FYS-STK3155 – Applied data analysis and machine learning

Autumn 2018

Project 1

Vincent David Friedrich &

Frederik Stihler

04.10.2018

Deadline: 08.10.2018

Prof. Morten Hjorth-Jensen

Abstract

The topics of this project are regression analysis and resampling techniques. Three different regression methods have been tested. The methods used to fit polynomials to a test function were Ordinary Least Squares (OLS) Regression, Ridge and Lasso Regression. In the process of training the models and tuning the parameters, the k-fold cross-validation was used as a resampling technique. It was found that the OLS provided the best approximation of the test function and no overfitting was detected. The Ridge Regression provided also an accurate model, following a low variance and higher bias approach. The Lasso regression was difficult to implement and resulted in convergence problems when using the integrated functions of *scikit-learn*. In the end, it offered a model with the highest prediction error on test data after cross-validation.

Thereafter, real terrain data was retrieved and the three methods were applied to the new setting. This time again, OLS and Ridge delivered the best results for parameterizing the terrain data. However, the size and complexity of the input dataset prevented the rather simple models from precisely capturing the data. As a consequence, higher degree polynomials up to 9^{th} order were used to increase accuracy. Nevertheless, it was found that the optimal degree to minimize the prediction error must be even beyond that.

During both experiments, a balance between bias and variance of the models was sought. In all cases, no overfitting occurred, leading to the fact that the advantages of the Ridge and Lasso method could not fully unfold, as they prevent models from growing to complex and keep the variance low.

1 Contents

2 Introduction

The main topics of this project are regression analysis and resampling methods. Regression analysis is a form of modelling technique, often used to find a predictive model, for describing the relationship of independent input variables, the predictors, and a target variable, depending on the predictors. As these kinds of models are used in practice for forecasting, regression analysis is an important tool for statistically analyzing and modelling data. Consequently, there exists a wide range of applications in the real world. For instance, it is used to predict risks and support decisions in a business environment. It follows that the study of different regression methods and the understanding of their suitability for different applications is a relevant research topic, in order to obtain realistic predictions and optimizing forecasting.

The three regression methods studied in more detail in this report are the Ordinary Least Squares (OLS) method, the Ridge regression and the Lasso regression. A major objective of this project is to implement and test these methods, as well as to compare them, highlighting advantages and drawbacks. In addition to that, the implementation of the regression methods is followed by resampling techniques, resulting in a model selection procedure. The models are tested against each other by fitting polynomials to the two-dimensional Franke's function (Franke, 1979) and comparing the mean squared error (MSE) and R^2-score calculated with the true data and the predictions. The assessment of the models is done with a k-fold cross-validation. Finally, real digital terrain data of a section of the surface of the earth is attempted to be reproduced with the selected models.

The codes in form of Jupyter notebooks, as well as other supporting material can be found in the github repository at:

https://github.com/Fredesti/Proj1_ML_FS_VF

3 Methodology

For the implementation of the project the programming language Python 3.0 was used. The relevant python packages for analysis, computations and visualization of the data were *numpy, scikit-learn, scipy* and *matplotlib.*

Regarding the information and theoretical background on regression analysis and resampling methods, most of the techniques applied are based on the lecture notes of the course FYS-STK3155 ("Applied data analysis and machine learning") of autumn semester 2018 at the department of physics at the university of Oslo in Norway, as well as on the book "The Elements of Statistical Learning" by Hastie et al. (2009) referenced in the appropriate sections of this report. The regression methods used in this project were OLS, Ridge and Lasso. The least squares approach assumes a model that is linear in the parameters. From a set of data, called the training data, the parameters β are derived. The estimation of parameters is based on minimizing a chosen cost function. In the case of OLS, β should minimize the following residual sum of squares (Hastie, Tibshirani, & Friedman, 2009, p. 44) :

$$\begin{aligned} \mathrm{RSS}(\beta) &= \sum_{i=1}^{N}(y_i - f(x_i))^2 \\ &= \sum_{i=1}^{N}\Big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\Big)^2 \end{aligned}$$

Here (x_i , y_i) for i = 1,…,N represents the training data and f the linear model. Further details about the theory behind the OLS are covered along the way in the result analysis of this method, when testing it on a test function.

The Ridge Regression uses a different cost function. It is a shrinkage method, preventing the model from growing to complex. According to Hastie et al. (2009, p. 63), "the ridge coefficients minimize a penalized residual sum of squares" :

$$\hat{\beta}^{\mathrm{ridge}} = \underset{\beta}{\mathrm{argmin}}\left\{\sum_{i=1}^{N}\big(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\big)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\right\}$$

In the Ridge Regression, λ is a non-negative parameter, controlling the complexity of the model by imposing a penalty on the size of the coefficients. Again, more theory is covered in the section for Ridge Regression when testing it on the test function "Franke's function".

The Lasso method is similar to the Ridge Regression. The problem for finding the coefficients in this method is as follows (Hastie, Tibshirani, & Friedman, 2009, p. 68) :

$$\hat{\beta}^{\text{lasso}} = \underset{\beta}{\operatorname{argmin}} \left\{ \frac{1}{2} \sum_{i=1}^{N} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

As this is not easy to solve, the computation was performed with the help of the *scikit-learn* python package, which uses a coordinated descent algorithm. Coordinated descent successively minimizes a function along the coordinate axes. As for the other methods, the remaining theory that has not been covered in the methodology section can be found in the literature of Hastie et. al and in the belonging report sections. A more extensive comparison between the methods is drawn in the critical evaluation at the end of chapter 4 in section 4.1.4.

