FYS-STK4155 Project 1: Linear Regression

Lidia Luque & Kosio Beshkov

Introduction

Regression is a common method used in statistics to find correlations in order to predict a statistical variable from measurements of another statistical variable. In this exercise we study how different linear regression models as well as re-sampling techniques compare to each other. We first test our models using data sampled from the Franke function, which is a well established testing function, before moving on to using real world terrain data on the same models.

1 Ordinary Least Squares (OLS) on the Franke Function

For this first exercise we apply the OLS algorithm to the Franke function, which has the following expression:

$$f(x,y) = \frac{3}{4}\exp(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}) + \frac{3}{4}\exp(-\frac{(9x+1)^2}{49} - \frac{(9y+1)}{10}) + \frac{1}{2}\exp(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}) - \frac{1}{5}\exp(-(9x-4)^2 - (9y-7)^2) \tag{1}$$

We sample the Franke function on a uniform grid where $x, y \in [0, 1]$. From this we get a vector with the data we want to fit, $\boldsymbol{z} \in \mathbf{R}^n$ where n is the total number of samples. In linear regression, as the name implies, the data is expected to be represented as a linear combination of the the model features:

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{2}$$

where $\boldsymbol{X}$ is the so-called *design matrix*. Each column of the design matrix represents a feature, while each row contains the feature values for one data point. In this project we choose to fit our data to a polynomial of d degree. The design matrix will thus consists of all the combinations of the features with degree less than or equal to d. ϵ refers to the stochastic noise of the data. The vector $\beta \in \mathbf{R}^f$ is a vector of parameters where f is the number of features. For this exercise, since the data is not the result of an experiment, we are free to choose the noise ourselves. We choose $\epsilon \sim N(0, \sigma^2)$, that is, we add a normally distributed noise with 0 mean and σ variance to each data point in $\boldsymbol{z}$. Our

goal now is to find the vector of regression parameters $\hat{\beta}$ that gives the best prediction for the fitted data vector $\hat{\mathbf{z}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$. In ordinary least squares, as the name implies, the goodness of the fit is measured by the mean square error (MSE), which we therefore must minimize. It can be shown that the analytic expression

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

minimizes the MSE with respect to β. Note that using equation eqn:OLS requires finding the inverse of the matrix $\mathbf{X}^T\mathbf{X}$. Because this matrix can be singular (the columns can be linearly dependent), we use the pinv function from numpy to find the pseudo-inverse. Using pinv gives the inverse of the matrix when the matrix is not singular, and it gives the matrix "closest" to the inverse when the matrix is singular (it minimizes the norm between $\mathbf{AA_{pinv}}$ and $\mathbf{I}$).

From equation 3 we get the regression parameters $\hat{\beta}$. Finding $\hat{\beta}$ is commonly referred to as *training* a model. From $\hat{\beta}$ we find the predicted data vector $\hat{\mathbf{z}} = \mathbf{X}\hat{\beta}$, and we have thus found the ordinary least squares fit to $\mathbf{z}$ given the features in $\mathbf{X}$.

However, instead of using all data points to train the model, it is common to divide $\mathbf{z}$ into a training dataset used to train the model, and a test dataset used to evaluate the trained model. This gives a more realistic picture of the model fit, given that the test data remains unseen by the model (i.e. is not used in training) and is only used in the model evaluation. In this project we split the data using the train_test_split function in SKlearn, and use 70% of our data to train the model and the remaining 30% to evaluate it.

In regression problems, and indeed in machine learning problems in general, the feature vectors may have very different scales. Scaling the feature vectors is a widely used prepossessing step which tends to lead to better models. How important scaling is depends on both the input data and the chosen model. In this case, the input data is quite well scaled in of itself, since the Franke function only varies between approximately -0.1 and 1.3 and it is only sampled for $0 <= x, y <= 1$. This means that both $\mathbf{z}$ and X are quite well scaled, and so for this project, scaling is not a must. Furthermore, in ordinary least squares, scaling will never change the outcome of the model, since there is no parameter scaling the regression parameters, as there is in lasso and ridge regression. Summing up: scaling tends to be important, but for this project, it will either not change the outcome at all (OLS) or only give a marginally better model (ridge and lasso regression). However, for pedagogical proposes, we have nevertheless chosen to scale the data. We do so using the simplest scaling method, removing the mean from the feature vectors of $\mathbf{X}$ and the $\mathbf{z}$ vector. We do this for both train and test data. It is also common to divide by the variance, but in this case, since the variance can be very small, this could lead to numerical errors, so we have decided against it.

The resulting $\mathbf{z}$ train vector is used to train an OLS model where we use polynomials of fifth degree to fit the data.

Figure 1 shows the values of all $\hat{\beta}$ for the model with their corresponding 95%

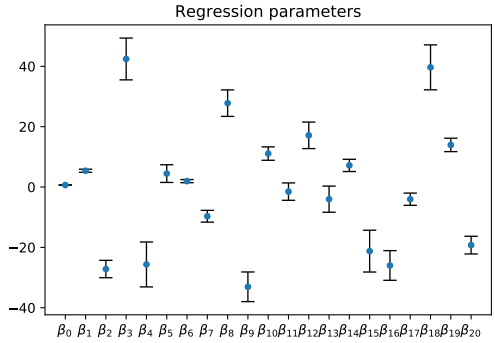

Figure 1: The values for regression parameters for a model of degree 5 with $n = 100$. The vertical lines show the 95% confidence intervals for each parameter.

confidence intervals. Here, the variance of the noise is 0.05 and total number of datapoints $n = 100$ (meaning the model is trained on 70 datapoints). The MSE between the predicted data and the test data (called the test MSE) for this example is 0.0042, while the $R2$ score is 0.9490. Keeping in mind that perfect model would have a test MSE of 0 and a $R2$ score of 1, this looks at face value to be a relatively good model.

