FYS-STK4155

Machine Learning

PROJECT 1

Henrik Løland Gjestang
henriklg@student.matnat.uio.no
Betina Høyer Wester
betiahw@student.matnat.uio.no
Mona Heggen
monahe@student.matnat.uio.no

Dato: October 9, 2018

1 Abstract

The aim of this project is to evaluate various regression methods - Ordinary Least Squares (OLS) method, Ridge regression and Lasso regression. In addition we will be assessing these methods bootstrap resampling techniques. To compare and evaluate these methods we will first apply two-dimensional function called Franke's function and in the second part of the project we implement the same methods on real terrain data.

It has been observed that OLS with polynomial of fifth degree gave the best fit for the data form Franke's function. However, Lasso model of fifth degree was most representative of the terrain data.

Contents

2 Introduction

This paper is written for Project 1 in the course FYS-STK4155, at the University of Oslo. The main objective of this paper is to study regression techniques of Machine Learning (ML). One part of ML investigates learning and fitting of models to a given set of data. The more advanced techniques lets one to make predictions outside the data set one is presented with.

ML techniques play an increasingly large role in many aspects of modern technology. Some examples are self driving cars, smart devices, optimizing energy consumption, advertisement and more. For physicists and people working in field of data science it is of high importance to understanding of the ML basics. As a For this reason we will be doing 3 projects that will take us through polynomial regression, neural networks and deep learning. In this report we will implement and evaluate three different regression methods:

- Ordinary Least Squares (OLS)
- Ridge regression
- Lasso regression

A central objective of this study is to learn, evaluate and find performance degree of various models to the same data set.

a) Ordinary Least Squares

OLS is a method which is used to estimate the unknown parameters in a polynomial regression model. It does so by choosing the parameters of a polynomial function of a set of explanatory variables by the principle of least squares, that is minimizing the sum of the squares of the differences between the observed variables in the given data set and those predicted by the polynomial function. As stated in *A high-bias, low-variance introduction to Machine Learning for physicists* [1] OLS is defined as the minimization of the L_2 norm of the difference between the response y_i and the predictor $g(\mathbf{x}_i;\beta) = \mathbf{x}_i^T\beta$:

$$\min_{\beta\in R^p} ||X\beta - \mathbf{y}||_2^2 = \min_{\beta\in R^p} \sum_{i=1}^{n} (\mathbf{x}_i^T\beta - y_i)^2.$$

In other words, we are looking to find the β which minimizes the L_2 error. The solution to this problem, after differentiation, is

$$\hat{\beta}_{\text{LS}} = (X^TX)^{-1}X^T\mathbf{y}.$$

b) Ridge regression

Ridge regression, on the other hand, reduces the standard errors by adding a degree of bias to the regression estimates. When choosing polynomial for our final model representative we need to make proper evaluations so that we do not underfit or overfit. Underfitting happens when a model with too low polynomials is chosen and thus does not make correct description of our data. Overfitting happens when we choose a model with too high polynomials resulting in a model with high variance and low bias.

$$\hat{\beta}_{\text{Ridge}}(\lambda) =_{\beta\in R^p} \left(||X\beta - \mathbf{y}||_2^2 + \lambda||\beta||_2^2\right).$$

c) Lasso regression

Least Absolute Shrinkage Selector Operator (Lasso), is a polynomial regression method that very similar to Ridge regression. Lasso also has hypermeter α, but when we use Lasso with increasing values of α, coefficients are reduced to absolute zeroes. This is because Lasso selects only the non correlated features while reducing the other coefficients (that correlates) to zero. This method is good for use cases with large number of features, because of this automatically feature selection.

$$\hat{\beta}_{\text{LASSO}}(\alpha) =_{\beta \in R^p} ||X\beta - \mathbf{y}||_2^2 + \alpha||\beta||_1.$$

d) Cross validation

K-fold cross validation is a technique that calculates the accuracy of the model. The more data you have the more accurate is this model. Most common partitioning of the original sample is into five to ten subsamples.

With k-fold cross-validation we can find MSE, bias and the variance of the obtained models. In the k-fold process we first split the data into two sets. The first validation subset (approximately 25%) and then the second subset consisting of training and test data (the remaining 75%). Then we split the training and test data into 25% test data and 75% training data. Following, we find the design matrix for the validation data and for the test data. We use the training data to find the β of our model. Then we use the validation data to calculate predicted z from β found from training data. Next, we calculate the bias and the variance for each fold. At last, we iterate through all the folds finding the final bias and the variance.

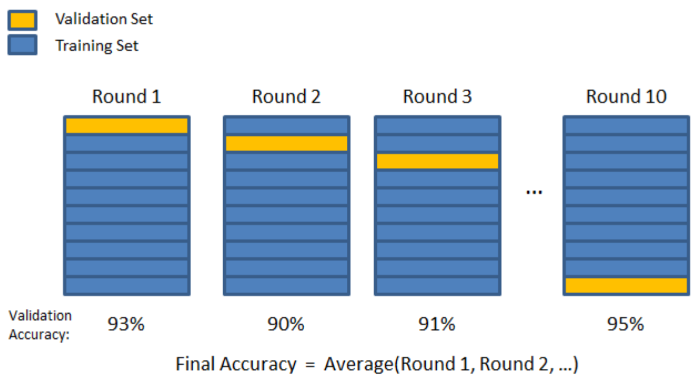


Figure 1: Cross validation

e) Bootstrap

Bootstrap is one of the most fundamental resampling methods. One difference is that cross-validation uses all of your data points, whereas bootstrapping, which resamples your data randomly, may not hit all the points. Since bootstrap resamples data randomly you can also run it for as long as you want. The main principles of the bootstrap algorithm is as follows:

- Draw with replacement n number for the observed variables $\hat{x} = (x_1, x_2, ..., x_n)$.
- Define a vector $\hat{x}*$ containing the values which were drawn from $\hat{x}$.
- Using the vector $\hat{x}*$ compute $\hat{\theta}*$ by evaluating $\hat{\theta}$ under the observations $\hat{x}*$.
- Repeat this process k times.

