Project 1 FYS-STK4155 Autumn 2018

Jon Audun Baar & Mikael Ravndal

8. oktober 2018

1 Introduction

A common problem in inferential statistics is to investigate dependencies among random variables. That is, given the outcome of one or more random variables, one wishes to develop a model that estimates the outcome of another random variable. A widely used technique for developing such a model is a method called linear regression. In this project we aim to evaluate the performance of three different types of linear regression using a polynomial as our model. The methods are: ordinary least squares regression (abreviated OLS), ridge regression and lasso regression.

First we evaluate our methods on a known function, that is we know the relationship between our variables. Then we apply the same methods to evaluate the performance of our methods on a real data set where we don't know the underlying relationship.

2 Method

The general idea of linear regression is to assume that the relationship between your dependent and independent variables is given by some function (which is linear in it's coefficients) and then try to estimate this function, also called a *model*, by minimizing some error term. As the scope of this text is to evaluate the performance of this method in some special cases rather than explaining it in general, we are not going to induldge in the full derivation of this method but rather assume that the reader is familiar with the basics of this concept. However we will need to establish some context, notation and assumptions.

In our case we are given two independent variables, also called *predictors*, and one dependent variable, also called *response*. We denote by N the number of datapoints/observations in our sampledata and gather the observations of the predictors in two vectors $\boldsymbol{x_1} \in \mathbb{R}^N$ and $\boldsymbol{x_2} \in \mathbb{R}^N$, and the observations of the response in one vector $\boldsymbol{y} \in \mathbb{R}^N$.

Further we assume the true relationship between the predictors and response to be:

$$y = f(x_1, x_2) + \epsilon \tag{1}$$

where ϵ is a stocastic variable which is normally distributed around 0 and f is some function. We will try to model the function f by a polynomial p of degree

k given by

$$
\begin{aligned}
p(x_1, x_2) = \beta_0 + &\\
&\beta_1 x_1 \;+\; \beta_2 x_2 \;+ \\
&\beta_3 x_1^2 \;+\; \beta_4 x_1 x_2 \;+\; \beta_5 x_2^2 \\
&+\; \dots \;+ \\
&\beta_{q-(k+1)} x_1^k \;+\; \beta_{q-k} x_1^{k-1} x_2 \;+\; \dots \;+\; \beta_{q-1} x_1 x_2^{k-1} \;+\; \beta_q x_2^k
\end{aligned} \tag{2}
$$

Here q is the sum of the natural numbers from 1 to $k+1$.

If we let $x_{i,j}$ be the j-th entry of $\boldsymbol{x_i}$ our design matrix is then given by

$$
X \;=\; \begin{bmatrix}
1 & x_{1,1} & x_{2,1} & x_{1,1}^2 & x_{1,1}x_{2,1} & x_{2,1} & \dots & x_{1,1}x_{2,1}^{k-1} & x_{2,1}^k \\
1 & x_{1,2} & x_{2,2} & x_{1,2}^2 & x_{1,2}x_{2,2} & x_{2,2} & \dots & x_{1,2}x_{2,2}^{k-1} & x_{2,2}^k \\
\vdots & \vdots & & & & & & & \vdots \\
1 & x_{1,n} & x_{2,n} & x_{1,n}^2 & x_{1,n}x_{2,n} & x_{2,n} & \dots & x_{1,n}x_{2,n}^{k-1} & x_{2,n}^k
\end{bmatrix} \tag{3}
$$

and from what we have established so far we get the equation

$$
\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{4}
$$

where $\boldsymbol{\epsilon}$ is a vector of errors. Solving this equation for $\boldsymbol{\epsilon}$, we get that $\boldsymbol{\epsilon} \;=\; \boldsymbol{y} - X\boldsymbol{\beta}$.

In the following we will discuss how to estimate $\boldsymbol{\beta}$ in order to obtain our model p. The general idea is the same for all three methods: we will try to get a good fit of our model to the sample data by minimizing the magnitude of the vector $\boldsymbol{\epsilon}$. In doing this we define a function $C := C(\boldsymbol{\beta})$ called the *cost function*, which we wish to minimize.

