A Study on Linear Regression Methods

Caspar William Bruenech

October 7, 2019

Abstract

Linear regression was studied using three different methods; ordinary least squares, ridge, and lasso. Their dependence on the number of data points available, the complexity of the model, and the value of the regularization parameter was analysed, resulting in a visual example of the bias-variance trade off in practice. While the ridge and lasso models were both observed to maintain stability (avoiding overfitting the data) for higher complexity regression models, the OLS model produced similarly low errors for the data analysed in this experiment. However, due to the dependency on using singular value decomposition when calculating the parameters in the OLS model to avoid singular matrices, it is uncertain which model is best suited when taking into account the numerical cost of the calculations. The analysis was also used on a real data set featuring elevation in terrain data, which produced good estimated of the general features of the terrain. Once again, the three different methods produced similar errors, while the OLS model could not maintain a low error for higher polynomial degrees, something both ridge and lasso managed to do.

1 Introduction

As one of the basic analysis tools in machine learning, linear regression allows us to create models to predict an outcome based on an input variable. This can be used for many situtations, such as predicting the housing cost in an area by looking at certain properties of that area, or getting an idea of how much of certain products a store is going to sell during a festivity. Linear regression is essentially used to explain a relationship between a dependent variable and one or more independent variables. The purpose of this experiment is to study various methods of estimating the parameters in linear regression. Three methods will be analysed: Ordinary Least Squares, Ridge Regression, and Lasso regression. The three methods will be compared in terms of accuracy of predicting the desired data. We will also look at how the bias-variance trade off plays a crucial role when doing machine learning, and how we can find the best suited model by tuning the model complexity and various regularization parameters. Finally, we will apply our methods to real terrain data, and study how we parametrize the terrain using linear regression.

2 Method

In order to perform a linear regression analysis, data is required. In this experiment, we will generate a dataset by evaluating the Franke function[1] $f(x, y)$ at $x, y \in [0, 1]$. By adding random, normally distributed noise to the data, a dataset on the form

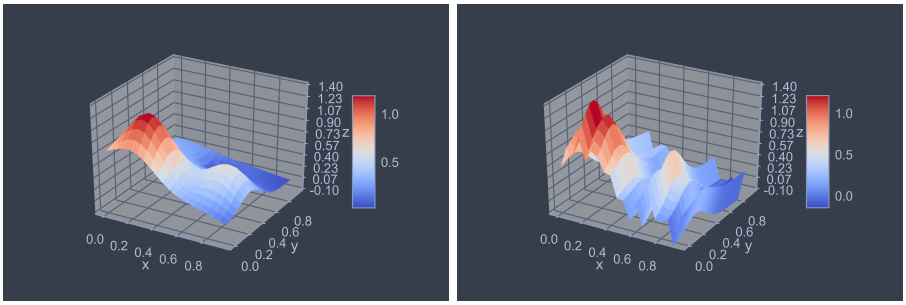

Figure 1: Two plots showing the Franke function data with and without added noise.

$$z = f(x, y) + \epsilon,$$

is produced. Our goal is therefore to use linear regression to approximate the true value $f(x, y)$ using polynomials of a certain degree. A plot of the data with and without noise is shown in figure 1

In linear regression, we want to approximate the true data f as set of linear equations using polynomial up to a degree p, i.e.

$$\tilde{y}_i = \sum_{j=0}^{p} \beta_j x_j^p$$

Where $\tilde{y}_i \in \tilde{y}$ is the predicted dataset based on the observational data, β are the parameters of the linear model, and $x_i \in x$ are the variables which we assume f is a function of. This can be written as a vector multiplication

$$\tilde{y}_i = X_{i,}\beta$$

Where X is the design matrix, which in the case of fitting to a polynomial, is the Vandermonde matrix for a polynomial of degree p with n observations:

$$X = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^p \\ 1 & x_1 & x_1^2 & \dots & x_1^p \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^p \end{bmatrix}$$

In our 2-dimensional case, the observations x are the values of x and y in the meshgrid in which the data is scattered. This means we have two-dimensional polynomials which we need to fit the data to. I.e.

$$\tilde{y}_i = \beta_0 + \beta_1 x_i + \beta_2 y_i + \beta_3 x_i y_i + \beta_4 x_i^2 + \beta_5 y_i^2 \dots \beta_{(p+1)^2} x_i^p y_i^p$$

2.1 Ordinary Least Squares

The first linear regression method we will look at is the Ordinary Least Squares-method (OLS). The idea of this method is to minimize the squared difference between the fitted model $\tilde{z}$, and the provided data points z. In other words, we want to minimize the value

$$\frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2 = \frac{1}{n}\left[(z-\tilde{z})^T(z-\tilde{z})\right]$$