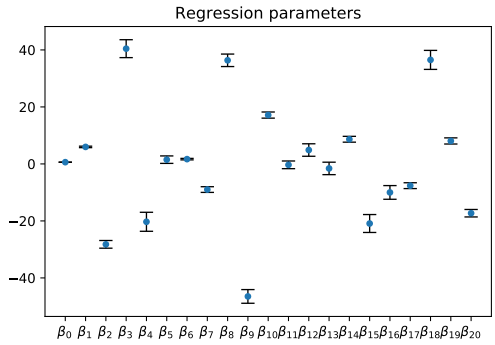

Figure 2: The values for regression parameters for a model of degree 5 with $n = 225$. The vertical lines show the 95% confidence intervals for each parameter.

Figure 2 shows the same plot for $n = 225$. As expected, increasing the number of datapoints keeps the values of the regression parameters similar but lowers the confidence intervals, since the confidence interval is inversely proportional to the number of datapoints. We would expect the variance of the regression parameters (and thus their confidence intervals) to be small for small i indexes in $\hat{\beta}_\mathbf{i}$, which refer to the lowest polynomial degrees. The variance in

the parameters should increase for larger i indexes where the model tries to fit the polynomials to the noise. This is not the case in neither of the plots. We have tried increasing the noise in order to increase the variance of the regression parameters of largest order without much luck, and the reason for this is unclear.

2 Bias-variance trade-off and resampling techniques

In this exercise we vary the complexity of our model by increasing the number and degree of polynomial regression variables and evaluate the bias and variance of the predictions as a function of said complexity. In order to get a better estimate of the bias and the variance, we use a resampling method called *bootstrap*. Resampling methods estimate statistics on a dataset by sampling it in a certain way, in the case of the bootstrap, sampling it with replacement. While using resampling in this case is not technically needed, since we can easily produce more data, resampling methods are key in more real-world cases, where more data cannot easily be produced.

We will first plot the test and the train mean square error of the model as a function of the model complexity. Figure 3 shows the test and train MSE for the model we used in exercice 1, that is, a model trained on 70% of $n = 225$ and tested on the remaining 30% with a noise variance of 0.05. Note that the test MSE is a lot more variable than the train MSE. This is because we are only using $225 \times 0.3 = 67$ data points to calculate the test MSE, while the train MSE is calculated from over two times as many datapoints. However, the trend for both is as expected: the train MSE falls continuously as the complexity of the model increases, while the test MSE falls before starting to increase at degree 7. This increase in the test MSE is due to *over-fitting*: with increasing complexity, the model starts increasingly fitting not just the underlying trends in the data but also the noise, and when we test the model on new datapoints, the model "fails", meaning the test MSE increases. We will now look at an anologous way of explaining this: the *bias-variance trade-off*.

In the previous problem we found the regression coefficients with the OLS algorithm. If we name the estimate of the true data z to be $\hat{\boldsymbol{z}} = \boldsymbol{X}\boldsymbol{w}$, we can write out the cost function (the MSE) as:

$$C(w, X) = \frac{1}{n}\sum(z_i - \hat{z}_i)^2 = E[(z_i - \hat{z}_i)^2] \tag{4}$$

Since the real data z can be written as the true function that generates data plus some zero mean noise $z = f(x, y) + \epsilon$ we can substitute this in the above expression.

$$E[(z_i - \hat{z}_i)^2] = E[(f(x_i, y_i) + \epsilon - \hat{z}_i)^2] \tag{5}$$

After adding and subtracting $E[\hat{z}]$ and removing expectations of zero mean

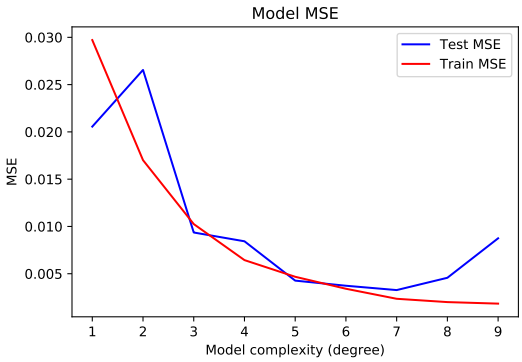

Figure 3: Train and test MSE without resampling for $n = 225$

terms like $E[\epsilon]$ the expression becomes:

$$\begin{aligned} &E[(f(x,y) + \epsilon - \hat{z} + E[\hat{z}] - E[\hat{z}])^2] = \\ &\quad = E[(E[\hat{z}] - f)^2] + E[(\hat{z} - E[\hat{z}])^2] + E[\epsilon^2] = \\ &\quad = \frac{1}{n}\sum (f_i - E[\hat{z}])^2 + \frac{1}{n}\sum (\hat{z}_i - E[\hat{z}])^2 + \sigma^2 \end{aligned} \tag{6}$$

The first term measures how far off the mean of our model is from the true values and is known as the squared bias, the second is simply the variance of our model and the third is the variance of the error. What this equation shows is that we get the smallest (test) MSE (assuming a constant noise variance) by minimizing the sum of the bias squared (we will from here on call it simply the bias) and the variance. Figure 4 shows the test MSE, the bias and variance as a function of complexity of out model, this time using the using bootstrap to resample 50 times in order to get better estimates for these statistics. The input data is the same as in figure 3. Here we see a general trend of decrease in the bias as the complexity increases, that is, increasing the complexity of the model makes the mean of the modelled data be closer to their true values. However, increasing the complexity of the model also increases the variance of the modelled data, leading to over-fitting. The complexity of the model is thus an important hyperparameter to tune in order to get a model with a low bias, while keeping the variance from being too high. Figure 4 shows that, in this model, the MSE is minimized around degree 5-6, which is thus the optimal complexity to model this data.

