke data at polynomial degree 10. $n = 100$, $\sigma^2 = 0.1$.

In figure 7 we see how the penalty effects the bias and variance of a model fit with Ridge regression on our generated Franke data. We can see that with increasing penalty, we get a significant decrease in variance. We can also see a decrease in bias, but not as steep as the variance.

3.3. *Lasso for Franke data*

In figure 8 we see the MSE for the test data for a model solved using Lasso regression. We see that an increased penalty results in a higher MSE over all. We see a minimum in MSE at high polynomial degree and low penalty. We see that for significantly high penalty, we get a very high MSE. We get a maximum MSE at when the penalty magnitude is 10^{-2} and polynomial degree is 1.

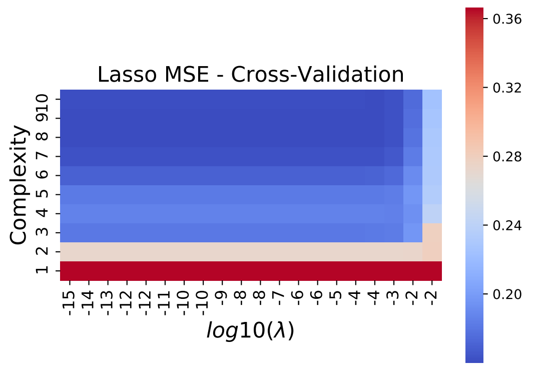

FIG. 8.— MSE of Lasso regression model as function of complexity and penalty, λ. Resampled using k-fold cross validation. $n = 400$, $\sigma^2 = 0.1$, $k = 5$, dataset: Franke

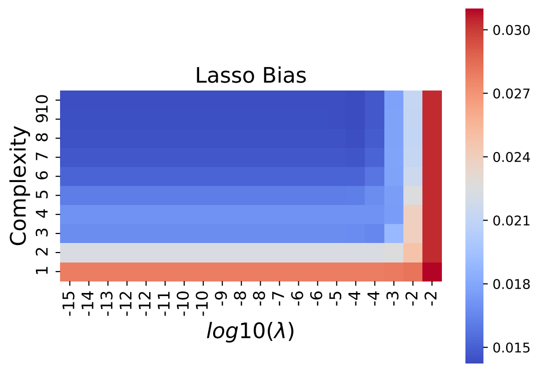

FIG. 9.— Bias of Lasso regression model as function of complexity and penalty, λ. Resampled using the bootstrap method. $n = 400$, $\sigma^2 = 0.1$, $n_{bootstraps} = 100$, dataset: Franke

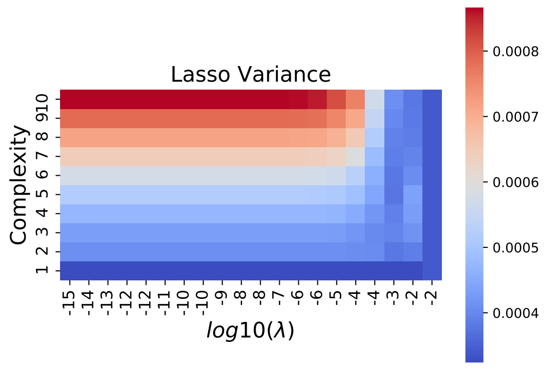

FIG. 10.— Variance of Lasso regression model as function of complexity and penalty, λ. Resampled using the bootstrap method. $n = 400$, $\sigma^2 = 0.1$, $n_{bootstraps} = 100$, dataset: Franke

In figures 9 and 10 We see the bias and variance for the Lasso model resampled using bootstrap resampling. The bias values get lower towards higher complexities and lower penalties, but there is not a clear minimum. The variance (seen in figure 10) has a maximum at high complexities and somewhat low penalty values. The variance is at its lowest for low polynomial degrees and high penalty values.

