Contents

List of Figures

List of Tables

1 Abstract

2 Introduction

The purpose of this article is to explore three different linear regression algorithms, their utility, application and some of the theory behind them. These are Ordinary Least Squares (OLS), Ridge regression and LASSO regression. These methods are tested on a model dataset constructed using the Franke function (2), then later applied to digital terrain data over South Korea in an attempt to accurately recreate the image. We explore the two resampling techniques K- fold cross validation and Bootstrapping. Studying the bias- variance trade off in model selection. Resampling becomes especially useful when evaluating the digital terrain data, with its limited resolution.

All methods are implemented using the Python programming language. The programs are available at the Github address

3 Methods

3.1 Linear Regression

The aim of Linear Regression is to find the conditional distribution $p(y|\boldsymbol{x})$ for y given $\boldsymbol{x}$. In Linear Regression, a linear functional dependence is assumed, and the aim is hence to create a linear function that fits a set of p predictor variables $\boldsymbol{x}$ to a target variable $\boldsymbol{y}$, where there are n discrete observations for each of the p predictor variables and the target variable. That is, the aim is to find a functional relationship of the form $f(\boldsymbol{x}) = y$.
We assume hence that each target variable y_i has a functional dependence of the predictor variables:

$$y_i = \tilde{y}_i + \epsilon_i = \beta_0 + \beta_1 x_{i0} + \beta_2 x_{i1} + ... + \beta_p x_{i(p-1)} + \epsilon_i$$

As this is true for all y, this can be written in matrix form:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\boldsymbol{y}$, and $\boldsymbol{\epsilon}$ are vectors of length n, $\boldsymbol{\beta}$ is of length p and $\boldsymbol{X}$ is a matrix of size $n * p$. X is called the design matrix. We now want to find fitting parameters $\boldsymbol{\beta}$ that represent the best linear fit of y to the input data $\boldsymbol{x}$. Hence, we want the fit

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

to be as close to the target data $\boldsymbol{y}$ as possible. There are several approaches to define the best linear fit, and in this article, we looked at three different methods that define the best fit in a different way each. All methods have in common that they define a loss function, which describes the deviation from the fit to the real data, that needs to be minimized.

3.1.1 Ordinary Least Squares Regression

In Ordinary Least Square regression, the loss function is defined as the following:

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

The solution minimizing this problem is found by deriving $C(\boldsymbol{\beta})$ with regards to each β_i and setting each derivative equal to zero. Doing this, we end up with the following equation (in matrix notation)

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

which gives us

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

which can be solved when $(\boldsymbol{X}^T\boldsymbol{X})$ is invertible. In the case of strongly correlated data and large matrices, There might be issues in inverting $(\boldsymbol{X}^T\boldsymbol{X})$ due to the matrix being singular. Also, for very large matrices, there might be issues caused by the fact that many mathematical operations can introduce a numerical error. Hence, an equivalent, but more stable way to implement the solution for $\boldsymbol{\beta}$ is to use Singular Value Decomposition (SVD). As every matrix has a SVD decomposition, we can write

$$\begin{aligned}
\boldsymbol{\beta} &= \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y} \\
&= \left(\boldsymbol{V}\boldsymbol{D}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T\right)^{-1}\boldsymbol{V}\boldsymbol{D}^T\boldsymbol{U}^T\boldsymbol{y} \\
&= \left(\boldsymbol{V}\boldsymbol{D}^T\boldsymbol{I}_p\boldsymbol{D}\boldsymbol{V}^T\right)^{-1}\boldsymbol{V}\boldsymbol{D}^T\boldsymbol{U}^T\boldsymbol{y} \\
&= \left(\boldsymbol{V}\boldsymbol{D}^2\boldsymbol{V}^T\right)^{-1}\boldsymbol{V}\boldsymbol{D}^T\boldsymbol{U}^T\boldsymbol{y} \\
&= \boldsymbol{V}\boldsymbol{D}^{-2}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{D}^T\boldsymbol{U}^T\boldsymbol{y} \\
&= \boldsymbol{V}\boldsymbol{D}^{-1}\boldsymbol{U}^T\boldsymbol{y}
\end{aligned} \tag{1}$$

.

Under the assumption that true data y is given as

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

where $\boldsymbol{\epsilon}$ follows a normal distribution $N(0, \sigma^2)$, information about the variance and the expectation value can be deduced. Assuming that $f(\boldsymbol{x}) = \boldsymbol{X\beta}$, it can be shown that

$$\mathbb{E}(y_i) = \boldsymbol{X_{i,*}\beta}$$

$$Var(y_i) = \sigma^2$$

where $\boldsymbol{X_{i,*}}$ is the i-th row of the design matrix.

