Exploring Linear Regression Methods: Validation and Terrain Model Applications

Project 1, FYS-STK4155, UiO

Varvara Bazilova, Sergio Andrés Díaz Meza

September 2020

Abstract

Regression methods are widely used Machine Learning algorithms. In this report we present the results of Ordinary Least Squares, Ridge and Lasso regression models with different degrees of complexity and penalty parameters. We also explore this with Bootstrap and K-Fold cross-validation techniques. We test these regression methods on the functional data (Franke's Function) and on a real terrain data (SRTM elevation model). We conclude that for the Franke's Function the best performances were achieved with the OLS regression from a 10 to a 15 complexity degree, however, based on our investigations, the optimal slope values were given by the complexity degree of 12. Ridge and Lasso regressions allow to achieve good solutions to overfitting problems by tuning the penalty parameter. However since this was a known function, there is a high chance of a good polynomial fitting as we saw, so this parameter does not affect the result as much as the complexity degree variations, but rather increase the performance errors. For the SRTM regression model performances seems to be achieved better in higher degrees of complexity than the ones with Franke's Function with penalty parameters ranging from 1 to 4, however more computational efforts are required to achieve better and reliable results.

1 Introduction

Several daylife data requires methods to predict values, no matter if they are for simple curves and geometries, or for complex multi-variable cases. For the latter the one might need to employ several combinations of complex functions and transformations that create non-linear problem. However, there are several regression methods that allow to simplify complex problems into a linear equation.

Therefore, the aim of this report is to explore various regression methods and to study, how to fit a function (using a polynomial function as an example) to a two-dimensional dataset. We consider the following methods: 1) OLS (Ordinary Least Squares) 2) Ridge Regression and 3) Lasso Regression to investigate how the complexity of the model affects the predictions of data (generated by Franke's Funcrion (Section 3.1). We also investigate the variability of model parameters and asses model performance using cross-validation and resampling techniques. And finally, following these resultsthe linear regression is applied to estimate the elevation data, based on SRTM didital elevation model scene (Farr et al. (2007)).

2 Background

2.1 Linear Regression methods

Regression methods are a family of Machine Learning algorithms, based on supervised learning. The main goal of regression method is to find a functional relationship between input data (independent variables/predictors/explanatory variables) and a reference dataset and to fit a continuous function with parameterization, that maps data to continuous output values. This allows to make predictions of the dependent variables using this functional. Despite not being a very advanced method among other supervised learning algorithms, regression is widely used due to it's simplicity.

Linear regression methods assume, that the the regression function $E(Y \mid X)$ is linear in the inputs $(X_1, X_2, ...)$ (Friedman et al., 2001).

Therefore, the general simplified target equation is given as follows:

$$y_i = f(\beta * X_i); \beta = [\beta_0, \beta_1, ..., \beta_{p-1}] \tag{1}$$

where y - is a dependent variable, X - is the matrix of predictors of independent variables (also called design matrix (3.2), β - is the regression parameter. The goal of the regression methods is to find the minimum of the cost function, which is expressed differently for every method.

To asses the model and to pick an optimal complexity, the so-called bias-variance trade-off is used. Bias-variance trade-off is a property of the model, that describes the difference between the variance (error originated from model applied to test data) and the bias (error originated from model performance on test data). Low variance together with high bias lead to overfitting problem: the model performs "too well" on the training data, that causes the poor performance on the training set.

2.1.1 OLS

Ordinary Leas Squares is one of the simplest methods to find β coefficients for (eq. 1). The optimal parameters β are derived from minimizing the sum of squared errors (residuals) and taking an average value of it. Therefore, the $\hat{\beta}$ is given by (explained in 3.2 in more detail):

$$\hat{\beta}_R = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y} \tag{2}$$

where $\mathbf{X}$ is the design matrix.

2.1.2 Ridge Regression

Ridge regression (also called Tikhonov regularization) is a method, which is similar to OLS, but it allows to avoid singular-matrix problem, that might occur while fitting β parameters to the design matrix in a simple way. In general, this method improves the efficiency of the fitting in exchange for an increase in bias. Ridge regression is one of the so-called shrinkage methods, because it allows to shrink the variance.

The $\hat{\beta}$ is given by (explained in 3.2 in more detail):

$$\hat{\beta}_R = (\mathbf{X}^\intercal \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\intercal \mathbf{y} \tag{3}$$

where $\mathbf{I}$ is an identity matrix, and λ is the penalty parameter. Ridge parameter is constrained: $\lambda \geq 0$ to avoid problems when minimizing a cost function. If $\lambda = 0$, then the $\hat{\beta}$ estimation reduces to OLS.

2.1.3 Lasso Regression

Lasso (Least Absolute Shrinkage and Selection Operator) is another shrinkage method. It uses λ is the penalty parameter, but it is applied on a slope and intercept of the model.

2.2 Resample Techniques

Resampling techniques are used to achieve a better result, if the number of datapoints is limited. These methods allow to estimate statistical parameters of the sample, using the subset of the available data. There are several resempling methods: Bootstrap, K-fold, Jackknife, Leave One Out cross-validation and many more. Resempling methods also allow to do cross-validation of the data.

2.2.1 Bootstrap

Bootsrtap is a method, which is often used in Machine Learning applications. Resampling methods consist of repeatedly drawing samples from a training set and refitting a model on each sample in order to obtain a statistical information about the model parameters and fitted model. In other words, bootstrap provides a way to build B-number of models from slightly different (resampled) data. The data is sampled with replacement, which allows to specifically to resample the training dataset, not just shuffle it. In principal, when B (number of bootstraps) goes to infinity, then the statistical properties of the sample, approach the statistical properties of the population.

2.3 Cross-Validation

Cross-validation is a resampling method, which is used for model evaluation, if the amount of data is limited. Random sampling and splitting for a train and test dataset might have some unbalanced influence on the model fit and the following model performance: some datapoints might be sampled multiple times, some might not be sampled at all. There are many ways of doing cross-validation of the model. The main idea is that the data, that is used for the validation, is a resampled training dataset.

