Report Project 1

Erlend Øydvin

October 11, 2021

Abstract

In this project I have adapted a polynomial model to fit Frankes Function and a weather radar image of a precipitation event. This was done using Ordinary Least Squares (OLS), Ridge regression and Lasso regression. The results show that OLS for noisy datasets and high polynomial degrees tends to overfit the data and that this overfitting can be dampened using Ridge or Lasso regression. The overfitting can also be reduced by increasing the samples (resolution) from the radar image. I also show that Ridge and Lasso shrinks the regression coefficients by reducing their variance and spread around their mean value.

1 Introduction

Historically, attempts to fit polynomials to observed data has been quite successful. An explanation for this could be that simple polynomial models makes intuitive sense and relates quantities in a way that often can be visualized. Another explanation could be that many physical models behave in a way that can be described more or less easily with polynomials. Mathematically polynomials behave nice with regards to for instance integration and derivation. In this report we investigate how an n-degree two dimensional polynomial of the form $[1,\ x,\ y,\ x^2,\ y^2,\ xy..]$ can be fit to Franke's Function and a weather radar image. This fitting is done using the three famous regression techniques: Ordinary Least square (OLS) regression, Ridge regression and Lasso regression. The code for this project can be found in the following git-repository:
https://github.com/erlnwind/FYS-STK4155/tree/main/Project_1

2 Theory

2.1 Franke's Function

In order to test the regression algorithms, Franke's Function with some added noise, was used to generate a noisy surface. Frankes's function consists of two peaks of various height and a smaller dip, see Figure 1. According to the lecture notes, it is widely used to test various interpolation algorithms. It reads as follows:

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

2.2 Weather radar image

As I have just started working on a PhD (6 weeks as of now) about precipitation, I thought I'd study some extra relevant data. Weather radar images of precipitation has much of the same characteristics

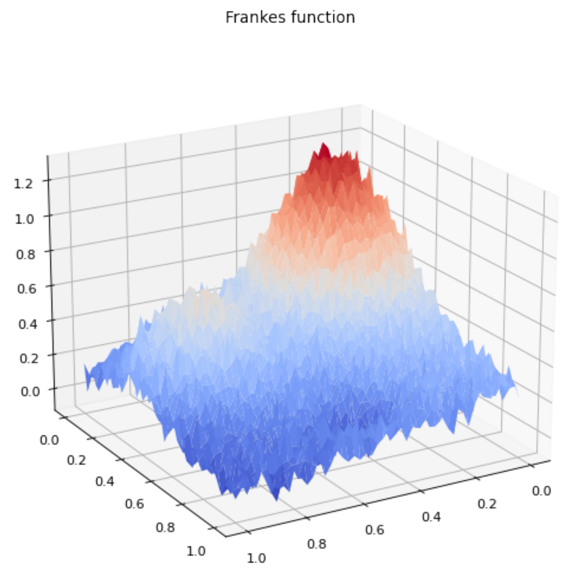

Figure 1: 3D-Plot of Franke's Function with some added noise. We can observe the two peaks in red and the smaller dip in dark blue

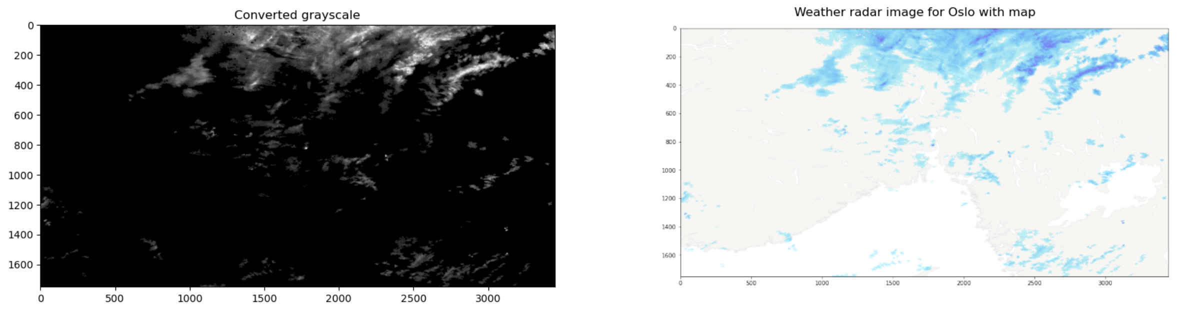

Figure 2: Left: Grayscale conversion of weather radar image. Right: Precipitation in shades of blue with a map of Norway behind.

as terrain data in that it can be thought of as a function z of x and y. It is also high resolution and has a structure where two points close to each other are more correlated than points lying far from each other. However, as I have not (yet) gained access to any raw radar data source, I have instead used an open API from GeoNorge to get real-time precipitation estimates based on radar reflectivity. The API can be found following this link:

https://kartkatalog.geonorge.no/metadata/meteorologisk-kartinformasjon/c247f4a3-56a8-49fa-afe0-a0329b9453e1