After implementing each of the regression models, a k-fold cross validation was performed as part of resampling. The purpose of these techniques is to tune important parameters and to validate the models by using randomly created subsamples. In the k-fold cross-validation, the shuffled original data is split into equally sized partitions. Each partition is one time used as test data, while the remaining k-1 partitions build the training set (Hastie, Tibshirani, & Friedman, 2009, p. 242). Thus, there will be k repetitions. Evaluating performance measures for each run and averaging the results gives an impression how appropriate a generalization of the model to new data would be.

Furthermore, the project also involved the usage of real digital terrain data, which was retrieved from the database of the U.S. Geological Survey (USGS) and the Earth Resources Observation and Science Center (EROS). The USGS is a science agency within the department of the interior of the United States of America, focusing on natural sciences and providing access to databases concerning earth and biological topics, such as energy, climate and geology. The terrain data introduced in this project was obtained via the EarthExplorer-tool on the USGS website. The area chosen to be analyzed was the greater area around Munich, Germany. The data format was SRTM, meaning that the data has been collected by NASA's Shuttle Radar Topography Mission in 2000 (Ramirez, 2018; Mehta, Wang, Day, & Richardson, 2018).

4 Implementation and Results

4.1 Testing the regression models with artificially generated data

In the following section the testing of the regression methods OLS, Ridge and Lasso is presented. For this purpose, it was attempted to fit a polynomial to the so-called Franke's function (Franke, 1979), which is a two-dimensional function that has been widely used when testing fitting algorithms. A visualization of the Franke's function, as a 3D-plot can be seen in Figure 1. The surface shows a minor dip and two Gaussian peaks with different maxima.

The input data for this function was generated artificially by drawing samples from a uniform distribution between 0 and 1 via the *numpy random uniform* generator. The sample size chosen was one thousand. Thereafter, a stochastic noise according to the normal distribution was added. The size of the noise was later varied, to study the sensibility of the produced results. The code for the data creation is presented below.

```
# use seed=0 to reproduce results
np.random.seed(0)

# creating the data
x = np.random.uniform(0,1,1000)
y = np.random.uniform(0,1,1000)
noise=0.1*np.random.randn(1000)
z = FrankeFunction(x, y) + noise
```

4.1.1 Ordinary Least Squares on the Franke function

First, the Ordinary Least Squares regression analysis was performed using polynomials in x and y, from second to fifth order. For this implementation, some facilitating functions were created, for instance a function *designmatrix()* that creates the corresponding designmatrix for a specified degree and a given dataset with two features. Another function that was defined calculates the regression coefficients. For this calculation, the matrix inversion method was applied. The code for this function is shown below. A singular value decomposition was not utilized. The MSE and R^2-score were also computed manually without using the integrated functions of *scikit-learn*.

```
# function that finds the coefficients of the OLS regression with matrix inversion

def betareg(x,y,z,degree):
    #input1 : input data vector 1
    #input2 : input data vector 2
    #input3 : target data vector
    #input4 : degree of polynomial for fit
    #output : coefficients for polynomial fit

    X = designmatrix(x,y,degree)          #designmatrix function is definded before

    # Calculation of beta
    XtX = np.transpose(X).dot(X)
    XtXinvers = np.linalg.inv(XtX)
    betareg = (XtXinvers.dot(np.transpose(X))).dot(z)

    return betareg
```

Afterwards, a short comparison of how the polynomial fits of different degrees performed was made. Note that for this analysis, no resampling techniques have been applied and that the scores are obtained only by comparing the true target values with the values that have been predicted with the coefficients, gained through the fitting to all available target values (sample size = 1000). The results can be found in Table 1.

Tab. 1: Comparison of MSE and R^2-score of predictions with fitted polynomials of different degrees

	Degree of Polynomial			
	2	3	4	5
MSE	0.01615	0.00734	0.00390	0.00191
R2-score	0.80779	0.91262	0.95365	0.97728

As expected, the MSE is decreasing, while the R^2-score is increasing when raising the degree. Due to the fact that the model is tested on its own training data in this case, a higher degree can only result in a better fit and more precise predictions. Hence, the mean squared difference between the predicted values and the true data gets smaller. Furthermore, a greater proportion of the variance in the target values can be predicted by the inputs, represented in a better R^2-score. However, when splitting the data into training and test data, there exists a possibility of overfitting, meaning that a higher degree polynomial could perform poorly on “new” data (test data).

This can happen when a model is too well trained on a specific dataset and fails to accurately capture the overall trend, leading to worse predictions than a lower degree polynomial would produce. This was not the case in this project, as comparing the degrees with the k-fold cross-validation technique showed the same outcome as in Table 1. This can be controlled in Table 3. To further highlight this important issue, the following visualization in Figure 1 should help understand the topic.

