FYS-STK4155 Project 1 - Regression analysis and resampling methods

Joakim Flatby, Linus Ekstrøm

August 2019

https://github.com/jflatby/FYS-STK4155

Abstract

In this paper we test ordinary least squares, ridge- and lasso-regression on a smooth function with noise, as well as real terrain data. This was first done on Franke's function, and our analysis showed that the ordinary least squares algorithm produced the best results over a range of model complexities. Additionally, we studied these methods on a sample of terrain data of the Grand Canyon. Also in this case the ordinary least squares performed the according to the mean squared error and R^2 metrics.

Contents

1 Introduction **3**

2 Theory **3**

2.1 Generated Data . . . 3
2.2 Ordinary Least Squares . . . 3
2.3 Variance of $\vec{\beta}_{\text{o.l.s.}}$ . . . 3
2.4 Mean Squared Error . . . 3
2.5 R^2-score . . . 3
2.6 k-Fold Cross-Validation . . . 4
2.7 Bias-Variance Trade-off . . . 4
2.8 Ridge Regression . . . 4
2.9 Variance of $\vec{\beta}_{\text{ridge}}$ . . . 4
2.10 Lasso Regression . . . 4

3 Method **5**

3.1 Code Structure . . . 5
3.2 Evaluating Franke's Function with Stochastic Noise . . . 5
3.3 Franke's Function with k-fold Resampling . . . 5
3.4 Regression types . . . 5

4 Results **5**

4.1 Evaluating Franke's Function with Stochastic Noise . . . 5
4.2 Franke's Function with k-fold Resampling . . . 6
4.3 Ridge Regression of Franke's Function . . . 6
4.4 Lasso Regression of Franke's Function . . . 7
4.5 Variance . . . 7
4.6 Terrain data . . . 7

5 Discussion **7**

5.1 Ordinary Least Squares Analysis of Franke's Function . . . 7
5.2 Ridge Regression Analysis of Franke's Function . . . 8
5.3 Lasso Regression Analysis of Franke's Function . . . 8
5.4 k-fold Resampling . . . 8
5.5 Variance . . . 8
5.6 Terrain data . . . 8

6 Conclusions **8**

7 Sources **9**

A Terrain Data Comparison **9**

1 Introduction

In this computational project we will be tackling three different kinds of regression methods: Ordinary Least Squares, Ridge regression and Lasso regression. Regression is a technique often used in machine learning to model and analyze the relationship between different variables. Often times relating to how these variables vary with regard to each other. To study these three methods we will use the famous Franke's function, which is commonly used as a test function for interpolation problems. In addition to our regression analysis we will also implement and use the k-fold cross-validation technique to evaluate our regression results. This is in order to better estimate the quality of our implemented models. We evaluate using the metrics: mean squared error and R^2-score. After this we will test our models on real terrain data consisting of data from the Grand Canyon AZ. U.S.

2 Theory

In the following section we will give a brief introduction to the necessary theoretical background for the work we have done in this report.

2.1 Generated Data

To generate our data we will be using Franke's Function $f(x, y)$ with added stochastic noise using the normal distribution $\mathcal{N}(', \infty)$. Franke's Function is defined by

$$\begin{aligned} f(x,y) &= 0.75\exp\Big(-\frac{(9x-2)^2}{4}-\frac{(9y-2)^2}{4}\Big)\\ &+0.75\exp\Big(-\frac{(9x+1)^2}{49}-\frac{9y+1}{10}\Big)\\ &+0.5\exp\Big(-\frac{(9x-7)^2}{4}-\frac{(9y-3)^2}{4}\Big)\\ &-0.2\exp\big(-(9x-4)^2-(9y-7)^2\big)\end{aligned}$$

our complete function then becomes

$$z(x,y) = f(x,y) + C\mathcal{N}$$

where C is a constant that represents the strength of the added noise.

2.2 Ordinary Least Squares

Ordinary least squares (OLS) is a method for estimating the unknown parameters in a linear regression model. The parameters are chosen by the principle of least squares: *minimizing the sum of the squares of the differences* between the observed dependent variable and those predicted by the linear function. Consider the system of linear equations

$$\sum_{j=1}^{p} X_{ij}\beta_j = y_i, \quad i \in \{1,2,...,n\}$$

of n linear equations in p unknown coefficients $\beta_1, \beta_2, ..., \beta_p$ with $n > p$. In matrix notation:

$$\mathbf{X}\vec{\beta} = \vec{y} \tag{1}$$

It is unusual for such a system to have an exact solution, so the goal of the algorithm is to find the coefficients $\vec{\beta}$ which minimize the distance from each predicted value to the test value. In other words: the coefficients which fit (2) the best.