We know that the predicted values can be expressed via the matrix-vector multiplication of the design matrix X and the parameters β;

$$\tilde{z} = X\beta.$$

Using this and denoting the resulting expression as a function C with β as the parameter, we get the so-called cost function:

$$C(\beta) = \frac{1}{n}\left[(y - X\beta)^T(y - X\beta)\right]$$

Since we want to minimize this function with respect to , we want to find

$$\frac{dC(\beta)}{d\beta} = 0$$

Which gives us

$$-2X\beta(y - X\beta) = 0$$

$$\beta = (X^TX)^{-1}X^Ty$$

So in order to find the optimal parameters which best fit our data set, we need to invert the matrix X^TX. If columns in the design matrix X are highly correlated, then the matrix X^TX is also nearly singular. To avoid this, we can calculate the β-parameters using the Moore-Penrose pseudoinverse and the singular value decomposition. Using that:

$$y = X^+\beta$$

Where X^+ is the pseudoinverse of X, combined with the singular value decomposition of X:

$$X = U\Sigma V^T$$

We get a new expression for the parameters β

$$\beta = V\Sigma^+U^Ty.$$

Another way to avoid a singular matrix is to add a small hyperparameter to the diagonal elements of the matrix X^TX. This gives us the new cost function

$$C(\beta) = \frac{1}{n}||(y - X\beta)||_2^2 + \lambda||\beta||_2^2$$

Which requires $||\beta||_2^2$ to be smaller than a value $t > 0$. This means that the resulting parameters are shrunk in magnitude depending on the value of λ. Which again means that the model will no longer be mainly driven by the features with the highest coefficient magnitude. . Finding the optimal parameters in ridge regression is done in the same way as OLS, by finding where the derivative of the cost function with respect to β is zero. The resulting expression is then

$$\beta^{Ridge} = (X^T x + \lambda I)^{-1} X^T y$$

By instead defining the cost function as

$$C(\beta) = ||y - X\beta||_2^2 + \lambda||\beta||_1$$

we are left with Lasso regression. This method is similar to Ridge in that it shrinks the parameters, however it also allows some parameters to be set to 0, which means that Lasso essentially does feature selection automatically. The problem with Lasso is that the minimum of the cost function does not have an analytical solution as with OLS and Ridge. Instead, a gradient descent has to be implemented in order to find the minimum of the Lasso cost function.

2.2 Variances and confidence intervals

To get an idea of how the different methods affect the resulting model coefficients, the 95 % confidence intervals for the parameters will be calculated. Since the ridge parameters are identical to the OLS parameters when the regularization parameter $\lambda = 0$, it is sufficient to find the expression for the variances of the ridge parameters, as we can also apply it for OLS. The variance of the ridge parameters is given as ([2]):

$$Var[\beta^{Ridge,OLS}] = \sigma^2 \left(X^T X + \lambda I\right)^{-1} X^T X \left(\left[X^T X + \lambda I\right]^{-1}\right) \tag{1}$$

Where σ^2 is the variance of the observational data. Since this is normally not known, we set it to 1. The variance of the i-th regression parameter is then given as the root of the diagonal elements ii of the resulting matrix. For the Lasso parameters, there is no analytical expression for the variances. This means we have to estimate them. To do this, we need a number of estimates for the parameters themselves (for example k estimated using a k-fold cross validation), which we can then use to calculate the mean and variance. To get the confidence interval for the different parameters in the different methods we can calculate

$$CI = 1.96 \times \sigma_i$$

Such that

$$\beta_i = \overline{\beta}_i \pm 1.96 \times \sigma_i$$

2.3 Resampling

In general, resampling methods are used for getting better estimates of the accuracy of the model. The name refers to the act of repeatedly drawing subsets (samples) of the original data, fitting the data to these subsets, and then performing calculations of the test statistics. These methods are very useful when data is sparse, as the amount of data is then not enough to train a reliable model. The simplest example of resampling is to split up the original data into a training and a test set, fit the model to the training set, and the calculating the prediction error on the test set. The resulting error estimate will then be closer to the error which the model will produce when it is used on new data. In this experiment we will look at one specific resampling method, namely the k-fold cross validation, which provides a good prediction of the true prediction error of the model. To perform a k-fold cross validation, the original data set is divided into k bins of equal size. The bins are then looped over, and for each iteration of the loop, a different bin is used as the testing (validation) set, while the rest are used for training the model. The prediction error (MSE in this case) is calculated for each model, before the model is discarded. This procedure is repeated for all k bins, before the average of the prediction error is returned. By performing the k-fold cv over an array of different models (for example combinations of hyperparameters and model complexities), one can then pick out the model which produces the smallest error, knowing that this will be a good estimate for when the model is applied to new data.

2.4 The Bias-Variance trade off

When fitting a model to a dataset, there is nothing stopping us (apart the higher numerical cost) from increasing the complexity of the model (in this case the degree of the polynomial) such that the mean squared error goes to zero, meaning that the fitted model *hits* every data point in the observations. The problem with this, however, is that the model will now be very bad at predicting data which is not in the set used for training the model. This is known as the model having high variance and low bias; the model will have low error on the training data, but high and inconsistent error on the test data. This is known as overfitting, and can happen when increasing the model complexity too much. On the other hand, if the model is not complex enough (for example fitting a straight line to a non-linear data set), the prediction error will be large, as the model is not a good approximation to the data. However, the variance is low, as the (high) error will be consistent for new data. The bias and variance are inversely correlated, meaning that as one increasing, the other tends to decrease. This is where the idea of the trade off comes in, as choosing the best model comes down to finding the one which has the smallest value of bias, variance, and error. To derive the formula for the bias and variance, we will assume we have some true data $f(x)$ which has some added noise ϵ, giving us the observed data as

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

From ordinary least squares, we know that we can approximate this function as

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

Where $\mathbf{X}$ is the design matrix, and β are the optimal regression parameters.