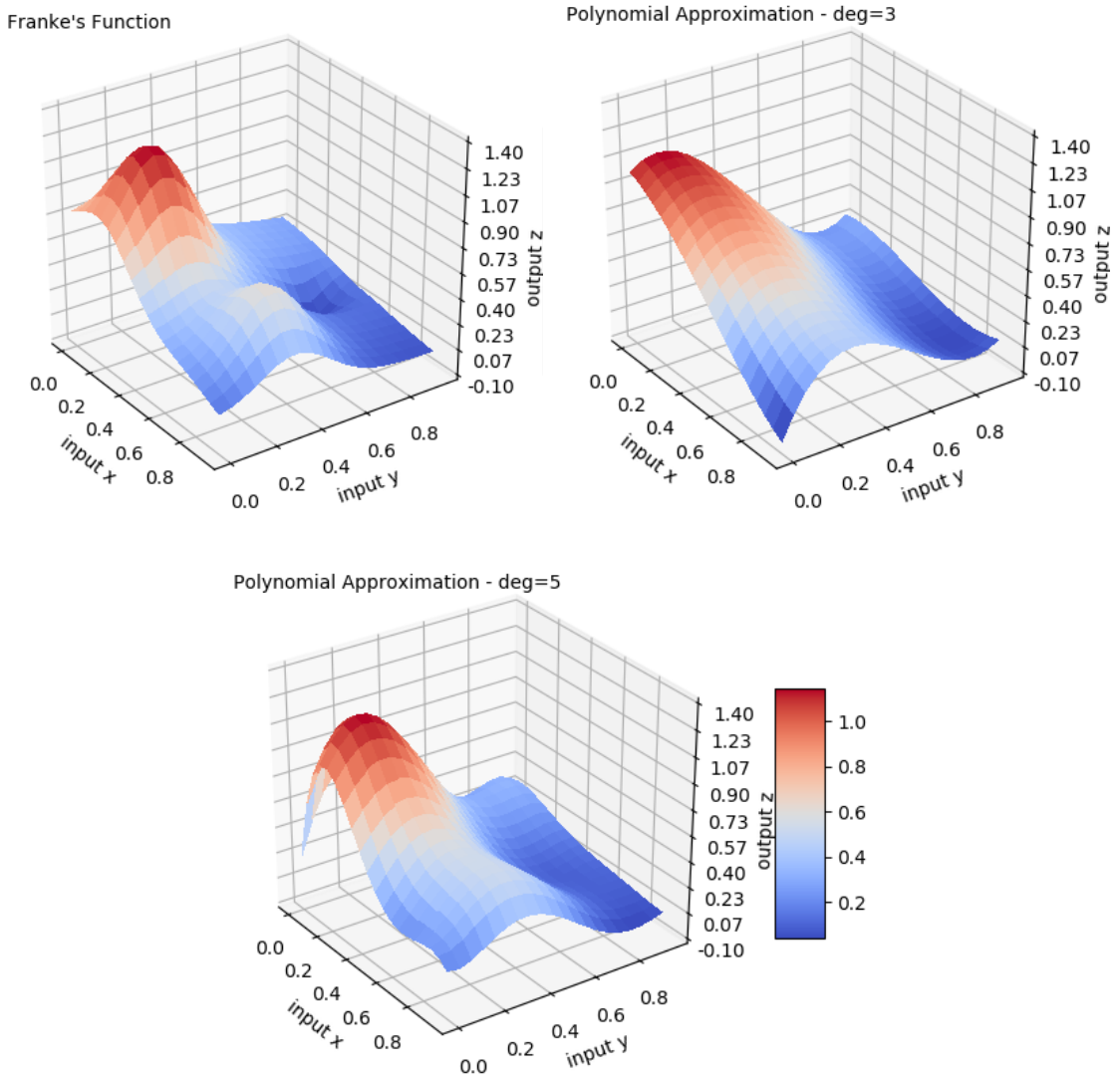

Fig. 1: 3D plot of the test function “Franke’s function” and a degree 3 and 5 polynomial fit with OLS

The lower degree polynomial fit, in this case degree 3, appears as a flatter surface, not capturing the peaks and smaller dips of Franke’s function very well. In contrast, the degree 5 approximation suggests a better reproduction of the test function without overfitting.

Variance

The variance of the regression coefficients $\hat{\beta}$ was calculated according to Hastie et al. (2009, p. 47) by applying the formula

$$\mathrm{Var}(\hat{\beta}) = (X^T X)^{-1}\sigma^2 .$$

The diagonal elements of this matrix represent the variance of the corresponding regression parameters in $\hat{\beta}$. For the variance σ^2, an unbiased estimate was used (Hastie, Tibshirani, & Friedman, 2009, p. 47). From these calculations, a 95% confidence interval was derived for the regression coefficients of a degree 5 polynomial fit. The values can be studied in Table 2. The 95% confidence interval implies that when collecting new data, assumed to be generated by the same distribution as before, the regression coefficient would end up being within the intervals in 19 out of 20 experiments.

Tab. 2: Variance and 95% confidence intervals for regression parameters beta (degree 5)

coef.number	variance	lower	beta	upper
1.	0.00076	0.18294	0.236985	0.29103
2.	0.088022	8.70296	9.284464	9.865968
3.	0.078723	4.229886	4.779816	5.329745
4.	1.857498	-42.230391	-39.559106	-36.887821
5.	1.218169	-22.146884	-19.983618	-17.820353
6.	1.65111	-13.488421	-10.969909	-8.451396
7.	8.823969	51.994813	57.817026	63.639238
8.	5.32782	48.589553	53.113639	57.637724
9.	4.929662	23.33942	27.691177	32.042934
10.	8.086609	-11.023763	-5.450118	0.123527
11.	9.339019	-38.100963	-32.11124	-26.121518
12.	5.91767	-64.867345	-60.099398	-55.331451
13.	5.015377	-19.247469	-14.858042	-10.468614
14.	5.222959	-38.03594	-33.556597	-29.077253
15.	8.74209	21.662934	27.458071	33.253208
16.	1.417136	2.075605	4.408858	6.742111
17.	1.092785	18.819025	20.867937	22.91685
18.	0.995	11.086678	13.041772	14.996866
19.	0.947959	-5.196427	-3.288108	-1.379789
20.	0.977021	15.210645	17.147995	19.085345
21.	1.334872	-18.131597	-15.867078	-13.602559

The number of the coefficient can be found in the first column and the value of the parameter itself is written below beta. In the second column, the variance for the coefficient is shown. In the column labelled “lower”, the lower boundary of the confidence interval is displayed, while the upper boundary is in the “upper” column.