f) Bias and variance

$$E[(y - \hat{f}(x))^2] = \text{Bias}(\hat{f}(x))^2 + \text{Var}(\hat{f}(x)) + \sigma^2 \tag{1}$$

$$= \Big(E[f(x) - \hat{f}(x)]^2\Big) + \Big(E[\hat{f}(x)^2] - E[\hat{f}(x)]^2\Big) + \sigma^2$$

Bias-variance trade-off tells us something about the relation between the bias, variance and the mean squared error (MSE) for a given model. These characteristics evaluate the performance degree of each model. Best fitting models are the ones with lowest bias and variance. See figure 2. MSE is represented as sum of bias and variance, the best fit to the data of interest would thus be a model with a polynomial degree that gives us the lowest MSE [2].

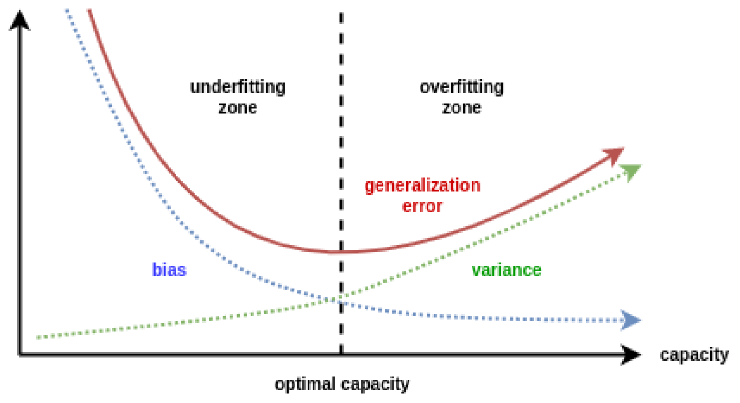


Figure 2: Relationship between Bias-Variance trade-off visualized.[2]

3 Method

a) Franke's function

To study the various regression methods we obtained our data by randomly generating a set of x and y values. Then we entered these into the two-dimensional Franke's equation and used as our observed data set. This function has been widely used when testing interpolation and fitting algorithms.

The Franke's function is expressed as a weighted sum of four exponentials,

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

This function will be defined for $x, y \in [0, 1]$. When we write this into code and run it we get something like this:

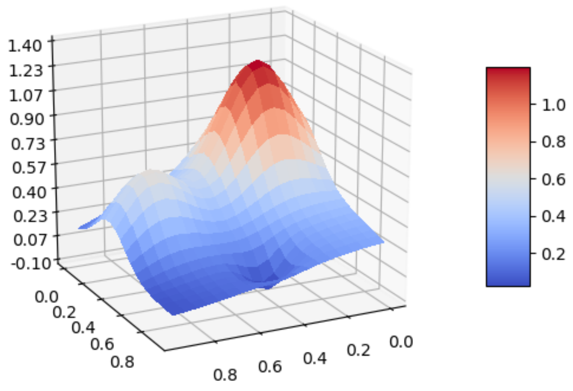


Figure 3: Visual representation of Franke's function

b) Testing our models with Franke's function

Our first step will be to perform OLS regression analysis of Franke's function. We then perform a resampling of the data using bootstrap resampling method, compute mean squared error and R^2-score and pick the model with the best R^2-score. Our β-values and 95%-confidence intervals is shown in table 1.

Here we are only interested to look at the true values of z, and the predicted values computed from OLS with the design matrix computed by true x and y.

$$MSE(\hat{z}, \hat{\tilde{z}}) = \frac{1}{n}\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2$$

We then find the least square errors with MSE with $ztrue$ and computed $zpredict$ from OLS.

$$R^2(\hat{z}, \tilde{\hat{z}}) = 1 - \frac{\sum_{i=0}^{n-1}(z_i - \tilde{z}_i)^2}{\sum_{i=0}^{n-1}(z_i - \bar{z})^2}$$

where,

$$\bar{z} = \frac{1}{n} \sum_{i=0}^{n-1} z_i$$

After implementation of this method we plot the obtained results in figure 4.

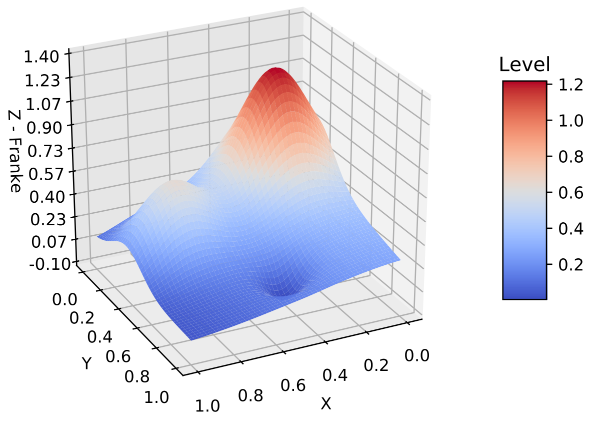


Figure 4: Fitted data with OLS regression

By visual examination we see the obtained surface has the similar form as the original plot of Franke's function. However, some of the most prominent differences is that the surface is somewhat skewed.

The variance of the parameters β in the linear regression method [3] is given by,

$$\text{Var}(\hat{\beta}) = \left(\hat{X}^T \hat{X}\right)^{-1} \sigma^2$$

with

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

This is the the variance-covariance matrix of the least squares parameter estimates. It finds the β-variance by computing the standard deviation that is covariant with the MSE and then finds the β-coefficients by taking the square root of the eigenvalues of the β matrix' variance. This gives us the β-coefficients and their confidence interval presented in figure 4.

Before we examine bootstrap resampling method, we evaluate the mean value, MSE and the R^2-score. We estimate this by using the results are found in table 1 on page 16.