For β, it can be shown that

$$\mathbb{E}(\boldsymbol{\beta}) = \boldsymbol{\beta}$$

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

3.1.2 Ridge Regression

The above mentioned problems with OLS regression, namely those of correlated data and near singular matrices, can be amended by implementing ridge regression. This is done by introducing the regularization parameter λ in the cost function:

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

effectively imposing a penalty on the size of $\boldsymbol{\beta}$. This has the effect of decreasing the variance of the parameters β, and thereby reducing the impact of overfitting.

The expression for the optimal value of $\boldsymbol{\beta}$ becomes

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

Where as before the matrix $\boldsymbol{X}^T\boldsymbol{X}$ could have been singular, by adding a non- zero, positive value to the diagonal, the eigenvalues become positive definite ensuring that the matrix is invertible.

Regularization parameter There are several factors that determine how different regularization parameters λ will minimize the cost function. A dataset with a lot of noise will cause more overfitting at higher model complexities. We expect that increasing the noise parameter σ in a model dataset will make models with a larger value of λ, perform better. We see an example of this in figure 1, where models with higher values of λ give the smallest MSE with increasing noise.

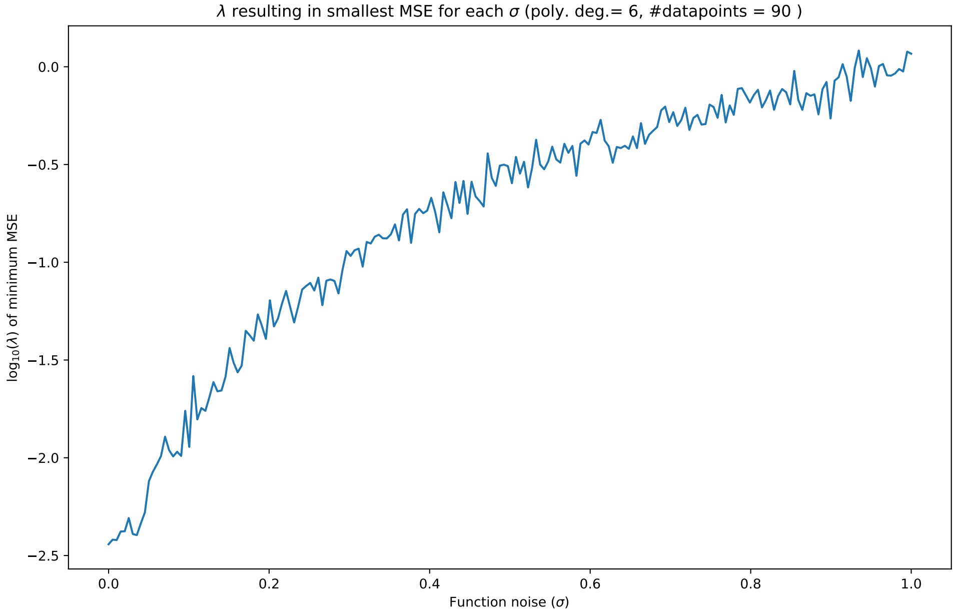

Figure 1: Performing Ridge regression on the Franke function (2), with a normally distributed error term $\sigma\epsilon$.

3.1.3 LASSO Regression

As with ridge, LASSO regression implements a new component to the cost function. Only, in this case the penalty is proportional to the L1- norm $||\boldsymbol{\beta}||$:

$$C(\boldsymbol{X},\boldsymbol{\beta}) = \frac{1}{n}\left\{(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{\beta})\right\} + \lambda||\boldsymbol{\beta}||.$$

One of the problems with ridge regression is that rather than eliminating covariant features, it will tend to shrink parameters, reducing their influence but not removing them. This is due to the behaviour of the squaring term in the L2- norm, which penalizes few large numbers more than many small numbers. For example, $||[\beta_1,\beta_2]||_2 \leq ||[\beta_1+\beta_2,0]||_2$. Whereas in LASSO, the area of the L1- norm in the parameter space is more likely to intersect the least square loss function along the orthogonal axes in the paramater space, along which, at least one of the parameters are zero. This has a greater effect on eliminating features in a model, although which of two collinear parameters are eliminated is arbitrary.

3.2 Resampling Methods and Data Splitting

In order to get an approximate estimate of the quality of a fit, it is necessary that the fit is tested on a set that it has not been trained on. As the training set determines the parameters $\boldsymbol{\beta}$, we need a set from the same distribution as the training set, which is different from the training set. For that reason, the full data set of predictors and targets is split into a test set and a training set. The test usually contains $20-25\%$ of the full set, while the training set contains $75-80\%$. It is also possible to use a further validation set to do a final estimate using data that has never been "seen" by the programmer or the code, however, this was not implemented in this project.