A close look at the coefficient reveals that there are some large values, for example coefficients 7 and 12 with around 58 and -60 respectively, although the true target values z and their prediction are not larger than 2. Furthermore, the variance of the coefficients 7 to

15 show a much higher variance in comparison to the rest. That means that in this case, a repetition of the experiment is more likely to produce significantly different parameters. Sometimes it can be useful to introduce a penalty on the size of the coefficients, in order to reduce variance and prevent the model from growing to complex and from overfitting. This topic is covered in the following sections about Ridge and Lasso regression. Another observation about the confidence intervals is the fact that there is no sign change in any of them. This is a desirable outcome, as the parameters are likely to have the same sign when repeating the experiment with new data. Consequently, there is no reversion of trends for the relation between inputs and target values in the model.

k-fold cross-validation

Next, a resampling of the data was performed by splitting the dataset into training and test data. For the purpose of implementing the k-fold cross validation algorithm, k partitions of the shuffled dataset were created. The code for this shuffle and split is presented in appendix 1.

For this cross-validation k=10 was chosen, following the recommendation of Hastie et al. (2009, p. 242) to find a compromise between bias and variance of the model. For every of the 10 randomly created partitions, a polynomial of degree 2 up to 5 was fitted to the training data represented by the other 9 parts. Afterwards, it was tested on the remaining test data and the MSE and R^2-score was calculated. The mean over all 10 MSE calculations and R^2-scores can be found in Table 3.

Tab. 3: Comparison of performance indicators of predictions to test data after 10-fold cross-validation

cross-validation and bias-variance decomposition	Degree of Polynomial			
	2	3	4	5
Prediction Error (MSE)	0.01963250	0.00756602	0.00373579	0.00186906
Bias ^2	0.01961638	0.00755610	0.00372871	0.00186459
Variance	1.6120e-05	9.92422e-06	7.07778e-06	4.4681e-06
R2-score	0.80319	0.91991	0.95915	0.97582

By implementing the cross-validation, the degree of the polynomial fit can be tuned. Similar to Table 1, in Table 3 it can be observed that the degree 5 polynomial fit performs the best, also when applying 10-fold cross-validation. This means that no overfitting is occurring and the model that is fitted with a degree 5 polynomial also is the best when using test data. The bias-variance decomposition, calculated as in Hastie et. al (2009, p.223), reveals that the greatest proportion of the MSE is made up of the squared bias. The bias is the deviation between the true mean and the mean of the model. As the variance is still very low, it means that the chosen final model of degree 5 is not too complex for the fitting of the test function.

4.1.2 Ridge Regression on the Franke function

Following the OLS, a Ridge Regression analysis was performed on the same data set. A main objective of the Ridge Regression according to Hastie et al. (2009, p. 61) is to "shrin[k] the regression coefficients by imposing a penalty on their size". Doing this can prevent the model from overfitting, as discussed later. The Ridge regression was performed with centering the data before executing the fit and without the utilization of an SVD. Using the formulas provided by Murphy (2012, S. 226) the coefficients for the Ridge Regression were calculated as:

$$\hat{\beta}^{Ridge} = (\lambda \mathrm{I} + X^T X)^{-1} X^T z.$$

Here X is the designmatrix, z are the target values and lamda is the parameter controlling the shrinkage of the regression coefficients. A larger lamda results in a stronger shrinkage. The code can be found here:

```
#function that returns ridge regression coefficients for centered data

def betaridge(x,y,z,degree,lamda):
    #input1 : indput data vector 1
    #input2 : indput data vector 2
    #input3 : target data vector
    #input4 : degree of polynomial for fit
    #input5 : chosen penalty
    #output : coefficients for polynomial fit of ridge regression

    #function for designmatrix is definded before (without the first column
    consisting of ones)
    X = designmatrix(x,y,degree)
    XtX = np.transpose(X).dot(X)
    ID = np.identity(XtX.shape[0])

    # Calculation of beta

    # intercept b[0] was calculated before as mean of target data
    betaridge0 = np.empty(1)
    betaridge0[0] = intercept
    betaridge2 = (np.linalg.inv(XtX+lamda*ID).dot(np.transpose(X))).dot(z)
    betaridge=np.concatenate((betaridge0,betaridge2),axis=0)

    return betaridge
```

After setting up the Ridge Regression, the tuning of the parameter lamda was done with a 10-fold cross-validation. The MSE and the R^2-score of the 10 partitions have been averaged and can be seen in Figure 2 for different values of lamda.