The images downloaded from Geo Norges API can be filtered so that they contain only estimated precipitation and is thus easier to handle than images downloaded directly from Meteorologisk Institutts own API, Frost. The downloaded image is a color image where rain with various intensities is represented by shades of blue (and red for large precipitation events) and zero rainfall is represented by white. Assuming that the shades of blue linearly represents measured rainfall these colors was then converted to grayscale, where zero rain is represented by black and maximum rain (found in this picture) is represented by white. See Figure 2. I have included in my GitHub repository an Jupyter Notebook that illustrates this workflow. Further work will be to make sure that the conversion of blue colors to grayscale is correct or to find an API that returns data that is easier to load. However, for this exercise the current conversion seems sufficient.

2.3 Regression methods

Linear regression is a supervised learning method, and thus requires labeled data for training the model. In the Franke function it is assumed that x and y represents coordinates in x-y plane, i.e in meter and that the resulting z-value represents height. For the weather radar image the x and y column represents coordinates with length of unknown unit. The z-value represents the amount of precipitation, also in an unknown unit. By scaling the data these 3 dimensions are made comparable. Common for the regression methods we use in this project is that they seek to minimize the following cost-function

$$C(\vec{\beta}) = \frac{1}{n}\sum_{i=1}^{n-1}\left(y_i - \tilde{y_i}\right)^2, \tag{2}$$

where $\vec{\beta}$ is the parameters to be minimized, y contains reference data and $\tilde{y}$ contains an approximation to y. Equation 2 is also called the mean sum of error squared and is in fact minimized using the Ordinary Least Squares method (OLS). OLS can be seen as the orthogonal projection of $\vec{y}$ onto the column space spanned by the polynomials

$$\vec{\beta} = (A^T A)^{-1} A^T \vec{y}, \tag{3}$$

where A is the design matrix for the p-th order polynomial and y is the labeled train data. A common problem encountered by OLS is that given a high enough polynomial the method tends to generate large regression coefficients, making the model overfit to data. The Ridge regression method seeks to solve this problem by introducing a penalty on the size of the regression coefficients. It can thus prevent over-fitting of datasets by preventing that large contributions from one regression coefficient cancels out the contribution from another. The Ridge method can be stated as follows

$$\vec{\beta} = min\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{i=1}^{n} x_{i,j}\beta_j\right)^2, \tag{4}$$

$$\sum_{i=1}^{n}\beta_j^2 \leq t. \tag{5}$$

where Equation (4) is recognized as the standard least square problem and Equation (5) is recognized as the bond. The Lasso regression method also introduces a penalty on the size of the regression coefficients, but in this case the beta coefficients are restricted by their absolute value. The Lasso method can be stated as follows:

$$\vec{\beta} = min\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{i=1}^{n} x_{i,j}\beta_j\right)^2, \tag{6}$$

$$\sum_{i=1}^{n}|\beta_j| \leq t. \tag{7}$$

where Equation (6) is recognized as the standard least square problem and Equation (7) is recognized as the bond.

2.4 Bias - Variance Trade-off

A model that has high bias is typically too simplistic to fit the data very well. Increasing the model complexity (ie. introducing a larger polynomial order) makes it easier for the model to fit to the data, thus decreasing the bias. However, if the data contains noise, giving too much freedom to the model (too large polynomial order) may cause the model to adapt to this noise, increasing its variance. See Figure 3. Both high bias and high variance is bad. A good compromise between these two sides is usually the best when doing regression. In Appendix A it is shown that Equation (2) can be rewritten into

$$\frac{1}{n}\sum_{i=1}^{n-1}\left(f_i - \mathbf{E}[\vec{\tilde{y}}]\right)^2 + \frac{1}{n}\sum_{i=1}^{n-1}\left(\tilde{y}_i - \mathbf{E}[\vec{\tilde{y}}]\right)^2 + \sigma^2. \tag{8}$$

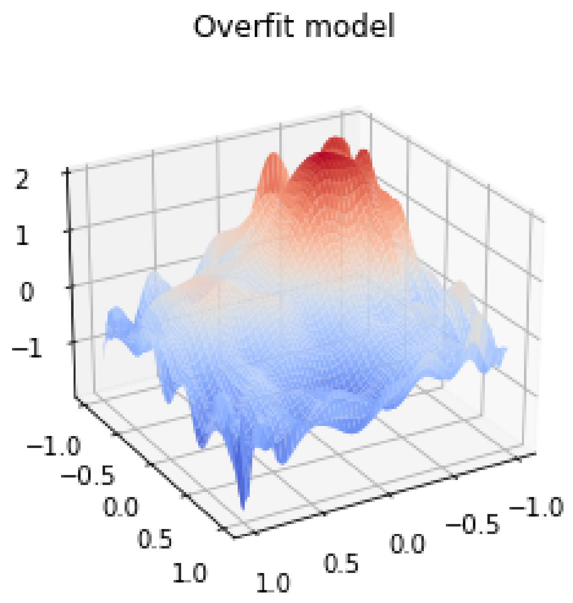

Figure 3: Over-fit polynomial fit to Franke's Function. This model has low bias and high variance.