$$\begin{aligned}\vec{\beta} &= \text{argmin}S(\beta)\\ S(\vec{\beta}) &= \frac{1}{n}\sum_{i=1}^{n-1}|y_i - \sum_{j=1}^{p}X_{ij}\beta_j|^2 = ||\vec{y}-\mathbf{X}\vec{\beta}||^2\end{aligned} \tag{2}$$

Provided the p columns of $\mathbf{X}$ are linearly independent then the problem has a unique solution:

$$\begin{aligned}(\mathbf{X}^T\mathbf{X})\vec{\beta} &= \mathbf{X}^T\vec{y}\\ \vec{\beta} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y}\end{aligned}$$

2.3 Variance of $\vec{\beta}_{\text{o.l.s.}}$

The variance of the least squares estimator is:

$$\begin{aligned}\text{Var}(\vec{\beta}_{\text{o.l.s.}}) &= \text{Var}\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y}\Big)\\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\text{Var}(\vec{y})\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)^T\\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\sigma^2\mathbf{I}\Big((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\Big)^T\\ &= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\\ &= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\end{aligned} \tag{3}$$

With

$$\sigma^2 = \frac{1}{N-p-1}\sum_{i=1}^{N}(z_i - \tilde{z}_i)^2$$

2.4 Mean Squared Error

One way to measure the accuracy of our model is through the mean squared error (MSE), this is often used instead of the absolute value due to it having a non-vanishing first order derivative. Due to it being the *square* of the error, the MSE is always non-negative with values closer to zero signifying a better fit.

$$\text{MSE}(\vec{y},\vec{\tilde{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 \tag{4}$$

2.5 R^2-score

Another way to measure the accuracy of the model is through the R^2-score

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

The R^2-score provides a measure of how well the observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.

2.6 k-Fold Cross-Validation

Cross-validation is a re-sampling technique used to evaluate machine learning models on a limited data sample. The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it. This is done to notice problems such as overfitting or selection bias. In k-fold cross-validation we split the original sample into k randomly partitioned equal sized subsamples. Of the k subsamples, $k-1$ of them are used as training data, while the remaining one is used as validation data for testing the model. The cross-validation is then repeated k times, with each of the k subsamples used exactly once as the validation data. Afterwards, the k results can be averaged to produce a single estimation for the predictive power of the model.

2.7 Bias-Variance Trade-off

When considering a data set $\mathbf{L}$ we assume that the true data is generated from a noisy model

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

with $\vec{\epsilon}$ normally distributed with mean zero and standard deviation σ. We have defined the approximation to the true data as $\tilde{y} = \mathbf{X}\vec{\beta}$ where we find the parameters $\vec{\beta}$ by optimizing the mean squared error (4). We can split the error into terms for the bias, the variance and the deviation for the noise.

$$\begin{aligned}
\mathbf{E}(\vec{y}-\tilde{y})^2 &= \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2 \\
&= \frac{1}{n}\sum_{i=0}^{n-1}\big(f_i+\epsilon-\tilde{y}_i+\mathbf{E}(\tilde{y})-\mathbf{E}(\tilde{y})\big)^2 \\
&= \frac{1}{n}\sum_{i=0}^{n-1}\Big(\mathbf{E}\big((f_i-\mathbf{E}(\tilde{y}_i))^2\big)+\mathbf{E}(\epsilon^2) \\
&+\mathbf{E}\big((\mathbf{E}(\tilde{y}_i)-\tilde{y}_i)^2\big)+2\mathbf{E}\big((f_i-\mathbf{E}(\tilde{y}_i))\epsilon\big) \\
&+2\mathbf{E}\big(\epsilon(\mathbf{E}(\tilde{y}_i)-\tilde{y}_i)\big) \\
&+2\mathbf{E}\big((\mathbf{E}(\tilde{y}_i)-\tilde{y}_i)(f_i-\mathbf{E}(\tilde{y}_i))\big)\Big) \\
&= \frac{1}{n}\sum_{i=0}^{n-1}\Big(\big(f_i-\mathbf{E}(\tilde{y}_i)^2\big)+\mathbf{E}(\epsilon^2) \\
&+\mathbf{E}\big(\mathbf{E}(\tilde{y}_i)-\tilde{y}_i\big)^2\Big)
\end{aligned}$$

Another way to write this is:

$$\begin{aligned}
\mathbf{E}(\vec{y}-\tilde{y})^2 = \frac{1}{n}\sum_i\big(f_i-\mathbf{E}(\tilde{y})\big)^2 \\
+\frac{1}{n}\sum_i\big(\tilde{y}_i-\mathbf{E}(\tilde{y})\big)^2+\sigma^2
\end{aligned}$$

Here we recognize the first term in the sum as the bias and the last as the variance. The bias error stems from erroneous assumptions in the learning algorithm, while the variance is an error from a too great sensitivity to small fluctuations in the training set. They embody respectively the concepts of under-fitting and over-fitting to the data. It is typically the case that the more complex the model is the more of the training data it will envelop creating a better fit to the data. However, the complexity will make the model vary more to capture all the training data, hence the larger variance for greater complexity models.