Next, we will explore how the bias-variance trade-off depends on the number of datapoints n. Figure 5 shows the bias, the variance, and the MSE for four models. All models use the same parameters, but are trained (and tested) on different number of datapoints (note when comparing the plots that the axis limits are different). The general trend of decreasing bias and increasing variance holds true for all four plots, but the model complexity at which the variance starts to grow increases as we increase n. This leads to the region with

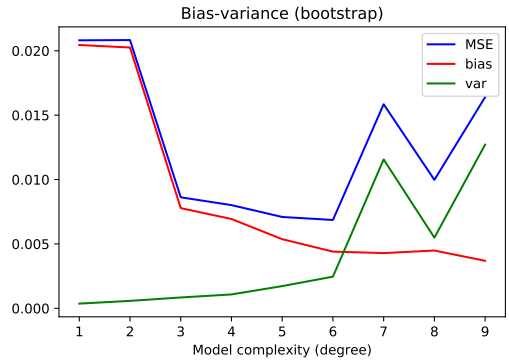

Figure 4: Bias, variance and test MSE for $n = 225$ using 50 resamples using the bootstrap method.

the lowest MSE occurring at different model complexities: for $n = 100$, the optimal degree of the model would be somewhere between 3 and 5, for $n = 225$ 5-6, for $n = 400$ 5-10 and for $n = 625$ 6-11. In other words: a model trained on more data will accept a higher complexity model before starting to over-fit.

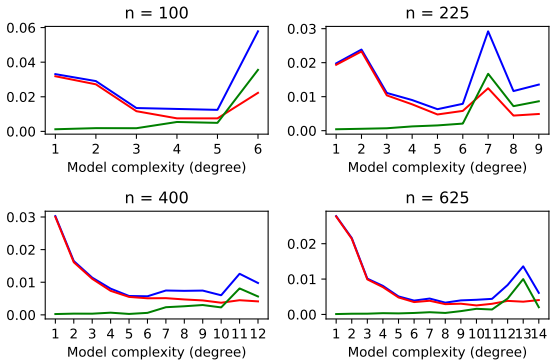

Figure 5: Bias (red), variance (green) and test MSE (blue) for different model degrees and number of datapoints.

3 Cross-validation as resampling techniques, adding more complexity

In this problem we analyze the MSE by using another resampling technique, k-fold cross-validation. We start by dividing the data into k number of subsets, called folds. We then compute the statistical variable we want to estimate, in

this case the MSE, by using one of the folds as test data and the rest as train data. This step is repeated k times, each with a different test fold. We then compute the mean of the MSE values to get its estimate. Figure 6 shows the test and train MSE computing using 50 bootstraps on the left, and the test and train MSE computed using 5-fold cross-validation on the right. While the train MSE is almost identical, the test MSE is a lot more variable when we compute it using the bootstrap method. This holds true even if we increase the number of bootstraps, so it seems that using the k-fold can produce to a smoother MSE estimate. Another thing to note is that the test MSE using bootstraping is, for most degrees, significantly higher than it is when using k-fold. A possibility is that this has to do with the amount of data used in training, since bootstrap is set up to use a 70/30 train-test split, while k-fold only uses $1/k$ of the data for testing and leaves $(k-1)/k$ data for training, which for $k = 5$ is a 80/20 train-test split.

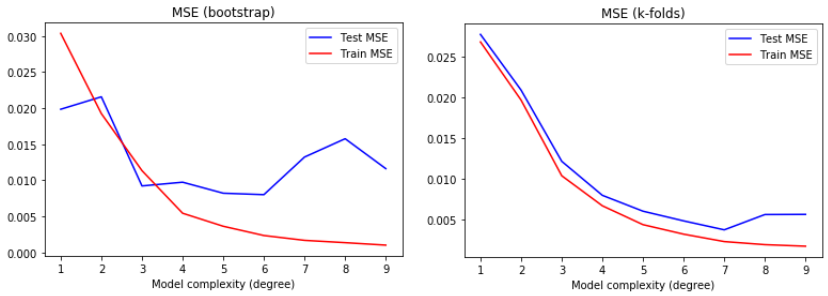

Figure 6: Test and train MSE computed using the bootstrap method with 50 resamples on the left, and using 5-fold cross-validation of the left for $n = 225$

Let's look at a model with $n = 625$, without otherwise changing any of the data or sampling parameters. Figure 7 again shows vary high (and variable) test MSE from the bootstrap method when compared to the k-fold with $k = 5$ for higher model degrees. What happens with the test MSE when we vary k? Figure 8 shows the MSE computed for two more values of k, 3 and 10. The test MSE (and indeed the train MSE) calculated with $k = 10$ and $k = 5$ are very similar, but the one calculated with $k = 3$ is, for high model complexities, almost two orders of magnitude larger. In fact, the $k = 3$ MSE test, which uses 66% of the data to train, resembles the bootstrap test MSE in figure 6, which trains on 70% of the data (however, the later still has significantly more variability). This is consistent with the model starting to overfit at a lower degree when using less training data, and is thus important to keep in mind when choosing the train-test split, either directly or indirectly by choosing k.

