FYS-STK4155 – Project 1 on Machine Learning

Regression analysis and resampling methods

Viggo Wetteland

October 10, 2020

Abstract

In this project, we have studied in detail the Ordinary Least Squares (OLS) method, Ridge regression, and Lasso regression. We have compared them on data generated by the Franke function which simulates terrain data, and we compared them on real terrain data. We also studied resampling techniques like the bootstrap method and cross-validation. We found that on relatively small datasets (20x20) the Ridge regression gave us the lowest test MSE on the Franke function dataset followed by OLS, but on the real terrain data both Ridge and Lasso started to produce very large errors, and the OLS stayed stable. The bootstrap technique performed better than the cross-validation method in all cases.

1 Introduction

The goal of this project is to study in detail various regression methods, including the Ordinary Least Squares method, Ridge regression, and Lasso regression on our own numerically generated data and on real terrain data. To generate our own dataset we're gonna make use of the Franke function which is a function that simulates terrain data. The Ordinary Least Squares method estimates the parameters in a regression model by minimizing the sum of the residuals, which is the difference between the observed value and the mean value that the model predicts for that observation. This method draws a line through the data points that minimizes the sum of the squared differences between the observed values and the corresponding predicted values. Ridge regression is a method for analyzing multiple regression data that suffer from multicollinearity, which is a phenomenon in which one predictor variable in a multiple regression model can be linearly predicted from the others with a substantial degree of accuracy. Lasso regression is an analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.

We will also try out two different resampling techniques on each of these methods. These resampling techniques are called the bootstrap method and cross-validation. The bootstrap method is a statistical technique for estimating quantities about a population by averaging estimates from multiple small data samples. Samples are constructed by drawing observations from a large data sample and returning them after they have been chosen. Cross-validation is a resampling procedure used to evaluate machine

learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=5 becoming 5-fold cross-validation.

We will try to evaluate which of these models best fit our simulated terrain data and the real terrain data, and which of the resampling techniques best improve the results.

2 Method

2.1 Ordinary Least Square (OLS) on the Franke function

We will first train our models on our own datasets, and for that, we need to generate our own data. To simulate a terrain dataset we're using the Franke function which needs two 2-dimensional inputs of data points. We generate these two sets of data points x and y like this:

```
x = np.arange(0, 1, 0.05)
y = np.arange(0, 1, 0.05)
x, y = np.meshgrid(x,y)
```

We then feed these two 20x20 matrices of data points into the Franke function which is defined as

```
def FrankeFunction(x,y):
    term1 = 0.75*np.exp(-(0.25*(9*x-2)**2) - 0.25*((9*y-2)**2))
    term2 = 0.75*np.exp(-((9*x+1)**2)/49.0 - 0.1*(9*y+1))
    term3 = 0.5*np.exp(-(9*x-7)**2/4.0 - 0.25*((9*y-3)**2))
    term4 = -0.2*np.exp(-(9*x-4)**2 - (9*y-7)**2)
    return term1 + term2 + term3 + term4
```

To make this simulated terrain dataset more realistic we will add Gaussian noise with NumPy's "random.normal", which is similar to the noise we get from measurements on real terrain data.

The next thing we're going to do is to perform a standard least square regression analysis using polynomials for x and y up to fifth order. This can be done by making a design matrix as a function of a given polynomial. Doing this manually would look like this

```
X = np.zeros((len(x),21))
X[:,0] = 1; X[:,1] = x; X[:,2] = y
X[:,3] = x*x
X[:,4] = y*y
X[:,5] = x*y
X[:,6] = x**3
X[:,7] = y**3
X[:,8] = x*x*y
X[:,9] = x*y*y
```

and continuing up until the wanted order. With a more convenient function where n = polynomial degree as seen in the code below we can generate design matrices with many different polynomial degrees without having to do it manually.

```
def create_X(x, y, n ,intercept=True):
        if len(x.shape) > 1:
                x = np.ravel(x)
                y = np.ravel(y)

        N = len(x)
        l = int((n+1)*(n+2)/2)          # Number of elements in beta
        X = np.ones((N,l))
        idx = 0
        for i in range(2-intercept,n+1):
                q = int((i)*(i+1)/2)
                for k in range(i+1):
                        X[:,idx] = (x**(i-k))*(y**k)
                        idx +=1
```

We want to find the confidence intervals of the parameters (estimators) β by computing their Mean Squared Error (MSE) and the R^2 score function. If $\tilde{\hat{y}}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the MSE is defined as

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

and the R^2 score is defined as

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

where we have defined the mean value of $\hat{y}$ as $\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$.

A scaling of the data will be included with sklearn.preprocessing's "StandardScaler", and we will split the data into train and test data where we want to validate the trained model against the untouched test

data. The splitting of data can be done with sklearn.model_selection's "train_test_split", and we choose a train data size of 80%.