In this equation the first term can be interpreted as the error resulting from bias, the second term can be interpreted as the error resulting from variance and the last term can be interpreted as the error resulting from irreducible variance. The best model is usually a good compromise between bias and variance.

3 Methods

3.1 Pre-processing and scaling data

When given a dataset the first steps are to look at the data. For the Franke Function this amounts to determine how much noise that are reasonable to apply. I added noise so that it was still possible to work out the contours of Frankes Function when looking at a 3D plot. For the weather radar data this process was more complicated as I had to relate shades of blue to numbers between zero and one. This process was done by filtering out irrelevant colors and relating the remaining shades of blue to numbers between zero and one (grayscale). More work is definitely needed in this step.

The next step was to scale the data. Scaling was applied in two steps, first subtracting the x, y, and z vector its mean value (centering) and then dividing by the same mean value. The first step, centering, makes sure that all data are centered around zero. This is advantageous as polynomials are good at estimating things centered around zero, but requires an massive effort to estimate things far away from zero. Also, not centering the data would imply a large β_0 coefficient had to be used to "lift" the estimated polynomial up to where the z-values are. This is disadvantageous when using Ridge and Lasso regression as they would punish large β_0. Further, we do not expect the weather radar images coordinates (longitudes and latitudes) to have any relation with observed rainfall, other that its spatial correlation. Thus dividing by its mean or variance makes the columns of the design-matrix dimensionless and more comparable.

3.2 Cross Validation

The idea behind k-fold cross validation is to divide the whole dataset into k folds. Then leaving out the first fold, train the model against the other k-1 other folds and test it against the left out fold. The process is repeated for all k-folds as test producing k estimates of the best polynomial fit. The best polynomial fit is then taken to be the mean of all k folds. The main advantage of k fold cross

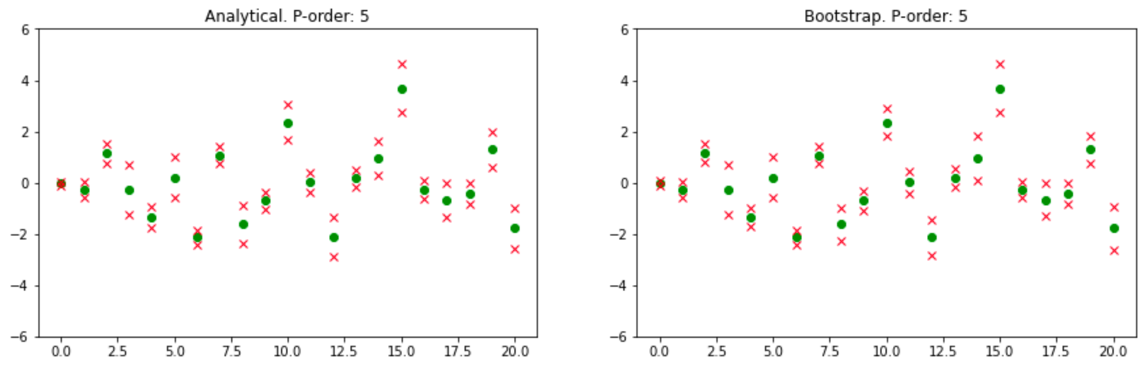

Figure 4: Plot of analytical derived standard deviation for beta, and bootstrap re-sampling estimate of beta variance for OLS. Green dots are expected values and the red crosses marks its sigma one confidence interval.

validation is that it runs quite fast and produces reliable estimates.

3.3 Bootstrap re-sampling

Bootstrap works by randomly drawing, with replacement, n samples from the dataset such that the same data can be chosen many times. Then the model is fit to this randomly sampled bootstrap sample. The process is repeated b times and the optimal model is the mean of all the bootstraps. Given that the data is independent and identically distributed, bootstrap also has the ability to estimate the uncertainty of the model. By taking the variance of the bootstraps we can do an estimate of the uncertainty in, for instance, the regression coefficients. This can, according to [Hastie et al., 2009] be though of as the Monte-Carlo estimate of variance. In Figure 4 we can see an comparison between the analytical derived uncertainties in beta and the bootstrap estimate of beta. By increasing the number of bootstraps the bootstrap estimate of variance in beta becomes more and more similar to the results of the analytical expression.