3.3 Bias-variance relationship

In order to evaluate the quality of a fit, it is customary to evaluate the Mean Square Error (MSE) of the test data to the prediction. The Mean Square Error is defined as

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = n^{-1} \sum_{i}^{N} (y_i - \tilde{y}_i)^2 = \mathbb{E}\left[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2\right]$$

where the left formula is the within-sample mean square error. The Mean Square Error can be decomposed into three parts: The unavoidable Error σ^2, the squared Bias (which is the deviation of the mean value of an estimator and the true value), and the Variance (which is a measure of the spread of the estimator). Hence

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = Bias[\hat{\boldsymbol{y}}]^2 + Var[\tilde{\boldsymbol{y}}] + \sigma^2 = \Big(f - \mathbb{E}(\tilde{f})\Big)^2 + \mathbb{E}\Bigg(\Big[\mathbb{E}\Big[\tilde{f}\Big] - f\Big)^2\Bigg] + \sigma^2$$

A proof of this decomposition can be found in the appendix. In this article, we will however calculate the following values:

$$Bias[\tilde{\boldsymbol{y}}]^2 = n^{-1} \sum_{i}^{N} (y_i - avg(\tilde{\boldsymbol{y}}))^2$$

$$Var[\tilde{\boldsymbol{y}}]^2 = n^{-1} \sum_{i}^{N} (\tilde{y}_i - avg(\tilde{\boldsymbol{y}}))^2$$

This is because the true values f_i are not known and impossible to obtain.
An important observation when fitting the data is that the Bias decreases as the model gets more complex, while the variance increases. When the Bias is large, important relations of the underlying data are not considered. When the Variance is high, small, unimportant features in the training set have too high of an impact. This is called overfitting. The aim is hence to find the model complexity that gives the lowest MSE. This is usually the case around the point where the Bias has the same magnitude as the Variance.

3.3.1 Bootstrap method

The Bootstrap method is a method that can be used to calculate statistical quantities. The main idea is the following: Let n be the size of the training set $Z = (z_1, z_2, ...z_n)$, where each z_i contains p predictors and one target value, that is $z_i = (x_{i,1}, x_{i,2}, ...x_{i,p}, y_i)$. Now, n data points z_i are drawn randomly from Z with replacement. This is done B times, creating B new sets Z'_j. We can now create a fitting model for each of the sets Z'_j. For each of these data sets Z'_j, we can now calculate any variable $\hat{\theta}_j$ given the data. Now, the distribution of $\hat{\theta}_j$ can be used to estimate the distribution of θ and get values of interest, such as the mean value or the variance. In this article, Bootstrap is used to get information about the test MSE, variance and bias in order to evaluate the quality of the fit. This is done by using the B bootstrap sets Z'_j to calculate the fitting parameters β and calculate B fits to the test set. $\tilde{Y}_j$. From these B fits, we can then approximate the test MSE, variance and bias.
TODO: Talk some more about the mathematical validity of Bootstrap.

3.3.2 K-fold Cross validation

K- fold Cross Validation is a useful technique for when you have a limited data set, and want to make sure that all features are captured in a training set. K- fold Cross Validation does this by partitioning the data set into k so called folds, assigning $k-1$ folds as the training set and 1 of the folds as the testing set. It does this k times, making sure to utilize each fold as the testing set once, as seen in figure 2. In this way, we end up generating k different models, whose performance can be measured with respect to the testing fold or additionally, a separate testing set withheld from the partitioning process.

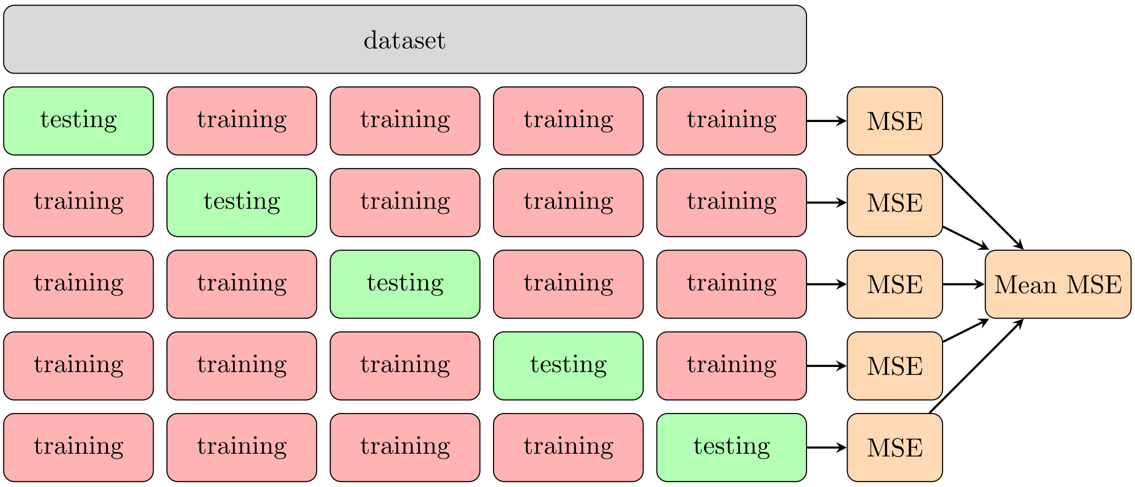
dataset
testing training training training training MSE
training testing training training training MSE
training training testing training training MSE
training training training testing training MSE
training training training training testing MSE
Mean MSE