3 Data and Methods

3.1 Data

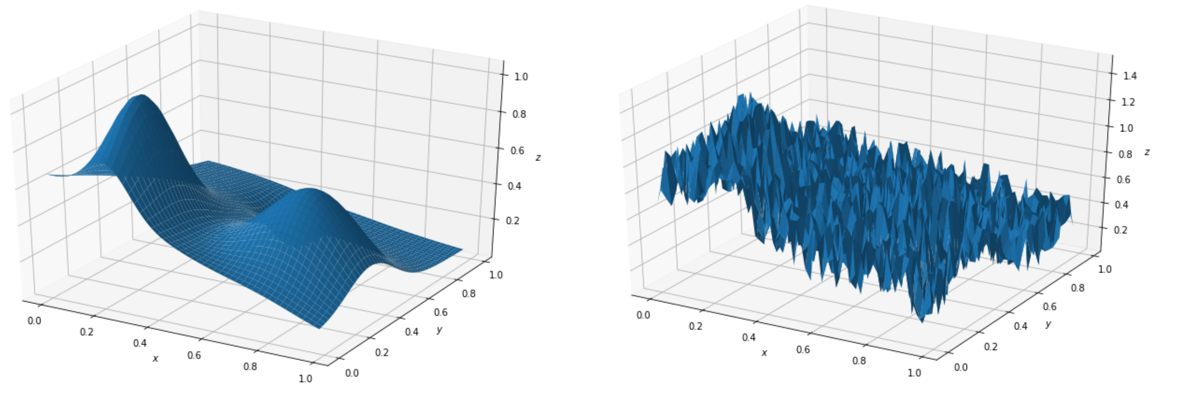
The data for the project is generated using the Franke's function. This is a two-dimensional function, that has two Gaussian peaks of different heights, and a smaller dip.

It is initialized as follows:

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

For this study the function was defined for $x, y \in [0, 1]$ to avoid further normalization of the variables. Also, we added stochastic noise to the 50% using the normal distribution $N(0, 1)$ since the use use of the dataset without noise will just give overfitting due to working with a smooth surface (see Figure 1). For the OSL regression we used 400, 1156 and 2500 data points to see, how the dataset size affects the train and test performances. For the rest of the regression methods (Ridge and Lasso), the dataset used consists of 2500 points. The difference in the results is described in analysis section (**??**).

The SRTM DEM (Shuttle Radar Topography Mission Digital Elevation Model) is a near-global digital elevation model Farr et al. (2007). It represents the terrain elevation values and is widely used for multiple geoscientific and many other purposes. The SRTM was flown on board NASA space shuttle Endeavour in 2000. The DEM was constructed by NASA from the processing of the Radar data (C-band and X-band). The resolution of the data is 1 Arc-Second, which corresponds to about 30 meters.



(a) (b)

Figure 1: Franke function without noise (a) and with noise(b).

For the current project, we have downloaded one square scene of the SRTM DEM, available from USGS (United States Geological Survey) archive. The data, used for the report is located in the southern Colorado (USA). We tried to find an ares, that would have different terrain properties (such as mountains, valleys, meadows and plain flat areas) to explore, how would regression models perform on a complex data which doesn't have any functional description.

The SRTM DEM data is stored in a standart *.tif* format as a geo-referenced raster. File size is approximately 25 MB.

3.2 Methods

We set the x and y points to be between 0 and 1 and then a matrix X is generated for every model independently, based on the polynomial degree of the function (model complexity) (see Equation 5). In this case, since the x and y points were generated through a meshgrid between 0 and 1, then there is no need to normalize the design matrix. Also, for testing the error performance, several error functions were introduced: Mean Square Error (MSE) and R^2 (See *project_1.ipynb* file, section A).

$$\begin{bmatrix} \bar{y}_0 \\ \bar{y}_1 \\ \vdots \\ \bar{y}_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^N \\ 1 & x_1 & x_1^2 & \dots & x_1^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^N \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{N-1} \end{bmatrix} \tag{5}$$

With the data made, we have done simple OLS regression (polynomial fit) with our own inversion and with the Scikit-Learn module (See *project_1.ipynb* file, section B) by varying the data set points and doing iterations up to 25 complexity degrees. Out method is generation a solution of the Equation 2. Through the function *np.arange* we produce different lengths of data points: 400, 1156 and 2500 points, both for the x and y axis for the meshgrid (See *project_1.ipynb* file, section A). For this method and the next to come we defined a test set equals to the 20% of the data set by making use of the $train_test_split$ function of Scikit-Learn. The number of data points was set to 2500. We made this in order to obtain an idea of the optimum polynomial degree fit.

Next, we implemented both algorithms as before but with a Bootstrap cross-validation technique of 500 bootstraps over a pre-defined train set by using the same splitting method with the same test ratio

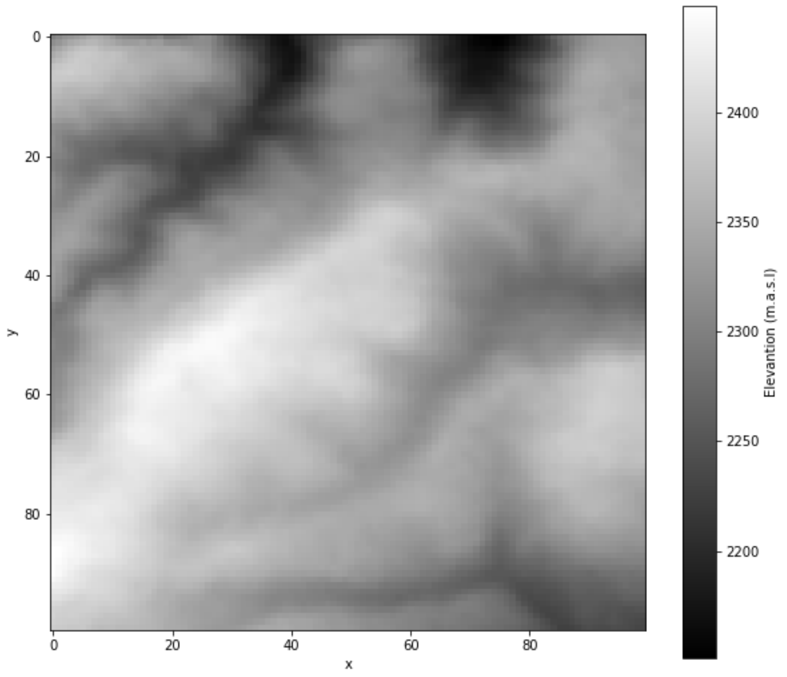


Figure 2: SRTM digital elevation model downsampled to 10000 points by taking the upper left-corner section of it.