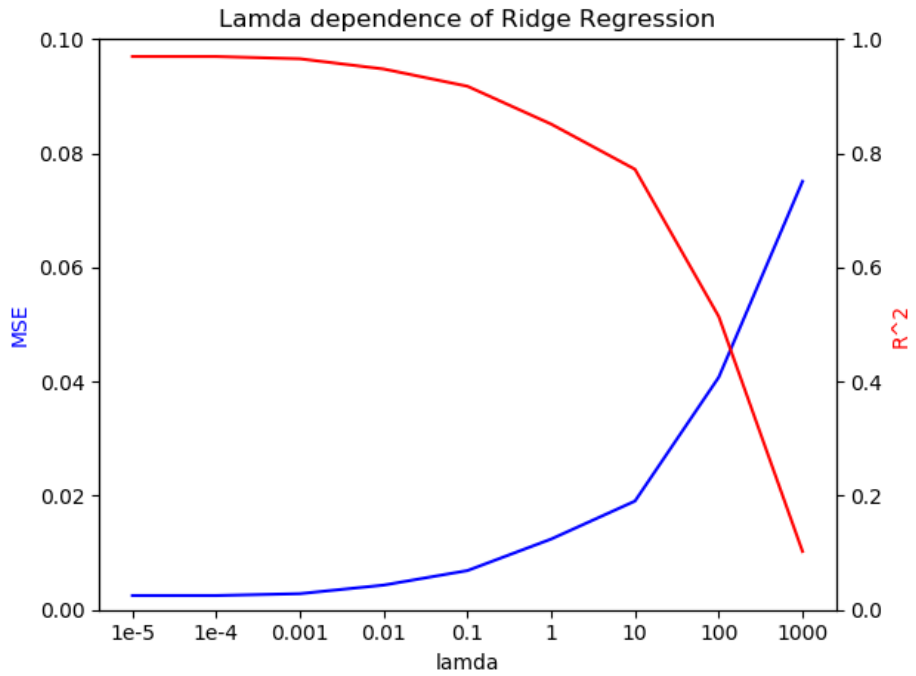

Fig. 2: Comparison of average MSE and R^2-score for different lamdas after 10-fold cross-validation

The graphs show that for increasing lamda, the R^2-score decreases, while the MSE increases. This means that the model gets less accurate in predicting the test data.

An aspect that can also be derived from Figure 2 is that the MSE / R^2-score only start to increase / decline rapidly when setting lamda larger than 0.001. Consequently, lamda was chosen to be 0.001 for the final model, to get the lowest possible variance in the parameters beta, without sacrificing much of the predicting accuracy. An even smaller lamda would impose an even lower penalty on the coefficient size, leading to a more complex model and consequently a higher variance in the parameters. Compared to the OLS, where the variance in some parameters can be up to 9, the variance of each parameter for a degree 5 polynomial fit with the Ridge Regression and lamda = 0.001 did not exceed 1.2 (see https://github.com/Fredesti/Proj1_ML_FS_VF for exact results).

4.1.3 Lasso Regression on the Franke function

Finally, the third regression method investigated was the Lasso method. Lasso stands for "least absolute shrinkage and selection operator" (Tibshirani, 1996, p. 1). The method often produces coefficients that are exactly zero, which makes the models less complex and easier to interpret. It also prevents overfitting. The implementation of the Lasso method was fully executed with the functionalities of *scikit-learn*. As the integrated Lasso Regression in *scikit-learn* uses a coordinated descent algorithm to fit the model, some convergence problems occurred for too small penalization parameters alpha. On the contrary, too large values for alpha was shrinking almost all the coefficients to zero. The problem was addressed by increasing the maximum number of iterations that the fitting algorithm performs. However, for small penalty parameters, the function still caused precision problems. As a consequence, the tuning of the parameter alpha was done by using the build-in LassoCV function of *scikit-learn,* which is returning the recommended amount of penalization selected by cross-validation. The code for the Lasso fitting can be found in appendix 2.

The Lasso model fitted by *scikit-learn* resulted in a penalization parameter alpha = 6.195e-05. The coefficients of the Lasso Regression can be seen in Figure 3. A plot of the Lasso approximation of Franke's function is also included in this figure.

```
Lasso coefficients
[[ 1.          1.003891]
 [ 2.          0.      ]
 [ 3.          0.672433]
 [ 4.         -3.764696]
 [ 5.          1.200551]
 [ 6.         -3.517855]
 [ 7.          3.773347]
 [ 8.          0.637246]
 [ 9.         -0.      ]
 [10.          0.      ]
 [11.          0.      ]
 [12.          0.      ]
 [13.         -0.      ]
 [14.         -0.6438  ]
 [15.          2.204063]
 [16.         -1.044952]
 [17.         -0.301484]
 [18.         -0.      ]
 [19.         -0.      ]
 [20.         -0.14924 ]
 [21.         -0.      ]]
```

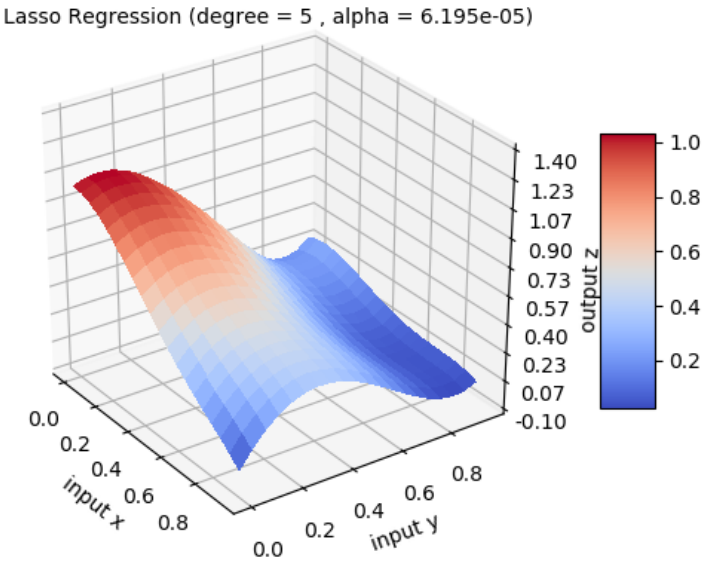

Fig. 3: Lasso Regression coefficients (left) and plot of the approximation of Franke's function with Lasso (right)

One can see that 9 out of the 20 coefficients (21 with intercept) were completely shrunken to zero. Consequently, the model is less complex than the degree 5 fit with OLS and rather resembles a degree 3 polynomial fit (see Fig.1 and Fig.3), as it looks more flattened.

4.1.4 Critical evaluation and comparison of the regression methods

After implementing the three regression methods OLS, Ridge and Lasso and performing a resampling and cross-validation, the final models were determined. As an indicator for how well the models can predict new data, the average MSE and R^2-score after a 10-fold cross-validation were computed and are presented in Table 4.