Figure 2: In this example, 5- fold Cross Validation is used on the entire dataset, with one model being generated for each row. The MSE on the test set is then calculated and averaged with the MSEs of every other model.

The value we choose for k will impact the accuracy of our model. To see this, we performed OLS regression on the same function (2) using different values for k resulting in figure 3.

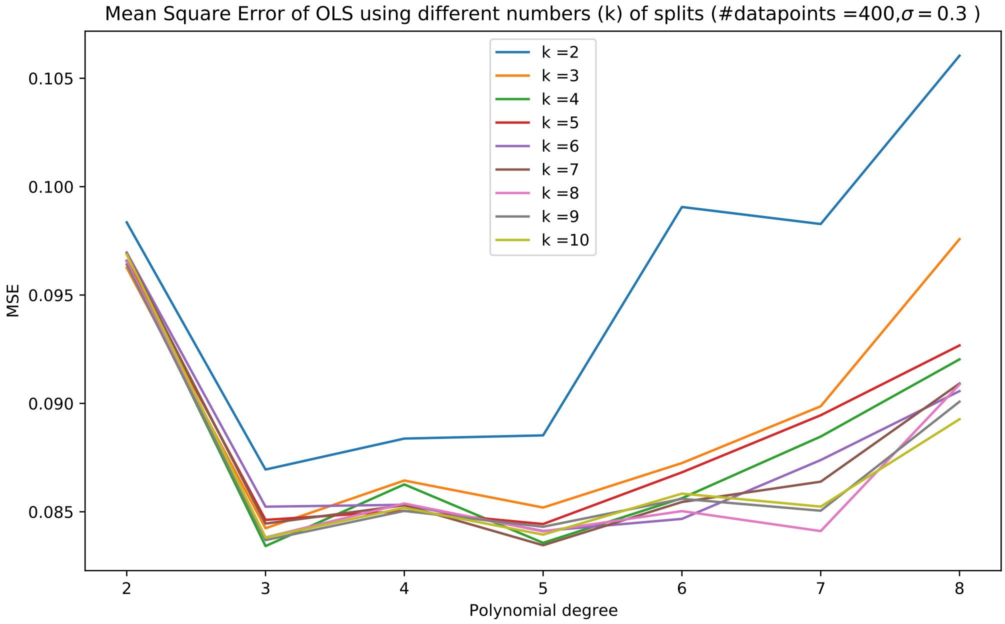

Figure 3: Mean squared error as a function of model complexity (polynomial degree) for different values of k, using k- fold CV OLS on the Franke function (2). We can see that higher values of k consistently perform better though there is no clear correlation among the highest few values.

3.4 Fitting the Franke function

3.5 Korea Data

4 Computational Implementation

4.1 Finding the ideal parameter λ

In order to find the ideal parameter λ for both LASSO and Ridge Regression, we looped over a set of lambda-values in different magnitudes and estimated the MSE using 4-fold Cross Validation. We chose to use the parameter λ that gave the lowest MSE, averaged over two uncorrelated runs. This was repeated for each fit, that is, λ was recalculated for each new Design Matrix X. This is because the ideal λ value changes with model complexity.

4.2 Error in LASSO-Regression

As there is no way to find the parameters ***beta*** for LASSO-Regression analytically, numerical methods need to be used. Here, we used Scikit-Learn in order to solve the problem numerically. We have observed that the LASSO-method converges rather slowly. For this reason, we have chosen the tolerance to be equal to 0.03 for fitting the Franke function, while the number of maximum iterations is chosen to be 10^5. We observed that decreasing the tolerance to 0.01 lead to no convergence even at 10^5 iterations. Similarly, we used XXX as tolerance at 10^5 maximum iterations for fitting the country data. As this is very time-consuming, we found it impossible to find the ideal parameters ***beta*** for LASSO-Regression. Each time the LASSO-fit was higher than the OLS fit, no ideal parameters have been found, as the ideal LASSO-fit cannot be no larger than the OLS-fit (because $\lambda = 0$ is the OLS fit).