2.8 Ridge Regression

The difference between O.L.S. and ridge regression is that we have added a penalty term to the cost function. This penalty is proportional to the square of the magnitude of the regression coefficients. The goal of this is to reduce the standard errors associated with analyzing multiple regression data which suffers from multicollinearity. More on how to deal with collinearity in data sets here[1]. This is realized by adding a term to our least squares cost function (2)

$$\min\frac{1}{n}||\vec{y}-\mathbf{X}\vec{\beta}||_2^2+\lambda||\vec{\beta}||_2^2$$

This is known as a Ridge regression minimization problem where we require $||\vec{\beta}||_2^2 \leq t$ with $t > 0$. Where the subscript signifies the norm.

2.9 Variance of $\vec{\beta}_{\text{ridge}}$

Due to the difference in the terms of the ordinary least squares and the ridge estimator, the variance is also different. We start from the variance of the o.l.s estimator (3), and writes the ridge estimator as a function of the o.l.s estimator

$$\begin{aligned}
\vec{\beta}_{\text{ridge}} &= \big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\vec{y} \\
&= \big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{X}\big(\mathbf{X}^T\mathbf{X}big)^{-1}\mathbf{X}^T\vec{y} \\
&= \big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{X}\vec{\beta}_{\text{o.l.s.}}
\end{aligned}$$

Thus, we write the variance of the ridge estimator as

$$\begin{aligned}
&\mathrm{Var}(\vec{\beta}_\lambda) \\
&= \big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{X}\mathrm{Var}(\vec{\beta}_\lambda)\big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{X}\big)^T \\
&= \big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{X}\mathrm{Var}(\vec{\beta}_\lambda)\mathbf{X}^T\mathbf{X}\big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1} \\
&= \big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\mathbf{X}^T\mathbf{X}\sigma^2\big(\mathbf{X}^T\mathbf{X}\big)\mathbf{X}^T\mathbf{X}\big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1} \\
&= \sigma^2\big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}\big(\mathbf{X}^T\mathbf{X}\big)\big(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}\big)^{-1}
\end{aligned}$$

2.10 Lasso Regression

The basic idea of Lasso regression is similar to Ridge regression. In Lasso regression we force the sum of the absolute value of the coefficients to be smaller than some value, $||\vec{\beta}||_2^2 \leq t$. This forces some of the coefficients to be set to zero, generating a new simpler model. This is opposed to Ridge regression because in Ridge regression the coefficients can be shrunk, but not set to zero.

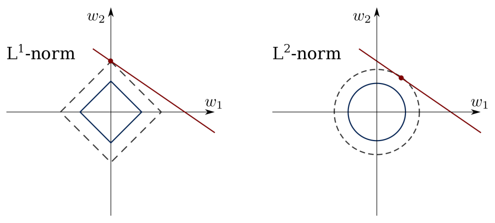


Figure 1: Difference in the constraint regions for the Lasso and Ridge methods.(2)

3 Method

In this section we explain how we went about getting our results presented in the results section.

3.1 Code Structure

We created a class in python called Regression whose constructor takes a couple arguments used to determine the size and shape of the data and how to model it. Based on the given number of points we create a dataset y using Franke's function as explained in section 2.1 with added noise of a chosen magnitude. We then populate a design matrix using the provided x and y-values(y here is the second dimension of the grid that defines the function, not the dataset), and the chosen polynomial degree. Having our design matrix X and our data y from Franke's function, we can then use the given regression method(OLS, Ridge, Lasso) to compute the β's, which in turn can be used to find our model's prediction by taking the dot product with the transposed design matrix of the data we want to make a prediction on

$$\tilde{y} = \beta \cdot X^T$$

In addition to this core functionality we created functions to do different types of error analysis. This includes the mean squared error, R^2 score, bias and variance. We also computed the confidence intervals for the β-values and plotted them.(Figure 4) We also implemented some command line-functionality to make customizing the different parameters easier, and the ability to choose between terrain data and Franke's function.

Lastly we created functions outside the class to loop over different parameters and do a full regression fit each time, saving the error values so we can plot them as a function of the variable we loop over. This gives us most of the plots shown in the results section.

3.2 Evaluating Franke's Function with Stochastic Noise

First we evaluated Franke's function with added stochastic noise. We calculated the mean squared error and the R^2-score. In addition we also computed variance of the predictors for the given polynomial degree. We compared our result to the pre-built functions in the Python library sci-kit learn. We then computed the error several times for a range of polynomial degrees from 0 to 30.