2.2 Bias-variance trade-off and resampling techniques

We consider a dataset $\{(y_j, x_j), j = 0 \dots n-1\}$, and we assume that the true data is generated from a noisy model $y = f(x) + \epsilon$ (with ϵ being normally distributed with mean zero and standard deviation σ^2). The parameters β are then found by optimizing the means squared error with the cost function

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

Here X is the design matrix and the expected value E is the sample value. Since we're going to study the bias-variance trade-off it will be useful to rewrite this equation in terms of bias and variance. We can use that y and ϵ have a variance equal to σ^2, and that the mean value of ϵ is zero. We can start by rewriting y as $f + \epsilon$

$$\mathrm{E}[(y - \tilde{y}_i)^2] = \mathrm{E}[(f + \epsilon - \tilde{y}_i)^2],$$

and we can add and subtract $\mathrm{E}[\tilde{y}]$ and get

$$\mathrm{E}[(y - \tilde{y}_i)^2] = \mathrm{E}[(f + \epsilon - \tilde{y} + \mathrm{E}[\tilde{y}] - \mathrm{E}[\tilde{y}])^2]$$

$$\mathrm{E}[(y - \tilde{y}_i)^2] = \mathrm{E}[(y - \mathrm{E}[\tilde{y}])^2] + Var[\tilde{y}] + \sigma^2$$

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

where the equation we end up with has the bias as the first term and the variance as the second term. In applications where the function values f_i are unknown we will replace them with the actual data points y_i.

We're going to perform a bias-variance analysis of the Franke function by studying the MSE value as function of the complexity of our model. We will also introduce the bootstrap resampling method here with sklearn.utils' "resample" and calculate the mean of the calculated sample statistics.

2.3 Cross-validation as resampling techniques, adding more complexity

Next, is updating our model with the k-fold cross-validation algorithm. K-fold cross-validation works by splitting the data into k subsets (called folds). Our model will have 5 folds. The model is then trained using all but one of the folds. The model is then evaluated using the unused fold, which acts as the test set. This can be implemented in our code with sklearn.model_selection's "Kfold" like this:

```
kfold = KFold(n_splits = k)
```

```
for train_inds, test_inds in kfold.split(X_train_scaled):
    x_cv_train = X_train_scaled[train_inds]
    z_cv_train = z_train[train_inds]

    x_val = X_train_scaled[test_inds]
    z_val = z_train[test_inds]

    beta = np.linalg.pinv(x_cv_train.T @ x_cv_train) @ x_cv_train.T @ z_cv_train
    ztilde = x_cv_train @ beta
    MSE_train_CV = MSE(z_cv_train,ztilde)
    zpredict = x_val @ beta
    MSE_test_CV[i] = MSE(z_val,zpredict)
```

We're going to run our code using 5 folds and compare these results with the results from the bootstrap analysis.

2.4 Ridge Regression on the Franke function with resampling

We're performing the same bootstrap analysis with the bias-variance trade-off, and the cross-validation method again, but now with the Ridge regression algorithm instead of OLS. This can be added to the code like this:

```
I = np.eye(X.shape[1],X.shape[1])
nlambdas = 20
MSEPredict = np.zeros(nlambdas)
MSETrain = np.zeros(nlambdas)
lambdas = np.logspace(-10, 1, nlambdas)

for i in range(nlambdas):
    lmb = lambdas[i]
    Ridgebeta = np.linalg.pinv(X_train.T @ X_train+lmb*I) @ X_train.T @ z_train
    B0 = np.mean(z_train)
    # and then make the prediction
    ztildeRidge = X_train @ Ridgebeta + B0 #adding the intercept
    zpredictRidge = X_test @ Ridgebeta + B0 #+intercept
    MSEPredict[i] = MSE(z_test,zpredictRidge)
    MSETrain[i] = MSE(z_train,ztildeRidge)
```

The reason why we're adding the intercept later in the code now is because of a problem of some unpredictable behavior of the MSE training data otherwise.

2.5 Lasso Regression on the Franke function with resampling

Again we will repeat the previous parts, but now with Lasso regression using sklearn.linear_model's "Lasso". The parts in the code where OLS and Ridge regression were calculated gets replaced by sklearn's functionality while the rest of the code stays the same.

2.6 OLS, Ridge and Lasso regression on real data with resampling

With our codes functioning on a satisfactory level we are now ready to take the next step and try our models out on real terrain data. The terrain data used in this project has GeoTIFF format. To read these data files in Python one can simply import "imread" from the imageio package and read the data with

```
terrain1 = imread('SRTM_data_Norway_1.tif')
```

To get the same x and y input-matrices we had earlier we can choose a smaller area of the terrain data and make use of meshgrid again. For example like this:

```
# just fixing a set of points
N = 20
m = 7 # polynomial order
terrain1 = terrain1[:N,:N]

# Creates mesh of image pixels
x = np.linspace(0,1, np.shape(terrain1)[0])
y = np.linspace(0,1, np.shape(terrain1)[1])
x_mesh, y_mesh = np.meshgrid(x,y)
# Note the use of meshgrid
z = terrain1.ravel() #height
```

The data is now ready to be used as input for the design matrix function as with our previous data set from the Franke function.