4.3 Sampling Franke Function

The Franke Function is given by

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

and serves as a test function to check how well the implemented methods beform. We sample randomly points x_i and y_i between 0 and 1 and calculate the target points

$$z_i = f(x_i, y_i) + \epsilon_i$$

where ϵ_i follow a normal distribution with standard deviation σ.

4.4 The Design Matrix

The Design Matrix implemented is a 2D-version of the Vandermonde Matrix, hence, approximate z_i as, for a given degree k

$$\tilde{z}_i = \beta_0 + \beta_1 x_i + \beta_2 y_i + \beta_3 x_i^2 + +\beta_2 x_i y_i + ... + \beta_p y_i^k$$

In the practical implementation, we "drop" β_0 by removing the first column from the Design Matrix, as our data is scaled.

4.5 Scaling?

The design matrix is scaled using Scikit-Learn's StandardScaler.

5 Results

5.1 Error comparison

5.1.1 Bootstrap vs. Cross validation

We're evaluating the performance of Bootstrap and K- fold cross validation in terms of MSE on the same dataset using a fixed value for σ, a fixed number of data points and fixed model complexity. The objective being to compare the two resampling methods, varying only the amount of folds used in K- fold cross validation. Using a dataset constructed from the Franke function, we get figure 4.

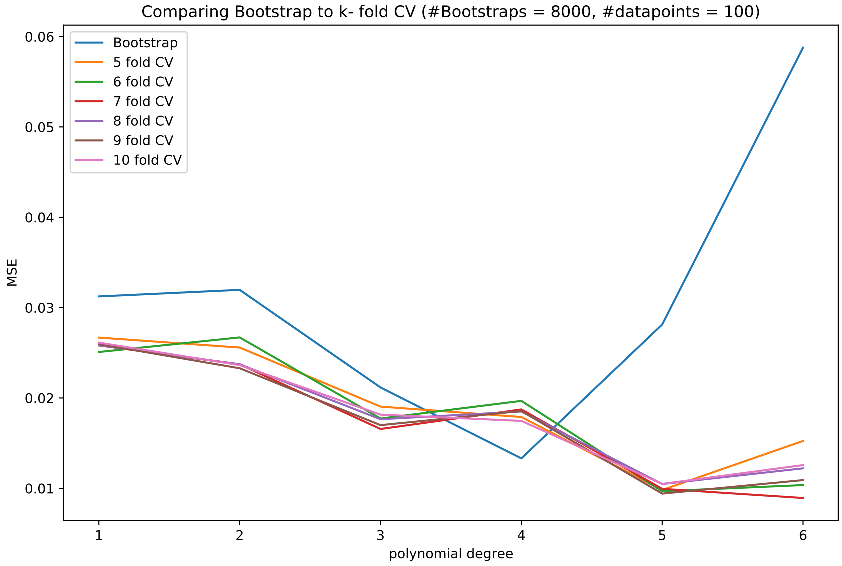

Figure 4: Mean squared error as a function of model complexity (polynomial degree) for different values of k, using k- fold CV and Bootstrap OLS on the Franke function (2). We see that all iterations of K- fold cross validation perform better than 8000 bootstraps on this very limited data set of 100 entries