3.4 Numerical uncertainties

In this project I have implemented my own code for OLS and Ridge. These methods was tested against the same methods implemented in scikit-learn. My methods seems to give similar results, but for noisy data with over-fitting and large polynomials my code seems to differ. In 5 we can observe the same OLS bootstrap fit to the same data, in the leftmost picture we see that my implementation and the sklearn implementation gives similar results, at least up to polynomial 20. After this the two methods diverge from eachother, this could be due to numerical errors. Another interesting behaviour I have observed is that the resolution (or smoothness) of the weather radar image or Franke's Function seems to affect if the OLS method over-fit or not. As a consequence of this I have tried to study both resolutions that generate over-fit and resolutions that do not seem to generate over-fit.

4 Results

4.1 OLS, Ridge and regression on Franke's function

In Figure 6 we can see a plot of the beta values and its confidence intervals of a p order polynomial that are being fit to Franke's function using Ridge regression. The same polynomial was fit to the same noisy data in Figure 4 using OLS regression. Here we can observe that using a low lambda ridge makes Ridge regression behave more like OLS regression, returning similar beta-values and similar confidence intervals. Increasing lambda reduces the variance of the beta values, driving them towards zero. In Figure 7 we can observe the same effect using Lasso. Notice also that since I did not implement my own version of Lasso the beta values are displayed in the order given by in SK-Learn, which is different from the order in my code. Another interesting result is that I'm not able to produce visible confidence

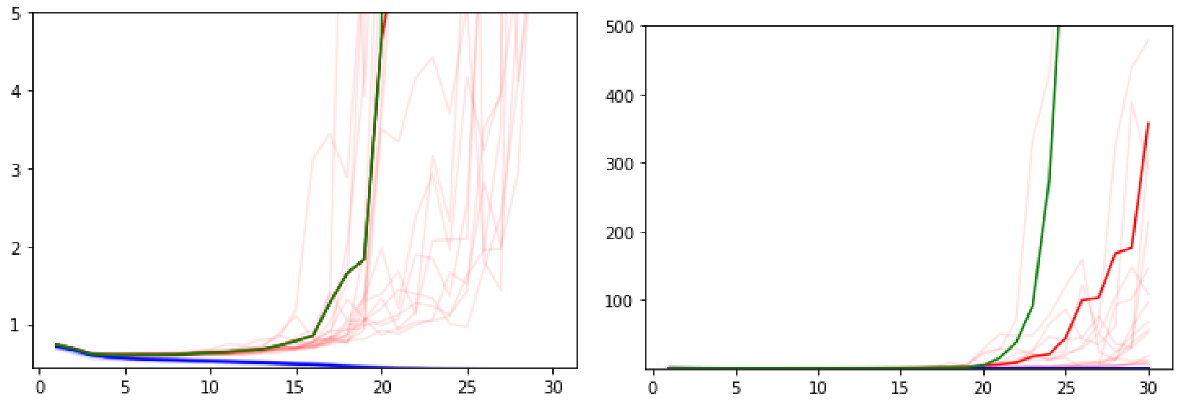

Figure 5: Blue line is train dataset, small red lines are different bootstraps from my code (fat red is the average of small red lines) and the green line is the average of scikit-learn's bootstraps. My code seems to produce the same results as scikit-learn up to polynomial 20, after this the methods diverge from each other.

intervals for Lasso regression. This might be due to error in my code, or the fact that Lasso shrinks the confidence intervals to zero very quickly. I was not able to run the code with lower lambda lasso values as that increased the computational time too much.

4.2 OLS, Ridge and Lasso on weather radar image

In Figure 8 we can observe the MSE of different polynomial fits to a low resolution version of the weather radar image in Figure 2 using OLS, Ridge and Lasso. For polynomials above 21 the OLS test group seems to explode, indicating that the polynomial above this degree seems to over-fit the data. The over-fitting effect is dampened using Ridge and Lasso, resulting in almost flat lines for polynomials lower than degree 5. Increasing lambda for Ridge and Lasso seems to increase MSE. If I increase the weather image resolution I do not get the same over-fitting tendencies. See Figure 9.

In Figure 10 I have plotted the optimal polynomial fit for different polynomial degrees and different regression methods for a low resolution weather radar image. We can see that the OLS fit seems to be able to reproduce more of the wavy behaviour of the radar image than Ridge and Lasso. Ridge is able to produce a quite detailed surface as well, and it does not change much after polynomial degree 16. Lasso have the fit that seems to change the least with polynomial order, even if its MSE is almost identical to Lasso MSE and OLS MSE for polynomial degree 16, see 8. Increasing the resolution of the image makes the OLS method able to capture much finer details, further lowering MSE.