3.4. *Terrain Data*

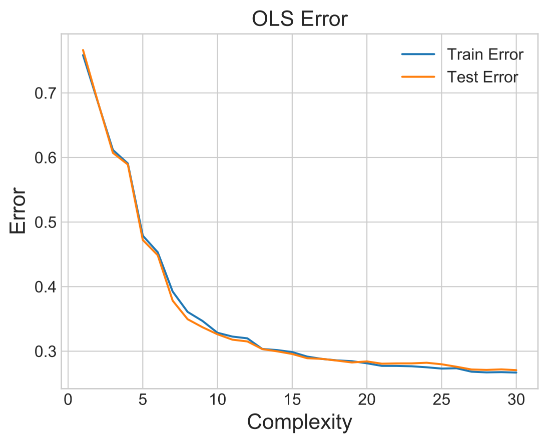

FIG. 11.— Train and test error for terrain data using OLS

In figure 11 We compare the test and training error from the OLS sampled from our terrain data. We see that, even for complexities up to polynomial degree 30, both the training and the test error follow the same path to 0. The MSE starts off high, and decreases exponentially, converging towards 0.

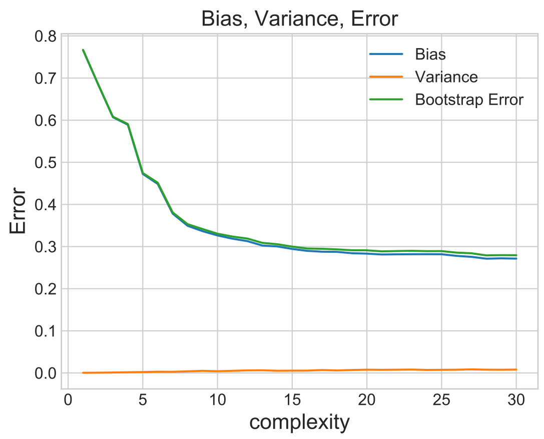

FIG. 12.— bias variance and MSE in test set as a function of complexity using terrain data. $n_{bootstraps}$=100

If we look at figure 12 we can see that the variance for our terrain data is very low, lingering around zero, making the error predominantly consist of the bias term. The bias and the error has a minimum in complexity = 5, and gradually increase after that.

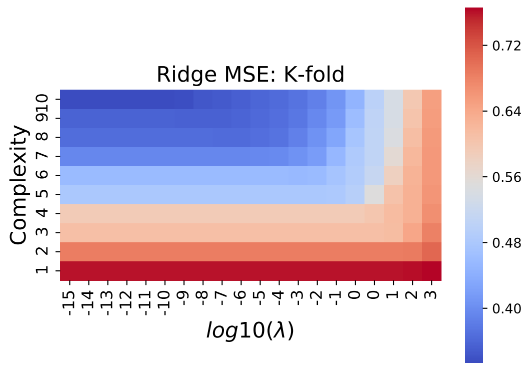

FIG. 13.— MSE of Ridge regression as function of complexity and λ. Resampled using k-fold cross validation. $k = 5$, dataset: Terrain

Figure 13 shows the MSE of the Ridge regression model on our terrain data, resampled using k-fold cross validation. We see an extremely high MSE for high complexities and low penalties, with a peak MSE at polynomial degree 10 and a penalty $\lambda \sim 10^{-10}$.

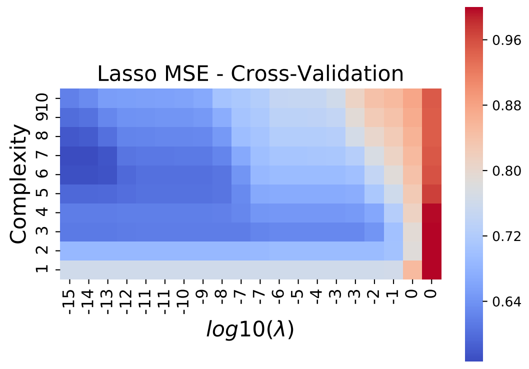

FIG. 14.— MSE of Lasso regression as function of complexity and λ. Resampled using k-fold cross validation. $k = 5$, dataset: Terrain

Figure 14 shows the MSE of our Lasso model on the terrain data. We see that increasing penalty causes increasing MSE for high complexities. For significantly high λ we see a rapid increase in MSE even for low complexities, but low penalties have otherwise little effect at lower complexities. We see a minimum MSE at polynomial degree 6 and $\sim 10^{-15}$.