Trying again, this time with 10 times as many data points. We get figure 5

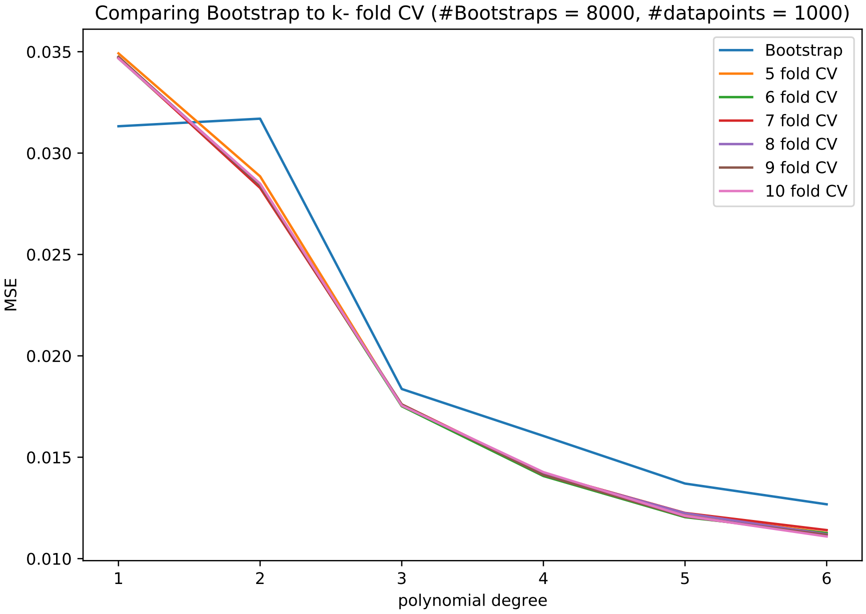

Figure 5: Mean squared error as a function of model complexity (polynomial degree) for different values of k, using k- fold CV and Bootstrap OLS on the Franke function (2). Again, all iterations of K- fold cross validation perform better than 8000 bootstraps. However, increasing the number of data points made the difference significantly smaller, especially for the more complex models.

Figures 4 and 5 give the impression that on a sufficiently large dataset, bootstrap competes with K- fold cross validation, if we can only increase the number of bootstraps sufficiently. This comes at a computational cost however and we've yet to quantize this.
Using the same model as above, we run K- fold cross validation with $k = 10$ and compare the

elapsed time with Bootstrap. The result is shown in figure 6.

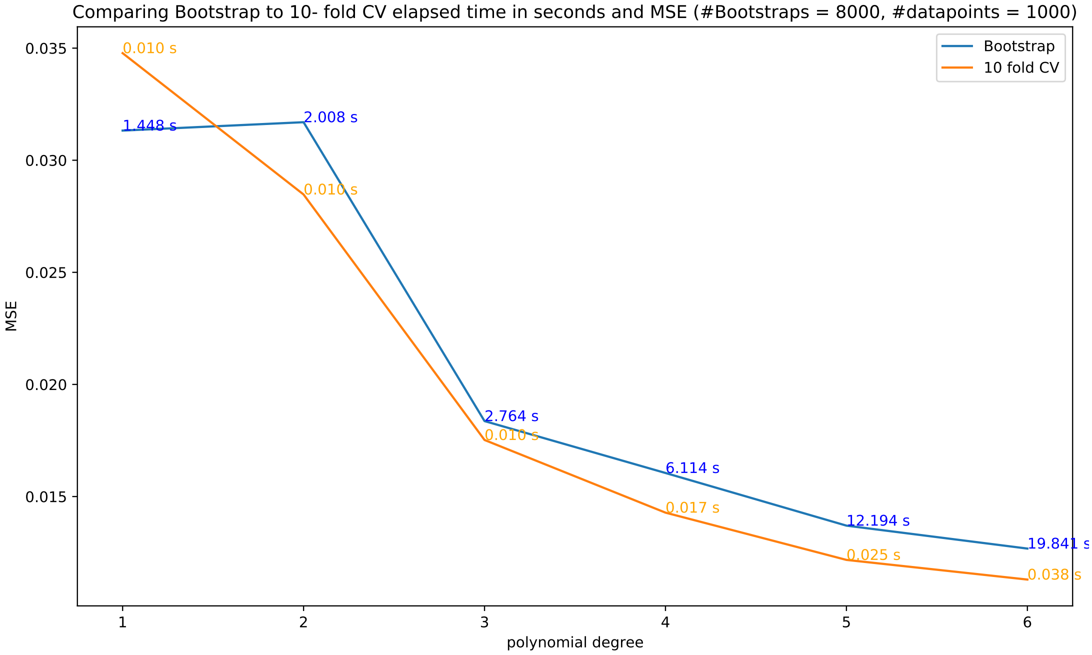

Figure 6: MSE as a function of model complexity for $k = 10$, using k- fold CV and Bootstrap OLS on the Franke function (2). The time elapsed for each model is plotted. Not only does K-fold CV take considerably less time, but the time increases much slower with increased model complexity compared to Bootstrap.

XXXXXXXXXXXXXXXXXXXXXX CAN THIS BE RIGHT? IT SEEMS RIDICULOUS. MAYBE I'M TIMING AT THE WRONG POINTS, ALTHOUGH I'M PRETTY SURE THEY'RE CORRECT XXXXXXXXXXXXXXXXXXXXXXXXXX