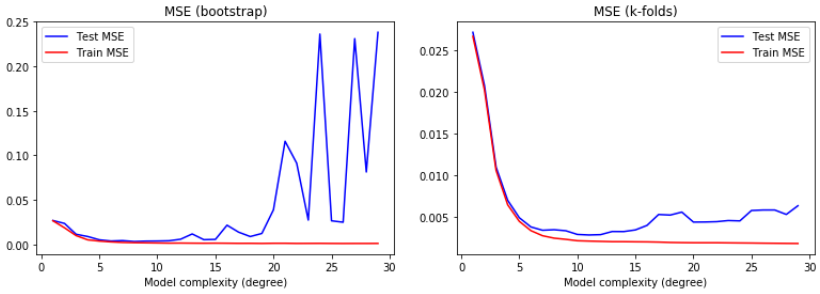

Figure 7: Test and train MSE computed using the bootstrap method with 50 resamples on the left, and using 5-fold cross-validation of the left for $n = 625$

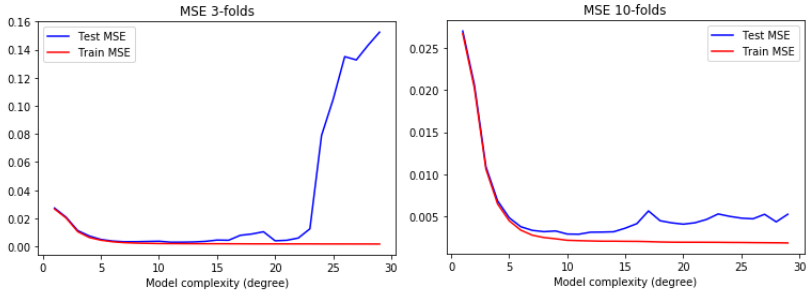

Figure 8: Test and train MSE computed using k-fold cross-validation with $k = 3$ and $k = 10$ for $n = 625$

4 Ridge Regression on the Franke function with resampling

In this exercise we implement the Ridge regression algorithm on the same dataset. Ridge regression is very similar to OLS except for the fact that one adds a regularization parameter λ which controls the size of the estimated weight coefficients. In other words the algorithm optimizes the loss:

$$C(w, X) = \frac{1}{n} \sum_i (z_i - \sum_j X_{ij} w_j)^2 + \lambda \sum_i {w_i}^2 \tag{7}$$

This optimization also has a one step solution using the pseudo inverse, which is:

$$w = (X^T X + \lambda I)^+ X^T z \tag{8}$$

With the plus in the exponent denoting the pseudo-inverse and I denoting the identity matrix.

First we show how both the train and test error depend on the regularization parameter λ (figure 9. In all Ridge calculations we choose 8 values of λ from 0.00001 to 100:

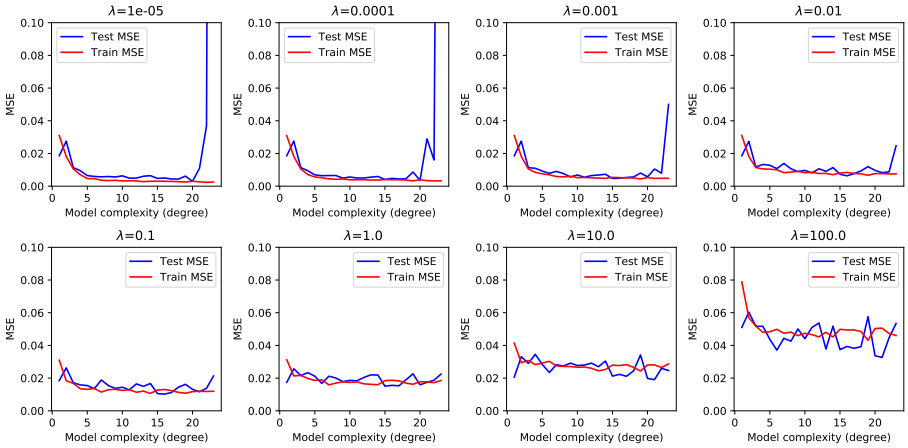

Figure 9: Test and train MSE for the Ridge method with varying λ values on Franke's funciton.

As one can see the best performance is achieved for low values of lambda, while higher values lead to a larger mean squared error. Additionally it seems that higher values of λ control the test error by assigning lower weights to polynomials with high degree which contribute to the test error in a more significant manner.

Next we use bootstrapping to study the bias-variance trade off as a function of the regularization parameter. The results can be seen in figure 10:

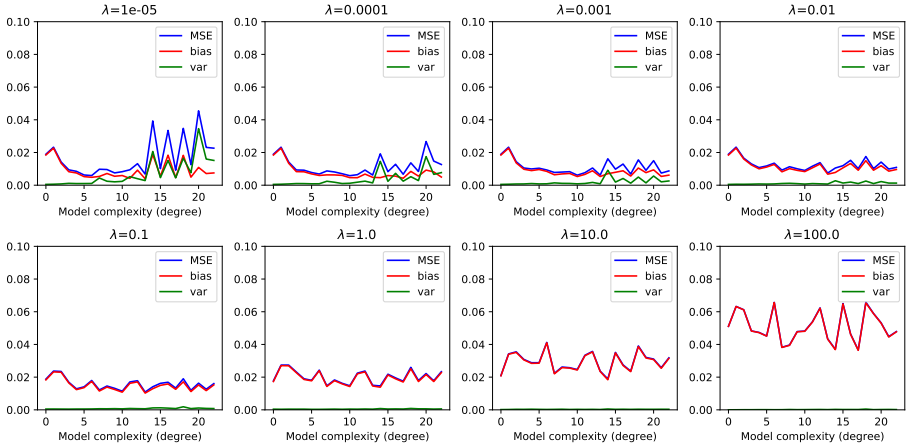

Figure 10: MSE, bias and variance in Ridge as a function of model complexity and λ value on Franke's function.