(See *project_1.ipynb* file, section B) and up to 20 complexity degrees. We have done this by doing a loop over the Bootstrap quantity, and averaging over the iterations.

Next, we did a K-Fold cross-validation by varying the number of K-Fold from 5 to 10 with the same regression method, but leaving a fixed polynomial degree of 15 (See *project_1.ipynb* file, section C). The way to approach this was by: dividing the entire data set into K groups, doing a loop over the groups and every time selecting one of them as a test data set and the rest as train set, fitting the model with the train set and doing predictions with the test, and finally averaging at the end of the loop.

To study the Ridge and Lasso regression, we applied out algorithm but introducing the λ variable as shown in Equation 3 for the Ridge. However, due to the difficulty of programming it, we decided to use the Scikit-Learn package for applying the Lasso regression. For both we varied the λ value from 0 to 20 for complexity degrees varying form 0 to 20 in order to study this trade-off influence, for both train and test data sets (split in the same way as in OLS). We applied also Bootstrap of 20 and K-Folding of 10 and 50. (See *project_1.ipynb* file, section D and E).

For the SRTM digital elevation model, we tried several iterations, but the data is too heavy. After this, we decided to downsample the data up to a section of it that contains a total amount of 10000 data points. To normalize the data, we redirect the pixels information to a meshgrid where the x and y variables ranges from 0 to 1 (See *srtm.ipynb* file).

We applied the OLS, Ridge and Lasso regression over the data set, with the same properties and conditions as the the Franke Function test, but with several changes (See *srtm.ipynb* file). In the OLS, we established a complexity degree range from 15 to 35. For both Ridge and Lasso regressions we set a maximum λ penalty value ranging from 0 to 10 and complexity degrees ranging from 5 to 15. We thought of this since higher degrees are reflected in a bigger design matrix X and thus there is more computational time implied (already seen with the simple OLS computation time). To make things efficient, we parallelized the process by running part of the code on a local machine and the most time demanding (Lasso) on a server with 32 CPUs and 256 GB of RAM).

4 Results

4.1 OLS Regression

In all the OSL regression (see Figure 3), regardless of the algorithm used, the train-curve decreases and flattens, when the complexity is increased. At the same time, the test-curve is trying to do the same at the beginning (with the low model-complexity). It changes after a after certain complexity value: the test-curve starts to do up (the high complexity model has a high MSE values). We see that the level of model complexity depends on the algorithm used and the number of datapoints.

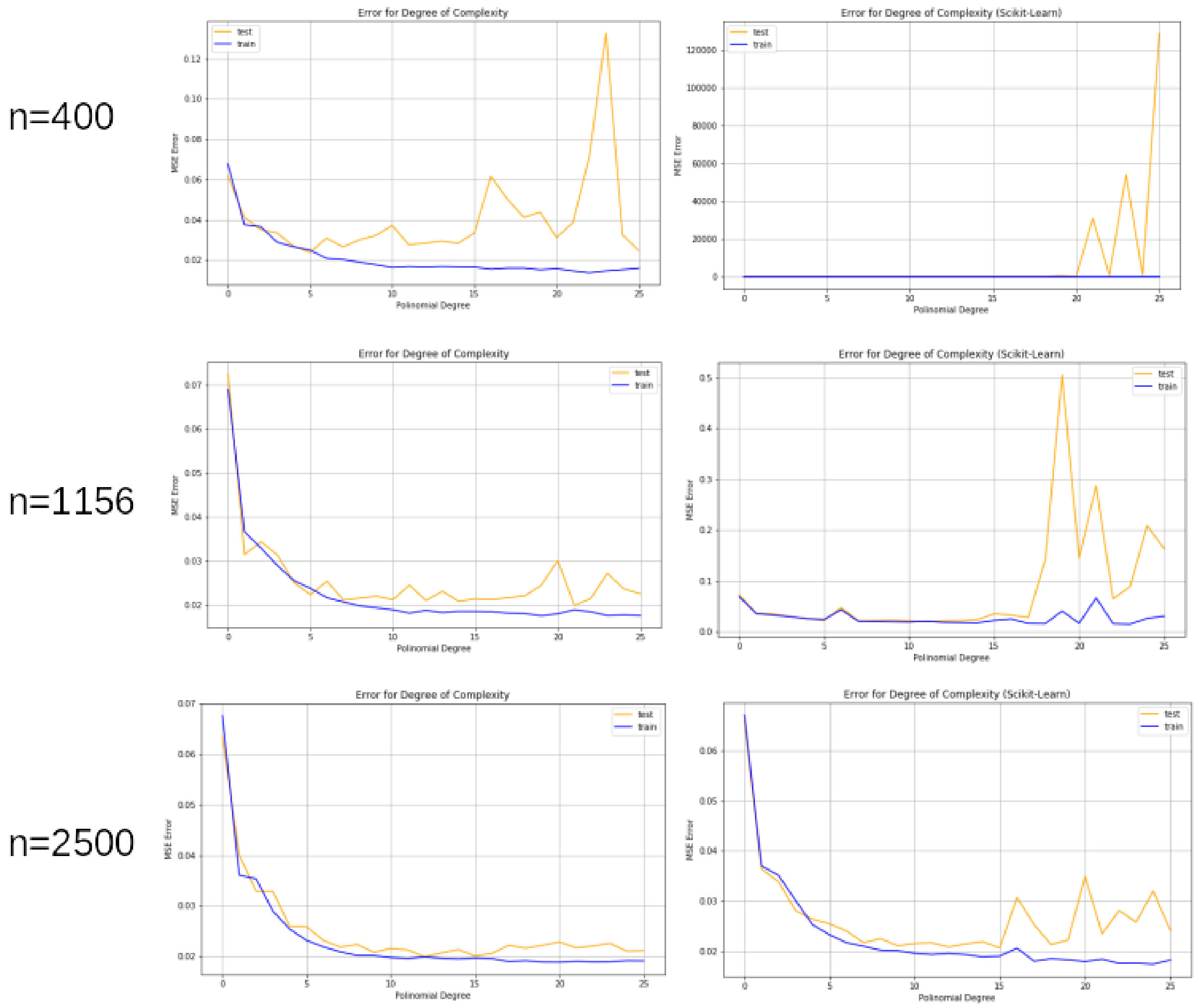
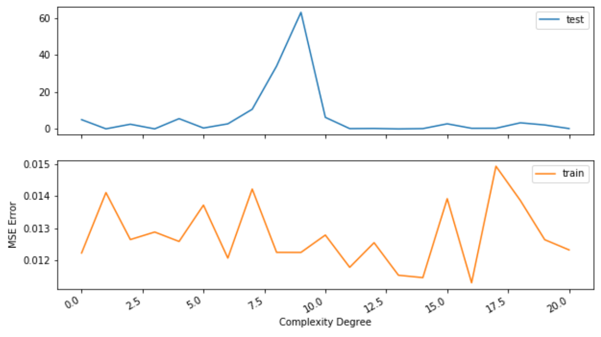
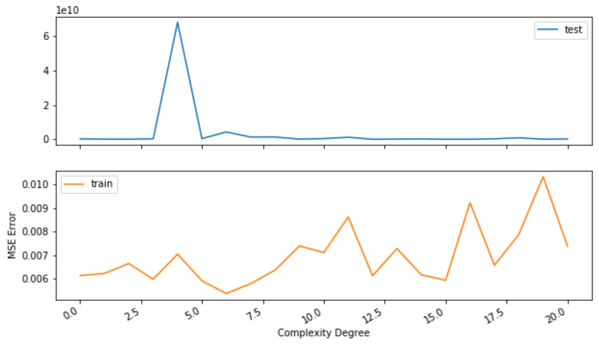