3.3 Franke's Function with k-fold Resampling

Next we implemented k-fold resampling. We first tried doing everything manually, but had a lot of trouble getting it to work. (We address this in the discussion section (5.4)) We ended up using sklearns KFold class to split our data(still doing everything else manually). With the KFold class, we are given the correct indexes for test and training data, and so all we need to do is compute our both our train- and test-design matrices and use the training matrix to compute beta-values for the training data fit. With these beta values we can take the dot product with the test design matrix, to produce our prediction on the test data without using the actual values in the process of making the prediction.

3.4 Regression types

Ridge regression was implemented using matrix inversion and Lasso regression is done using sklearns Lasso-function. We have a separate function for each regression type in our class, and a parent function find-fit which uses the correct regression type based on the type specified in the command line. The find-fit function returns all the beta values for the computed fit.

4 Results

In this section we will give a brief run-down of the data generated from our regression analysis of Franke's function and height data taken from `https://earthexplorer.usgs.gov/`. We chose to use date from the Grand Canyon, Arizona U.S. Our results will be more thoroughly discussed in the discussion section (5).

4.1 Evaluating Franke's Function with Stochastic Noise

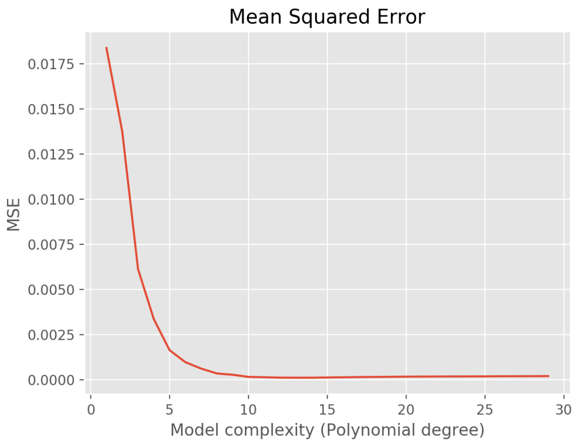


Figure 2: The Mean Squared Error as a function of the model complexity for Ordinary Least Squares regression

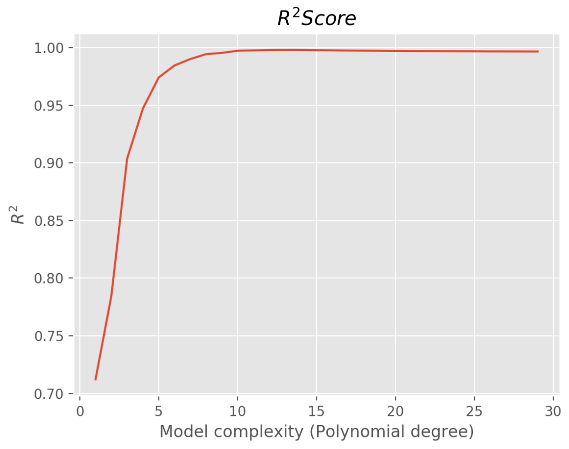


Figure 3: The R^2 as a function of the model complexity for Ordinary Least Squares regression

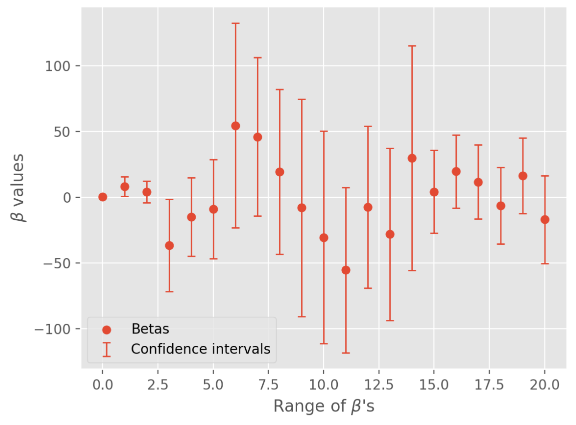


Figure 4: Confidence intervals for the different values of beta resulting from a 5th degree fit of the Franke function. We used a added noise value of $\epsilon = 0.1$

In figure 4 we have a scatter plot of all the beta-values produced when using ordinary least squares to fit the Franke function with a 5th degree polynomial. The vertical bars represent the confidence interval for each given beta-value. Note that there is no correlation between the different beta-values and

4.2 Franke's Function with k-fold Resampling

Following are the graphs of our ordinary least squares regression for respectively the mean squared error and R^2-score with k-fold re-sampling.