In this case there is a noticeable difference from the bias and variance in the OLS. While in both cases the bias goes down and the variance goes up with model complexity, when regularization is added the variance becomes really small, except for models with very high complexity. This is most likely due to the fact that regularization prevents overfitting on the test data by restricting the model complexity, which excludes highly variable solutions (large coefficients are excluded, so the variance of the candidate coefficient distribution is smaller). It can also be seen that there is a dependence on λ where for large values, the bias becomes a much larger part of the MSE than the variance.

Finally we perform k-folds cross-validation with which to study the MSE (figure 11). Similar to the OLS k-folds cross-validation, the MSE comes out smoother than the bootstrap, but the general pattern is the same. Namely larger regularization parameters lead to both higher train and test error.

5 Lasso Regression on the Franke function with resampling

The last algorithm that we explore is the Lasso regression, which optimizes the same loss function as the Ridge regression except for the fact that it uses the l1-norm instead of the l2-norm to constrain the coefficient values. In other words the loss becomes:

$$C(w, X) = \frac{1}{n} \sum_i (z_i - \sum_j X_{ij} w_j)^2 + \lambda \sum_i |w_i| \tag{9}$$

Unlike OLS and Ridge regression, this objective function can not be minimized in a single step calculation. For that reason we use the **Scikit-Learn** function

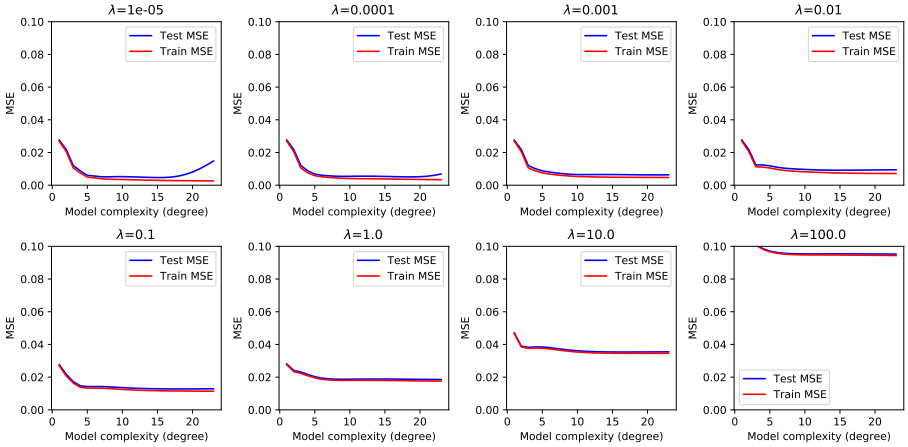

Figure 11: Test and train MSE of the Ridge method computed using k-fold cross-validation with k=5.

linear_model.Lasso to find the regression coefficients. To get sensible results we had to choose a set of regularization parameters with smaller values than when applying Ridge regression. We ended up choosing 8 values from 1e-7 to 1. First we show the MSE as a function of λ (figure 12).

The outcome is very similar to that of the Ridge regression, where small regularization values lead to a smaller test error. However in this case the test error is low even for high degrees which shows that Lasso regression punishes small coefficients more harshly than Ridge. This makes sense as in Ridge regression coefficients smaller than 1 contribute much less to the cost than in Lasso regression (due to the fact that they are squared, whereas in Lasso they are constant) and high order polynomials that contribute to overfitting are not removed.

The results are not very different from the ones in Ridge regression for the bootstrap and k-folds algorithms (figure 13 and 14).

Like in the Ridge estimation, here the bias dominates over the variance.

Using k-folds, we get a more clear picture that large λ values lead to worse solution for both training and test datasets. One additional observation that seems reasonable to make is that by increasing λ the dependence on model complexity also starts to disappear (it seems completely absent in the last two plots at the bottom right).

In general both Lasso and Ridge regression help to prevent overfitting and allow one to use more basis functions which to fit the training data to so as to improve model prediction.

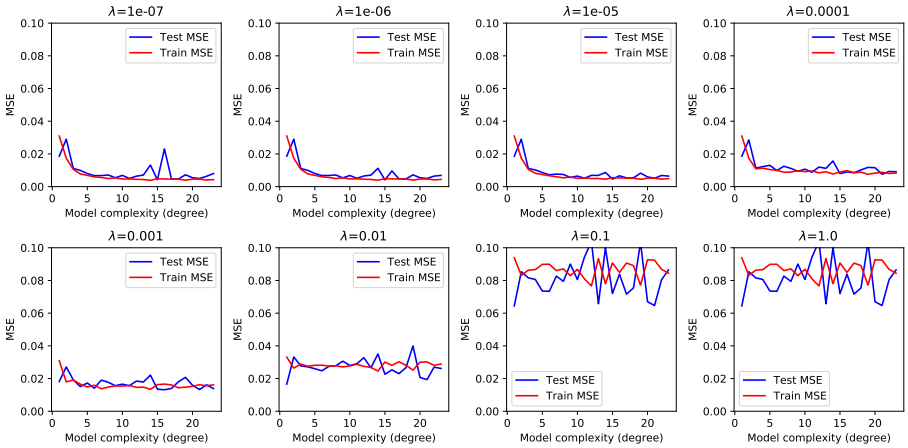

Figure 12: Test and train MSE for the Lasso method with varying λ values on Franke's function.