Substituting $\mathbf{y} = f(\mathbf{x}) + \epsilon$, followed by adding and subtracting $\mathbb{E}[\tilde{\mathbf{y}}]$ we get (f(x) will be abbreviated to f):

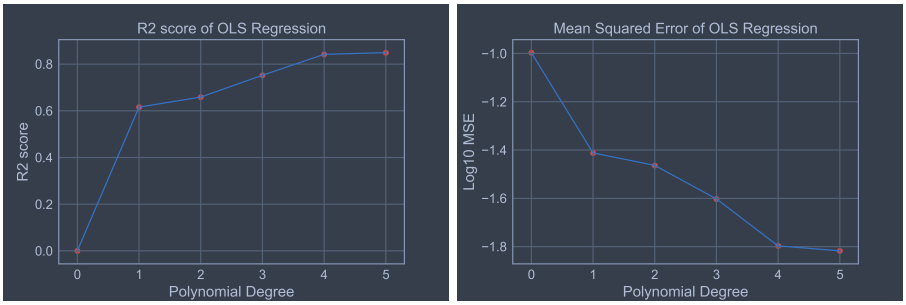

Figure 2: Plots of the R2 score and MSE score resulting from a standard OLS regression without resampling.

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

Since f is a deterministic value, we know that $\mathbb{E}[f] = f$. Additionally, we we have assumed that the noise ϵ has mean value of 0, which means that $\mathbb{E}[\epsilon] = 0$. By expanding the brackets and using this we are eventually left with

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

Where $\mathbb{E}[(y - \tilde{y})^2]$ is the total error in the fitted data, $(f - \mathbb{E}[\tilde{y}])^2$ is the so-called bias^2, which tells us how much the model deviates from the true data, $\mathbb{E}[\epsilon^2] = \sigma^2$ is the variance of the noise, which we usually don't know, and finally $\mathbb{E}[(\mathbb{E}[\tilde{y}] - \tilde{y})^2]$ which is the variance of the predicted data (the model). A visual example of this is shown in the analysis.

3 Implementation and Results

All the code written for this project can be found in the following jupyter notebook:

https://github.com/casparwb/FYS-STK3155/blob/master/1st_project/code.ipynb

The ordinary least squares regression was performed on the data set by calculating the analytical solutions to the parameters β. The resulting MSE and R2 score without resampling for varying model complexity is shown in figure 2.

A resampling was implemented in the form of a K-fold cross-validation with K=5 folds. OLS was then used on the training data to tune the parameters, before calculating the MSE on the test data. The result, compared to regression without resampling is shown in figure 3

Figure 4 shows three plots of the prediction error again as function of model complexity, featuring resampling by the cross-validation method, this time for model complexity of higher order than 5. Additionally, each plot shows the result with an increasing amount of data points.

Ridge regression was implemented using the analytical expression for the ridge parameters. The results were compared to OLS, which produced the figure 5

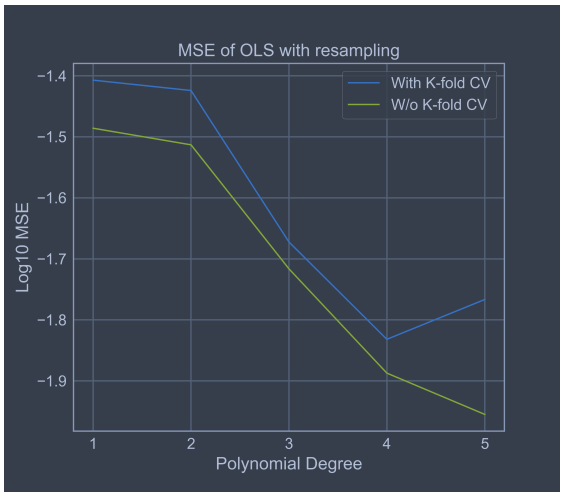

Figure 3: A plot showing comparing the MSE of a OLS regression as a function of model complexity, with and without simple resampling.

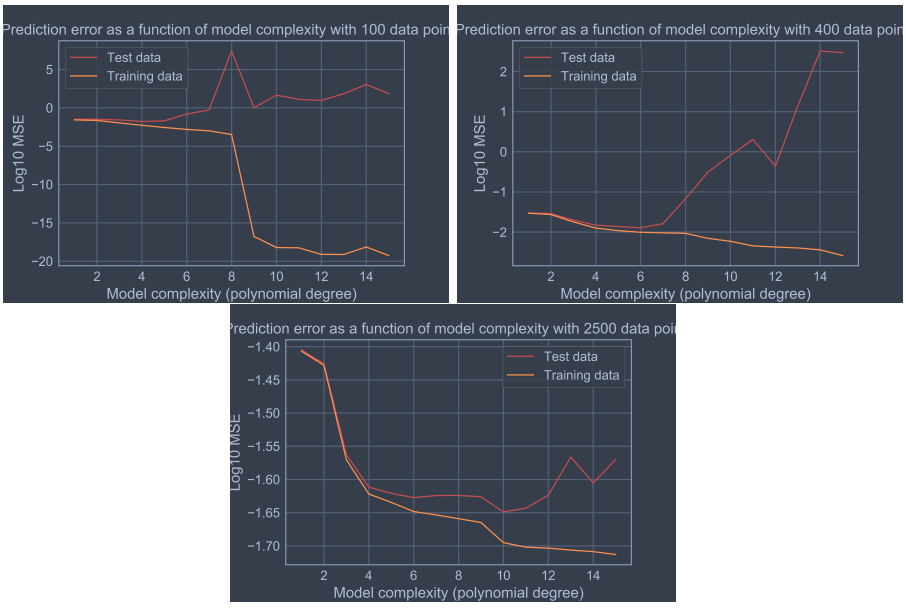

Figure 4: Plots of the R2 score and MSE score of the model on the training and test data as a function of model complexity. Each plot has a different number of data points.