5 Discussion and conclusion

5.1 Ridge and Lasso on weather radar data

In Figure 8 we can see that the OLS test group for a polynomial fit to weather radar image above degree 16 rises rapidly. This indicates that the polynomial is over-fit to data and are adapting to noise. In Equation 8 the second term expresses this increase in variance as the complexity of the model is a larger source of error than what is gained from reducing the bias. By comparing the visual results in Figure 10 it looks like the polynomial model is able to capture the larger trends in the dataset, but not the finer details in the weather radar image. On the other side, weather radar is known to produce measurements that have a lot of uncertainties and noise. Thus a polynomial fit might be a better representation of the real world. However, rainfall is known to have a high spatial variability, making the radar picture more likely to be correct, and the polynomial model more likely to be false. Ridge and Lasso results seams able to effectively prevent over-fitting by posing a bond on the beta values. Ridge is able to fit to finer details as I am able to set lambda low enough. I believe Lasso would be able to do the same given enough computational resources. Around polynomial 16 both methods seem to stop producing more details. This could indicate that the polynomial model is not a good

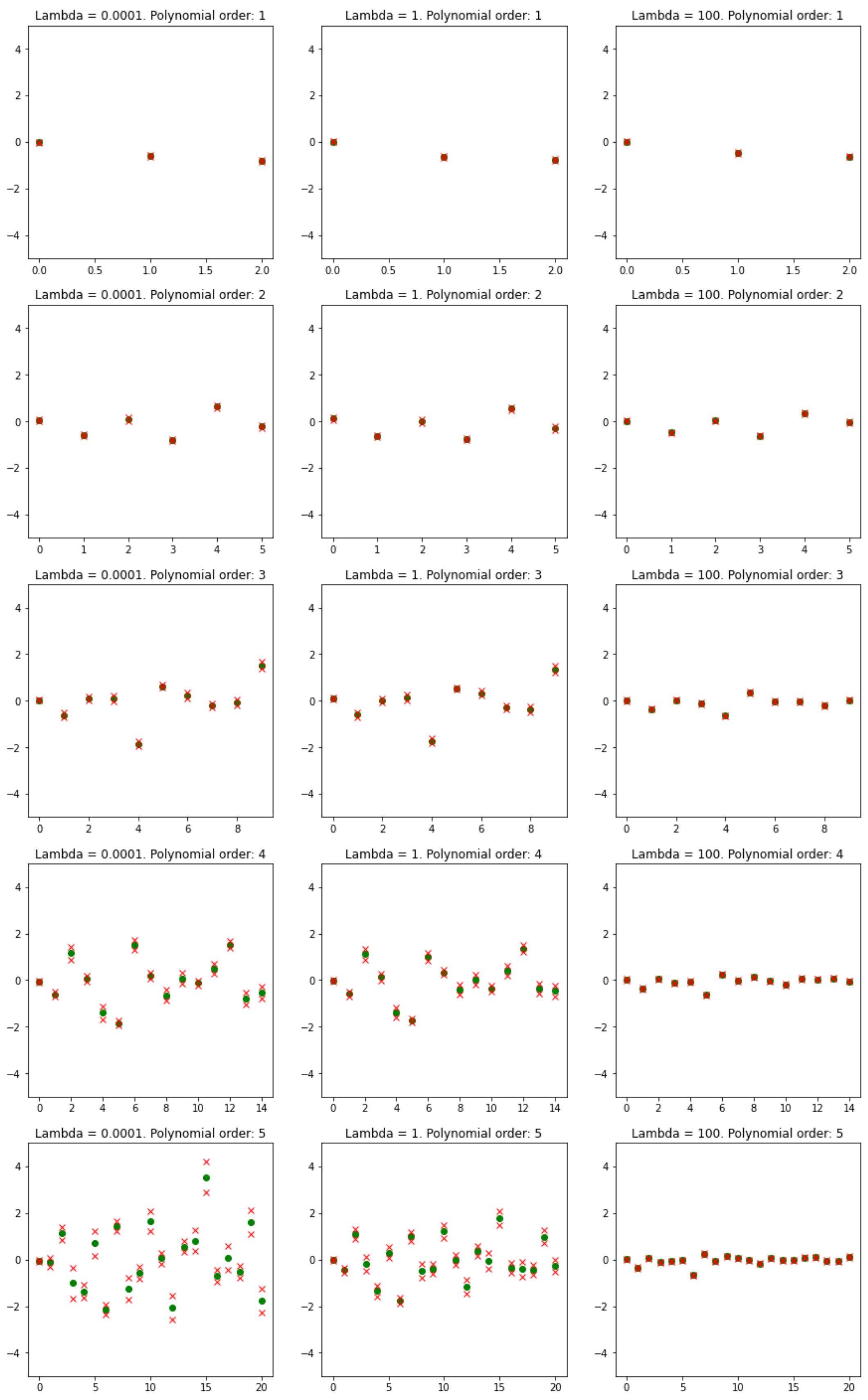

Figure 6: Beta values (green o) and their confidence interval (red x) for Ridge regression for polynomial degrees up to degree 5.