2.1 Ordinary least squares regression

In the case of OLS regression the cost function is defined by

$$
C(\boldsymbol{\beta}) = \sum_{i=1}^{N} (y_i - (X\boldsymbol{\beta})_i)^2 \tag{5}
$$

One great benefit of this method as opposed to for instance the later explained lasso method, is that the problem of minimizing C over all $\boldsymbol{\beta} \in \mathbb{R}^N$ has an analytical solution. One can with some elementary calculus show that the solution is given by (6) given that X is non-singular. For a thorough explanation of this identity consult [**?**]

$$
\boldsymbol{\beta} \;=\; (X^T X)^{-1} X^T \boldsymbol{y} \tag{6}
$$

2.2 Ridge regression

One major weakness of OLS regression is that noisy sample data can cause large coefficients, that is a model with large variance. One way to approach this problem is to impose a restriction to the magnitude of our coefficients $\boldsymbol{\beta}$. Ridge regression does this by adding the 2-norm of $\boldsymbol{\beta}$ to the cost function of OLS regression, and in that way penalizing large betas. We call this term a *penality*. The Ridge cost function is therefore given by

$$C(\boldsymbol{\beta}) = \sum_{i=1}^{N}(y_i - (X\boldsymbol{\beta})_i)^2 + \sum_{i=1}^{N}\beta_i^2 \tag{7}$$

As with the case of OLS, we are in a good position with respect to solving this minimization problem. By applying the method of Lagrange multipliers one can show that the solution for $\boldsymbol{\beta}$ is given by

$$\boldsymbol{\beta} \ = \ (X^T X + \lambda I)^{-1} X^T \boldsymbol{y} \tag{8}$$

where I is the identity matrix and λ is a positive real number. Choosing λ big then more emphazises restricting the magnitude of $\boldsymbol{\beta}$, while choosing a small λ more emphazises minimizing the magnitude of $\boldsymbol{\epsilon}$.

2.3 Lasso regression

Just as for ridge, the lasso method solves the problem of high variance by adding a penality to the cost function. The difference is that lasso adds the 1-norm of $\boldsymbol{\beta}$ to C instead of the 2-norm. Thus we get

$$C(\boldsymbol{\beta}) = \sum_{i=1}^{N}(\boldsymbol{y_i} - \boldsymbol{y_i'})^2 + \sum_{i=1}^{N}|\beta_i| \tag{9}$$

The advantage of this approach over ridge is that it tends to set coefficients that are irrelevant to zero instead of very small values. One might then end up with a smarter model with fewer predictors than one started out with. For a thorough explanation of this consult [?]. The draw back however is that we no longer have an analytical solution for $\boldsymbol{\beta}$ and have to resort to numerical methods to minimize C.

2.4 Performance measures

2.4.1 The mean squared error

An intuitive way of measuring the performance of a model would be to try it out on real data (which is not fitting data) and compute some sort of a mean value of the errors. This is in fact also the standard way of doing it! The *mean squared error*, or just *MSE*, is defined by

$$MSE(\boldsymbol{y}, p(\boldsymbol{x})) = 1/N \sum_{i=1}^{N}(y_i - p(x_i))^2 \tag{10}$$

and gives us a measure of how good the model predicts on new data.

2.4.2 Resampling

Usually when one have fitted a model to the given sampledata, the task of obtaining new sample data for testing might be hard, expensive or even impossible. Since the MSE and most other error measures don't give us any new information if we use it on the same data as we used for fitting, a technique called *resampling* is often applied.

The general idea of resampling is to divide your sample data into two parts, *trainingdata* and *testdata*. The trainingdata is used to fit your model while the testdata is used to evaluate the model by computing different statistical estimators such as the MSE.

2.4.3 Resampling using bootstrapping

There are many ways to perform the task of resampling. Crossvalidation, jackknife and bootstrapping are common ones. In this text we will discuss and use bootstrapping in our evaluation of our regression methods.

The principle of bootstrapping is to obtain your training data by randomly drawing N datapoints from your sample data with replacement B times. B is a pretty large number, usually 1000 or more or as much as your computer can handle. In this way one obtains B sets of training data for fitting. The huge advantage with bootstrapping is that you don't have to know the underlying probability distribution of your population. In addition it can be shown that when B tends to infinity your bottstrapestimators tend to the real value of the parameter.