Next up is to look at the variance and bias of our models. Since

$$MSE = Variance + Bias$$

we can derive the expression in function 2 on page 5.

From table 1 we can see that the squared bias and the variance for the model does not correspond to the mean squared error (MSE = Bias + Variance). This must be fixed before we can proceed with evaluation and application of the model. This fix is done by using a resampling method like bootstrap or k-fold cross-validation. We started with the k-fold, but because of some issues with finding the right bias we switched to bootstrap.

The results from bootstrap are presented in table 2. In contrast to the evaluation of data before the application of bootstrap we obtain that MSE = Bias + Variance this time.

Next, we perform the same procedure for the Ridge regression. Before doing the bootstrap validation, we looked at different values of λ together with the R^2-score and noticed that we got the best fit if we chose a small λ-value. When we do the bootstrap validation, we use $\lambda = 0.5$. We get the β-values and confidence interval shown in table 5 in the appendix.

When we use these values of β to test the model on the Franke's function we get,

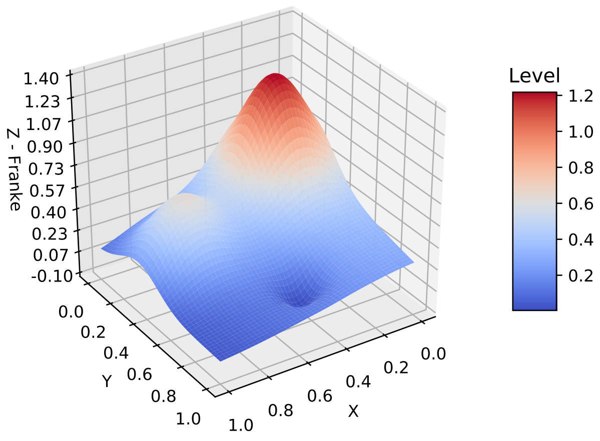


Figure 5: Fitted data to Franke's function with Ridge regression, $\lambda = 0.5$

For Lasso method we are using SKLearn instead of implementing our own code. This is both due to time constraints and usefulness to be familiar with SKLearn for further projects.

Our β values from Lasso regression are presented in table 1 and when we use the β values to train the model we get this estimation of Franke's function (6). Our β values are shown in table 6 in the Appendix.

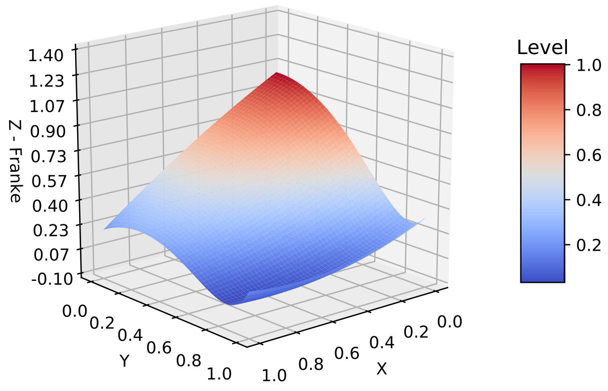


Figure 6: Estimation of Franke's function using Lasso method trained on given betas. $\lambda = 10^{-5}$.

In this example we use Franke's function to train our regression model we have data that does not serve well as real world. To try to show that our model also would work with 'imperfect' data we introduce some noise into the Franke's function. When we introduce noise into the Franke's function we can plot MSE for different values of λ.

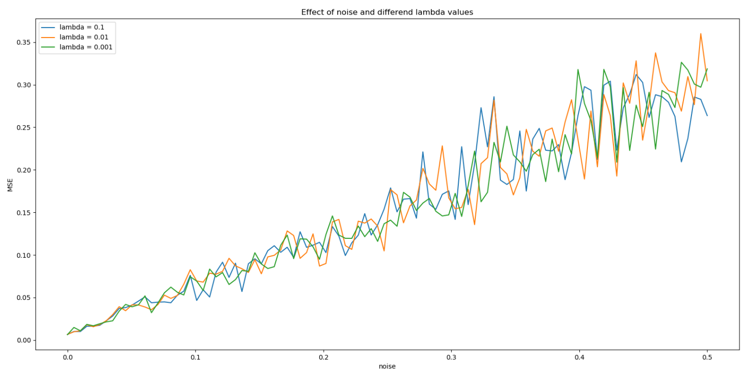


Figure 7: Effect of noise on λ-values [0.1, 0.01, 0.001]

From 7 we see that for $\lambda = [0.1, 0.01, 0.001]$ there is not much difference in MSE error. In further studies, we would have tried a larger interval for λ-values so we could see for which values of λ the model would start to over fit the data.

In figure 8, 9 and 10 we look at MSE and R^2-score for the three regression methods with a polynomial with changing degree. From these plots we can see how the MSE and R^2-score changes when the degree changes. When the degree exceed 10, we start to accumulate more and more MSE error, while the R^2 error dips. The R^2 is expected to dip because by using such a high value of degree our model becomes over-fitted. You can see this better in figure 15, 16 and 17 in the appendix, where we have plotted the MSE and R^2 for degree 1 to 20. The ideal is a high R^2-score (up to one) and at the same time a low mean squared error-score. A negative R^2-score indicates that the model with the given parametres does not fit the dataset. R^2 is negative only when the chosen model does not follow the trend of the data. It simply means that the chosen model (with its constraints) fits the data really poorly.