Figure 3: OLS results for the different number of points (400, 1156 and 2500). Left: OLS algorithm, constructed manually, Right: is the standard Linear Regression OLS from the Scikit-Learn library. Blue and Orange lines represent the MSE error of the Train and Test data, respectively.

Mainly, the Scikit-Learn algorithm shows the increase in trade-off in a higher complexity degree than our manually-written algorithm (for the same amount of points). So the trade-off, produced by the Scikit-Learn algorithm is more explicit. Also, as the number of points increase, the complexity level increases



(a) Developed Algorithm.



(b) Scikit-Learn Algorithm.

Figure 4: OLS regression with the bootstrap resampling method up to 20 degrees of complexity. We used a bootstrap value of 20 and a number of 2500 data points. (a) and (b) show results with manual algorithm and with Scikit-Learn algorithm, respectively. Blue and orange lines are the test and train sets.

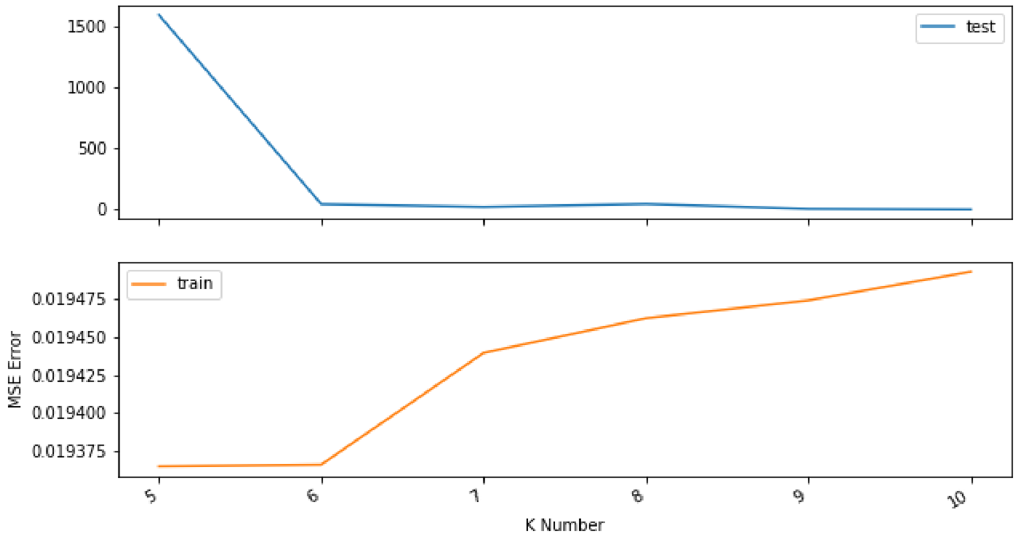


Figure 5: OLS regression into a 15 polynomial degree with variation from 5 to 10 K-Fold resampling numbers.

where the MSE training curve and the trade-off is low.

With the bootstrap resampling method (see Figure 4), the model performance on the training set has extremely low MSE values (up to the order of 10^{-3} in the Scikit-Learn algorithm) for all complexity degrees considered.

However, the test set are extremely high, even at low complexities. Note, that for the manual-method, the values are also high, but only in one order or magnitude (10^{1}). For the OLS regression using K-Fold resampling method for train/test splitting (see Figure 5), the MSE value decrease for the test set. We see a clear behavior of the MSE: values decrease (with the increase of the K number) and approaches 0. The train set just reveals MSE values in the order of 10^{-1} but as the K number increases the MSE value does it as well and tries to converge to a maximal value in the same order of magnitude.

4.2 Ridge Regression

The results for Ridge regression performance are all shown in the Figure 6. For Simple Ridge and Bootstrapped regressions, first thing noticed is that the model is less sensible to λ value changes than to complexity degrees. Nevertheless, Test set always exhibits more error than the Train set, which is the expected behaviour.

The K-Folding give less variability of the λ value influence on the regressions, but it shows it in a more gradient way, increasing the performance error towards the increase of λ. However, as the K number is higher (from 10 to 50), the error in the Test set decreases. Something noticeable is the pop-up MSE values for $\lambda = 0$.

4.3 Lasso Regression

Lasso regressions are shown in Figure 7 in the same way as the Ridge regression. It shows constantly low error values, with some random variations on the λ influence over the performance, but thew complexity degree dominates in this domain as in the Ridge regressions.