2.4.4 Decomposing the MSE

When fitting a model to a set of data one will always encounter the balancing act of determining the complexity of the model: Of course one wants a high complexity in order to get a good fit to the training data. The drawback with this is however that as you crank up the complexity of your model, it will usually start to predict worse on the test data at some point. This problem is called *overfitting*. On the contrary as you lower the complexity of the model it's predictions on the training data will get worse, and if you go to far you get the problem that the model lacks the flexibility to give good predictions on any data. This problem is called *underfitting*.

Two statistical parameters that are very helpful in the above mentioned balancing act are the *variance* and *bias* of your model. If we let p_j be the model fitted to bootstrap sample j, the bias squared is given by (11), while the variance is given by (12).

$$Bias^2 \ = \ 1/N\sum_{i=1}^{N}(y_i - 1/B\sum_{j=1}^{B}p_j(x_i))^2 \tag{11}$$

$$Var \ = \ 1/N\sum_{i=1}^{N}1/B\sum_{j=1}^{B}(p_j(x_i) - 1/B\sum p_j(x_i)) \tag{12}$$

A high variance indicates overfitting, while a high bias indicates underfitting. If we let σ^2 be the variance of the true probability distribution one can show that there's a connection between the MSE, Bias squared, variance and true variance. This is called the decomposition of the MSE and is given by

$$MSE \ = \ \sigma^2 \ + \ Bias^2 \ + \ Var \tag{13}$$

For a thorough explanation of this decomposition consult [?]

In the context of this text the degree of our polynomial p is a good measure of it's complexity and we will thus use k as a measure for the complexity of our model.

2.4.5 R2 score

Another measure for how well our model predicts outside the training data is the *R2Score* which is given by

$$R^2(\boldsymbol{y}, p(\boldsymbol{x_1}, \boldsymbol{x_2})) \;=\; 1 - \frac{\sum_{i=1}^{N}(y_i - p(x_i))^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2} \tag{14}$$

where $\overline{y}$ is the mean of the responses in the sampledata, that is the mean of the scalars in the vector y.

2.4.6 Variance of $\boldsymbol{\beta}$

The final measure for model performance that we will apply is the variance of the coefficients of our polynomial, $\boldsymbol{\beta}$. For OLS regression one can show that the covariance matrix of $\boldsymbol{\beta}$ is given by

$$Var(\boldsymbol{\beta}) \;=\; (X^TX)^{-1}\sigma^2 \tag{15}$$

where $\sigma^2 = 1/(N-p-1)\sum_{i=1}^{N}(y_i - p(x_i))$ (p is the number of predictors). Thus we have an analytical expression for the variance of $\boldsymbol{\beta}$.

For ridge and lasso this is not possible and we will resort to the sample variance of $\boldsymbol{\beta}$ computed by using bootstrapping.

3 Implementation

We have used python to implement our own classes and functions for fitting and evaluating models as described in section 2. The complete code can be found in [**?**]. The main part of the code are the classes OLSLinearModel, RidgeLinearModel and LassoLinearModel.

We found the choice of implementing our three different models as classes a rather intuitive and oversiktlig way of doing it. The advantage of this implementation is obviously that as we store most of the computations done in the model, excessive computing is not a problem. The weakness might then be excessive memory usage. This was hovewer not a problem in our application of the program.

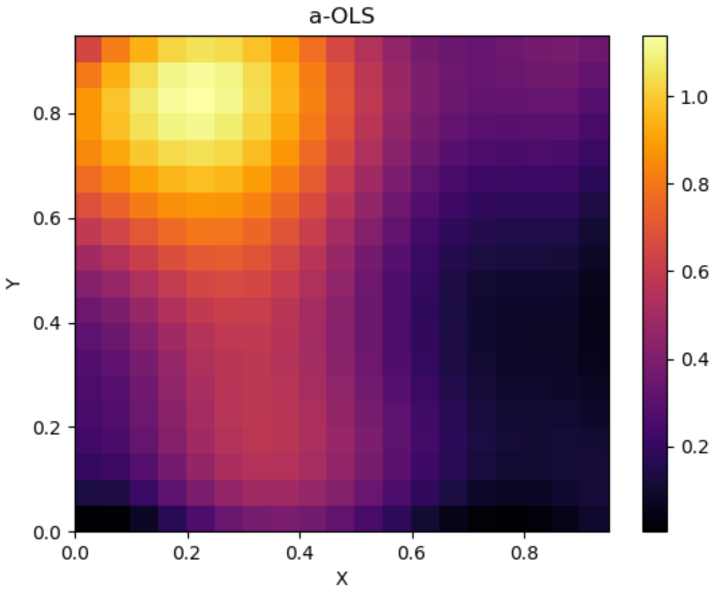
We tested our classes and methods against the classes and methods in scikit-learn and thus got a verification that our implmentations works correctly.