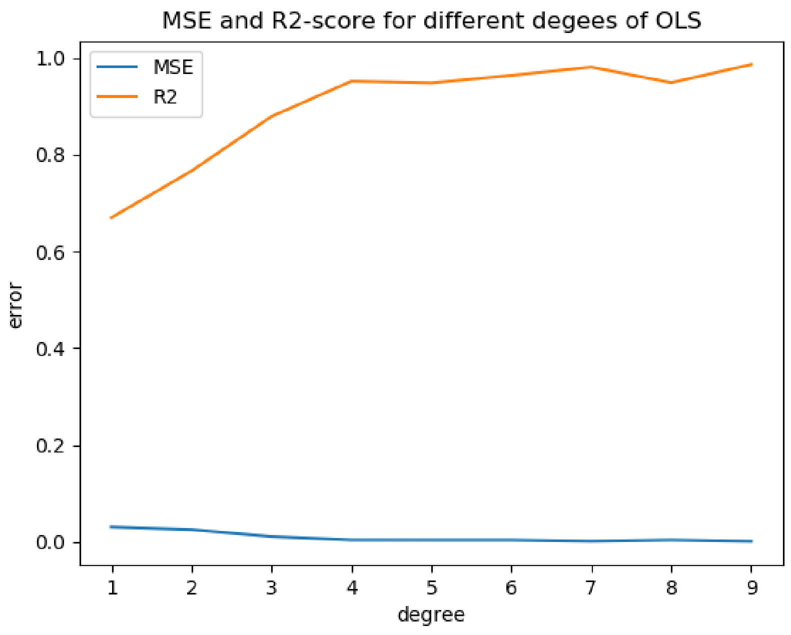


Figure 8: MSE and R2-score for different polynomial degrees fitted with OLS.

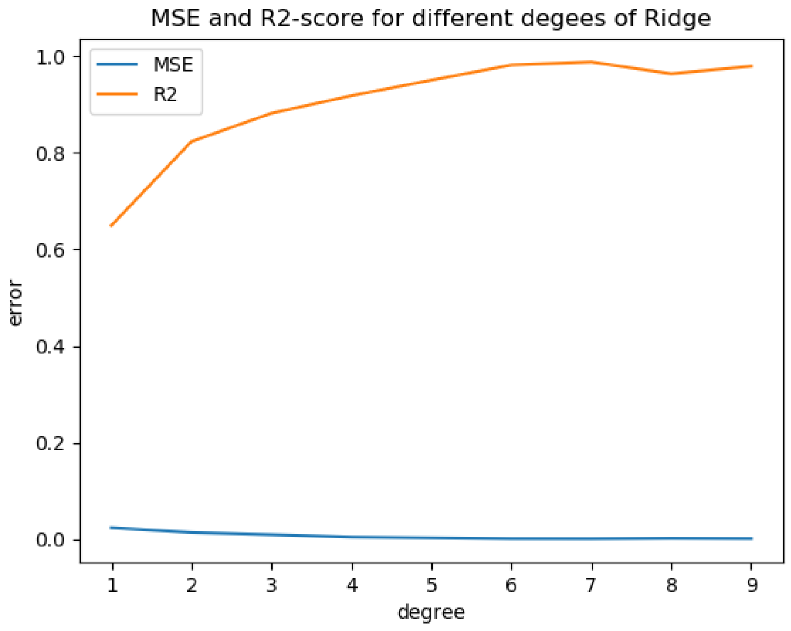


Figure 9: MSE and R2-score for different polynomial degrees fitted with Ridge regression.

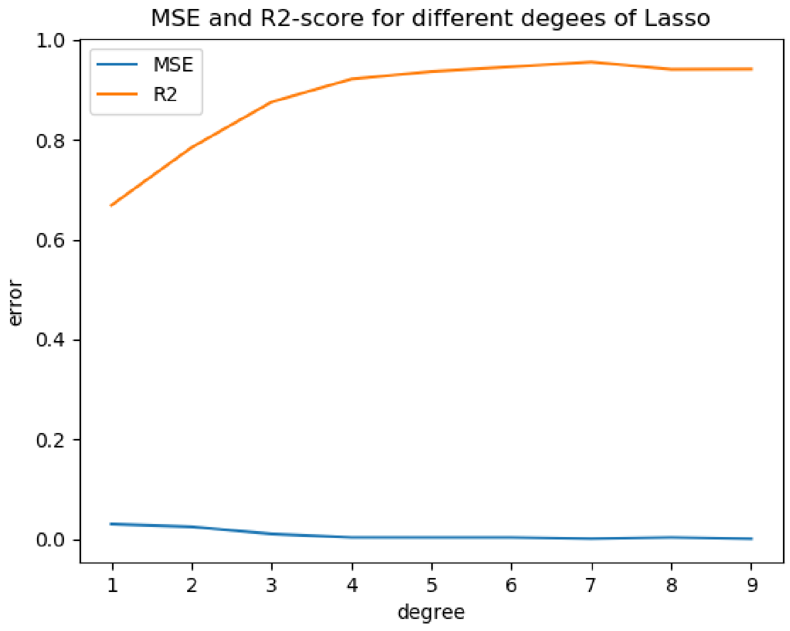


Figure 10: MSE and R2-score for different polynomial degrees fitted with Lasso.

c) Testing our models with real data

We now wish to test ordinary least squares, Ridge regression and Lasso regression with real data, using bootstrap as resampling technique. The set we are using is data of a terrain over Norway, shown in figure 11.

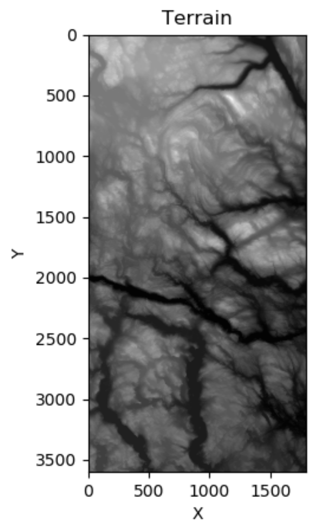


Figure 11: Our data set: Terrain over Norway

Since this is a big dataset, we will only focus on small part of the terrain area, thus choosing subset of the total dataset. Since we in Ridge regression have a parameter λ, and in Lasso regression have a parameter α, we need to optimize these values before we start comparing regression models. By doing this we first shuffle our data and split it into a training set and test set (these sets are the same size). Then we look at the mean squared error and R^2-score of the models for different λ-and α-values. The results are shown in figure 12 and figure 13.