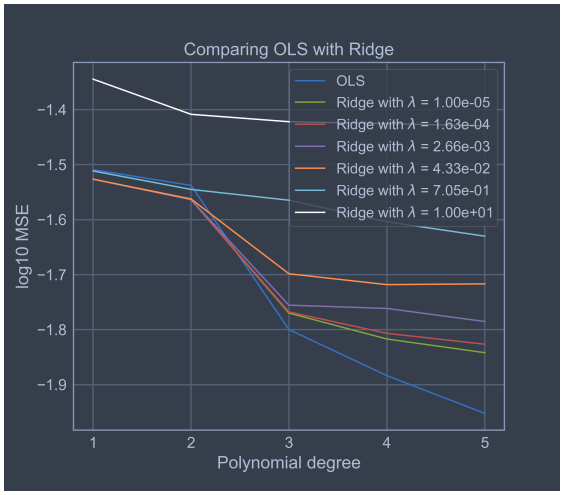

Figure 5: A figure showing MSE resulting from Ridge regression with various values of the hyperparameter λ, compared to the MSE of OLS regression

The K-fold cross-validation resampling method was then specialized for the case of Ridge regression. For each iteration of the K-fold, a variety of λ-values was tested, and the resulting error as a function of both model complexity and λ was produced. See figure 6

The same analysis was performed with Lasso regression. The prediction error was estimated for varying combinations of λ and the polynomial degree, with degrees up to 18. This result is shown in figure 7.

3.1 Confidence intervals

The variances for the parameters in OLS, Ridge and Lasso were estimated with a model complexity of polynomial degree 5 with varying degrees of the hyperparameters. The resulting plot in figure 8 shows then the confidence interval for $\beta_0, \beta_1, \ldots, \beta_{35}$.

3.2 Real data

The different regression methods we performed on terrain data gathered from [1]. Due to the size of the dataset and the limitations of the hardware used in this experiment, only a small selection of the data was used for analysis. A simple OLS regression was then used with polynomial degree 5, to visualize the parametrized curve. This is shown in figure 9.

The data was then parametrized with OLS, Ridge and Lasso. These results are shown in figure 10.

[1] https://earthexplorer.usgs.gov/

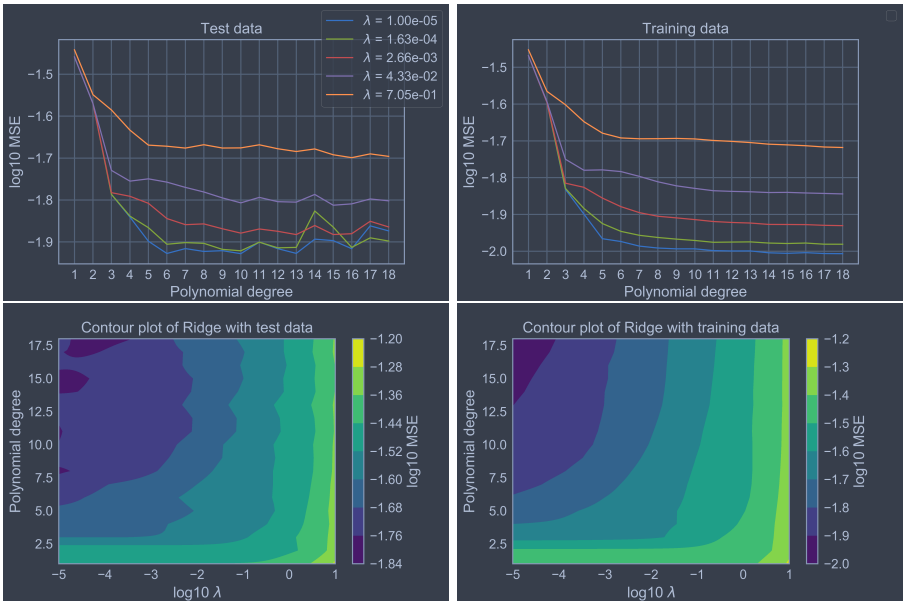

Figure 6: Plots showing how the λ-parameter affects the prediction error when using Ridge regression. The first show shows the error as a function of polynomial degree for a selection of λ-values, while the two contour plots show the continous plot of the error for all combinations of λ and polynomial degree. The first column shows the error when predicting the test data, while the second column shows when predicting the training data.

4 Analysis

4.1 OLS analysis

Figure 2 shows that the prediction error and R2 score of the OLS regression goes, respectively, down and up as the complexity of the model increases. This is to be expected, as we are simply increasing the polynomial degree, allowing to to get closer to more data points. Once we introduced resampling in the form of a 5-fold cross-validation, we suddenly see that the error is not only generally higher for all polynomial degrees in resampled case, but it also suddenly spikes when the model complexity reaches a polynomial degree of 4. This visualizes the problem of not dividing the data into a training and a test set, as it is not possible to know whether the model is over- or underfitting if this is not implemented. In this case, it appears as if the model is overfitting with polynomials of a degree higher than 4, while 4 is the model which best fits the data. This is even better shown in figure 4, where we also varied the number of data points in the data set. We see that in all the three plots, the prediction error of the training data always goes down as the complexity increases. In fact, there is no reason to even calculate the prediction error on the training data, since, as mentioned before, we can theoretically increase our model complexity indefinitely until the error goes to zero, but this will result in our model predicting poorly. The figure shows that the prediction error on both the training and test data decrease more or less similarly for the first few polynomial degrees, but at some point, the test data error always diverges from the training data and spikes. Where this spike happens, however, changes depending on how much data we are trying to fit the model to. With 100 data points, the two error start diverging at polynomial degree 4, while with 400 data points, it doesn't happen