Most of our code is in the GitHub repository. We have chosen to not use the code in our report. We have, however, made a python file, project01.py, that provides most of our code in sequence, so that it is possible to follow the code that produce our testresults and plots in order.

4 Analysis of the models

4.1 Ordinary least square on the Franke function

First we have generated some test data, with the noise being very little, which we have plotted to get a more intuitive feel for how it looks. We have done this for all the models.
Ordinary Least square of the Frankefunction(the same as Ridge with $\lambda = 0$):



As you can tell from our code, we haven't made an own function for ordinary least square, since it is the same as Ridge, just with the λ set to 0. This will also show later that when the λ gets low it is very similar to OLS.
Here is the values for calculating the five first degrees and their MSE and R2-score:
OLS Test Data We can tell that our model is fitting our test data better and

k	MSE	R2
1	0.020836568820240136	0.7159598591230791
2	0.015631136506458913	0.7869193218104034
3	0.007175355204257789	0.9021869233537347
4	0.003960217232650187	0.9460150723293508
5	0.0018602479352178107	0.9746414541595732

better with a higher degree. Now we have to check with bootstrap as well to see if our model fits the data good.
Before the resampling we have also calculated the betas of $k = 5$:

Var of Beta, degree 5

$$\begin{array}{rrrr}
[9.47238055e-03 & 8.01217226e-01 & 4.12183842e-01 & 9.89887798e+00 \\
8.39455751e+00 & 4.14211294e+00 & 2.71771303e+01 & 3.28888083e+01 \\
1.78285914e+01 & 1.31093669e+01 & 1.91368860e+01 & 2.67004422e+01 \\
1.91391193e+01 & 9.34737951e+00 & 1.13215581e+01 & 2.55450522e+00 \\
4.40599813e+00 & 3.60735967e+00 & 1.80107762e+00 & 1.00979408e+00 \\
 & & & 1.50021893e+00]
\end{array}$$

Also the 95-percentage confidence interval of the betas: 95-percentage CI of betas, degree 5

$$\begin{array}{rr}
[[7.25873511e-02 & 4.54098870e-01] \\
[7.23335269e+00 & 1.07421092e+01] \\
[3.44378188e+00 & 5.96043621e+00] \\
[-4.41211552e+01 & -3.17880887e+01] \\
[-2.42828028e+01 & -1.29254533e+01] \\
[-1.57022805e+01 & -7.72437196e+00] \\
[4.22571363e+01 & 6.26923830e+01] \\
[4.18269196e+01 & 6.43072224e+01] \\
[1.67482166e+01 & 3.32996879e+01] \\
[-1.02120404e+01 & 3.98078784e+00] \\
[-3.30453851e+01 & -1.58973753e+01] \\
[-7.33667324e+01 & -5.31114961e+01] \\
[-1.94596236e+01 & -2.31061330e+00] \\
[-3.98768180e+01 & -2.78922323e+01] \\
[1.90552365e+01 & 3.22448231e+01] \\
[-2.43921072e+00 & 3.82593943e+00] \\
[1.88142647e+01 & 2.70423776e+01] \\
[8.68290825e+00 & 1.61280472e+01] \\
[-7.84195698e+00 & -2.58124771e+00] \\
[1.66275865e+01 & 2.05666637e+01] \\
[-1.80168037e+01 & -1.32155417e+01]]
\end{array}$$

The code and commenting for the calculations is to be found in python-file project01.py