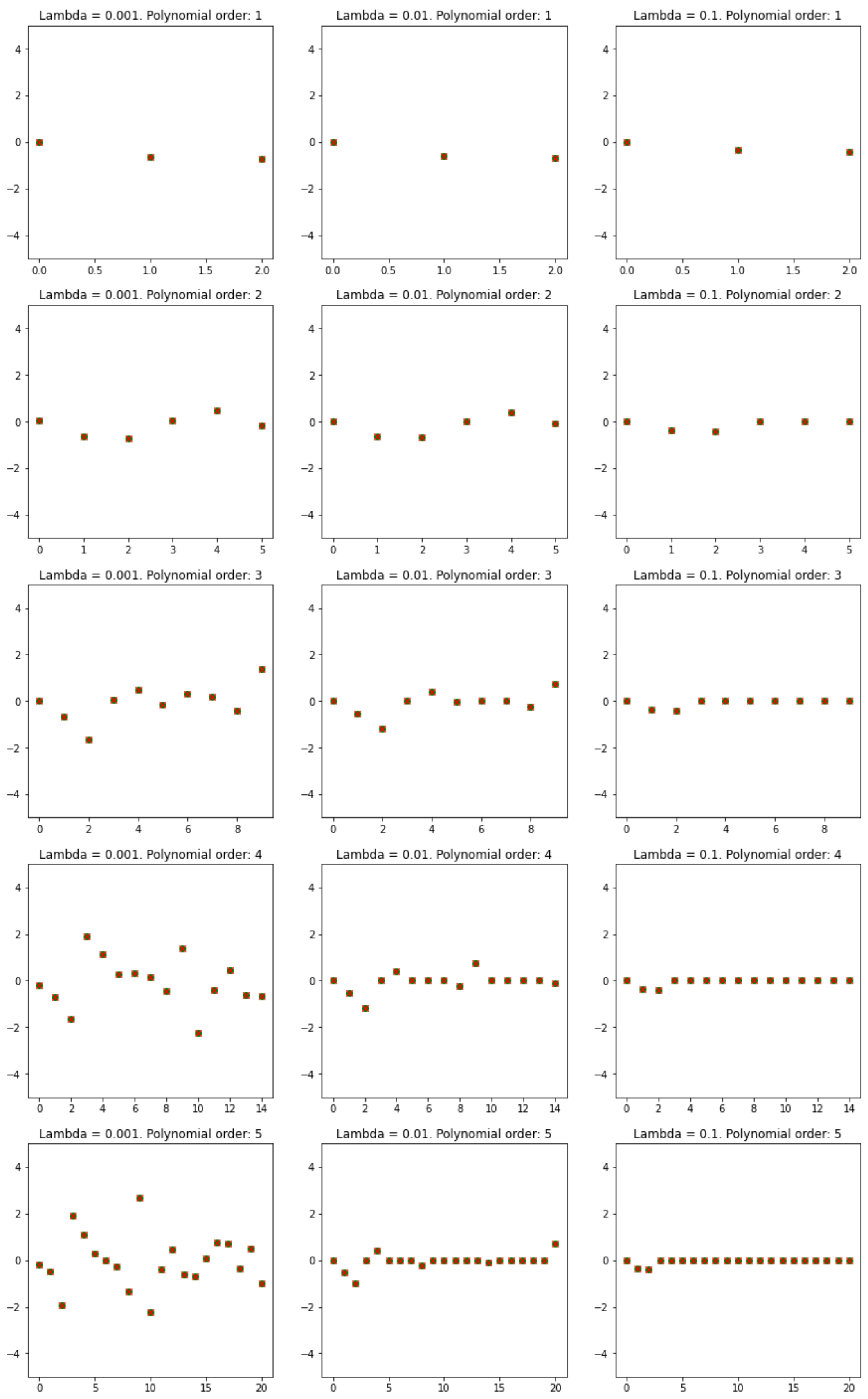

Figure 7: Beta values (green o) and their confidence interval (red x) for Lasso regression for polynomial degrees up to degree 5.

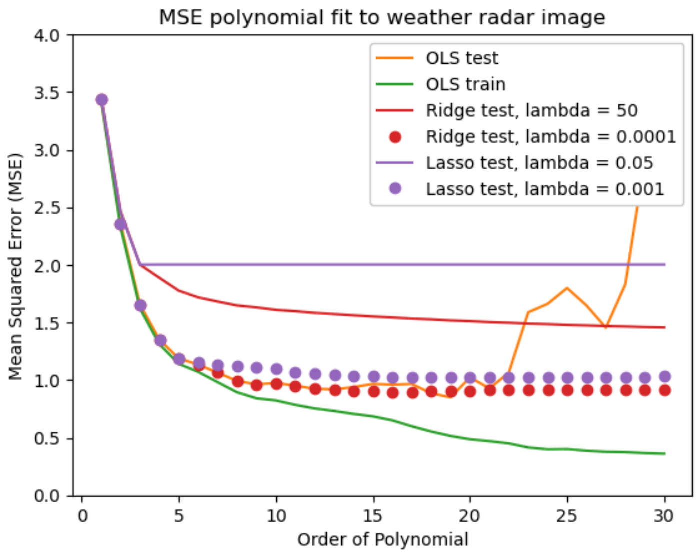

Figure 8: Mean squared error for a polynomial fit using OLS, Ridge and Lasso on a low resolution weather radar image.