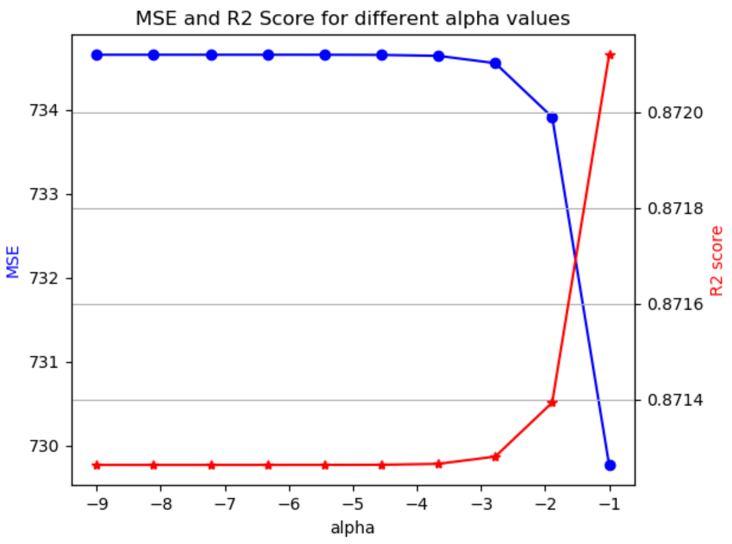


Figure 12: α-values compared to mean squared error in Lasso regression. In the figure the α-values represent the 10^n, where n is the α-values.

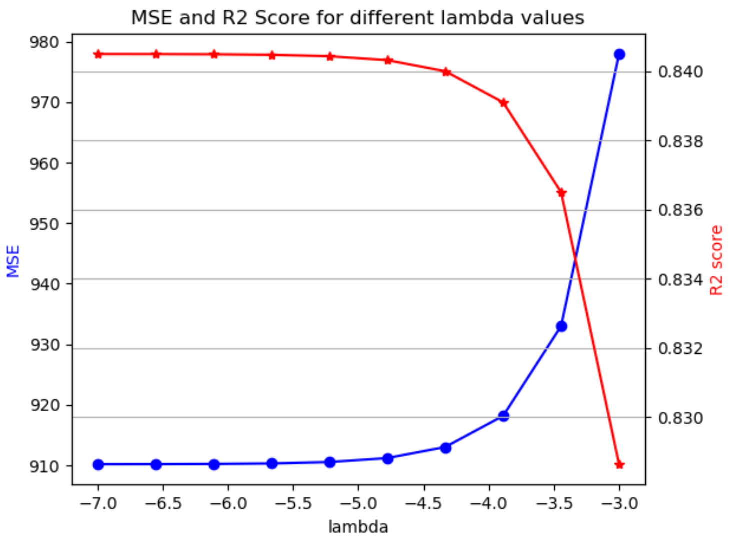
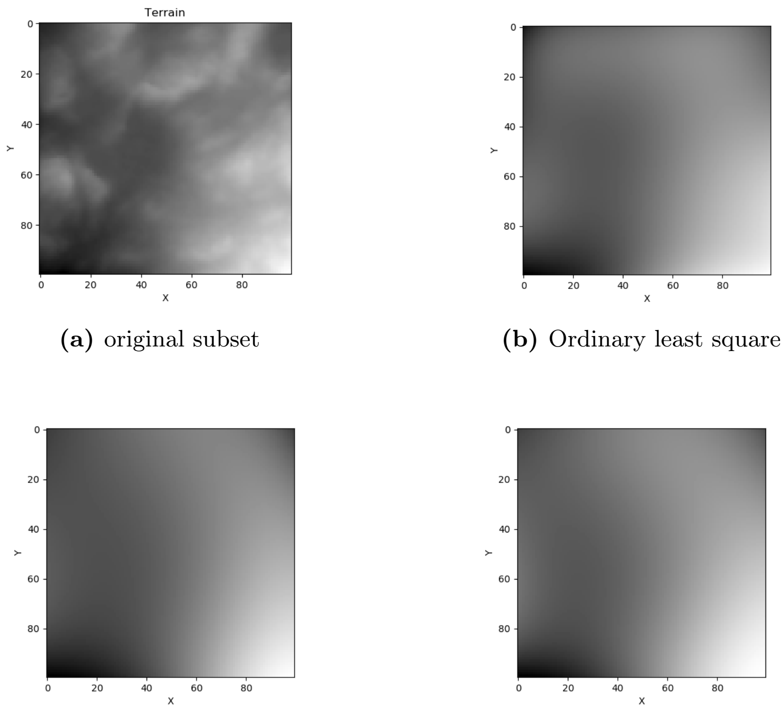


Figure 13: λ-values compared to mean squared error in Ridge regression

We see from the figures that we get the best models my choosing small α and λ. We therefore choose $\alpha = 10^{-3}$ in Lasso-regression and $\lambda = 10^{-5}$ for Ridge regression, and we are now ready to compare regression models. We use bootstrap to find the model which fits our terrain-data best. We compare mean squared error, R^2-score, bias and variance provided by each regression method.

As mentioned, we look at a subset of our terrain data to make computations go faster. Our result is shown in table 3.



(a) original subset

(b) Ordinary least square

(c) Ridge regression

(d) Lasso regression

Figure 14: Our terrain computed by our regression models

4 Results and discussion

When we generated our data-set from the Franke's Function, the model generated with Ridge regression got the best R^2-score, the lowest mean squared error and the lowest variance, but ordinary least squares got smaller bias. When we introduced real data, the model generated by ordinary least squares got the best R^2-score, the lowest mean squared error and the lowest bias, and Ridge got the lowest variance. In general, ordinary least squares and Ridge regression gave us good models for computing terrain, and Lasso regression provided the model with the worst fit. In general, bootstrapping with Lasso regression became more computationally heavy than the two other models.