4.1.1 Resampling

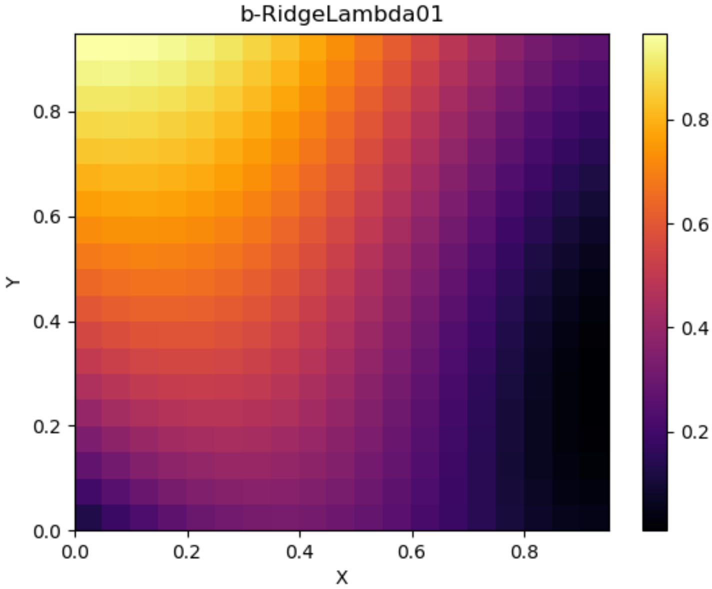
Using our bootstrapping algorithm with a resampling of 100, degree of five, we get these values:
VAR: 0.000052
BIAS: 0.001933

Bootstrap mean of MSE: 0.0020
Bootstrap mean of r2Score: 0.9757

The bootsrap values aligns pretty well with our original ones.

4.2 Ridge regression

Ridge Regression with $\lambda = 0.1$
Graphic plot of how it looks:



Ridge Test Data

k	MSE	R2
1	0.025965982389446095	0.6906471643146342
2	0.018247163430545398	0.7826074259086047
3	0.010258352759015437	0.8777843076427536
4	0.009382588378732645	0.8882179662029949
5	0.009143926633340667	0.8910613282063765

Compared to OLS, we can tell that Ridge does significantly worse then OLS.
Var of Beta, degree 5

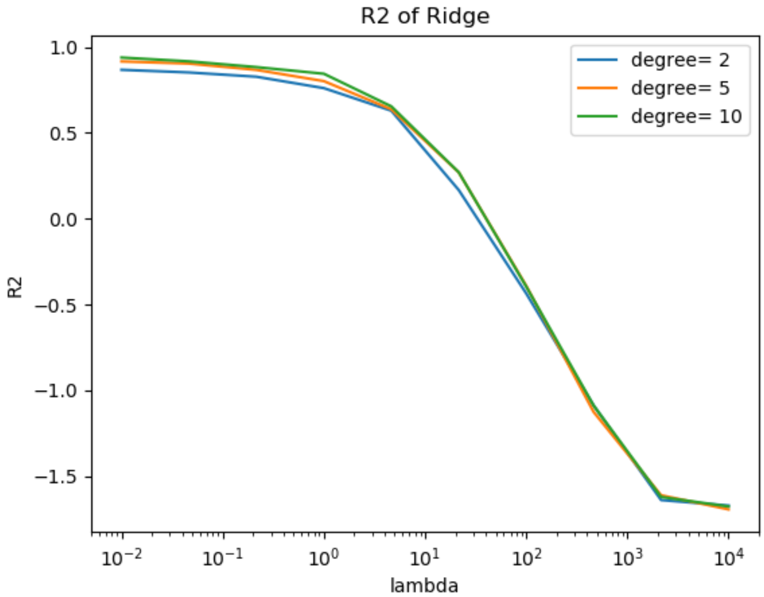
[0.00077004 0.01729839 0.01384995 0.06816008 0.06826285 0.05696662
0.0417508 0.02034709 0.06671475 0.03500783 0.02294549 0.01073849
0.03470761 0.01261125 0.02127092 0.03349824 0.01062464 0.03548007
0.03422875 0.04204926 0.02520149]

Also the 95-percentage confidence interval of the betas: 95-percentage CI of betas, degree 5

$$
\begin{array}{rr}
[[8.39518095e-01 & 9.45603016e-01] \\
[6.00189901e-01 & 1.10406786e+00] \\
[9.22637923e-01 & 1.52633094e+00] \\
[-6.51421032e+00 & -5.30387095e+00] \\
[5.84279933e-01 & 1.67511842e+00] \\
[-5.82384066e+00 & -4.85326698e+00] \\
[3.42317950e+00 & 4.16050815e+00] \\
[1.31502300e+00 & 2.10193829e+00] \\
[-2.00335018e+00 & -1.07460992e+00] \\
[1.38884378e+00 & 1.92131292e+00] \\
[3.21253289e+00 & 3.98989176e+00] \\
[5.79118852e-01 & 1.17557901e+00] \\
[-3.41043372e-02 & 7.77207310e-01] \\
[-8.02285903e-01 & -2.23570539e-01] \\
[3.31692208e+00 & 3.80174824e+00] \\
[-3.69457810e+00 & -2.99905374e+00] \\
[-2.10226995e+00 & -1.54360569e+00] \\
[-5.64130918e-03 & 8.94344017e-01] \\
[-1.07674065e+00 & -1.87377295e-01] \\
[3.14047856e-01 & 8.79022520e-01] \\
[-1.87908328e+00 & -1.35609332e+00]]
\end{array}
$$