This final part deals with the parameterization of our real terrain data. We will apply all three methods for linear regression as we did with our own generated dataset and evaluate which model fits the data best. The bootstrap analysis and the cross-validation as resampling techniques will also be tested and compared with the real terrain data.

3 Results and discussion

3.1 Ordinary Least Square (OLS) on the Franke function

In table 1 we see that the errors on the test data are greater than on the training data, which is to be expected. What is interesting to note here is that when increasing the added noise to 0.3 or 0.5 we start getting more unreliable results. It is possible that this happens because the model tries to fit the noise instead of the actual data.

OLS model	Training data	Test data
MSE	0.010452	0.014055
R^2	0.881796	0.835514

Table 1: The OLS model's MSE and R2 score with our own generated dataset with the Franke function. The polynomial degree is n = 5. The added noise was $\sigma^2 = 0.1$. 20x20 dataset.

3.2 Bias-variance trade-off and resampling techniques

When implementing the bootstrap resampling technique we can see in Figure 1 how the train and test data error behaves with varying model complexity. A model complexity in the 8-10 area seems to give us the lowest errors on this dataset.

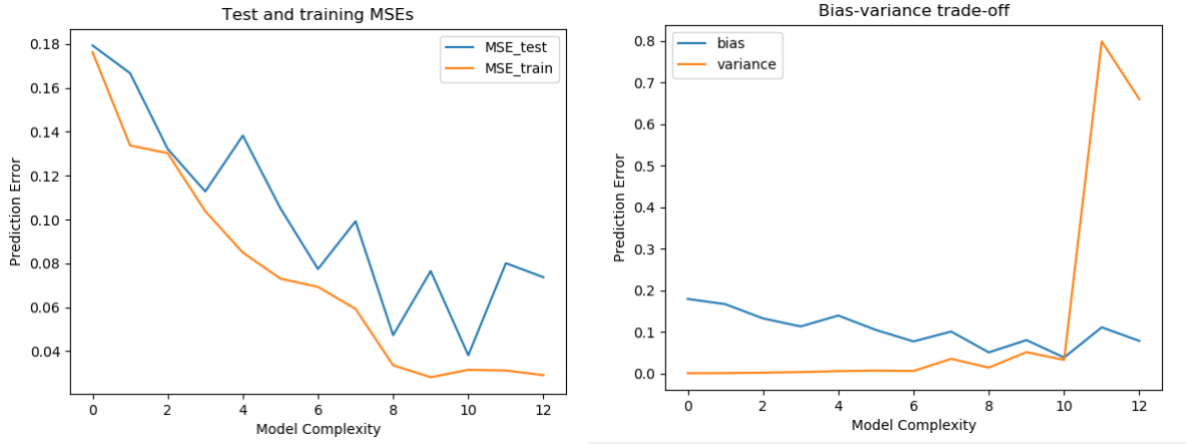

Figure 1 (left): A comparison of the test and training MSEs with 200 bootstraps and added noise $\sigma^2 = 0.3$. The model complexity refers to the polynomial degree. 20x20 dataset. Figure 2 (right): The bias-variance trade-off by implementing the bootstrap resampling technique. Same parameters as in figure 1.

After studying the bias-variance trade-off in figure 2 it becomes evident why we see the increase in MSE test error when the model complexity goes higher than 10. The variance becomes very high and results in the model overfitting the data. Increasing the number of data points decreases the variance because as we add more data we get increasingly precise estimates of group means, so that's one good way to prevent the overfitting.

3.3 Cross-validation as resampling techniques, adding more complexity

The cross-validation method has very similar results compared to the bootstrap method. The 10^{-1} area on the y-axis corresponds to the 0.1 range we can see in the previous figures, and the overfitting starts to happen a little sooner as shown in figure 3. A model complexity of n = 7 gives us the best results on this dataset.

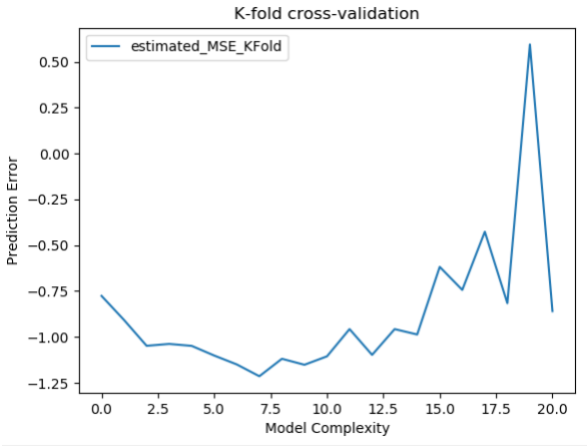

Figure 3: The cross-validation approach with 5 folds. The data is still generated from the Franke function with added noise $\sigma^2 = 0.3$. 20x20 dataset. The reason for negative y-values is because the y-axis is a log10 plot.