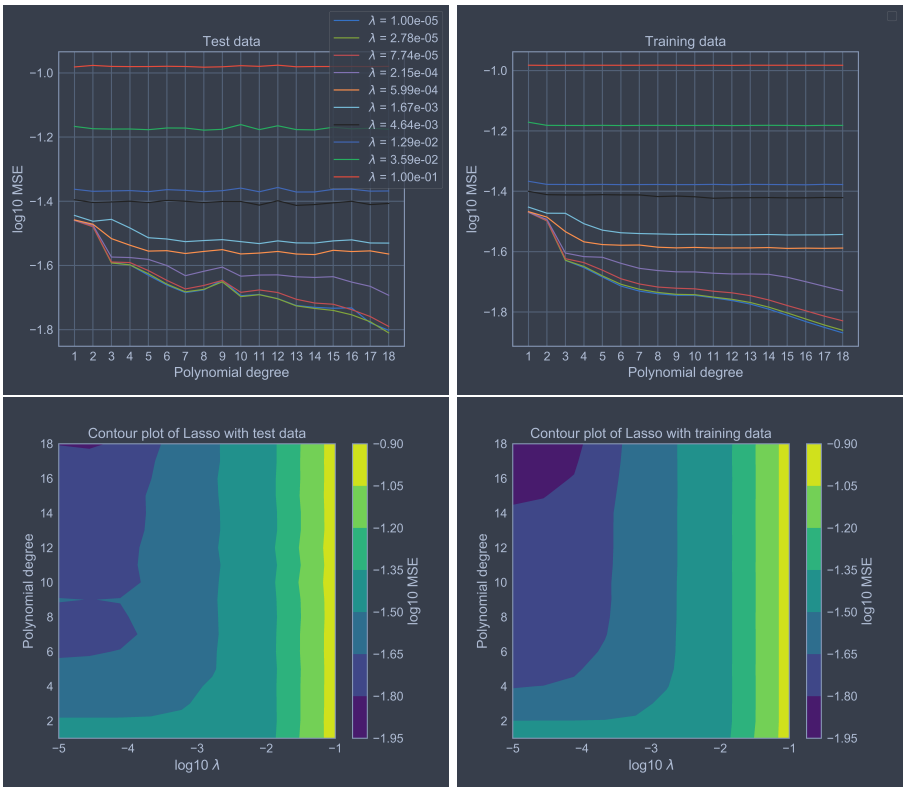

Figure 7: LASSO: Plots showing how the prediction error varies as a function of both the regularization parameter and the model complexity when using Lasso regression.

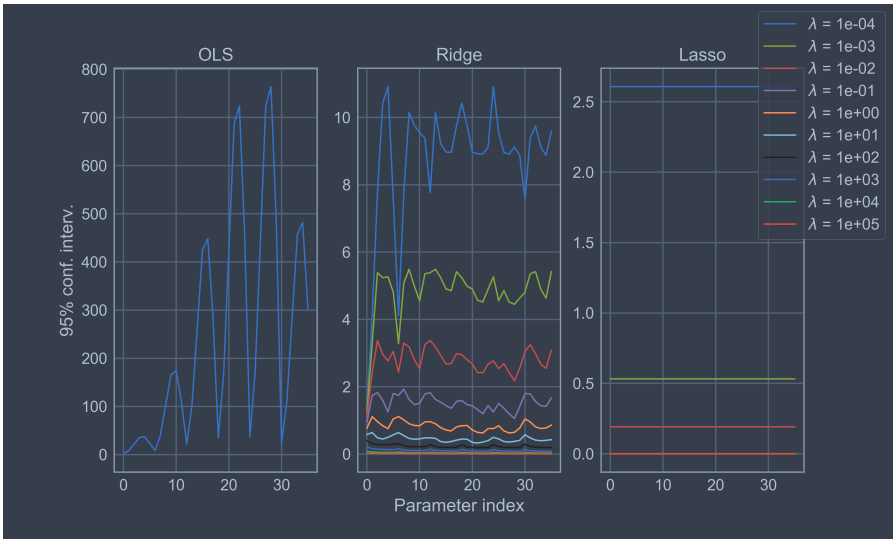

Figure 8: Three plots showing the confidence intervals for the parameters in each of the three regression methods.