4.2.1 Resampling

We can take a look at how different lambdas and different degrees of the polynomial makes a change in the R2-score and the MSE.
Here is a plot to show how they develop as a function of lambda.



We can tell pretty easily that the degree of the predictions doesn't matter much compared to how much the choice of lambda do. We can still tell that a lower degree function does worse then the other functions.

Some interesting values from bootstrap:

Bootstrap-values from degree of 5, lmb = 0.1 and 100 bootstrap-samples
VAR: 0.000067
BIAS: 0.008640
Bootstrap mean of MSE: 0.0087
Bootstrap mean of r2Score: 0.8980

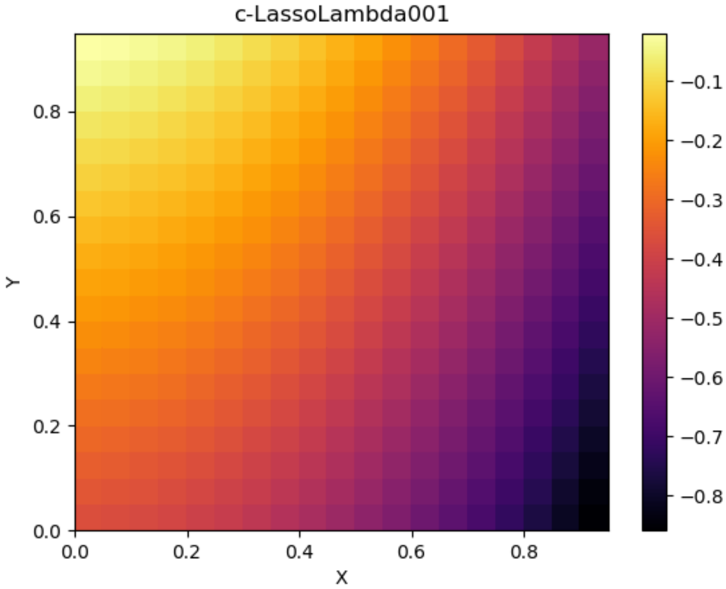
Bootstrap-values from degree of 5, lmb = 1 and 100 bootstrap-samples
VAR: 0.000059
BIAS: 0.011915
Bootstrap mean of MSE: 0.0120
Bootstrap mean of r2Score: 0.8597

Bootstrap-values from degree of 5, lmb = 10 and 100 bootstrap-samples
VAR: 0.000066
BIAS: 0.020274
Bootstrap mean of MSE: 0.0203
Bootstrap mean of r2Score: 0.7617

Bootstrap-values from degree of 2, lmb = 10 and 100 bootstrap-samples
VAR: 0.000057
BIAS: 0.022754
Bootstrap mean of MSE: 0.0228
Bootstrap mean of r2Score: 0.7327

4.3 Part c)

Lasso Regression with $\lambda = 0.01$



Ridge Test Data

k	MSE	R2
1	0.25272407945476655	-2.01089746779934852
2	0.25272407945476655	-2.0108974677993485
3	0.25272407945476655	-2.0108974677993485
4	0.25272407945476655	-2.0108974677993485
5	0.25272407945476655	-2.0108974677993485

Var of Beta

[0. 0.00025829 0.00394575 0. 0. 0.00324571
0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0.
0.]