3.4.1 Ridge Regression on the Franke function with bootstraps

Performing the same bootstrap analysis as in section 3.2, but now with Ridge regression, we find in figure 4 that a lambda of $10^{-3.5}$ gives a smaller MSE on the test data compared to the regular OLS algorithm (when lambda approaches zero we end up with OLS).

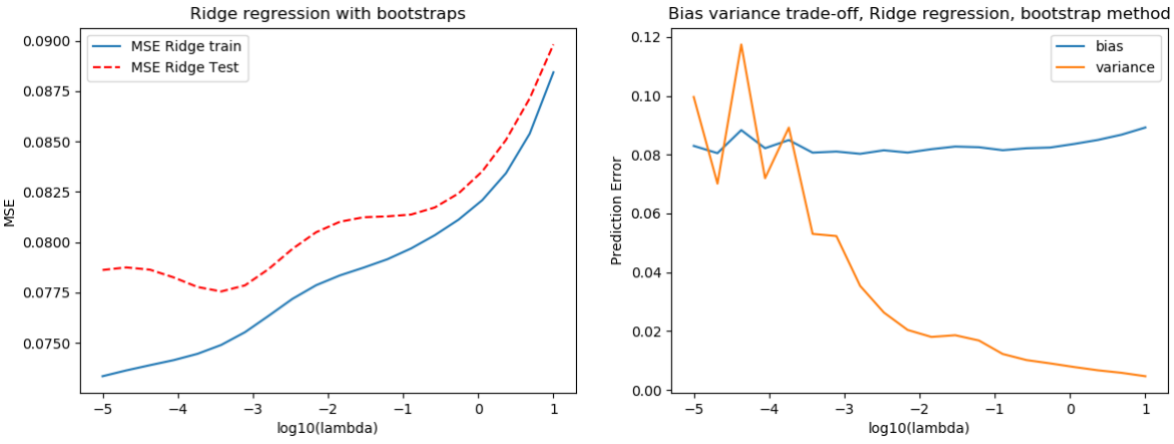

Figure 4 (left): 200 bootstraps using the Ridge regression algorithm on the Franke function dataset. Added noise is $\sigma^2 = 0.3$. Polynomial degree n = 7. 20x20 dataset. Figure 5 (right): The bias-variance trade-off with Ridge regression using the same parameters as in figure 4.

In figure 5 we can see that as λ increases, the flexibility of the ridge regression fit decreases, leading to a shrinking variance but increased bias. This is the tendency we're expecting to see in ridge regression. The reason for this is that ridge keeps all variables and shrinks the coefficients β down to zero.

3.4.2 Ridge Regression on the Franke function with cross-validation

Implementing now the cross-validation algorithm with Ridge regression we see that the value of lambda is in the same area near $10^{-3.5}$, but the MSE is considerably higher than with bootstraps. The K-fold cross-validation algorithm performs more than three times worse than the bootstrap algorithm as seen in figure 6 below.

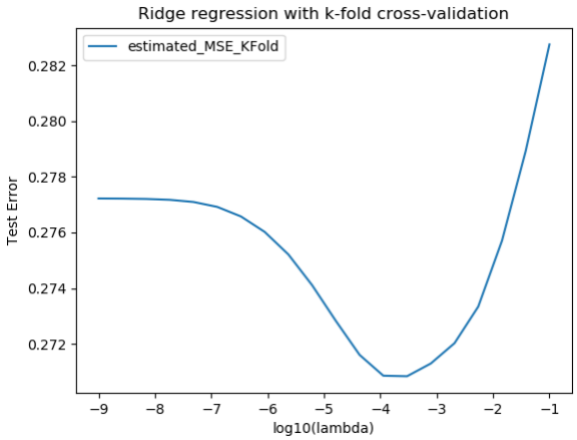

Figure 6: Ridge regression with cross-validation on the Franke function dataset. Here we used 5 folds, n = 7 polynomial degrees, and added noise $\sigma^2 = 0.3$. 20x20 dataset.

3.5.1 Lasso Regression on the Franke function with bootstraps

With Lasso regression and the bootstrap algorithm, we see that the error always increases as lambda increases, so in this case, it would be better to revert back to the regular OLS algorithm instead (with λ = 0) as seen in figure 7.