An important point, about the Ridge regression is that in all plots, the $\lambda = 0$ make the error to increase dramatically, compared with other combinations of λ values. We have seen this in Ridge, but for Lasso it is more enhanced. Once we check the lower MSE values in the regressions, they remains very near to 0.5 without significant changes.

4.4 SRTM

Figure 8 shows the results from 15 to 35 complexity degrees for the Train and the Test sets established from the SRTM data set. even though the trade-off seems to be always the same, it is slightly different.

this is just due to the plotting properties and the high error values. In fact, the trade-off is increasing with the increase in complexity degree.

For Ridge and Lasso regressions on the SRTM data set (see Figure 9), the results were using lower complexity degrees than the one from the OLS regression and lower range of λ penalty values due to computation time limits. $\lambda = 0$ values are noticed by the low performance error respect to the other values. Ridge regression gives a gradient-increase performance error towards moving to higher λ values. On the other hand, Lasso regression exhibits a strong influence of the λ penalty in the error performance, way more than the complexity degree.

5 Discussion

5.1 OLS

The two methods used (our algorithm and Scikit-Learn algorithm) seem to be somewhat different in their result behaviour. Figure 3 shows more enhanced peaks for the Test set in the Scikit-Learn, meaning a higher bias-variance trade-off. Our manual algorithm makes use of the *np.linalg.pinv* method to invert the matrices for the equation 2 instead of *np.linalg.inv*. If we use the latter, then the results are similar or worse to the ones from Scikit-Learn. The difference between these methods is that *np.linalg.pinv* allows us to avoid the problem of singular matrix by returning the normal inverse, if the matrix is singular, and a pseudo-inverse if the matrix is not. This is what is called a Moore-Penrose inversion (Strang, 2006). Also there are some rounding errors in floating arithmetic operations that might explain differences between the algorithms. This is very important to notice since our manual algorithm was used to test up to Ridge regression due to it's performance.

Due to the Moore-Penrose inversion, our manually-written algorithm allows to reduce the bias-variance trade-off faster than the Scikit-Learn in this exercise (see Figure 3) and reduce some of the peaks or outliers.

By doing it with the *np.linalg.inv* it is not only worse, but also would show the train set error over the test error in regressions for high complexity degrees.

The high peaks that are even more noticeable with the Scikit-Learn algorithm (right column of Figure 3) are because of pair and unpair polynomial degrees fit. The function seems to be pair since we have error peaks in mostly unpair numbers, and thus, it will explain the increase-decrease MSE values while scanning through unpair-pair polynomial degree sequences. But at the end we see in each plot a general expected behavior; the training performance curve flattens while the test performance diverges after a certain complexity value, which is the expected overfitting phenomenon. However, we can see that the overfitting occurs at different complexity degrees depending on the number of points.

To see the nice train and test error decrease and the overfitting, it is necessary to do a trade-off between the different noise added to the data. With a fixed noise value, we see that increase in the number of data points gives better results, since it means more available data to train the model and describe the problem better, also more data is available to validate the score of prediction.

For the Bootstrap resampling method, the difference between the two algorithms is the same. Scikit-Learn method performs the test set worse than with our algorithm, however, the train set performs worse with our algorithm, and for both train and test sets the performance is more stable.

An important thing to notice is that for the same amount of points, the algorithm still finds a suitable minimum error and optimal bias-variance trade-off at a complexity degree of 16 (Figure 3 with n=2500 (left panel), figure 4(a)). This seems to be a tendency that also bootstrap can notice. However, the spike at degree 9 and 4 for each subfigure at Figure 4 seems random. As mentioned earlier the good performance of pair number polynomial degrees on fitting our data and bad performance with the unpair, but in Bootstrap we have a random peak in a pair and an unpair degree for the test set. This can be due to the simple Bootstrap method. Resampling from the train set randomly might cause to select several data points more than once, for example. If we imagine a scenario when the Bootstrap select points better described by pair functions rather than unpair (or vise versa), and that those points repeats more than one in the new training set, then a situation like these peaks show, is very likely.

The K-Folding resampling method seems to to be more prone to bias-variance trade-off phenomenon. In Figure 5 we see that after K = 6 (with Folding equal to 6), the error of the test set decreases dramatically from 1500 to near 0, while the train set begins to increase the error but in a very good range of performance. This kind of behaviour is expected, since by increasing the K number, we are approximating to the resampling technique of Leave-One-Out cross-validation technique.

The Bootstrap and the K-Fold cross-validation techniques, due to resampling, is supposed to evaluate, how well the model could predict new data that it has not seen before. The Bootstrap method shows a more extreme situation due to the resampling technique, where information is repeated (sampling with replacement), and thus, train diversity is lost. K-Fold assures to take into account all the data points in a same level of importance. Also, by increasing the K number we assure more data for the train set, and thus, a better prediction of test data. For now, based on the Figure 5 the model with a good amount of data (better with all the points) seems it could be able to predict well new data points, that has never been seen before (outside the generated data set).

5.2 Ridge Regression

As seen in both Ridge and Lasso regressions that the penalty λ doesn't have as much impact on the model performance (both in test and train) as the complexity degrees. This is as expected, since the dataset is generated from a known function (Franke's function), and so it could be described by a polynomial equation. This can also bee seen in the first row of Figure 6 with $\lambda = 0$. Therefore, we expect that when using SRTM DEM (digital elevation model) as an input to the model the λ parameter will be even more important and have a significant impact on the MSE values for both train and test sets.

On the other hand, we can see the effect of penalty parameter, when we move from train and test datasets. Figure 6 shows the effect of λ penalty on the bias-variance trade-off. For a penalty of $\lambda = 0$ we see no major effects in regard to the complexity degree. However, for $\lambda > 0$ there is an expected effect of the penalty between the train and the test data sets. In principal, the penalty parameter can upgrade the model so that it works on new input data with a small sacrifice. The performance of the train set will be worse, but it will improve the performance for the test set. That is what we can see in Figure 6 (first row). There is a significant reduction of performance error for certain complexity degrees in the test set compared to the train set.