5.1.2 Franke Function

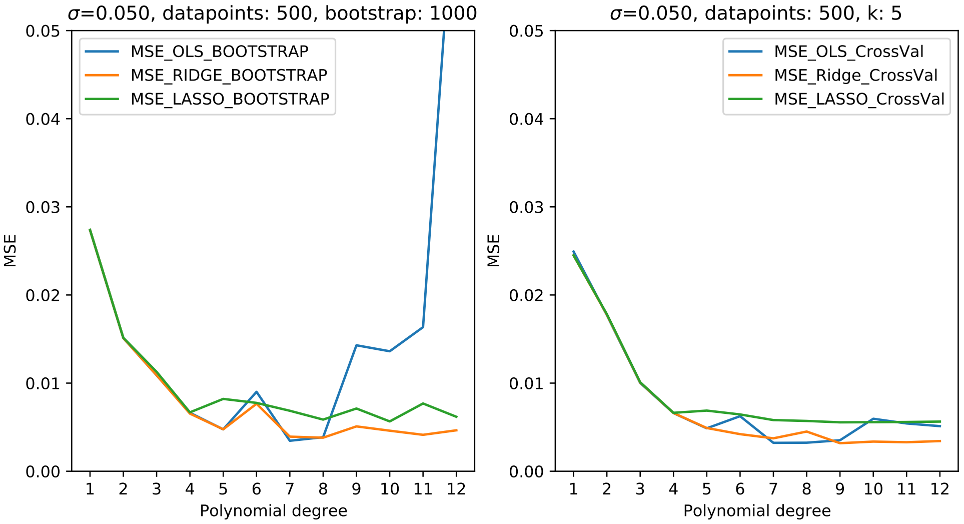

Figure 7: Test MSE for LASSO, Rigde and OLS regression. Estimated using Bootstrap (left) or k-Fold Cross validations. The exact parameters can be found in the graph's title.

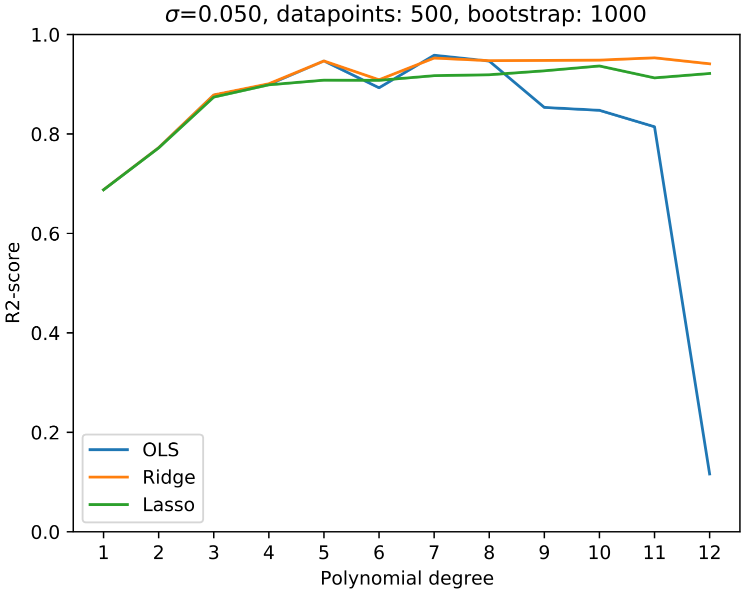

Figure 8: Test R^2-value for LASSO, Rigde and OLS regression. Estimated using Bootstrap (left) or k-Fold Cross validations. The exact parameters can be found in the graph's title.

As expected from the theory, increasing the model complexity lead to a decreased Mean-Square-Error and an increased R2-Score for all three methods when fitting the Franke Function. However, as can be seen in figures 7 and 8, the mean square error for the Ordinary Least Square approximation suddenly raises sharply (and the R2 score drops sharply) after some complexity is reached: This behaviour cannot be observed with LASSO and Ridge. Neither LASSO nor Ridge give conceivably better results than OLS when looking at the complexity giving the best results, that is, with the chosen parameters, OLS, LASSO and Ridge regression are about equally fast. As OLS is the most basic method, easiest to implement and also fastest, we figured out that it is the most suitable method for this problem - adding more complexity while avoiding a high variance (Ridge or LASSO) does not improve the quality of the fit. Another interesting observation is that there is much more fluctuation in the MSE when using bootstrap than when using 5-fold Cross validation. We are not quite sure why this happens, but we assume that one possible cause is that when using bootstrap, some very ”undesirable” combinations of values might be chosen, which leads to a very high Bootstrap. Thus, the graphs might look smoother when running more Bootstrap-samples. Another reason is that the training set in the Bootstrap method is not equally large as the training set when using 5-fold cross validation (75% vs 80%). We also observed that Bootstrap is more time-demanding than k-fold Cross Validation. The large deviation between LASSO and OLS is due to the aforementioned error.