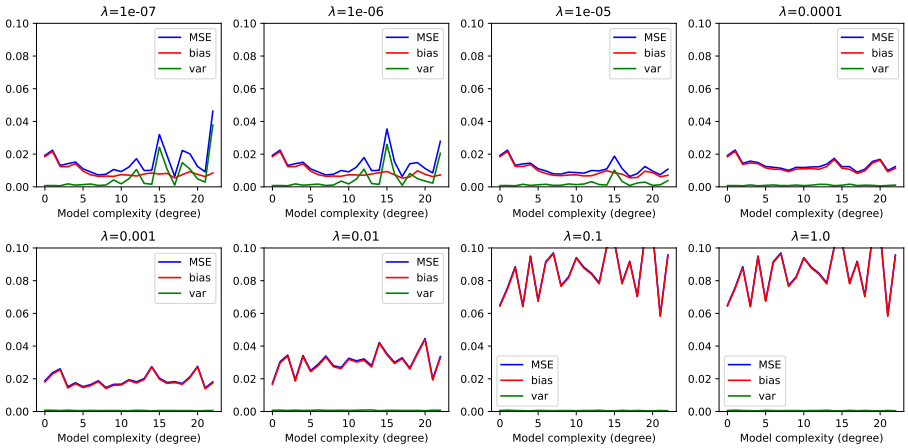

Figure 13: MSE, bias and variance in Lasso as a function of model complexity and λ value on Franke's function.

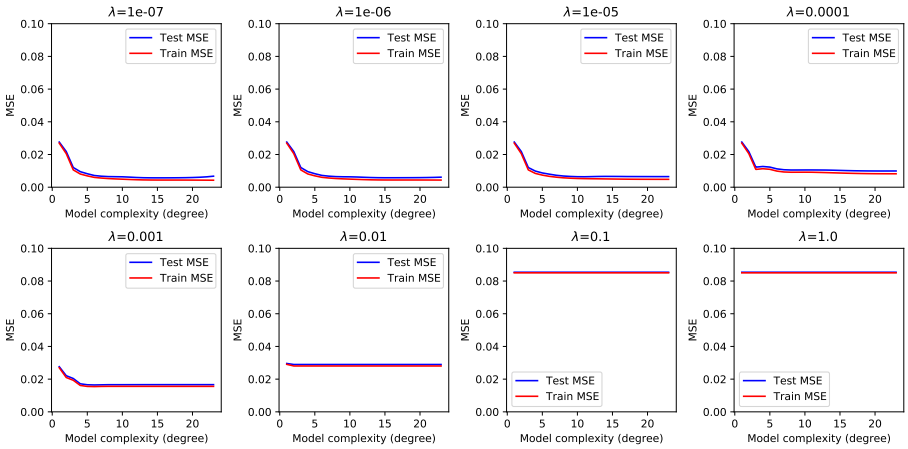

Figure 14: Test and train MSE of the Lasso method computed using k-fold cross-validation with k=5.

6 Analysis of real data

Finally it is time to apply all three types of regression on real data. For this purpose we use the provided digital terrain data and pick out a 50×50 pixels patch (figure 15 shows both the true terrain as well as a prediction made using OLS).

Again the only scaling that was performed on the data is the removal of the mean. Since we were not sure what the right model order is we performed OLS for up to model degree 50. Overfitting started to occur only after around model degree 40s. However training with so many polynomials made the Lasso regression way to slow to compute so we stuck with up to model degree up to 10 for that method.

6.1 Ordinary Least Squares

In the OLS estimate we can see that the MSE falls quickly with model complexity an slows down after around 10 degrees (figure 16). As for the bootsrap and the k-folds. They show similar patterns (only showing up to 25 model degrees to save computation time) to what we observed for the Franke function, with the bias going down as a function of model complexity (figures 17 and 18).

6.2 Ridge regression

For the Ridge regression we tried out the same λ parameters as in the Franke function. The MSE for all of them went up with higher labmda values and was consistently higher than the OLS equivalent. These results are shown in figure 19.

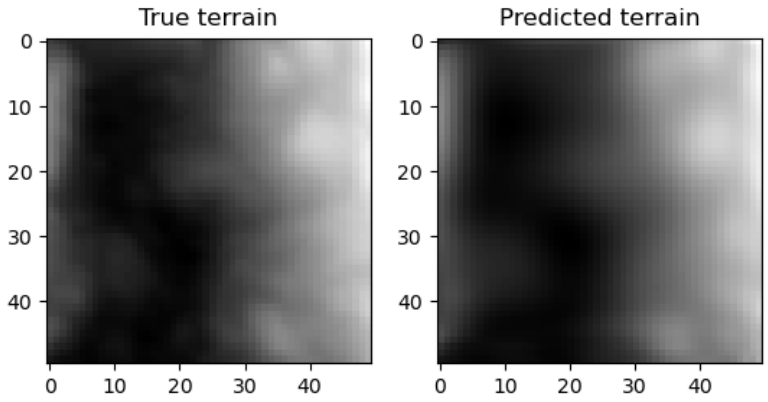

Figure 15: Comparison of real and predicted (OLS) Norwegian landscape

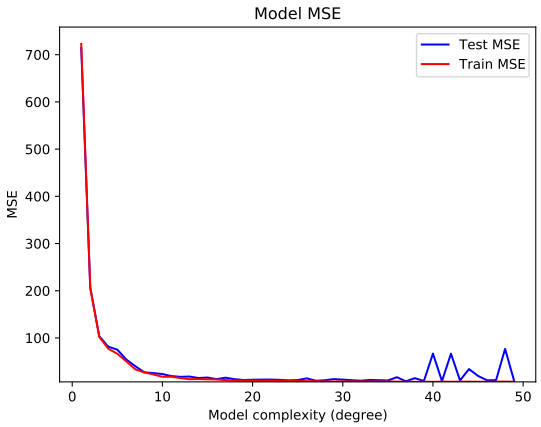

Figure 16: Test and train MSE for the OLS method on terrain data.