4. DISCUSSION

4.1. *OLS on Franke data*

Looking at the increased test error and variance for the OLS in figures 2 and 3, we see clear cases of overfitting. The reason we have lower test error than training error at some lower complexities is that the model hasn't fit either one very well, and which error is the largest is completely random. While we do work with the same data for every complexity for this particular model, we also split the training and test data into new sets with every new complexity. This may cause some instability in our model, but the overall trend is sound. The increase in test error polynomial degree 5 tells us that this is when the model will start overfitting. This is where we start seeing the flaw of the OLS. We see that simply fitting the data perfectly to the training data, leads to poor predictions once we introduce new data (in our case the test set). When resampling, we see a high variance for the OLS, meaning The optimal polynomial degrees seem to lie between 4 and 6, with the best fit obtained for degree 5 for this particular dataset, since this is where bias and variance both have low values.

We can see that with bootstrap we get a lot of overfitting with higher complexities, but this does not happen with k fold. Since k-fold makes predictions on a varied set of training data (in our case we chose 5 folds), we won't see overfitting to as high a degree as we do with bootstrap resampling. Note that we generate a new dataset for figure 4.

4.2. *Ridge regression*

We verify that our Ridge method works for small penalties since it behaves like the OLS in these regions, as is expected since Ridge is just the OLS plus penalty. This applies when we resample with the bootstrap method and K-fold cross validation. We note that the effect of the penalty becomes noticeable around $\lambda \sim 10^{-10}$. At this point, the bias introduced by the penalty starts reducing the variance of our model, resulting in a more general fit. Thus reducing the error for the test data. Pushing the penalty passed $\lambda \sim 10^{-1}$, we start seeing underfitting, where the bias has increased to the point where our model doesn't fit either the training or the test data very well. In figure 7 we can clearly see the variance decrease, as expected. When our model is overfit, we expect the bias in the test data to be high, seeing as the model only cares about the training data. When the overfitting is reduced we see the bias in the test data decrease, since our model will fit the test data slightly better. The bias would then increase again once we reach the region of underfitting. This bias behavior is exactly what we see in the figure. We also note that the penalty affects higher complexities more that lower. We believe this is because of the already high bias in these regions, meaning adding a small amount of bias will not have any great effect. As we discussed in section 4.1, and as can be seen in figure 4, k-fold cross validation handles overfitting better than bootstrap resampling. We clearly see that this is the case in figures 5 and 6. And the error for the k-fold resampled Ridge model gives us lower MSE over all.

4.3. *Lasso regression*

We note that for small penalties, the Lasso model does not behave like our OLS model. We believe this is because we use sci-kit Learn's method, which uses coordinate descent to find the best fit for the model. This minimisation method is used even if $\lambda = 0$, and will thus yield a different result than our OLS method, which finds the best fit analytically. It would seem coordinate descent is more accurate than our analytical method, seeing as we do not see any overfitting for low penalties. In figure 8 we only generate models up to polynomial degree

10. We would expect to see overfitting at higher degrees, as Lasso should behave like OLS at the lowest penalties. We have tried going to degree 15, but we still could not see any overfitting. Higher complexities would result in very high computational time. Since we got the same result for complexities up to degree 15, we chose to stick to degree 10, as this yields a more readable figure.

Since we do not have any overfitting in our model, we will only see the MSE increase with the penalty. This is again due to underfitting. Our model is already slightly underfit, meaning the penalty will only make this worse. If we ran the method for higher complexities we would eventually expect to see Lasso reduces the MSE faster than we saw in Ridge. We believe this is the case since Lasso forces the coefficients to 0 instead of having them converge. We can see this effect for $\lambda > 10^{-4}$ where the underfitting happens much faster for Lasso than for Ridge. The penalty rapidly increases the bias, and lowers the variance. We also note that the over all MSE for Lasso is much lower than for Ridge. We believe this might be due to the coordinate descent used in sci-kit Learn's model, or due to the fact that we generate new data sets for every test. Meaning Lasso is used on a different dataset than Ridge.

4.4. *Terrain*

The terrain data has significantly more datapoints than the Franke set generated in the previous section. It will thus take a higher complexity to get overfitting. In fig 11 we would expect to see the train and test error split at higher complexities, giving us a similar result as we saw for the Franke data (fig 2). We would also expect to see a similar result for the bias-variance tradeoff, seeing the bias steadily increase and eventually a rapid increase in variance once we reach overfitting. We can already slightly see this effect in figure 12.