5.1.3 Map data

5.2 Bias-Variance tradeoff

5.2.1 Franke Function

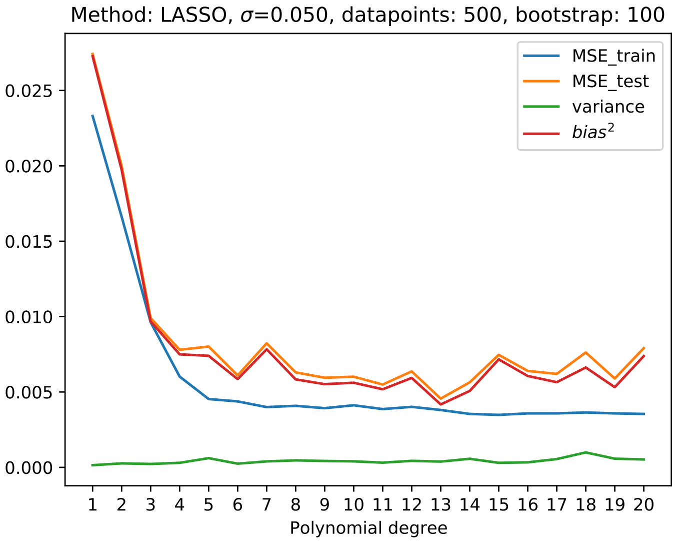

Figure 9: Bias, Variance, MSE & training MSE as a function of the polynomial degree using LASSO regression. The values for σ, the size of the dataset and the number of bootstraps can be found in the title.

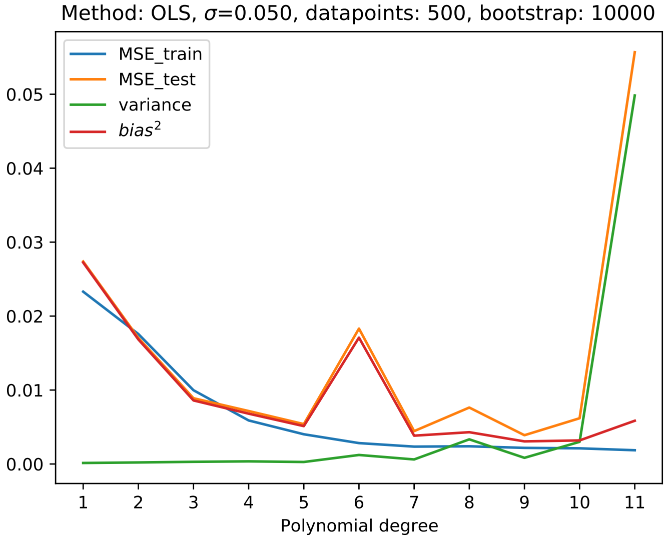

Figure 10: Bias, Variance, MSE & training MSE as a function of the polynomial degree using OLS regression. The values for σ, the size of the dataset and the number of bootstraps can be found in the title.

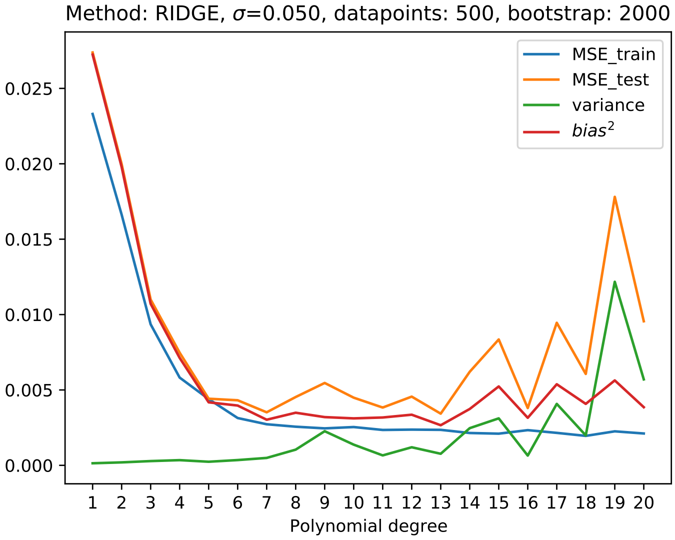

Figure 11: Bias, Variance, MSE & training MSE as a function of the polynomial degree using Ridge regression. The values for σ, the size of the dataset and the number of bootstraps can be found in the title.

Figures 9, 10 and 11 show the MSE, the Training-MSE, the Variance, and the bias for the Franke Function. In all figures, the aforementioned statement that the variance increases and the bias decreases with increasing model complexity, hold true. Especially for the OLS, we see that this resembles figure 2.11 on page 38 in Hastie et al. by a lot[1]. One can see that OLS is more prone to a higher variance at higher polynomial degrees, as is expected - shrinkage methods explicitly reduce the variance. There seems to be especially little variation in the LASSO approach, however, it is possible that this is due to the aforementioned issue with LASSO regression, as we have not found the ideal LASSO-coefficients due to the large error.

5.2.2 Map data

6 Conclusion

“I always thought something was fundamentally wrong with the universe” [1]

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.