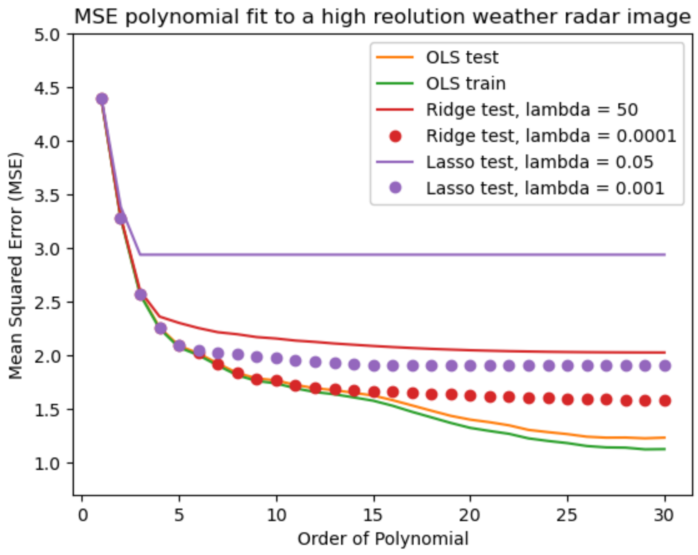

Figure 9: Mean squared error for a polynomial fit using OLS, Ridge and Lasso on a high resolution weather radar image.

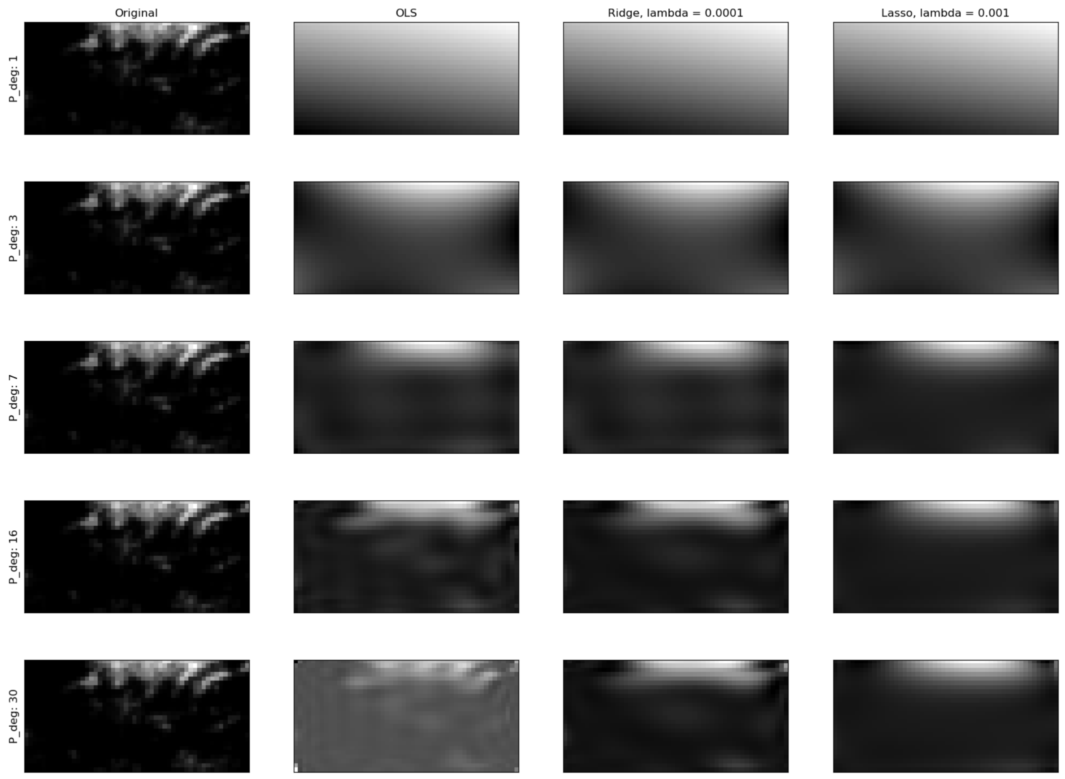

Figure 10: Optimal fit to weather radar image using different polynomial degrees and OLS, Ridge and Lasso.