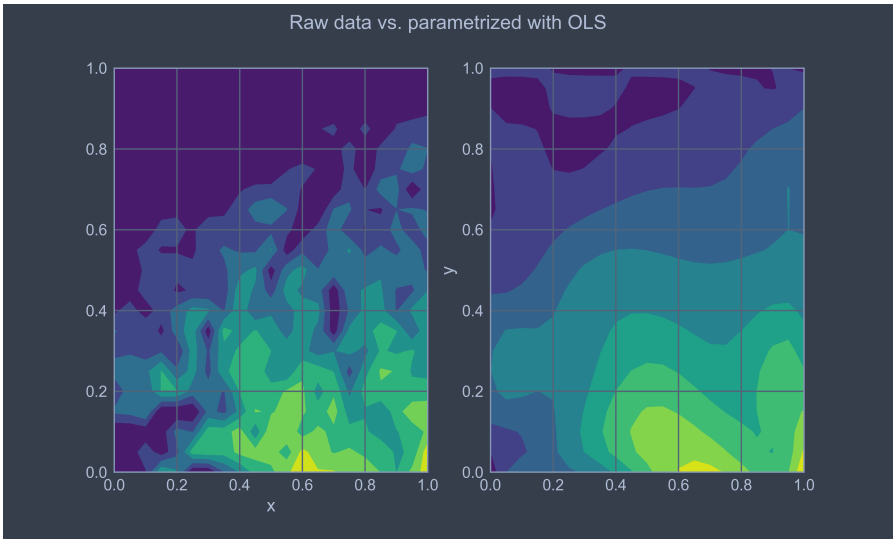

Figure 9: A figure showing a selection of the terrain data vs. the resulting parametrized data using OLS with polynomial degree 5. Lighter areas (yellow) correspond to higher altitudes in the terrain.

until polynomial degree 7. At 2500 data points, the testing error starts flattening out at degree 6, before spiking at around degree 10-12. In the plot with 100 data points, we also observe a massive negative spike in the prediction error for the training data at polynomial degree 8, before flattening out at higher orders. This is most likely the result of the model being able to hit more or less every point in the data set with a polynomial of degree 8. These results are then in line with that we would expect. More data points allows for higher complex models, while less data points increases the chance of overfitting at lower complexity. The plots also show the bias-variance trade off in practice. When we have a low complex model, we will produce results which are inaccurate, but which will not vary much. The model has low variance, but high bias, as the flexibility of the model is high. This is equivalent to the left hand side of the plots, where the estimated error is similar for both the training and the test data. On the other side, we have the case where the complexity of the model is so high that we are overfitting the data. This is very apparent in plot with 400 data points. In this case, the results will be accurate on average, but they will vary significantly from data set to data set. The variance of the model is high, but the bias is low since the flexibility of the model is so low. When this happens, we have essentially specialized the model for this specific data set, so since data is not usually the same, we will have a big difference in the results with small changes in the data we are fitting to. The ideal case is that in which the prediction error on both the training and test data is minimal, since this means we have optimized our model for multiple data sets, showing that the model is flexible enough to produce accurate results with varying data. This is the essence of the k-fold cross-validation, as it gives a good estimate of the prediction error when applying the model to new data by training the model on different combinations of the available observational data.

4.2 Ridge

In figure 5 we see how Ridge regression compares to OLS for various values of the parameter λ over the same model complexities. We see that as the value of λ increases, the error also decreases, with the highest prediction error corresponding to the lowest value of λ, and the smallest error corresponding to $\lambda = 0$, which is equivalent to standard OLS regression. To understand why the error is larger for Ridge regression, we need to go back to the what actually differentiates these two regression methods. As we saw when developing the ridge formula, the β-parameters in Ridge are calculated using the same formula as with OLS, except this with an added value of λ to the diagonal of the design matrix. By assuming the design matrix is orthonormal, we can write the expression for the Ridge estimators as

$$\beta^{Ridge} = (I + \lambda I)^{-1} X^T y$$

where y is the data set. This also means that the expression for the OLS-coefficients simplifies to

$$\beta^{OLS} = X^T y$$

Inserting into the expression for the Ridge coefficients we get

$$\beta^{Ridge} = \frac{1}{1+\lambda} \beta^{OLS}$$

Which shows that the Ridge model coefficients are simply the OLS coefficients shrunk by a factor of $1 + \lambda$. The Ridge method essentially adds bias to the model by shrinking the parameters, reducing the chance of overfitting since the coefficients are shrunk closer to the true value of the parameters. As we saw in the expression for the bias-variance trade off, increasing the bias also increases the total error.

An overview of the Ridge regressors dependence on λ is shown in the plots in figure 6. Here we see how the prediction error depends on λ and the model complexity, and how this changes the error on both the training and test data. All the models are trained with a total of 400 data points. When compared to the equivalent plots of the OLS regressor (figure 4), we see that as with OLS, the error decreases for the first few polynomial degrees as the complexity increases. However, instead of the prediction error on the test set blowing up at degree $> 5, 6$, the error actually stabilizes and remains more or less constant with a slight negative slope for all higher order polynomials. This is exactly the behaviour we would expect from the Ridge regressor, as the model coefficients which are responsible for the overfitting in OLS get shrunk by a factor related to the value λ, allowing the model to retain a low variance even for higher polynomials due to the bias added by the λ factor. We also see that the smaller the value of λ, the smaller the value of the prediction error at which it stabilizes. The contour plot shows the continuous prediction error as a function of both the polynomial degree and the value of λ. The plot visualizes the areas (combinations of λ and polynomial degree) which produces certain errors. It shows how you can tune the parameter to produce the same low error with varying model complexity. In other word, how you have both a low complexity and small λ, or a higher complexity with a higher λ, and still produce the same prediction error. This makes sense, as a higher value of λ shrinks the coefficients further, allowing a higher complexity model without risking overfitting.

To produce these results, the k-fold cross-validation resampling method was again implemented, and this time specialized for Ridge regression. This means that for each value of λ in the specified array, a 5-fold cross-validation was performed. By performing this operation for a selection of polynomial degrees, the cross-validation gives us an overview of the best model, which in this case means the best combination of λ and polynomial degree. This is, however, computationally expensive, as the model has to be trained $K * n_\lambda$ times, where n_λ is the number of λ-values we want to evaluate the model for.