Tab. 4: Comparison of average MSE and R^2 -score for different regression methods after 10-fold cross-validation

10-fold CV	Regression Method		
	OLS	Ridge	Lasso
MSE	0.00201	0.00282	0.00621
R2-score	0.97582	0.96578	0.91082
		λ=0.001	α=6.2e-05

By these indicators, the OLS method performed the best with the lowest MSE and highest R^2-score. This result contradicts with the expectation that the OLS will perform worse under certain conditions on test data in comparison to the Ridge and Lasso method. It was expected that the OLS method starts to overfit the data and consequently fails to accurately predict the target values when using new data. The coefficients of the OLS are not restricted and can have a large variance compared to for instance the Ridge Regression coefficients. A reason for this could be that only polynomials up to degree 5 have been fitted. Further increasing the degree could have led to passing by a minimum in the prediction error for test data and thus resulting in a growing error. The Ridge Regression and Lasso Regression that were shrinking the coefficients and therefore reducing the models complexity are performing (with regard to MSE and R^2-score) in the range of a degree 4 or degree 3 OLS fit respectively. However, selecting a smaller lamda for the Ridge Regression could improve the MSE and R^2-score at the cost of a higher variance in the parameters. The Ridge regression then performs at a comparably good level as the OLS. For example choosing lamda = 1e-4 leads to an MSE of 0.00249 and an R^2-score of 0.96990.

Another reason for the best performance of OLS could have been the choice of noise in the data creation. For most of the experiments, the noise was chosen to be 0.01*N(0;1), hence $N(0;0.01^2)$ distributed. While the datasets created with this level of perturbation are not too noisy, the Ordinary Least Square Regression can deliver high quality predictions. When increasing the level of noise to a factor of 0.1 of the standard normal distribution, the performance of Ridge, Lasso and OLS move closer together, although still the OLS was found to be the best, before Ridge and Lasso. It was expected that for noisier data, the less complex and “flatter” Ridge and Lasso Regression are more suitable, as they do not react sensitively on outliers.

One of the differences between the Ridge and the Lasso method is that Lasso actually shrinks coefficient to zero, meaning it shrinks and selects some of the parameters, while Ridge is only shrinking the coefficients without setting them to zero concretely. Furthermore, the Ridge Regression is easier to implement and it was observed that it takes considerably less time to fit a model with Ridge, while the Lasso method, due to the many iterations required for convergence of the algorithm, was very slow.

4.2 Parametrization of digital terrain data with regression methods

As described in the methodology section, the regression methods have been tested on real digital terrain data from the USGS. Munich has been selected as the geographical area to be analyzed. The target values consequently are the altitude measured in meters. Officially, Munich's altitude is around 520m. The objective of this experiment was to parameterize this area and attempt to reproduce it with the above evaluated regression methods. As the initial data set was too large to lead to effective outcomes and resulted in computational problems, it was decided to just look at a smaller patch of the area. A surface plot of the data can be seen in Figure 4. The surface shows many peaks and dips, leading to the assumption that a higher degree polynomial would be required to obtain low prediction errors.

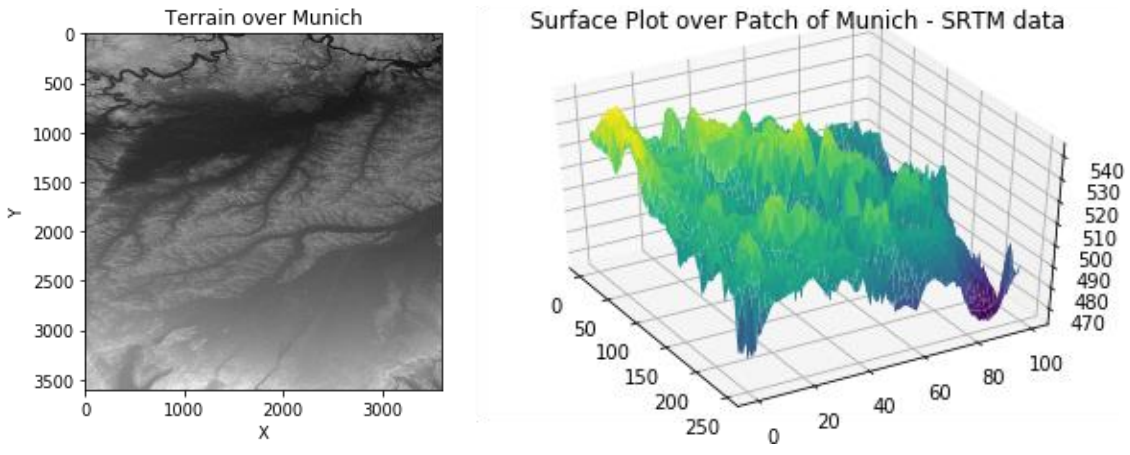

Fig. 4: SRTM data of area around Munich – picture (left) and surface plot of data (right)

4.2.1 OLS, Ridge and Lasso regression with resampling on terrain data

Simply applying the OLS method in the same way as with the artificial data gave an MSE of 65.73 and an R^2-score of only 0.629. This level of accuracy was not satisfactory and the functions have been adapted to fit a degree 7 and degree 9 polynomial, to study the fitting quality with a more complex model. After performing a 10 fold cross-validation, the average MSE and R^2-score of the partitions were calculated. The result was as expected, as the 7^{th} order polynomial fit improved the accuracy compared to degree 5. The 9^{th} order polynomial fit delivered the best scores. This lead to the conclusion that the degree 9 approximation suits this type of complex data the best. The scores of the final model fitted to all the available data can be seen in Table 5.