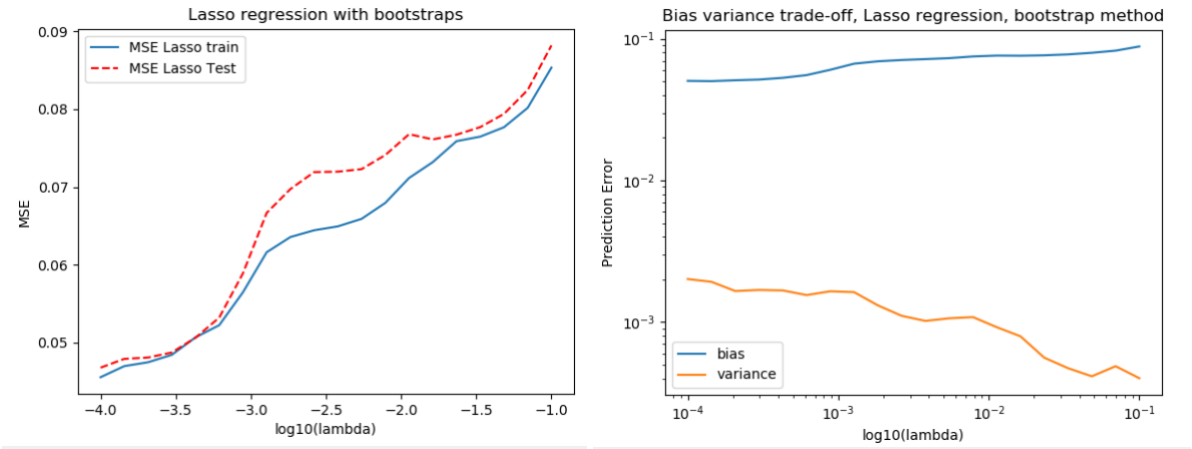

Figure 7 (left): 200 bootstraps using the Lasso regression algorithm on the Franke function dataset. Added noise is $\sigma^2 = 0.3$. *Polynomial degree n = 7. 20x20 dataset. Figure 8 (right): The bias-variance trade-off with Ridge regression using the same parameters as in figure 7.*

The bias-variance trade-off in this case is shown in figure 8 and follows the expected behavior as with the Ridge algorithm. Lasso regression is also a type of linear regression that uses shrinkage of data points towards a mean or a central point. The reason why Lasso regression doesn't work that well here could be because the Lasso procedure encourages simple, sparse models (i. e models with fewer parameters)

3.5.2 Lasso Regression on the Franke function with cross-validation

When implementing the cross-validation algorithm for the Lasso regression as seen in figure 9 below, we find that Lasso doesn't suffer from overfitting as easily, as the MSE keeps dropping with increasing lambda. However, the test error is still almost three times higher than the corresponding K-fold cross-validation with the Ridge method. Comparing the bootstrap method and the cross-validation method within Lasso regression we can see that the cross-validation MSE is approximately 50% higher.

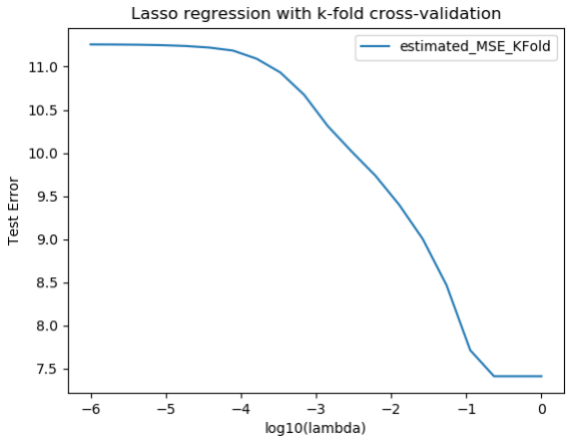

Figure 9: Lasso regression with cross-validation on the Franke function dataset. Here we used 5 folds, n = 7 polynomial degrees, and added noise $\sigma^2 = 0.3$. 20x20 dataset.

3.6 OLS, Ridge and Lasso regression on real terrain data with resampling

3.6.1 Ordinary Least Square (OLS)

As seen in table 2 the test errors are greater than the training errors like the results from table 1 earlier. When running the regular OLS algorithm on the terrain data we sometimes got results that weren't expected, similar to when we increased the added noise to the dataset generated by the Franke function. This observation makes sense because this real terrain data naturally has noise added to it as well.

OLS model	Training data	Test data
MSE	512.414	525.741
R^2	0.896034	0.887153

Table 2: The OLS model's MSE and R2 score performed on the real terrain dataset. 100x100 data points were used here.

3.6.2 Bias-variance trade-off and resampling techniques

When implementing the OLS bootstrap algorithm on the real terrain data we can see in figures 10 and 11 that the results look similar to what we got with the Franke function dataset. Overfitting seems to start a little earlier here in figure 10, at around a model complexity of 5-7. The bias-variance trade-off in figure 11 shows that the variance increases rapidly in the 5-7 area too, so the model fits the data best around this complexity. The fact that the bias doesn't decrease with increasing model complexity doesn't necessarily point to a mistake or bug in the code, since it at least stays relatively constant.