	OLS	Ridge	Lasso
MSE	$2.0154e^{-3}$	$5.82719e^{-3}$	$7.25932e^{-3}$
R2	0.987521	0.992817	0.909034

Table 1: Calculated error of OLS, Ridge and Lasso methods, polynomial degree = 5, $\lambda = 0.5$ and $\alpha = 10e^{-5}$

	OLS	Ridge	Lasso
MSE	$2.5854e^{-3}$	$2.5246e^{-3}$	$3.7561e^{-3}$
R2	$9.5230e^{-1}$	$9.6755e^{-1}$	$9.5664e^{-1}$
Bias	$1.9637e^{-3}$	$2.1510e^{-3}$	$3.4443e^{-3}$
Variance	$6.2168e^{-4}$	$3.7361e^{-4}$	$3.1183e^{-4}$

Table 2: Bootstrap validation of different regression methods with Franke's function. Degree = 5, $\lambda = 10^{-5}$ and $\alpha = 10^{-3}$

	OLS	Ridge	Lasso
MSE	454.75	482.03	529.55
R2	$9.0416e^{-1}$	$9.0074e^{-1}$	$8.8903e^{-1}$
Bias	453.42	480.77	528.22
Variance	1.3288	1.2539	1.3277

Table 3: Bootstrap validation of MSE, R2-score, bias and variance for OLS, Ridge and Lasso, using real data. Degree = 5, $\lambda = 10^{-5}$ and $\alpha = 10^{-3}$

5 Conclusions and perspectives

When we used our generated data from Franke's function, all three models provided satisfactory models with small deviations in mean squared error, R^2-score, bias and variance, but when we introduced real data, Ridge regression and ordinary least square regression gave us better models than Lasso regression. Considering the heavy computations in bootstrapping with Lasso Regression, we can conclude that ordinary least square regression and Ridge regression are the best methods for providing models for our terrain data. We see that we can provide good models using simple regression methods. When we tested our regression model with Franke's function, we observed that we got the best fit choosing a model with polynomial degree 5. Since Lasso forces some of the β values to be zeros, and since the terrain-data is so complex, we need all parameters to get a good fit. But since Ridge regression got a good fit, this indicates that some of the parameters are more significant than others.

This project in Machine Learning was a major commitment. We split most of the work and shared large parts of it between the group members but some of it required all members to comprehend. Studying and understanding the main concept was time consuming and we have spent a lot of time on this project - but in the end we feel that it was rewarding and we have learned a lot.

6 Appendix

a) Figures

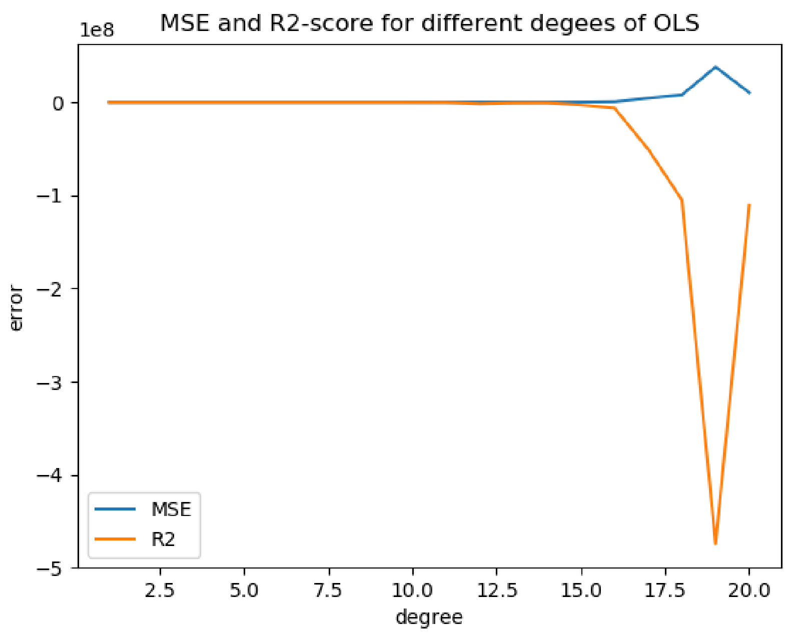


Figure 15: MSE and R2-score for different polynomial degrees fitted with OLS.

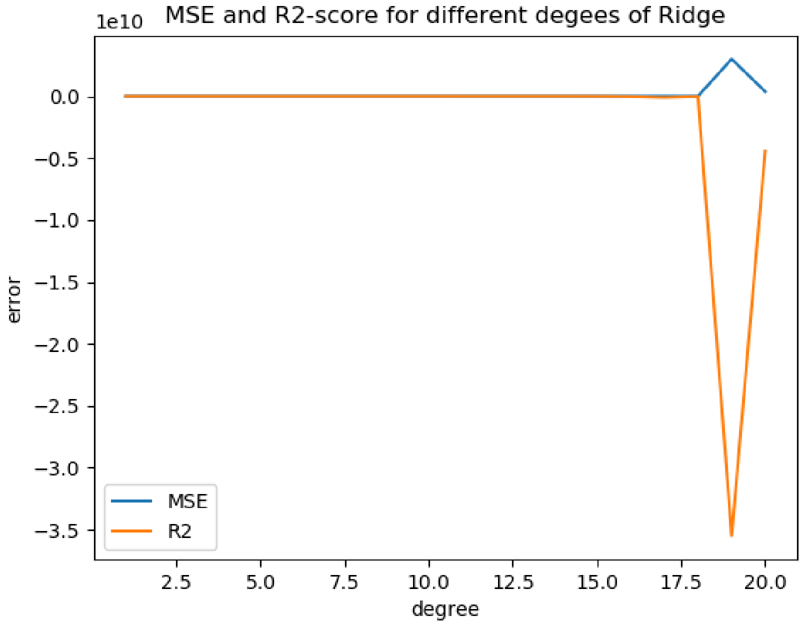


Figure 16: MSE and R2-score for different polynomial degrees fitted with Ridge regression.