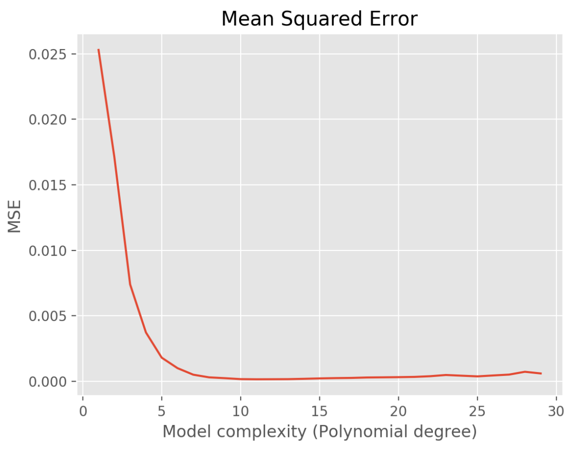


Figure 5: The Mean Squared Error as a function of the model complexity for Ordinary Least Squares regression with k-fold re-sampling

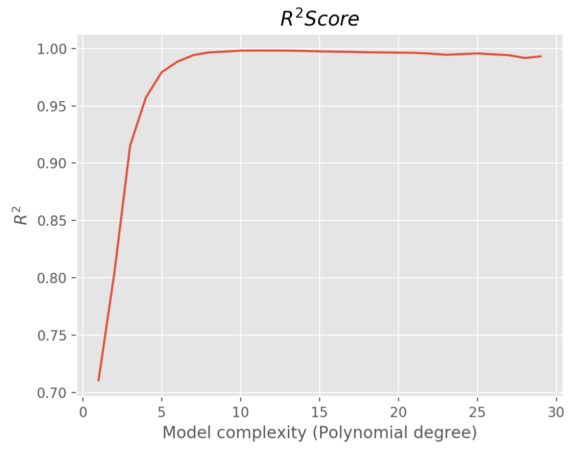


Figure 6: The Mean Squared Error as a function of the model complexity for Ordinary Least Squares regression with k-fold re-sampling

4.3 Ridge Regression of Franke's Function

Following are the graphs of the mean squared error and R^2-score for our ridge regression with varying values of λ

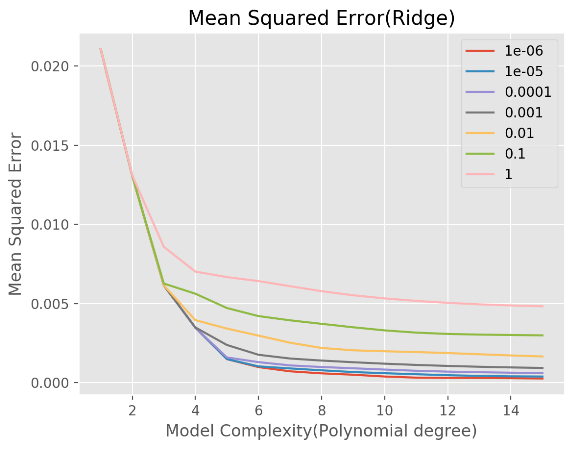


Figure 7: MSE for our ridge regression as a function of the penalty parameter λ

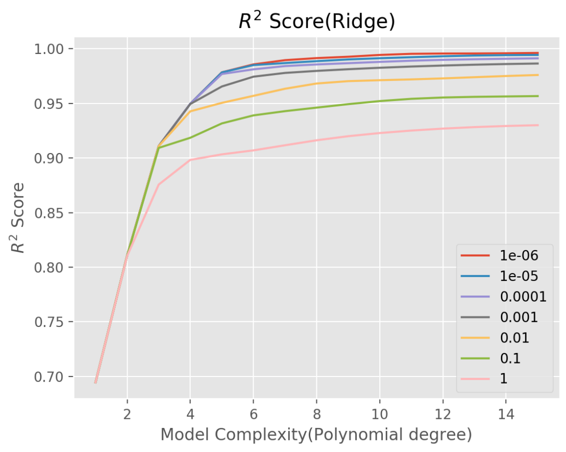


Figure 8: R^2-score for our ridge regression as a function of the penalty parameter λ

4.4 Lasso Regression of Franke's Function

Following are the graphs of the mean squared error and R^2-score for our lasso regression with varying values of λ

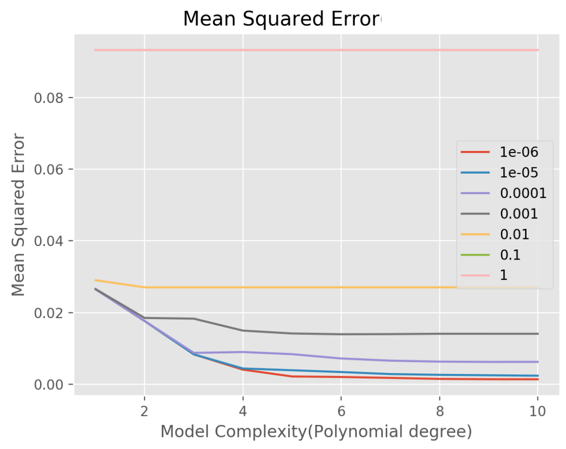


Figure 9: Mean squared error for lasso regression plotted for several values of λ