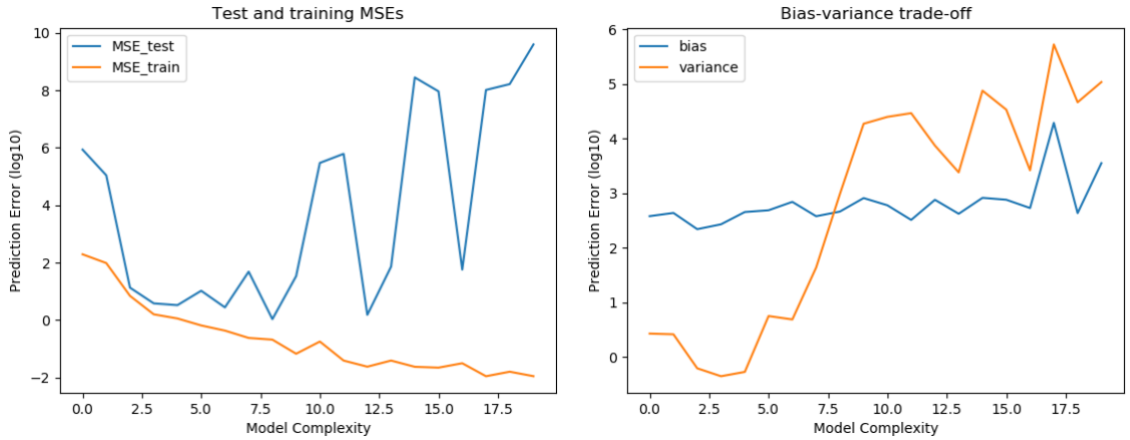

Figure 10 (left): A comparison of the test and training MSEs with 200 bootstraps on the real terrain data. The model complexity refers to the polynomial degree. 20x20 datapoints. Figure 11 (right): The bias-variance trade-off by implementing the bootstrap resampling technique on the real terrain data. Same parameters as in figure 10.

3.6.3 Cross-validation as resampling techniques, adding more complexity

Applying cross-validation to the OLS algorithm for the terrain data we see that with large datasets like in figure 12 the MSE continues to decrease as the model complexity increases. The overfitting starts to show up again when reducing the number of data points to 20x20 as seen in figure 13. Comparing figures 10 and 13 then reveals that on the real terrain data cross-validation isn't a more accurate algorithm than bootstrap, as the overfitting happens a little earlier and the error is about the same.

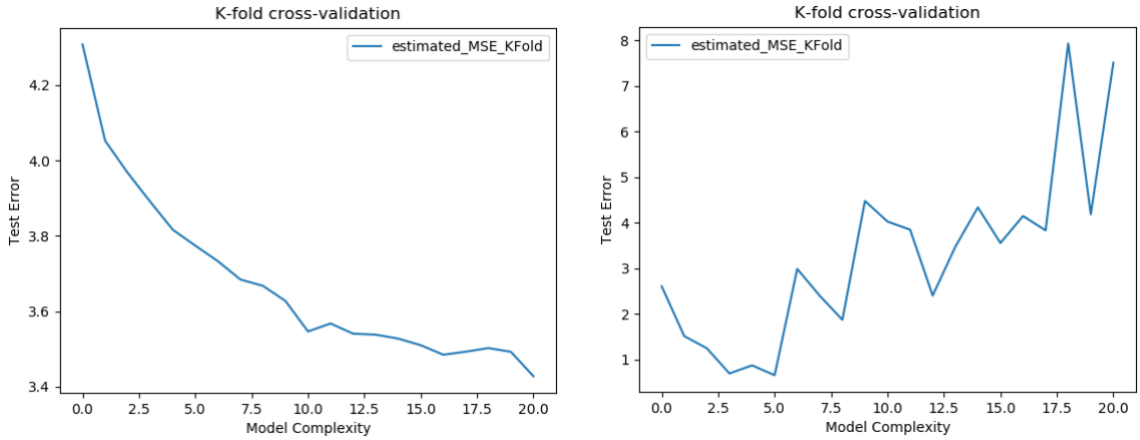

Figure 12 (left): The cross-validation approach with 5 folds. The test error on this figure is generated from the real terrain data with 1000x1000 datapoints and the y-axis is a log10 plot. Figure 13 (right): The same algorithm but with 20x20 datapoints this time.

3.6.4 Ridge Regression with bootstraps

Performing the Ridge regression bootstrap analysis on the terrain data we see that the model fits best when lambda is near 10^{-4}, and better than the OLS method in that area as seen in figure 14. The variance behaves like expected for increasing lambda values in figure 15 where it approaches zero. It's not immediately obvious why we get such extreme values on the y-axis in figure 15. It seems like the model doesn't perform very well on the terrain data.