4.3 Lasso

The result from the regression analysis is shown in figure 7. We see that for λ-values smaller than $\approx 5 \times 10^{-3}$, the error has very small variations as the model complexity is increased. As Lasso allows parameters to be set to 0 [3], with higher values of λ resulting in more parameters going to 0, this result visualizes what happens when λ is too big. So many of the coefficients have been removed such that the model has no variation when increasing the complexity, because only the coefficients corresponding to a certain polynomial degree are left. However, for smaller values of λ, we begin to see the effect of the model complexity. As opposed to OLS, the error does not spike after degree 6, but rather continues to decrease as the complexity increases. With ridge regression, we also saw that the error flattened at around degree 6, while with Lasso this only occurs for certain hyperparameter values. For lower values, the error keeps decreasing, even at polynomial degree 17 and 18. This shows that lasso regression does not only make the model more stable for higher complexity, as with ridge, but it also makes it much more flexible to the extent where it fits the data better for extremely complex models which would otherwise overfit the data. In the contour plots we again see how the model complexity does not matter for certain higher values of λ, and what combinations of the hyperparameter and polynomial degree gives the best error, which in this case is $\lambda \approx 3 \times 10^{-5}$ and $p = 18$. Were we to plot for even higher polynomial degrees, we would most likely see an even better error.

4.4 Parameter variance

The results of the confidence intervals as shown in figure 8 shows us how the variance of the parameters in a polynomial of degree 5 changes for each of the three regression methods. For OLS, we see that the variance is low for the first $\approx 7 - 10$ coefficients, which correspond to a 2d polynomial of degree $3 - 4$. Following these, the magnitude of the variance of some of the remaining coefficients spike significantly, while others maintain a lower value, but still higher than the first few. This is equivalent to the results we saw in the training and test error when performing the OLS. We saw that the error began spiking when using polynomials of degree > 4, meaning that the model was starting to overfit. As this happens, there is a substantial uncertainty in the higher order coefficients, as they are fitted for "hitting" these exact data points, and not the general shape of the data. For the Ridge regressor, we see that the confidence interval of the parameters decreases as λ gets smaller. We already saw how the λ-parameter in ridge regression shrinks the parameters by a factor of $\frac{1}{1+\lambda}$, which means that smaller values of the regularization parameter further shrinks the magnitude of the coefficients. As a result, the shrunked coefficients will be closer to their true values, which also means that the variance and confidence interval gets increasingly small, as we see in the figure. Finally, for Lasso, we see that the variance is the same for all parameters, but different for different values of the regularization parameter. What this tells us is that the parameters estimated by the lasso model all have the same confidence interval, while this value decrease for smaller λ. Since Lasso, as with ridge, puts a penalty on

the parameters, we would expect to see a similar behaviour in their variances.

4.5 Real data

The results of the parametrization of the terrain data using OLS (figure 10) shows a similar figure to that when we were analysing the Franke function. We observe that the training error decreases with the same apparent constant gradient, while the test error follows the training error before starting to increase at polynomial degree around 6, again similar to what we observed with the Franke function data. For the ridge model, we see that for higher values of λ, the error decreases slightly for the first few polnymial degrees, before more or less flattening out at a relatively high prediction error, while for smaller values of the hyperparameter, the model achieves errors around the same magnitude as that from the OLS model, but while allowing significantly higher complex models without overfitting. Finally, the lasso regressor has a somewhat similar result to the ridge model, except that the error is even more constant for higher complexity models. For higher values of λ, so many of the coefficients in the model have been set to zero that it has no variance, but it also has a relatively high prediction error. Essentially, the bias is so high that the variance is almost zero, but the error is as a result very high. As the hyperparameter is decreased, the prediction error decreases slightly as the complexity is increased for the first few polynomial degrees, before flattening at around degree 10 at an error of around $10^{-1.85}$, similar to that of the ridge model. While it is possible that increasing the complexity of the model even further would have decreased the error even more, it appears as if this would have happened so slowly, that the increase in complexity would not outweigh the gain in accuracy. Based on these results, all the models produced similar low errors. However, the OLS model did so at he lowest polynomial degree, which also means it would require the least amount of computational power to fit the data. While the bias added by regularization parameters in both ridge and lasso allows for more flexible models (higher complexity), the payoff is simply not worth it for this specific data set, as there would be an increase in computational power required for fitting data to more features. However, to avoid singular matrices when calculating the parameters in OLS, the singular value decompostion was used, which is, numerically, very expensive. Therefore, it is possible that this could actually be more expensive than a ridge or lasso model despite fewer features. The better canditate in that case would be lasso, as this regressor actually does remove features, meaning that a model complexity of polynomial degree 18 does not necessarily mean having a design matrix with 18 features, as some of them might have been set to 0 by the lasso model.