Tab. 5: Comparison of average MSE and R^2-score for different degrees of OLS on full data

	Degree of Polynomial			
	4	5	7	9
MSE	73.793	65.732	58.649	53.224
R2-score	0.583	0.629	0.668	0.699

Subsequently, the approximations for degree 5 and 9 have been plotted and can be seen in Figure 5. Compared to the real data, it is obvious that the plots are much smoother and flatter. However, there is also a significant difference between the 5^{th} order and 9^{th} order plot. The higher degree fit reproduces more of the bulges and resembles the bumpiness of the real digital terrain data in a more detailed way.

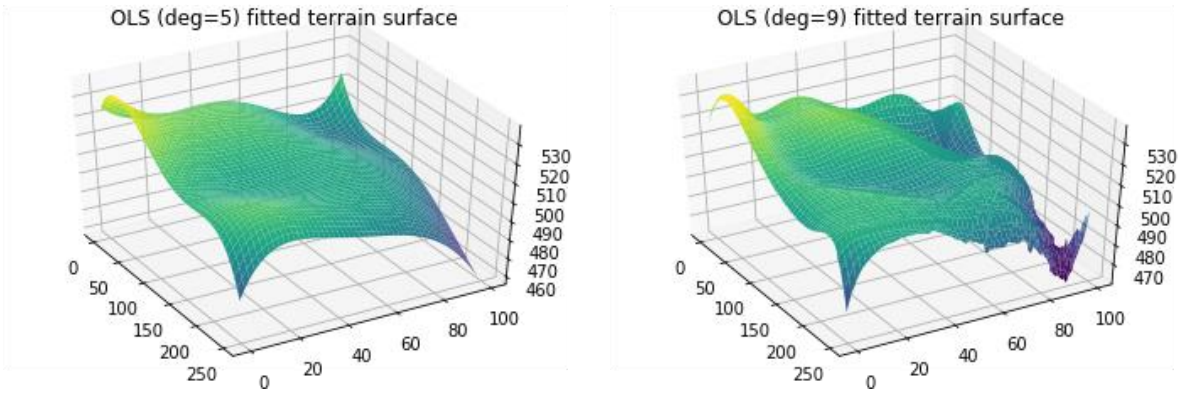

Fig. 5: Fitted terrain surface plots for degree 5 OLS (left) and degree 9 OLS (right)

Due to runtime problems, a polynomial fit of degree 9 was the maximum degree that was tested. It is expected that the minimum prediction error on test data when further increasing the degree of the polynomial to be fitted is reached even beyond degree 9.

The coefficients found with minimizing the penalized residual sum of squares for the Ridge Regression were extremely low, independent of the choice of lamda, except for the intercept. Yet, the method performed equally good as the degree 5 approximation with OLS (see Tab.6, Fig.5 and Fig.6). When running the Ridge Regression analysis, the variance of the parameters turned out to be very low, nearly equaling zero for many of the coefficients. The exact values can be studied in the codes at the provided github repository. An alternation of lamda had basically no effect on this matter. Even when choosing lamda very small and therefore not strongly penalizing the size of the coefficients, for instance lamda = 1e-7, the variance remained little. Thus, the tuning of lamda with cross-validation was not relevant. The performance measures of the final model of Ridge with lamda = 0.001 can be seen in Table 6.

Tab. 6: Comparison of MSE and R^2-score for final models with OLS, Ridge and Lasso

terrain data	Regression Method		
	OLS	Ridge	Lasso
MSE	65.732	65.799	93.150
R2-score	0.629	0.628	0.474

The value for the Lasso method was achieved with a complexity parameter alpha selected by *scikit-learn*. However, the application of the Lasso method also caused sum unforeseen outcomes. As in the previous experiment, the penalization parameter alpha was first chosen by *scikit learn.* As this lead to a value of alpha > 1 billion, all coefficients were shrunk to zero except for the intercept, resulting in a flat plane when plotting the approximation. To counter this, the value of alpha was manually decreased to alpha = 1e-4, leading to a convergence warning in the algorithm, but resulting in realistic coefficients. Nevertheless, the fitting was still flat, but better resembled the original data than a plane. The plot of the Lasso approximation can also be seen in Figure 6. Compared to the Ridge and degree 5 OLS plot, it shows a less extreme “hill” structure.

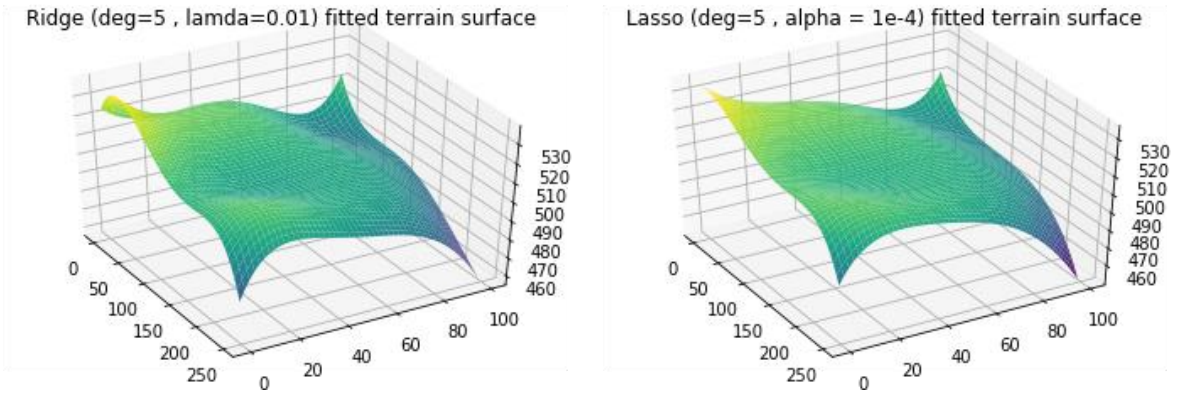

Fig. 5: Fitted terrain surface plots for degree 5 Ridge (left) and Lasso (right)

4.2.2 Critical evaluation of application of regression methods to this data type

Due to the retrieved dataset being very large, the computational time was high, as for instance the cross-validation for the ridge regression with real data took up to 4 minutes. Another reason could be some numerically expensive implementations in the code, as some loops were used. In addition, the computational cost may have been high because many of the functions were calling previously defined functions that sometimes newly created the designmatrix and calculated the regression coefficients repeatedly without storing variables.