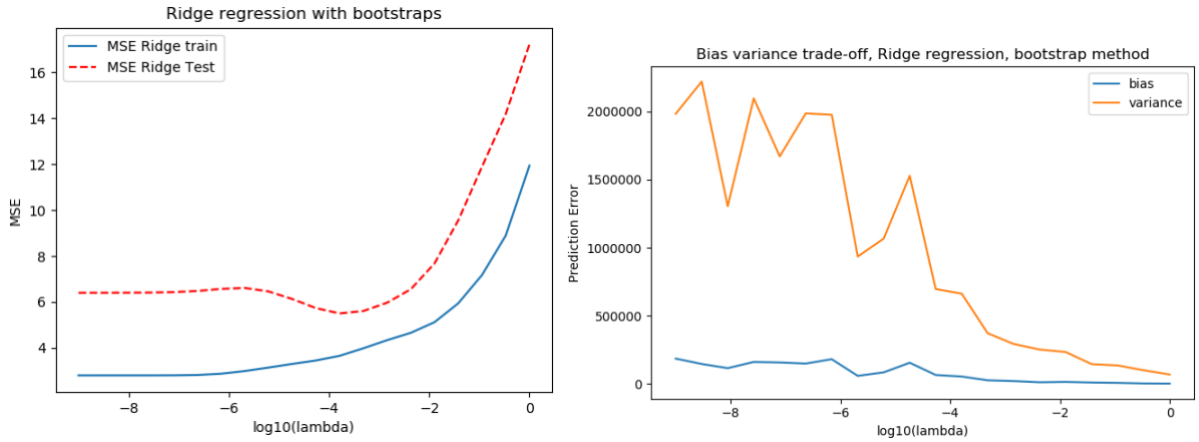

Figure 14 (left): 200 bootstraps using our Ridge regression algorithm on the real terrain dataset. Polynomial degree n = 7. 20x20 datapoints. No log plot on the y-axis. Figure 15 (right): 200 bootstraps using the Ridge regression algorithm on the real terrain dataset. Same parameters as in figure 14.

3.6.5 Ridge Regression with cross-validation

With K-fold cross-validation we see that the model fits better and better with increasing lambda, but the MSE is still extremely high. Also here we have to assume that the model struggles with handling the terrain data.

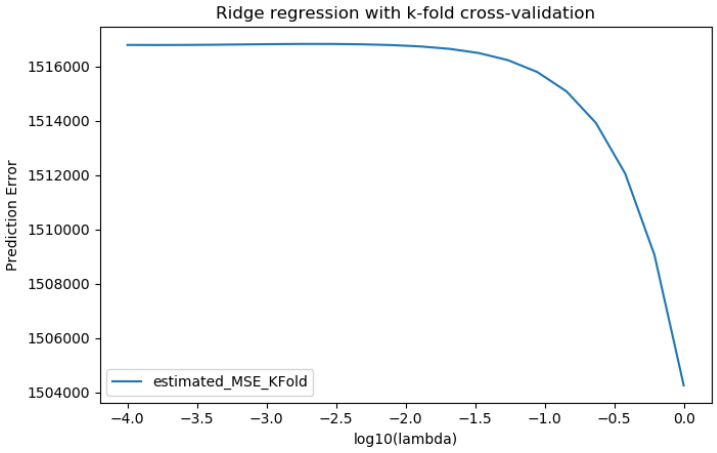

Figure 16: Ridge regression with cross-validation on the real terrain dataset. Here we used 5 folds, n = 7 polynomial degrees. 20x20 dataset.

3.6.6 Lasso Regression with bootstraps

The Lasso regression with bootstraps on the terrain data performs about equal to the OLS method for low lambdas but never starts to perform better at any point as seen in figure 17. And in figure 18 we see that the bias now is extremely high, and the variance actually increases with increasing lambda. This algorithm doesn't seem to handle the terrain data well either.

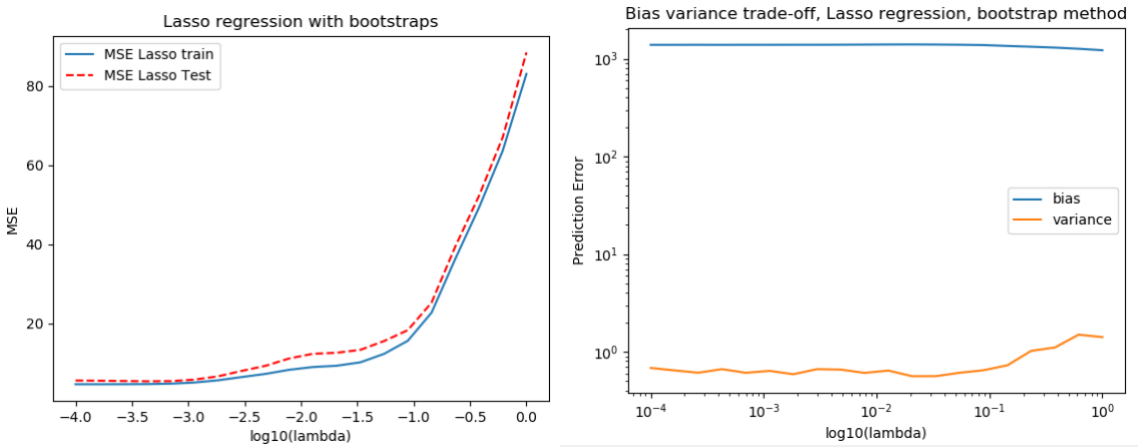

Figure 17 (left): 200 bootstraps using sklearn's Lasso regression algorithm on the real terrain dataset. Polynomial degree n = 7. 20x20 dataset. Figure 18 (right): 200 bootstraps using the Lasso regression algorithm on the real terrain dataset. Same parameters as in figure 17.