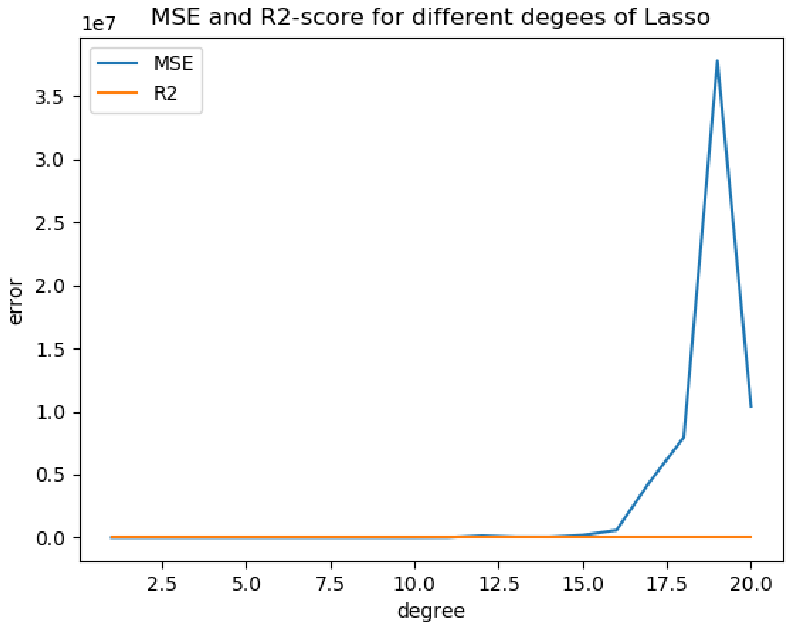


Figure 17: MSE and R2-score for different polynomial degrees fitted with Lasso.

b) Tables

β	value	confidence interval
0	0.38359657	0.38057137, 0.38662177
1	7.94267297	7.90898627, 7.97635966
2	3.87636858	3.84268189, 3.91005528
3	-34.45872345	-34.62109373, -34.29635316
4	-15.47516698	-15.60203592, -15.34829804
5	-8.96755549	-9.12992577, -8.8051852
6	47.89482466	47.52967076, 48.25997855
7	45.31689036	45.0460256, 45.58775512
8	21.3859254	21.11506064, 21.65679015
9	-7.65086618	-8.01602008, -7.28571228
10	-22.60705746	-22.98949106, -22.22462386
11	-54.39054128	-54.68283382, -54.09824874
12	-7.88625491	-8.15834028, -7.61416953
13	-30.61261916	-30.9049117, -30.32032662
14	28.99898026	28.61654666, 29.38141387
15	0.87617274	0.72560151, 1.02674397
16	19.51194634	19.38091607, 19.64297661
17	10.50598785	10.37812768, 10.63384802
18	-5.18996586	-5.31782603, -5.06210568
19	17.07823636	16.94720609, 17.20926663
20	-16.46899467	-16.6195659, -16.31842344

Table 4: β-values and 95%-confidence interval for OLS regression

β	value	confidence interval
0	0.88636603	0.88166076, 0.8910713
1	1.0834995	1.0311046 , 1.13589439
2	1.40390196	1.35150706, 1.45629686
3	-6.64179612	-6.89434017, -6.38925206
4	1.01375842	0.8164317 , 1.21108514
5	-8.96755549	-6.25287919, -5.74779108
6	4.18671518	3.61876985, 4.75466051
7	1.77602348	1.35473159, 2.19731537
8	-1.31413489	-1.73542678, -0.892843
9	2.025216628	1.45727129, 2.59316195
10	4.11764721	3.52282573, 4.71246868
11	0.69398982	0.23937005, 1.1486096
12	0.98765134	0.56446096, 1.41084173
13	-0.6771815	-1.13180128, -0.22256173
14	4.06261958	3.46779811, 4.65744106
15	-3.71558518	-3.94977746, -3.48139289
16	-1.99757225	-2.20137133, -1.79377317
17	0.58162334	0.3827549 , 0.78049178
18	-1.04349527	-1.24236372, -0.84462683
19	0.52127009	0.317471 , 0.72506917
20	-2.00947876	-2.24367105, -1.77528647

Table 5: β-values and 95%-confidence interval for Ridge regression

β	value
0	1.47011903
1	-1.88225667
2	-0.12909919
3	-0.64570042
4	2.55091299
5	-3.18649382
6	1.18242671
7	1.30276291
8	-0.77232804
9	0.22859632
10	0.13969607
11	-0.28062852
12	-0.5287407
13	0.25317329
14	1.22264699
15	-0.21907454
16	-0.98128879
17	0.16383744
18	-0.36739656
19	-0.24234818
20	0.75930537

Table 6: β-values for Lasso

c) Code

Franke's Function implemented in python

```python
import numpy as np

def FrankeFunction(x,y):
    term1 = 0.75*np.exp(-(0.25*(9*x-2)**2) - 0.25*((9*y-2)**2))
    term2 = 0.75*np.exp(-((9*x+1)**2)/49.0 - 0.1*(9*y+1))
    term3 = 0.5*np.exp(-(9*x-7)**2/4.0 - 0.25*((9*y-3)**2))
    term4 = -0.2*np.exp(-(9*x-4)**2 - (9*y-7)**2)
    return (term1 + term2 + term3 + term4)
```

How we generate the surface plot of Franke's function

```python
fig = plt.figure()
ax = fig.gca(projection='3d')

# Make 20 data points for x and y
x = np.arange(0, 1, 0.05)
y = np.arange(0, 1, 0.05)
x, y = np.meshgrid(x,y)

z = FrankeFunction(x,y)

# Plot the surface
surf = ax.plot_surface(x, y, z, cmap=cm.coolwarm, linewidth=0, antialiased=False)