In the Bootstrap part (Figure 6, second row) we still see the complexity degree effects in a more marked way than the λ parameter, but in a different way. Now there is no gradational change of the error performance for a λ variation in a same polynomial degree as with the normal OLS (Figure 6, first row), but some random variability on the train set. Bootstrap methodology might explain this similarly to the Section 5.1. And similarly, the test set exhibits higher error performance than the train set. Notice how in $\lambda = 0$ has a peak in complexity degree 19 (Figure 6, second row, right panel). This is related to the phenomenon, explained in Section 5.1 and compared to the peaks seen in Figure 4 for the test set since when we have $\lambda = 0$, the Ridge regression is basically the same standard OLS regression.

For the K-Folding with K = 10 (Figure 6, third column) we see now a gradational MSE increase in the train set when we increase the λ value in regard to the complexity degree, but the test set gives strange results. We expected for $\lambda = 0$ and 15 complexity degree to have a similar error performance as in Figure 5 at 10. However, the same bias-variance trade-off when we compare the two heatmaps are the expected. In $\lambda = 0$ the train set erorr performance just flatterns even more while the test set began to rise after a complexity degree of 12 (Figure 6, third row).

K = 50 case is different. The train set has the same behavior, so it seems that the train performance still is not gonna change regarding the K-Fold number. This is expected since the idea of a Cross-Validation is to give an estimation of the performance with new unknown data, so it doesn't affect the training sets too much. But we can see that the error performance for the test set decreased by moving from K = 10 to K = 50, which makes sense because now there is more population for training the data. Still, for $\lambda = 0$ we see higher error (less than with K = 10), but now the error is more distributed in the complexity degrees.

5.3 Lasso Regression

Figure 7 shows the results for the Lasso regression, organized in the same way as the Ridge regression (Figure 6) and it shows similar tendencies, as the Ridge regression. Similar features as the error strides in the complexity degrees, indicating a low influence of the λ penalty compared to the complexity degree can be seen. Just as in Ridge, for the same reason. Bootstrap behaves in the same way, with the error strides but adding some random variations with λ, for the same reason explained in Ridge.

For K-Folding, the behavior is expected to be similar to the one we saw with Ridge, however here is not the case. The fact is that there is a similar behavior, but due to the high errors at $\lambda = 0$ it is not noticeable, and this peaks are consistent in all cross-validation attempts. This is mostly due to the Scikit-Learn algorithm of Lasso. The function warns the user to avoid using $\lambda = 0$ and just use the OLS regression since with this value the function will encounter some problems.

5.4 SRTM

Both Figures (8 and 9) or regressions shows a tendency to even get higher performance at more complexity degrees that the ones explored (greater than 35). Due to computational time limits, we were not able to reach even further. Figure 8 present a continuous shift between positive and negative values for the trade-off, which were similar to the part of lower complexity degrees (0 to 5) of our manual algorithm in Figure 3 (left column). This is another argument that makes us think that the optimal complexity degree is even higher than the values that we explored in Figure 8.

Ridge and Lasso were mostly used for studying the influence of the penalty parameter λ in the SRTM data set compared to the Franke Function, and some behaviors that were expected since the discussion with Franke Functions (Ridge and Lasso) are actually shown here. Lasso shows the high influence of the λ parameter on the performance error, which means that in this kind of data set, a penalty value applied to the coefficients is necessary to avoid overfitting. We predict based also on the OLS regression that if we would have explored higher polynomial degrees, we could have reached an area of optimal performances with a penalty value that would have range from 1 to 4.

6 Conclusions

In regard to the model performances with the Franke's Function, the best performances were achieved with the OLS regression from 10 to 15 complexity degree, however, based on our investigations, the optimal parameter values were given by the complexity degree of 12. Ridge and Lasso regressions give good dynamic solutions to overfitting problems by setting penalties in the the slopes (β_i). However since this was a known function, there is a high chance of a good polynomial fitting as we saw, so the penalty parameters would actually not affect as much as the complexity degree variations.

Regarding the OLS regression, other inversion methods could have been tested (such as *np.linalg.solve*) for invertible matrices and to test their performances. But this is just a point for playing with our own manually written algorithm. Also, it might have been better to explore the model performance with more K-Numbers (during the K-fold application). That is more computationally expensive, so might not be possible without more powerfull machines. Thus, for a better visualization of the result, a heatmap, which shows the interaction of K-number with polynomial degree would be better. From statistical point of view, a good experiment would have been to compare a normal *test-train-split* method K times with the K-Fold. The mentioned split method does it by selecting random points (non-repeated) for a defined ratio of test and train from the entire data set, and the idea is now to do this K times or in the same number as the Bootstrapping.

The dataset was not shuffled, and that could have affected in the cross-validation methods due to the resample techniques. This means that the cross-validation iterations were done using several parts of the function that might not represent well the general phenomenon but instead they represented another function that would fit better for that local problem. Maybe that was necessary and thus, the bootstrap would have performed better, and even the K-Fold.

With the SRTM elevation data the performance was not good, however, it is expected since fitting a polynomial function (which obeys patterns) into a real elevation is not so accurate. However, the pattern of the observed results makes sence: the qauality of the model increases with the model complexity. We believe, that the optimal performance might have been at higher complexity degrees, nevertheless, this was not reached because of computational time and efforts. We tried to run each regression in a parallel way through 15 hours, one in a local machine and another one in a server (with 32 CPUs and 256 GB of RAM), however, we didn't achieve any progress in neither of the 3 regression since the data was too large and the design matrix gets bigger proportionally to the complexity degree.

We tried to downsample (resample to the lower resolution image) the data, but the result information still was big and we were concerned that downsampling or downscaling it in such a way would not describe well a real variation of a terrain, but just a hill that would be very similar to some of the Franke Function parts with random noise. For future, we suggest to use more high resolution SRTM and then resampling it even more so this concern might not show up in the future.

Furthermore, from a geoscientific prospective, using the x and y coordinates values as predictors to estimate the elevation values does not really make sense in terms of physical processes, even though, the elevation example is just used as a 3-dimensional dataset. However, this might be a part of the explanation of a poor model performance and should not be disregarded.

7 References

References

Farr, T. G., Rosen, P. A., Caro, E., Crippen, R., Duren, R., Hensley, S., Kobrick, M., Paller, M., Rodriguez, E., Roth, L., et al. (2007). The shuttle radar topography mission. *Reviews of geophysics*, 45(2).

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.

Strang, G. (2006). Linear algebra and its applications.