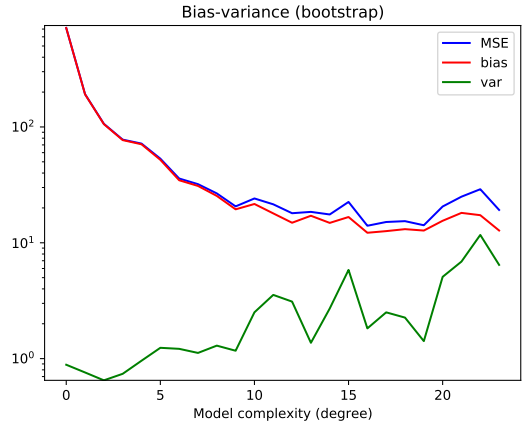

Figure 17: MSE, bias and variance in OLS as a function of model complexity on terrain data.

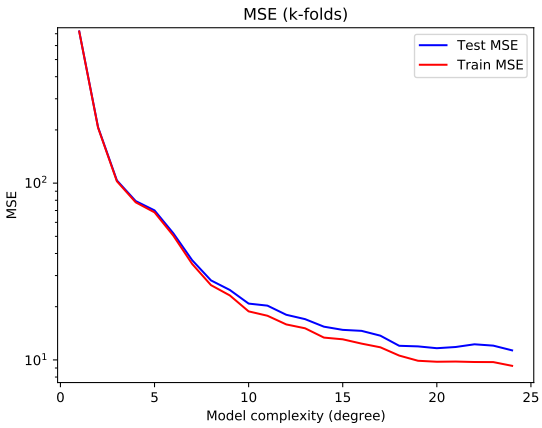

Figure 18: Test and train MSE of OLS method computed using k-fold cross-validation with k=5.

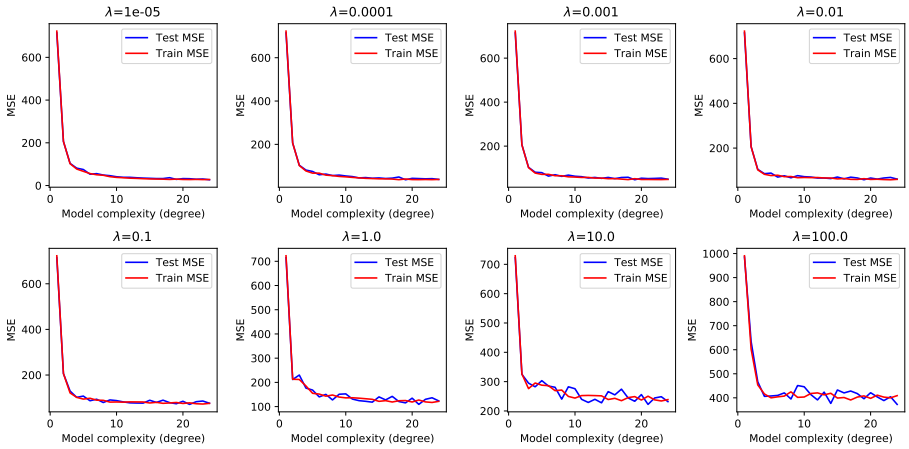

Figure 19: Test and train MSE for the Ridge method with varying λ values on terrain data.

The bootstrap method showed the bias variance trade off (figure 20) as expected and the k-folds performance is shown in figure 21.

In general the lack of overfitting for high model complexity is still apparent, so it is possible that better solutions can be obtained if the model complexity was to be increased even further.

6.3 Lasso regression

As previously mentioned we restricted the model order to 10 in the Lasso regression as it uses a gradient descent algorithm and required too many iterations to converge and takes several hours to complete.

The results are thus a bit harder to compare, but the same pattern as in the Ridge regression emerges. Namely higher λ values led to worse model performance and overfitting was not observed for these model values. These results can be seen in the following three figures (22 - MSE, 23 - bootstrap and 24 - k-folds).

In conclusion the OLS regression gave the best performance on the dataset, however it leads to faster overfitting for larger model complexity. Thus with more model complexity one might be able to find better solutions using regularized methods.

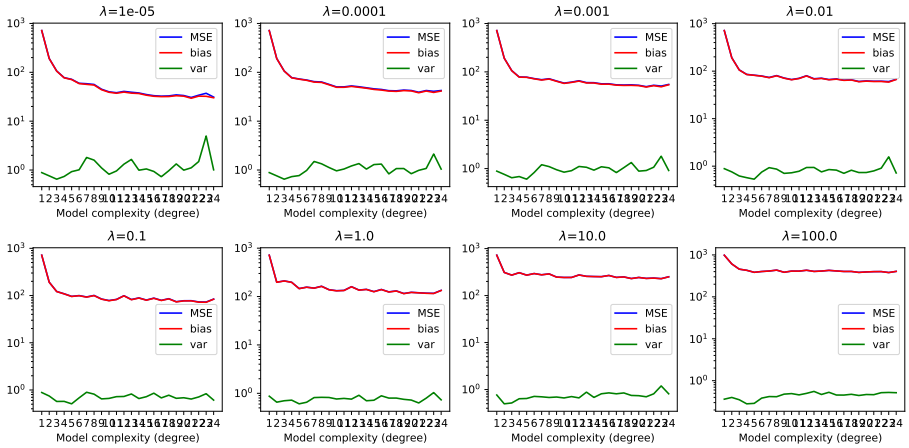

Figure 20: MSE, bias and variance in Ridge as a function of model complexity and λ value on terrain data.