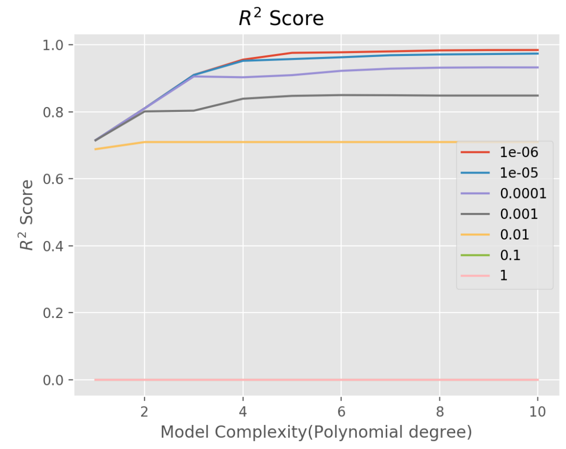


Figure 10: The R$^2 - score$ for lasso regression plotted against multiple λ values

4.5 Variance

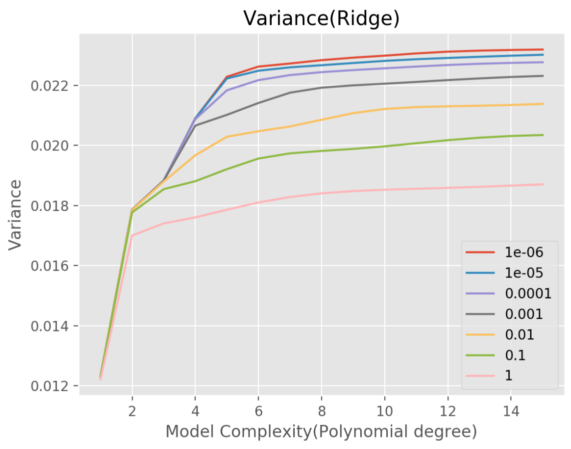


Figure 11: Variance plotted against model complexity using ridge regression on Franke's function with noise

Figure 12: Variance of the lasso regression on Franke's function with noise plotted against model complexity

4.6 Terrain data

All the methods used to evaluate Franke's function were applied to real terrain data from the Grand Canyon. A comparison of how the terrain is fitted using each regression method for different model complexities is shown in the appendix section (Figure 14).

5 Discussion

Throughout this project we have analysed various aspects of the ordinary least squares, ridge and lasso regression algorithms. We used the metrics of the mean squared error in addition to the R^2-score to evaluate the different methods. In addition we added resampling to give a better estimate of the quality of our algorithms.

5.1 Ordinary Least Squares Analysis of Franke's Function

As expected the mean squared error decreases as our model complexity increases. The R^2-score increases to almost 1.0 around the same polynomial degree as the mean squared error reaches close to zero. The polynomial degree at which these values reach their convergence is dependent on the data we are fitting to. So for fitting to Franke's function with added noise $\epsilon = 0.1$ using Ordinary Least Squares, we have found that a 10th degree polynomial will give the best result, and increasing the model complexity past that is unnecessary.

5.2 Ridge Regression Analysis of Franke's Function

Similarly to the ordinary least squares analysis we observe that the mean squared error decreases as our model complexity increases. We also observe the R^2-score increases with the complexity. However, differing from the o.l.s. results we see the MSE and R^2-score respectively increase and decrease once our polynomial degree reaches approximately $d = 15$.

5.3 Lasso Regression Analysis of Franke's Function

Again, the mean squared error decreases and the R^2-score increases with the model complexity. However, where o.l.s. and ridge stabilized at approximately equal values, the lasso regression stabilized at a significantly higher value for both metrics across all λ's.

5.4 k-fold Resampling

As mentioned in section 3.3 we first tried implementing k-fold all from scratch. From what we could tell we were doing everything correctly, looping through each fold, creating training data from different chunks every time, predicting a fit for the test data, and saving all the error values. We then computed the mean value from all these errors.

However something about our code was not correct at this point, so when looping over the polynomial degree and plotting the errors, the results showed periodic spikes in all the error values. The spikes would gradually(or rather exponentially) increase in magnitude. So if our MSE was around 0.01, it would spike to 1.0, back to ≈ 0.01, up to 100, back to ≈ 0.01, up to 10000 and so on. We were advised by the instructor to not spend too much time on this part, so we ended up using sklearns KFold class to split our data. At this point we did write the whole code from scratch, so the error could also have been from something else.

5.5 Variance

As we can see from figure 12 and (Insert variance lasso), the variance increases as our model complexity increases, which is what we expect. A fit using a lower degree polynomial will be smoother and thus have less variance.

We can also see that the higher value of λ, the less variance we have. This also is expected, since λ is used as a penalty term in ridge regression to reduce potential errors, and a high lambda will smooth out our function and decrease the accuracy of the estimation drastically.