As we saw when discussing the Lasso method for the Franke data, the bias introduced by the penalty will only increase the MSE by causing underfitting. We see the same result when applying Ridge to the Terraindata. In figure 13 we see no overfitting for Ridge. We thus see underfitting for higher penalties. For higher complexities, we would again expect to see similar results as for the Franke data. We might find that we would need even smaller penalties to see a significant effect on overfitting, as we have a lot more datapoints, giving a larger accumulated effect of λ.

When using the Lasso method on the terrain data, we start seeing a higher MSE for high complexities, even at very low penalties. Judging from figure 14 the MSE will only increase with complexity. We know that the penalty will have a stronger effect on higher, more variance dominated complexities. What we see here appears to be underfitting. We believe this early underfitting is due to high pre-existing underfitting because of too low complexity for the complex data set. This means that we would probably have to use extremely low penalties to see a proper result for the Lasso regression. So in an ideal case, we would study very high complexities together with much smaller penalties to properly get an idea of how Lasso works on our training data.

5. CONCLUSION

We saw that linear regression models are computationally demanding and take a long time to run, which makes it difficult to examine large datasets, but our methods work really well even on small datasets. It is easy to see that this is very useful in situations where the data is limited which is so often the case in real life. Since linear regression is best for smaller datasets we would love to try other machine learning methods on the data too see how well they could model the data. When working on larger sets, however, problems arise. Even when reducing the Terrain data by a factor of 1000, it still proved itself difficult to get a proper analysis of the various methods presented, due to the high computational time required to get adequate results.

We see clearly see overfitting when using the OLS on Franke data, This makes K-fold or Lasso the better choice and we get the best results for Lasso, since it is better at reducing the overfitting. As for resampling, we saw that K-fold cross validation results in less overfitting compared to bootstrap as an effect of the way we vary what is training data and what is test data. For the terrain data overfitting is not a problem. This makes it so that any penalty introduced with either Ridge or Lasso only serves to make a worse fit because it gets underfit.

It would be nice to use our methods on different data to be able to more clearly see the differences between Ridge and Lasso and confirm that lasso does better when we have a lot of features but the data is only really dependant on a small set of them, and ridge does better when the data is dependant on a lot of the features.

If we could make any improvements we would have liked to be able to plot terrain data for higher complexities. We would also like to implement ridge in the correct way, excluding the intercept when applying the penalty and implementing the proper version of K-fold where a test set is kept away from the start. Another thing that could be done is to split the data before we do the scaling so our test and training sets are kept completely separate and not splitting on every complexity would help create more stability. It would also be nice to look at different scaling methods to see how they might affect the result.

6. ACKNOWLEDGEMENTS

We acknowledge all help received from the group teachers of this course. These have been some difficult concepts to wrap our heads around, and their endless help in the group sessions have made this journey much easier. We also acknowledge the guidance recieved from M. H. Jensen in the lectures and lecture notes (3) following this course, providing us the much needed knowledge to complete this task. Lastly we acknowledge J. Starmer of StatQuest for his continuous uploads of witty and easy to understand videos on the concepts covered in this paper.

7. APPENDIX

7.1. *Additional results*

7.1.1. *MSE and R^2 score for OLS up to polynomial degree 5*

We study the MSE and R^2 score for the OLS for polynomial degrees from 1 to 5.

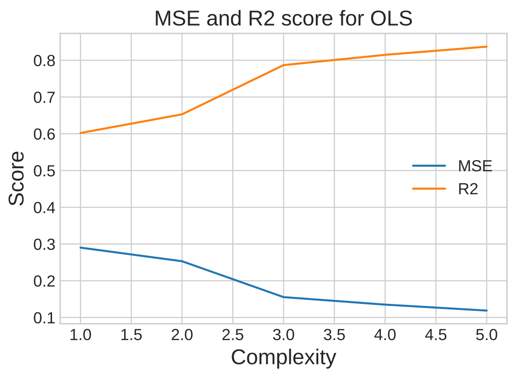

FIG. 15.— R^2 and MSE score for OLS as function of complexity. $n = 400$, $\sigma^2 = 0.1$, dataset: Franke

We can see in ifugre 15 that the R^2 score and the MSE are opposites of each-other. This makes sense seeing at R^2 is just $1 - MSE_{scaled}$.