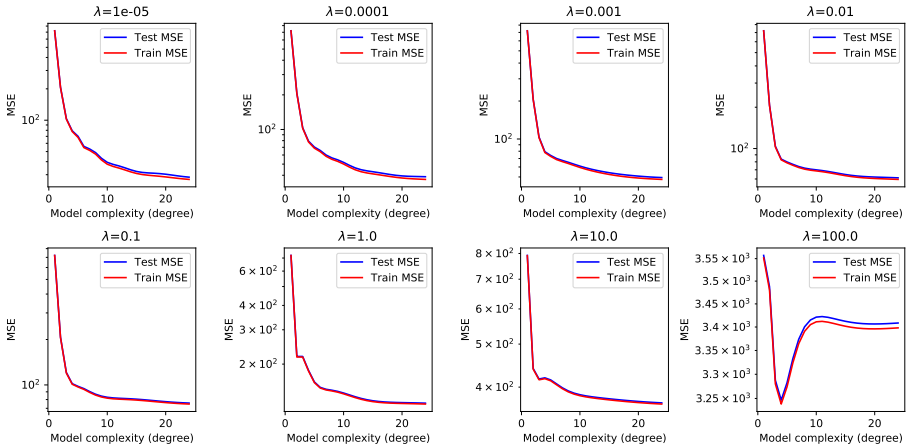

Figure 21: Test and train MSE of the Ridge method computed using k-fold cross-validation with k=5.

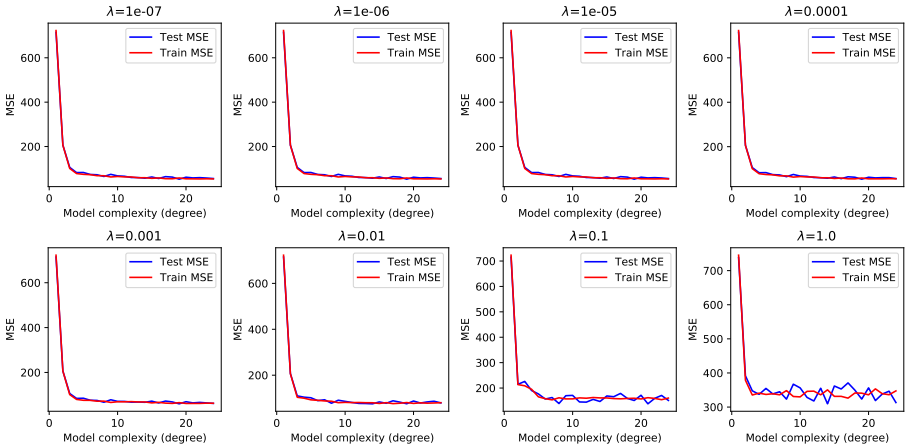

Figure 22: Test and train MSE for the Lasso method with varying λ values on terrain data.

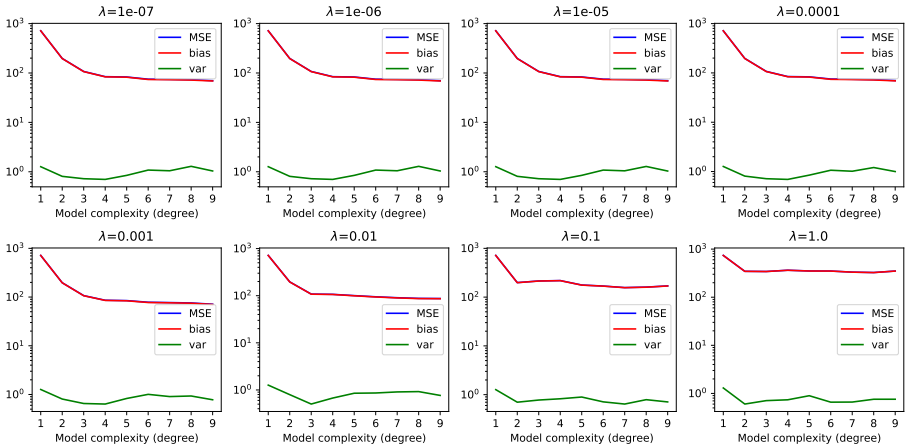

Figure 23: MSE, bias and variance in Lasso as a function of model complexity and λ value on terrain data.

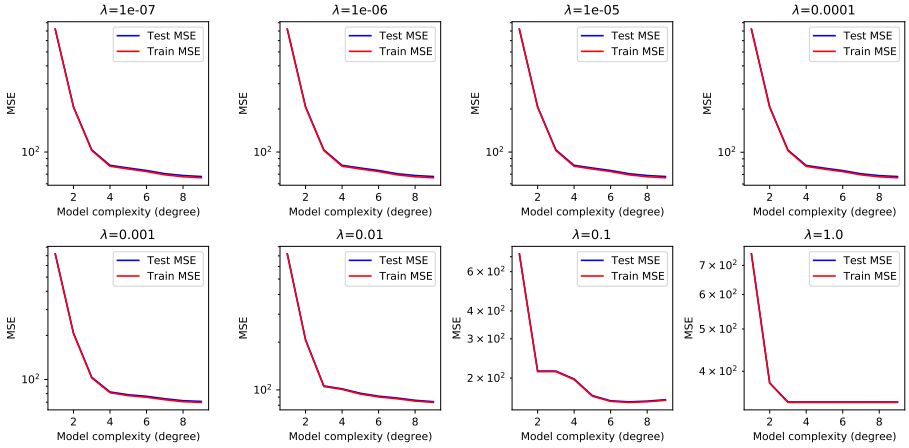

Figure 24: Test and train MSE of the Lasso method computed using k-fold cross-validation with k=5.