3.6.7 Lasso Regression with cross-validation

The K-Fold cross-validation algorithm also fails to improve the Lasso regression method on the real terrain data. In figure 19 we see for the first time in this report that the test MSE increases with increasing lambda. This is yet another clue that points to this method being unsuited for the terrain dataset.

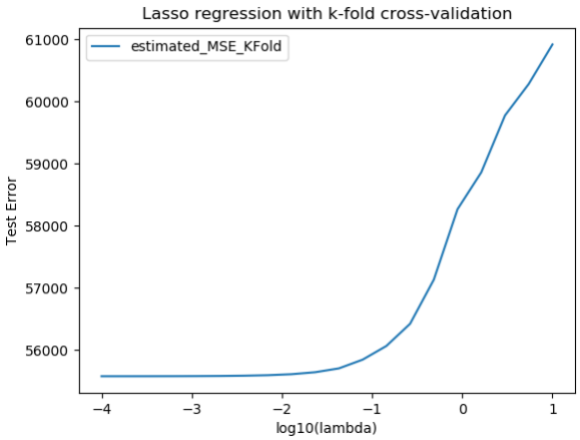

Figure 19: Lasso regression with cross validation on the real terrain dataset. Here we used 5 folds, n = 7 polynomial degrees. 20x20 dataset.

4 Conclusion

Looking first at the Franke function data we learned that with the OLS algorithm, the bootstrap resampling technique got slightly better results than the k-fold cross-validation technique, but nothing conclusive. With the Ridge regression algorithm, we learned that the bootstrap method performed more than three times better with an MSE of 0.078 compared to the cross-validation MSE of 0.270. Lastly, with the Lasso regression algorithm, we learned that the cross-validation MSE was approximately 50% higher than with the bootstrap method. From these results, we have to conclude that in these cases combined the bootstrap resampling technique outperforms the cross-validation technique.

Moving on to the real terrain data we learned that with the OLS algorithm, the bootstrap analysis and the cross-validation analysis produces very similar MSEs and that overfitting happened slightly quicker with cross-validation. We Ridge regression we saw that the bootstrap analysis performed around 250000 times better than the cross-validation analysis, but with these numbers, we can only assume that something has gone terribly wrong for the ridge regression on the real terrain data. The results with Lasso regression also give extremely high values and the parameters also behave strangely with variance increasing in the bootstrap analysis and the MSE increasing in the cross-validation analysis. Looking past these unexpected values we still see that the bootstrap resampling technique seems to perform better than cross-validation again.

After comparing the three algorithms together we learned that for the Franke function data Ridge regression produced lower MSE than OLS in lambda range around $10^{-3.5}$, and that OLS performed better than Lasso regression for all lambdas. On the real terrain data, we learned that the Ridge regression produced lower MSE than OLS in the lambda range of 10^{-4}, but that both Ridge and Lasso regression ran into problems and ended up being too unstable compared to OLS. To conclude, Ridge

regression came out as the winner on the Franke function data, but the simplicity of OLS came out as the winner on the terrain data.

Future work:

In the future, it will be interesting to find out how these different algorithms compare when dealing with much larger datasets, and if Ridge and Lasso get the same problems on the terrain data. It would also be interesting to see how the algorithms compare with other polynomial degrees, as n = 7 was mostly used here. How cross-validation compares to the bootstrap method on larger datasets also remains to be seen.

5 Appendix

Here you can find the terrain data and the python codes used in addition to a test of input and output for each code:

https://github.com/gery2/FYS-STK4155---Project-1

6 References

Project 1, *Regression analysis and resampling methods,* Department of Physics, University of Oslo, Norway, Fall semester 2020: https://compphysics.github.io/MachineLearning/doc/Projects/2020/Project1/html/Project1.html

Jim Frost, *Ordinary least squares* 2020, accessed 9 October 2020: https://statisticsbyjim.com/glossary/ordinary-least-squares/#:~:text=Ordinary%20least%20squares%2C%20or%20linear,and%20the%20corresponding%20fitted%20values.

NCSS Statistical Software, Chapter 335 *Ridge Regression* n.d, accessed 9 October 2020: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf

Jason Brownlee, A gentle Introduction to the Bootstrap Method 2018, accessed 9 October 2020: https://machinelearningmastery.com/a-gentle-introduction-to-the-bootstrap-method/#:~:text=The%20bootstrap%20method%20is%20a%20statistical%20technique%20for%20estimating%20quantities,after%20they%20have%20been%20chosen.

Jason Brownlee, A Gentle Introduction to k-fold Cross-Validation 2018, accessed 9 October 2020: https://machinelearningmastery.com/k-fold-cross-validation/#:~:text=Cross%2Dvalidation%20is%20a%20resampling,is%20to%20be%20split%20into.