5.6 Terrain data

A contourf with colorbar, made from 0.15% of the original data, using matplotlib's pyplot module is shown in figure 13. The selection of the data was chosen so our lasso analysis would be able to complete in a reasonable amount of time. Even though we are using a very small part of our data, the result is able to capture the complexity of the grand canyon as well as highlight the differences in the regression methods. Another positive with using a limited piece of the data set is that overfitting is slightly more possible, giving yet another way to highlight the differences in the methods.

Figure (14) shows a matrix of figures containing o.l.s, ridge and lasso regression of the above terrain image. We can see that for higher degrees that o.l.s over-fits more than the other algorithms. This has to do with the shrinkage and removal parameter of respectively ridge and lasso. We used a low shrinkage parameter, giving a result fairly close to the o.l.s algorithm while at the same time having pronounced differences. We found that both ridge and lasso tend to smooth out more than o.l.s with the smoothness being more pronounced in the lasso regression.

In the table (1) we wanted to include the values for the lasso regression, however we mistakenly did not save them and due to time constraints we were not able to reproduce the plots. Running scikit-learn's lasso algorithm takes a really long time with bigger data sets and design matrices. This is because it optimizes the learning rate while running to find the optimal fit.

6 Conclusions

Ordinary least squares was successfully used to fit two-dimensional data. Inspection of the different error measurements on Franke's function showed the expected behavior. We saw that the mean squared error decreased as the model complexity increased, and converges on a number where the function will no longer benefit from a higher polynomial degree. The opposite behavior was observed on the r^2 score, which increases and converges close to 1.

Using k-fold resampling, we started seeing some evidence of overfitting. As the model complexity increased beyond a certain point(around 15), the MSE would start going up and the R^2 score would start to decrease. As far as we understand, this is due to the function doing complex fits to a certain part of the data. So when testing this fit on the test-data, the function is trying to imitate small variations that aren't even in the test data.

We were successful in implementing ridge- and lasso-regression as well, however neither of them seem to be the better choice for any of our data sets. From what we understand our data sets are too smooth for ridge and lasso to make a better prediction, since overfitting and collinearity are not a problem.

7 Sources

(1) https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf
(2) https://en.wikipedia.org/wiki/Lasso_(statistics)#/media/File:L1_and_L2_balls.svg
(3) Taboga, M. (2010) ”Lectures on probability and statistics”, https://www.statlect.com
(4) earthexplorer.usgs.gov/ (5) Hastie, Tibishirani, Friedman, Elements of Statistical Learning, Springer.

A Terrain Data Comparison

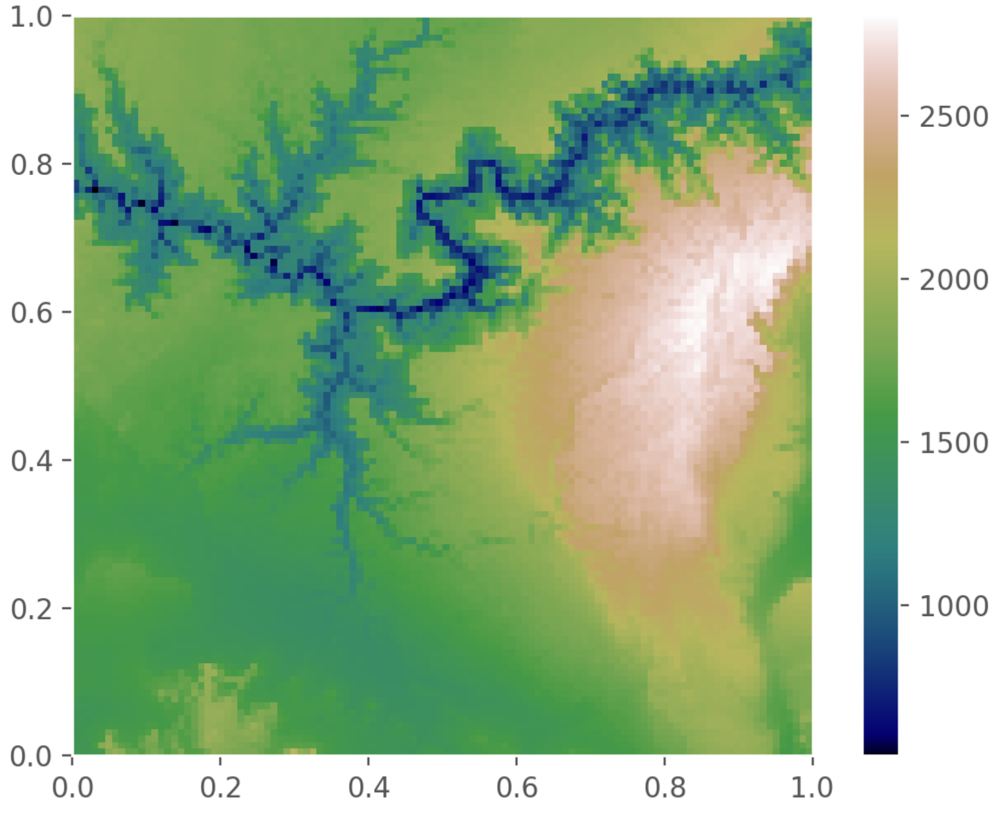
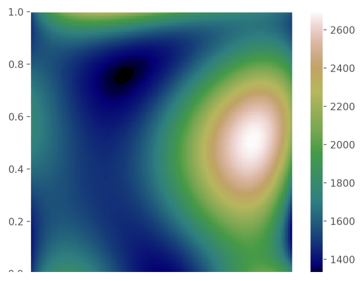
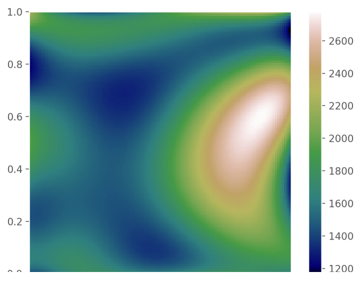
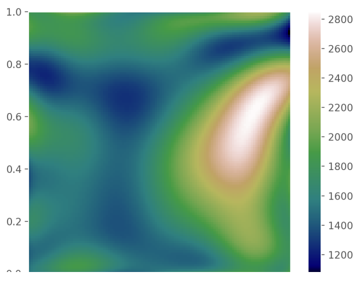
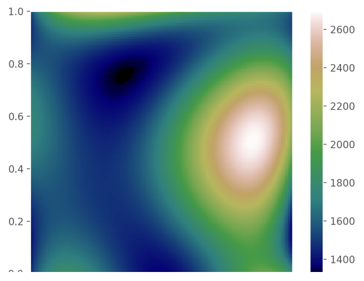