```

```python
# Customize the z axis
ax.set_zlim(-0.10, 1.40)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors
fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

Lasso method with SKLearn and trained on Franke's function data

```python
import numpy as np
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from frankeFunction import FrankeFunction
from scipy import misc

def OSLRegression(x, y, z, degree=5, l=0.1):
    # Split into training and test
    x_train = np.random.rand(100,1)
    y_train = np.random.rand(100,1)
    z = FrankeFunction(x_train,y_train)

    # traning data and finding design matrix X_
    X = np.c_[x_train,y_train]
    poly = PolynomialFeatures(degree)
    X_ = poly.fit_transform(X)
    clf = linear_model.LassoCV()
    clf.fit(X_, z)
    beta = clf.coef_

    return beta
```

Bootstrap validation

```python
def bootstrap(x, y, z, p_degree, method, n_bootstrap=100):
    # Randomly shuffle data
    data_set = np.c_[x, y, z]
    np.random.shuffle(data_set)
    set_size = round(len(x)/5)

    # Extract test-set, never used in training. About 1/5 of total data
    x_test = data_set[0:set_size, 0]
    y_test = data_set[0:set_size, 1]
    z_test = data_set[0:set_size, 2]
    test_indices = np.linspace(0, set_size-1, set_size)

    # And define the training set as the rest of the data
    x_train = np.delete(data_set[:, 0], test_indices)
    y_train = np.delete(data_set[:, 1], test_indices)
    z_train = np.delete(data_set[:, 2], test_indices)

    Z_predict = []

    MSE = []
    R2s = []
    for i in range(n_bootstrap):
        x_, y_, z_ = resample(x_train, y_train, z_train)

        if method == 'Ridge':
            # Ridge regression, save beta values
            beta = RidgeRegression(x_, y_, z_, degree=p_degree, l=0)
        elif method == 'Lasso':
            beta = Lasso(x_, y_, z_, degree=p_degree)
        elif method == 'OLS':
            beta = ols(x_, y_, z_, degree=p_degree)
        else:
            print('ERROR: Cannot recognize method')
            return 0

        M_ = np.c_[x_test, y_test]
        poly = PolynomialFeatures(p_degree)
        M = poly.fit_transform(M_)
        z_hat = M.dot(beta)

        Z_predict.append(z_hat)

        # Calculate MSE
        MSE.append(np.mean((z_test - z_hat)**2))
        R2s.append(R2(z_test, z_hat))

    # Calculate MSE, Bias and Variance
    MSE_M = np.mean(MSE)
    R2_M = np.mean(R2s)
    bias = np.mean((z_test - np.mean(Z_predict, axis=0, keepdims=True))**2)
    variance = np.mean (np.var(Z_predict, axis=0, keepdims=True))
    return MSE_M, R2_M, bias, variance
```

Some Various functions. MSE, R2, Variance, Mean, Sigma, Bias, Confidence interval

```python
def mu(z,n):
    #Gjennomsnittsverdien
    z_mean = (1/n ) * np.sum(z)
    return z_mean

def calc_Variance(z, z_mu,n):
    #Sample variance:
    var_z = (1/n)* sum((z-z_mu)**2)
    return var_z

def MSE(z, z_tilde, n):
    #Mean Squared Error: z = true value, z_tilde = forventet z utifra modell
    MSE = (1/n)*(sum(z-z_tilde)**2)
    #error = np.mean( np.mean((z - z_tilde)**2, axis=1, keepdims=True) )
    return MSE

def calc_R_2(z, z_tilde, z_mean, n):
    R_2 = 1- ((sum(z.reshape(-1,1)-z_tilde)**2)/(sum((z-z_mean)**2)))
    return R_2

def MSEk(z, z_tilde, n):
    MSE = (1/n)*(sum((z-z_tilde)**2))
    return MSE

def sigma_2(xyb, z_true, z_predict, N, p):
    sigma2 = (1/(N-(p-1 )))* (sum(z_true-z_predict)**2)
    varBeta = np.linalg.inv(xyb.T.dot(xyb))* sigma2#**2
    #Intervall betacoefisienter:
    s_beta_c_int = np.sqrt(np.diag(varBeta))
    return  s_beta_c_int

# Finner Confidens Intervallet
def confidenceIntervall(z_true, z_predict, N, p, xyb, beta):
    #N = z punkter , p = ant polynom.

    sigma2 = (1/(N-(p-1 ))) * (sum((z_true-z_predict)**2))
    # Betas varians:
    varBeta = np.linalg.inv((xyb.T.dot(xyb)))* sigma2

    # estimert standardavvik pr beta.
    betaCoeff = (np.sqrt(np.diag(varBeta))).reshape(-1,1)
    #Intervall betacoefisienter:
    beta_confInt = np.c_[beta-betaCoeff, beta+betaCoeff]
    return beta, betaCoeff, beta_confInt

# Finner modellens bias og varians:
def bias(z_true, z_predict, N):
    bias2 = np.sum((z_true - np.mean(z_predict))**2 )/N
    #bias = np.mean( (z_true - np.mean(z_predict, axis=1, keepdims=True))**2 )
    return bias2

def var2(z_predict, N):
    var = np.sum((z_predict - np.mean(z_predict))**2)/N
    #variance = np.mean( np.var(z_predict, axis=1, keepdims=True) )
    return var
```

Bibliography

[1] Pankaj Mehta, Ching-Hao Wang, Alexandre G. R. Day, and Clint Richardson. *A high-bias, low-variance introduction to Machine Learning for physicists.* Page 21-22. (March 26. 2018).

[2] Daniel Saunders. *Minds, Brains, and Programs: The Bias-Variance Tradeoff* https://djsaunde.wordpress.com/2017/07/17/the-bias-variance-tradeoff/ (July 17, 2017).

[3] Trevor Hastie, Robert Tibshirani, Jerome H. Friedman. *The Elements of Statistical Learning*, Chapter 3, equation (3.8)