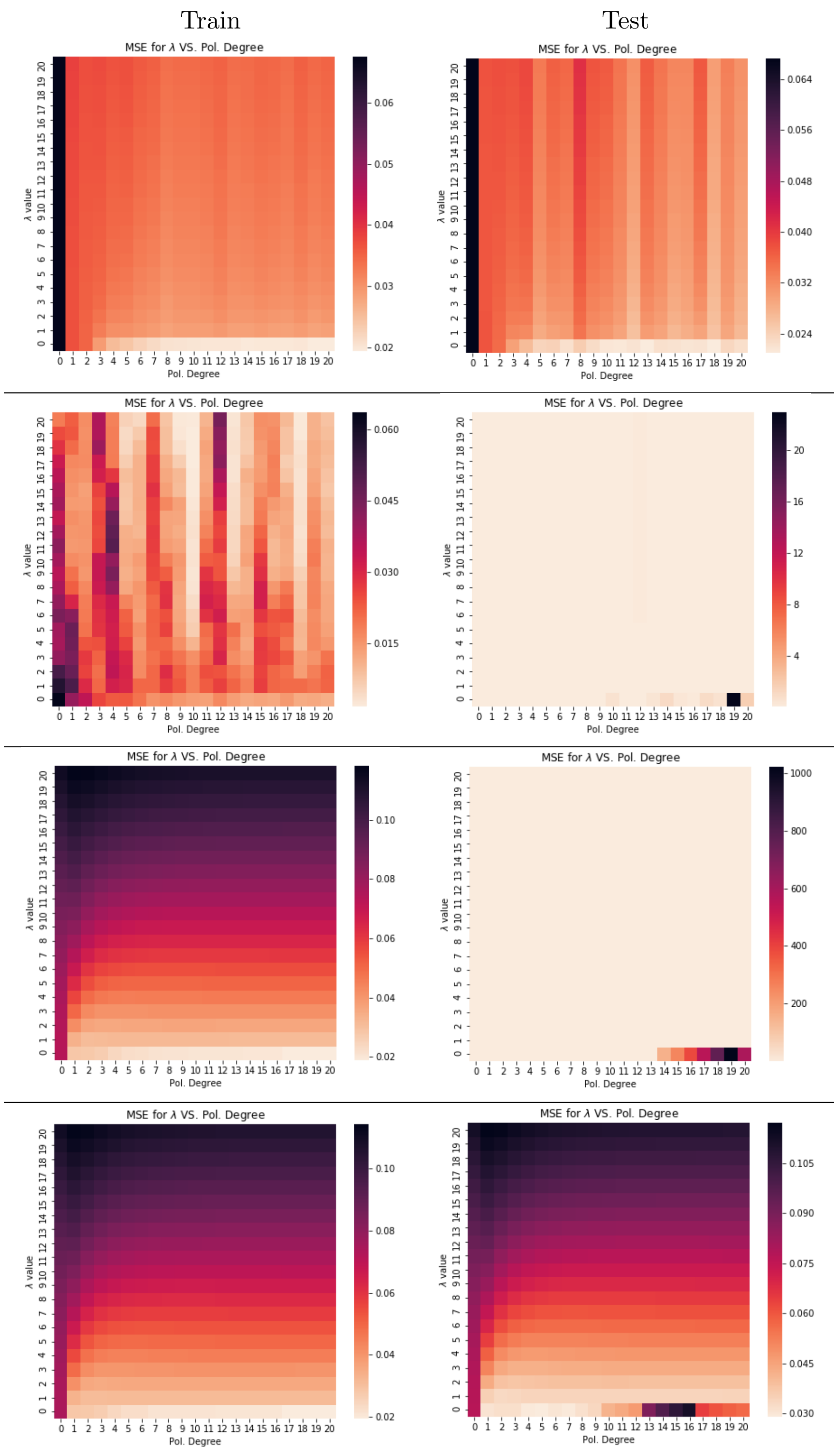


Figure 6: Ridge regression performance (MSE) for Train and Test sets varying complexity degree and lambda parameter, both up to 20. First to Fourth lines are; simple Ridge, Ridge bootstrapped by 20, Ridge with K-Folding of K = 10 and K = 50, respectively. Number of points = 2500.