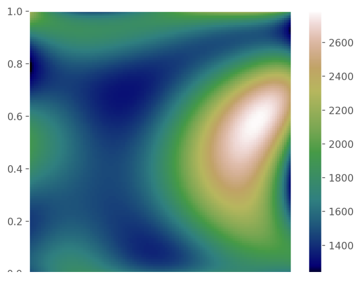
Figure 13: Top down representation of the Grand Canyon AZ. which was used for the following regression using o.l.s

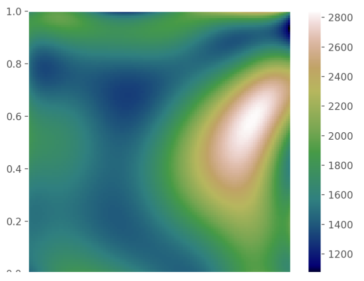

(a) o.l.s, $d = 5$

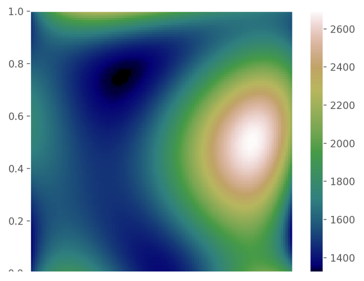

(b) o.l.s, $d = 7$

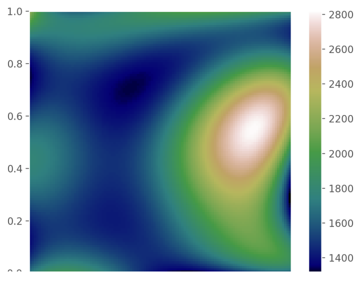

(c) o.l.s, $d = 10$


(d) ridge, $d = 5, \lambda = 1e-6$


(e) ridge, $d = 7, \lambda = 1e-6$


(f) ridge, $d = 10, \lambda = 1e-6$


(g) lasso, $d = 5, \lambda = 1e-6$


(h) lasso, $d = 7, \lambda = 1e-6$

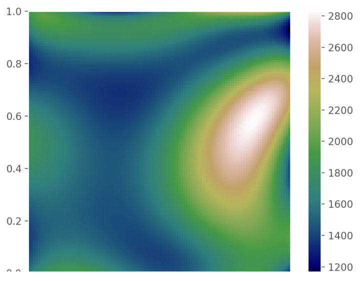

(i) lasso, $d = 10, \lambda = 1e-6$

Figure 14

Polynomial degree	5		7		10	
	MSE	R^2	MSE	R^2	MSE	R^2
o.l.s	0.0080	0.6461	0.0058	0.7424	0.0047	0.7935
	MSE	R^2	MSE	R^2	MSE	R^2
ridge	0.0080	0.6461	0.0058	0.7413	0.0052	0.7686

Table 1: Normalised values of the mean squared error and the R^2-score for the regression fits in the figure above.