7.1.2. *Beta confidence intervals*

We study the confidence intervals β by calculating their variances for a model of polynomial degree 5:

$$var(\beta) = \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} \tag{11}$$

The variances are along the diagonal of this matrix.

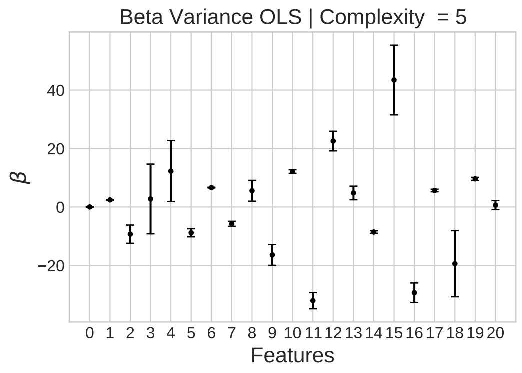

FIG. 16.— Coefficients with convidence intervals for OLS at polynomial degree 5. The features correspond to the coloumns in the design matrix (see section 2.1). $n = 400$, $\sigma^2 = 0.1$, dataset: Franke

We see in figure 16 that come coefficients have high variance. These coefficients correspond to the higher polynomial terms. These coefficients will cause the major variance in our model. When applying the penalties from Lasso and Ridge, these high variance coefficients will be shrunk down, causing the over all variance to go down as well.

7.2. *Bias-Variance decomposition (eq. 5)*

We start off with the cost function:

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

We can rewrite the right hand side:

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + \mathbb{E}\left[\tilde{\boldsymbol{y}}\right] - \tilde{\boldsymbol{y}})^2\right]$$

$$= \mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 - 2(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]) + (\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right]$$

Now the middle term becomes 0, giving us:

$$\mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right]$$

We recall that $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$ and expand the left hand term:

$$\mathbb{E}\left[(\boldsymbol{y} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] = \mathbb{E}\left[\boldsymbol{y}^2 - 2\boldsymbol{y}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + (\mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right]$$

$$= \mathbb{E}\left[f^2 + 2f\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^2 - 2f\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] - 2\boldsymbol{\epsilon}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right] + \mathbb{E}\left[\tilde{\boldsymbol{y}}\right]^2\right]$$

$$= \mathbb{E}\left[(f - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[2f\boldsymbol{\epsilon}\right] + \mathbb{E}\left[\boldsymbol{\epsilon}^2\right] - \mathbb{E}\left[2\boldsymbol{\epsilon}\mathbb{E}\left[\tilde{\boldsymbol{y}}\right]\right]$$

Since f and $\boldsymbol{\epsilon}$ are independent of each other we know that $\mathbb{E}\left[2f\boldsymbol{\epsilon}\right] = 2\mathbb{E}\left[f\right]\mathbb{E}\left[\boldsymbol{\epsilon}\right]$. And with $\mathbb{E}\left[\boldsymbol{\epsilon}\right] = 0$ and $\mathbb{E}\left[\boldsymbol{\epsilon^2}\right] = \sigma^2$ we then get for the left hand term:

$$= \mathbb{E}\left[(f - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \sigma^2.$$

Putting this back into the cost function, we get

$$\mathbb{E}\left[(f - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \mathbb{E}\left[(\tilde{\boldsymbol{y}} - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2\right] + \sigma^2$$

Finally, we get our result by switching to a sum notation for the expectation values.

$$\mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right] = \frac{1}{n}\sum_{i=0}^{n-1}(f_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \frac{1}{n}\sum_{i=0}^{n-1}(\tilde{y}_i - \mathbb{E}\left[\tilde{\boldsymbol{y}}\right])^2 + \sigma^2$$

Github with source code: https://github.com/OrjanStr/FYS-STK4155_Project1

REFERENCES

[1]USGS Earth Explorer. https://earthexplorer.usgs.gov/.

[2]T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2009.

[3]M. H. Jensen. Fys-stk4155, lecture notes. https://compphysics.github.io/MachineLearning/doc/web/course.html.

[4]F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.