When looking at the resulting terrain data in figure 9, we see that a polynomial of degree 6 manages to predict the major features of the landscapes, mainly the areas with the largest extremities, but it fails to capture more details about the terrain elevations. Nevertheless, it can be said that the result is satisfactory for getting a general overview of the height levels in the terrain

5 Conclusion

In this experiment we have seen how three different linear regression methods can be used to train a model using supervised learning. We have seen how resampling is a necessary tool in order to avoid overfitting our data, and also how the number of data points available affects the complexity of the model which gives the best error. Additionally, we observed how the regularization parameter in both ridge and lasso regression adds bias to the model, allowing us to

use a higher complexity model without the risk of overfitting, as the model parameters are shrunk by a value related to λ, allowing us to fine-tune our model with the appropriate combination of the hyperparamater and the model complexity which gives the best error with the lowest amount of CPU hours necessary for the calculations. We also saw how this parameter affects the confidence intervals of the parameters, and how the shrinkage factor causes the parameters to get closer to their true values. The lowest error on the Franke function data was produced with a polynomial of degree 6, and while both ridge and lasso regression are often chosen in favour of OLS as they allow more flexibility in the model and therefore a better error, our OLS model actually produced an error which was very similar to that of the ridge model with $\lambda \approx 10^{-5}$. The difference was that the ridge model maintained this error for higher order polynomials, while the error of the OLS model spiked. Compared to ridge, the lasso model continued to produce lower errors for extremely high complexity models, due to its ability to remove the parameters which would otherwise overfit or stall the model. Since we did not observe a bottom value for the error of the lasso model, it is hard to conclude which model was the best suited. If we assume the error of the lasso model would continue decreasing for even higher degrees, one might say that that is the model to choose as it produces the smallest error. However, high complexity models can be computationally expensive, depending on how many features the lasso regressor has removed. Therefore, we would have to benchmark the various models to see which produced the best error while also maintaining a reasonable numerical complexity. However, looking at only the prediction error, the best model we observed was that of the ridge regressor with polynomial degree 6, and $\lambda = 10^{-5}$.

We also looked at how linear regression could be used to parametrize real terrain data. The result was similar to that when analysing the data from the Franke function, and produced satisfactory results for predicting the general shape of the terrain.

Appendix: Terrain data

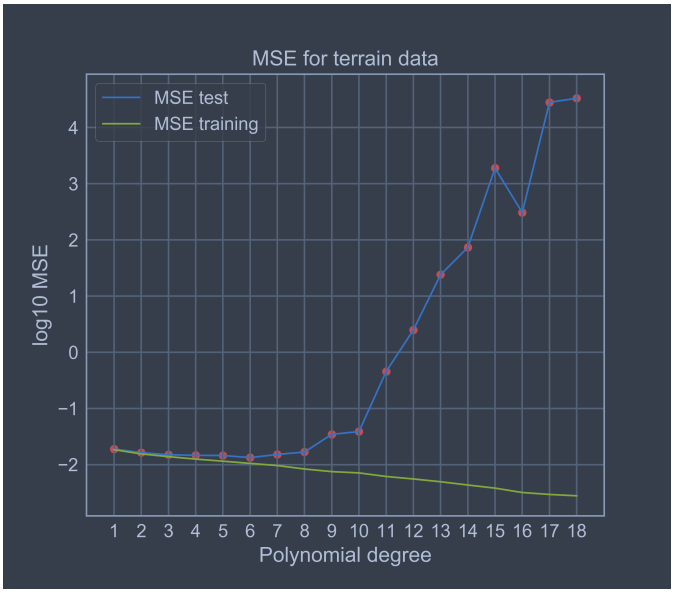

Figure 10: Figures showing the prediction error when performing OLS on the terrain data.

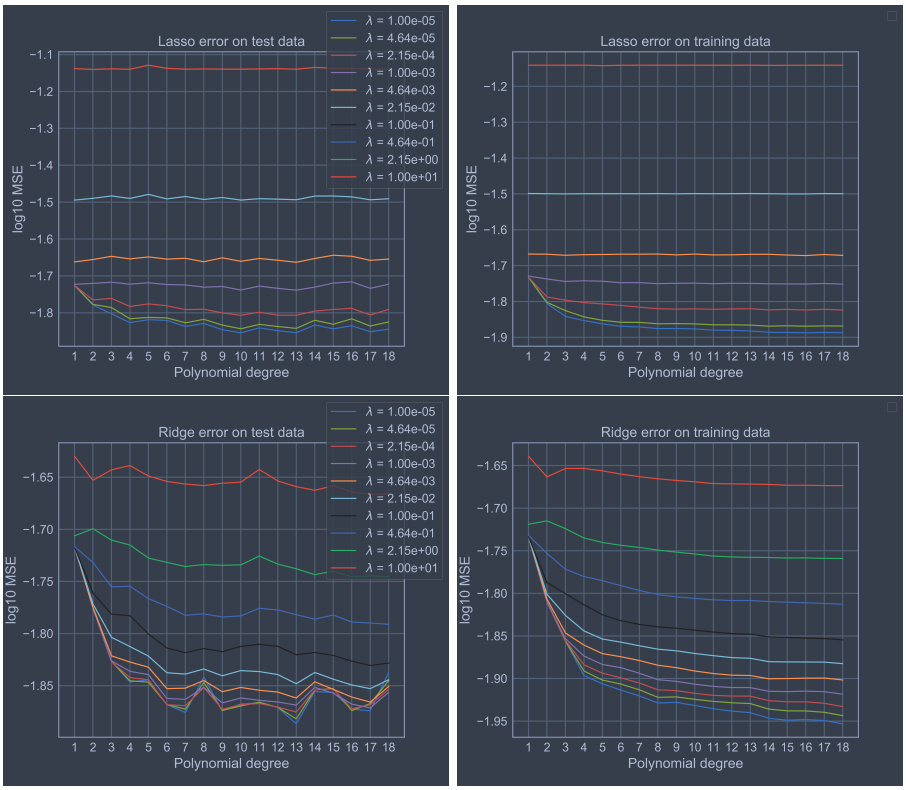

Figure 11: Figures showing the prediction error on the terrain data from Ridge and Lasso regression.