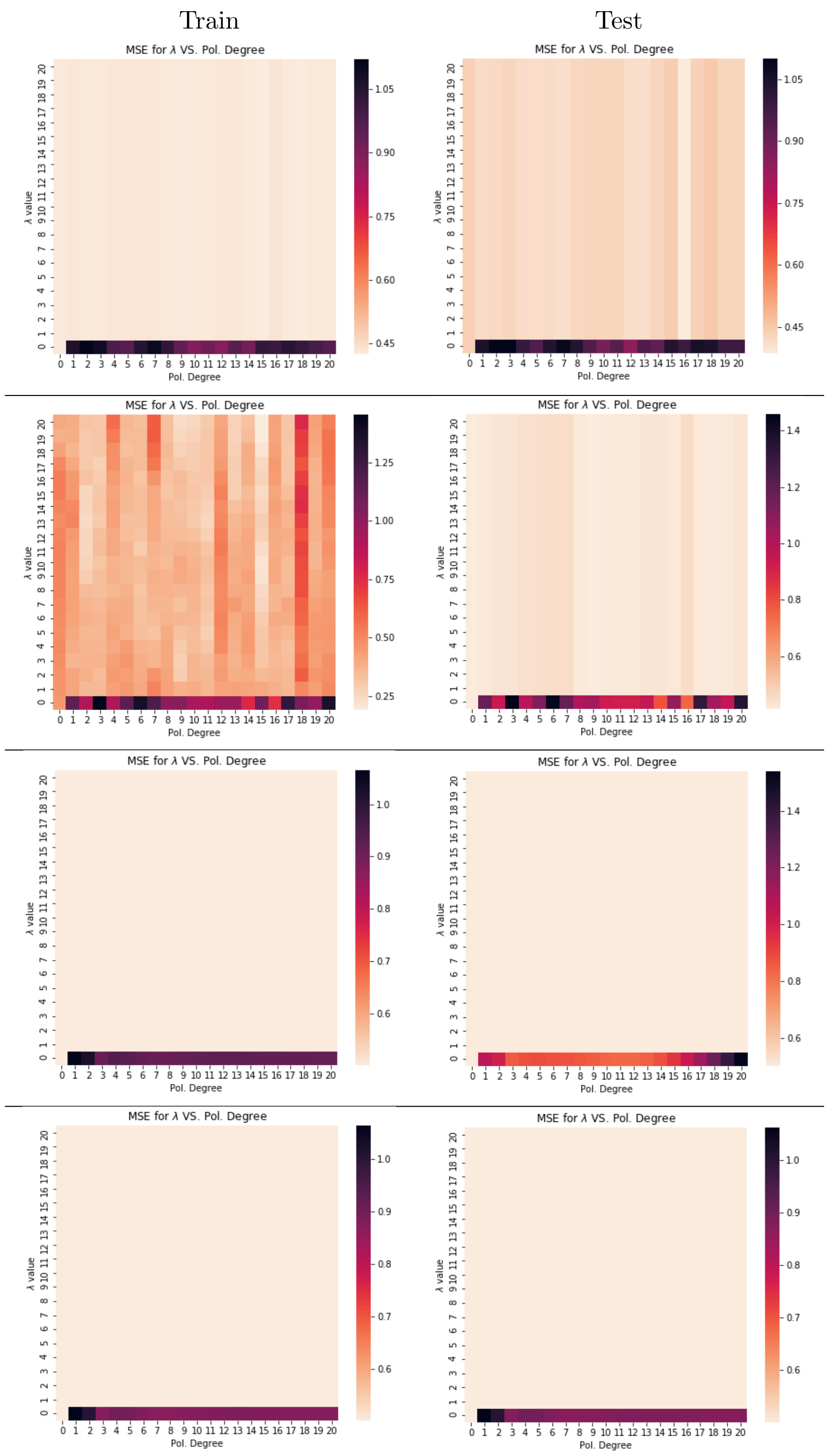


Figure 7: Lasso regression performance (MSE) for Train (left panel) and Test (right panel) sets varying complexity degree and lambda parameter, both up to 20. First to Fourth lines are: simple Ridge, Ridge bootstrapped by 20, Ridge with K-Folding of K = 10 and K = 50, respectively. Number of points = 2500.

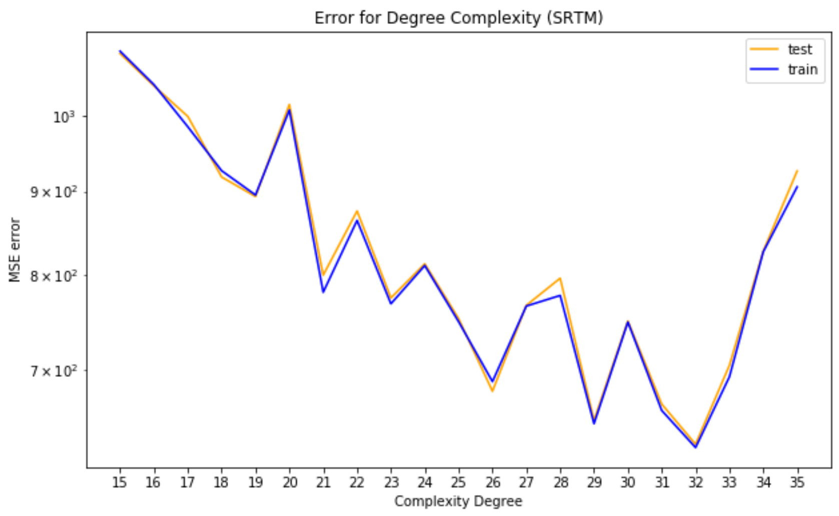


Figure 8: OLS regression for the SRTM data, from 15 to 35 polynomial degrees of complexity. Blue and orange lines curves are the Train and the Test model performance, respectfully

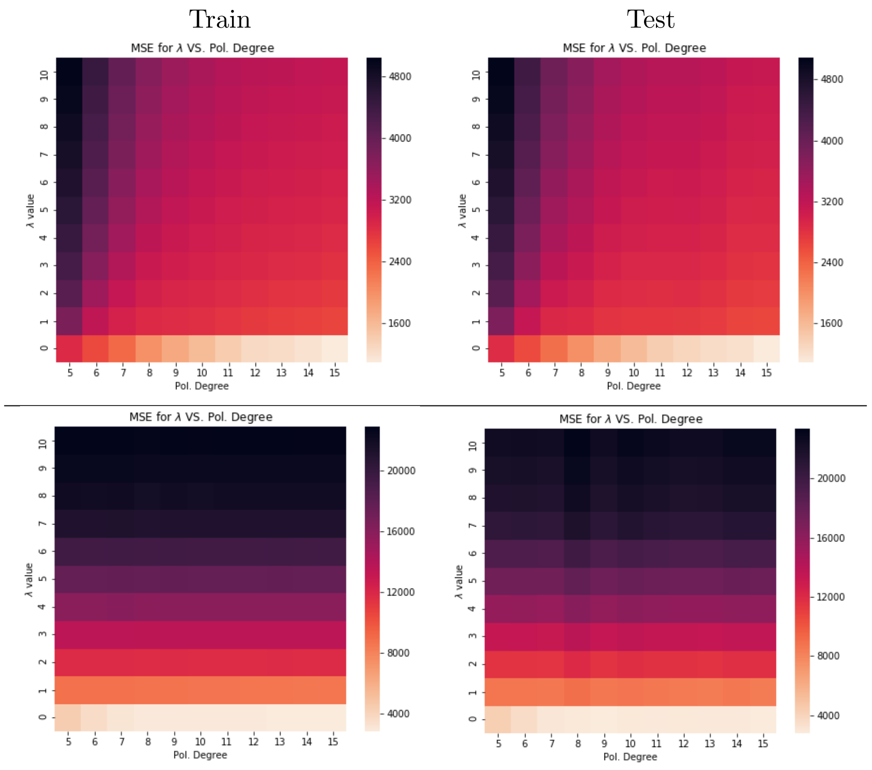


Figure 9: Ridge (first row) and Lasso (second row) for Train and Test sets of the SRTM data. λ penalty values ranges from 0 to 10 and complexity degree form 5 to 15.