model to model the finer details in a weather radar image. It could also indicate that noise in the the weather radar image is just as large as the finer details itself, raising the question of what is noise and what is details. Still, they are able to model the larger trends pretty well. Increasing the resolution of the weather radar image makes the OLS test follow the train group, at least up to polynomial degree 30. This indicates that increasing the resolution reduces the chance of OLS over-fitting to data. One reason for this could be that including more points makes it more expensive (according to the cost function) for OLS to not follow the larger trends, thus dampening the fit to noise. It could also indicate that including more points makes the noise to details ration smaller, increasing the likelihood of the polynomial adapting to the real rain details.

By comparing Lasso and Ridge MSE in Figure 8 we observe that its MSE is almost similar. We could thus expect the resulting fit to also be almost identical. However, this is not the case. In Figure 10 we can observe that Ridge regression makes a better job than Lasso regression in reproducing finer details, already at polynomial degree 7. This behaviour can be explained by noting that Lasso regression shrinks its regression coefficients to zero, suggesting that the best visual fit requires small beta values to play a role (Ridge allows small beta values).

This analysis could for instance be used as a way to classify precipitation events by grouping them by which polynomials where important in producing the fit, using for instance SVD.

5.2 Further work

This project have generated many new ideas which I have not had time to explore. Further work on these topics would definitely include studying different radar images with polynomials or maybe other functions and see if it possible to distinguish them from each other using for instance SVD. Another cool ting to do could be to study what happens if I hold one beta-value constant, and let the rest vary. How would that affect the confidence intervals and would it be possible to see which of the beta values are more dependent on each other? Another interesting task could be to instead of blurring the weather image to lower resolution, would sampling randomly from the weather radar image improve the predicted z-values? I have implemented this idea as a last minute bonus ipython notebook, sadly it does not show any improvements from a higher resolution weather radar image. More work on this could include using more bootstraps and higher polynomial orders.

References

References

[Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning, Second Edition.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

Appendix A: Bias-variance trade off

Equation (2) can be written on the more compact form

$$\mathbf{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] \tag{9}$$

Suppose the true data was generated from a noisy model

$$y = f + \epsilon \tag{10}$$

Inserting this into Equation (9) we get

$$\begin{gathered}\mathbf{E}[(f + \epsilon - \tilde{y})^2] \\ \mathbf{E}[(f + \epsilon - \tilde{y})(f + \epsilon - \tilde{y})] \\ \mathbf{E}[(f^2 + \epsilon f - f\tilde{y} + f\epsilon + \epsilon^2 - \epsilon\tilde{y} - f\tilde{y} - \tilde{y}\epsilon + \tilde{y}^2]\end{gathered}$$

Since ϵ is assumed to have mean equal to zero, terms with a single ϵ becomes 0

$$\begin{gathered}\mathbf{E}[(f^2 + f\tilde{y} + \epsilon^2 - 2f\tilde{y} + \tilde{y}^2]. \\ \mathbf{E}[f^2] - 2\mathbf{E}[f\tilde{y}] + \mathbf{E}[\tilde{y}^2] + \mathbf{E}[\epsilon^2]\end{gathered}$$

The variance of a vector y can be expressed as follows

$$\begin{gathered}\mathbf{E}[\tilde{y}^2] - \mathbf{E}[\tilde{y}]^2 = \mathbf{E}[\tilde{y}^2 - \mathbf{E}[\tilde{y}]^2] \\ \mathbf{E}[\tilde{y}^2] = \mathbf{E}[\tilde{y}]^2 + \mathbf{E}[\tilde{y}^2 - \mathbf{E}[\tilde{y}]^2]\end{gathered}$$

Thus we have that

$$\begin{gathered}\mathbf{E}[f^2] - 2\mathbf{E}[f\tilde{y}] + \mathbf{E}[\tilde{y}]^2 + \mathbf{E}[\tilde{y} - \mathbf{E}[\tilde{y}])^2] + \mathbf{E}[\epsilon^2] \\ \mathbf{E}[(f - \mathbf{E}[\tilde{y}])^2 + \mathbf{E}[\tilde{y}^2 - \mathbf{E}[\tilde{y}]^2] + \mathbf{E}[\epsilon^2]\end{gathered}$$

Since

$$\mathbf{E}[\epsilon^2] = \sigma^2$$

We get

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

$$\frac{1}{n}\sum_{i=1}^{n-1}\left(f_i - \mathbf{E}[\vec{\tilde{y}}]\right)^2 + \frac{1}{n}\sum_{i=1}^{n-1}\left(\tilde{y}_i - \mathbf{E}[\vec{\tilde{y}}]\right)^2 + \sigma^2.$$

Which is the bias variance relation in Equation 8.