When applying regression methods to the data type presented, terrain data, it is only used to parameterize the surface of the earth in this selected area. It should help find the altitude of coordinates that are not within the original input data, but still within the same total area to be parameterized. In other words, it should help fill the gaps within the input grid and the respective target values. It is not used to predict values based on completely new measurements. Thus, in this specific case, a closer fit to the training data or the full data set might be desirable, because it is about an accurate parametrization of the true data, that is not going to be changed. An effect of this issue was that the three regression methods performed significantly worse on test data during cross-validation in comparison to the previous experiment with Franke's function.

5 Conclusion

Overall, after studying the implementation and application of the three regression methods Ordinary Least Squares (OLS), Ridge Regression and Lasso, it was found that there are positive aspects as well as disadvantages about every method. While the OLS method is the easiest to implement and can deliver good results, the risk of overfitting is the highest and the models can grow complex very easily. For OLS, the variance of the model's parameters is the largest, which can lead to worse predictions when using test data. However, in this project, the OLS method performed the best when approximating the test function (Franke's function). Furthermore, when fitting complex data such as the real terrain data, it might be useful to use a more complex model.

On the other hand, Ridge and Lasso are useful methods when trying to avoid overfitting in forecasting models. The penalization on the size of the parameters helps to lower their variance and shrinks their values. This is the reason why for instance the Ridge Regression is considered a high bias and low variance method. The Lasso and the Ridge method differ in how they shrink the coefficients. Lasso can set parameters to zero. The experiments showed that the process of finding a suitable penalization parameter requires fine tuning and cross-validation. Resampling and cross-validation can in addition help to simulate the model's performance on new data, by splitting the data into training and test sets.

When applying the methods on the real terrain data, the difference between fitting the model to training data and testing it on the same data or on test data became obvious. The performance on new test data was significantly worse. Still, one must keep in mind the application of the model to assess its suitability for a task. When only parameterizing available data, such as in the case of the terrain data, the performance on new data might not be as important as in a forecasting setting.

For further and deeper analysis of the discussed topics, the implementation of the methods in the code and the programming can be improved. There were problems with some ineffective iterative methods that could have been solved easier. Furthermore, the use of more *scikit-learn* features could reduce runtime and simplify the programs. For future projects, fragments of the code can be used, in order to implement and study more advanced machine learning methods.

6 Appendices

Appendix 1. Code for shuffle and data split in k-fold crossvalidation

```
# k-crossvalidation, we choose k =10, that means we get 10 partitions
k=10

#generating random order of data points
for j in range(0,len(z)):
    switchindex = random.choice(range(0,len(z)))
    switchvalueX = x[j]
    switchvalueY = y[j]
    switchvalueZ = z[j]
    x[j]=x[switchindex]
    y[j]=y[switchindex]
    z[j]=z[switchindex]
    x[switchindex]=switchvalueX
    y[switchindex]=switchvalueY
    z[switchindex]=switchvalueZ

partitionsize = int(len(z)/k)
index = 0

for k in range(1,k+1):
    testx = x[index:index+partitionsize]                              #testdata
    testy = y[index:index+partitionsize]
    testz = z[index:index+partitionsize]
    trainingx =  np.concatenate((x[:index],x[index+partitionsize:])) #trainingdata
    trainingy =  np.concatenate((y[:index],y[index+partitionsize:]))
    trainingz =  np.concatenate((z[:index],z[index+partitionsize:]))

    [#In this section a model was fitted to the training data and afterwards tested
    on test data. The full code can be found in the github repository]

    index = index+partitionsize
```

Appendix 2. Code for Lasso regression

```
# create design matrix
xy=np.c_[x,y]
poly5 = PolynomialFeatures(degree=5)
XY = poly5.fit_transform(xy)

# find penalty with sklearn cross validation
lassocv=linear_model.LassoCV(max_iter=50000)
lassocv.fit(XY,z)
alpha=lassocv.alpha_
print('alpha= ',lassocv.alpha_)

# perform lasso regression with sklearn
lasso5=linear_model.Lasso(alpha,max_iter=50000)
lasso5.fit(XY,z)
zpredictlasso=lasso5.predict(XY)

# lasso coefficients
betalasso=lasso5.coef_
betalasso[0]=lasso5.intercept_
```

7 References

Franke, R. (1979). *A critical comparison of some methods for interpolation of scattered data.* Naval Postgraduate School Tech.Rep.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning - Second Edition.* Springer.

Mehta, P., Wang, C.-H., Day, A. G., & Richardson, C. (2018). A high-bias, low-variance introduction to Machine Learning for physicists. *Department of Physics, Boston University*.

Murphy, K. (2012). *Machine Learning - A Probabilistic Perspective.* Cambridge, Massachusetts: The MIT Press.

Ramirez, E. (2018). *Jet Propulsion Laboratory - California institue of Technology*. Retrieved from https://www2.jpl.nasa.gov/srtm/

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267-288.