95-percentage CI of betas

$$
\begin{aligned}
&[[0. \quad 0.]\\
&[-0.40937802 \quad -0.34637908]\\
&[-0.17825992 \quad 0.06797132]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0.58986796 \quad -0.36654512]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]\\
&[-0. \quad 0.]]
\end{aligned}
$$

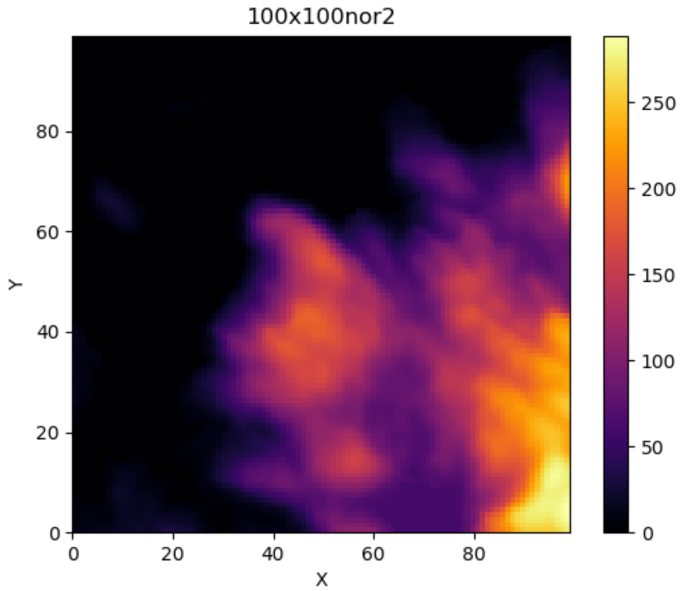
There is obviusly something happening with Lasso that doesn't work. Our Lasso method works great with the real data.

4.3.1 Resampling

Bootstrap-values from degree of 1, lmb = 0.1 and 100 bootstrap-samples
VAR: 0.000000
BIAS: 0.239704
Bootstrap mean of MSE: 0.2397
Bootstrap mean of r2Score: -2.0278
This was the same for all degrees and values of lambda, so I only included this.

4.4 Part d)

Imports 100x100 chunk of real data from top left corner of dataset nr.1.
Plot of real data

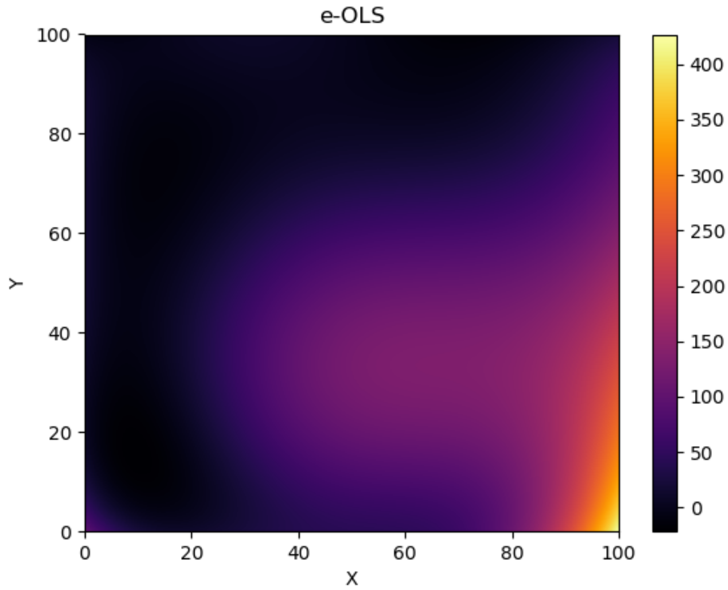


4.5 Real Data

Repeat of the previous method and data, but with real data.

4.6 OLS

Plot of the data with the OLS-method of degree 5:



OLS Score of the Real Data:

k	MSE	R2
1	1912.190996	0.587579
2	1177.397306	0.746059
3	1023.042624	0.779351
4	824.587589	0.822153
5	791.268452	0.829340

Var of Beta

$$[2.32350868e-17 \quad 3.59021583e-14 \quad 9.26320612e-14 \quad 6.67623760e-12$$
$$2.76179532e-12 \quad 2.44594128e-11 \quad 1.00899287e-09 \quad 2.82477031e-08$$
$$3.26102575e-08 \quad 4.03014136e-09 \quad 7.93068781e-13 \quad 8.29815731e-12$$
$$2.72417498e-12 \quad 8.99036913e-12 \quad 2.47909085e-12 \quad 4.07929803e-17$$
$$2.67723887e-16 \quad 5.18767547e-16 \quad 4.23509770e-16 \quad 1.84571052e-16$$
$$9.42227510e-17]$$

4.6.1 Resampling

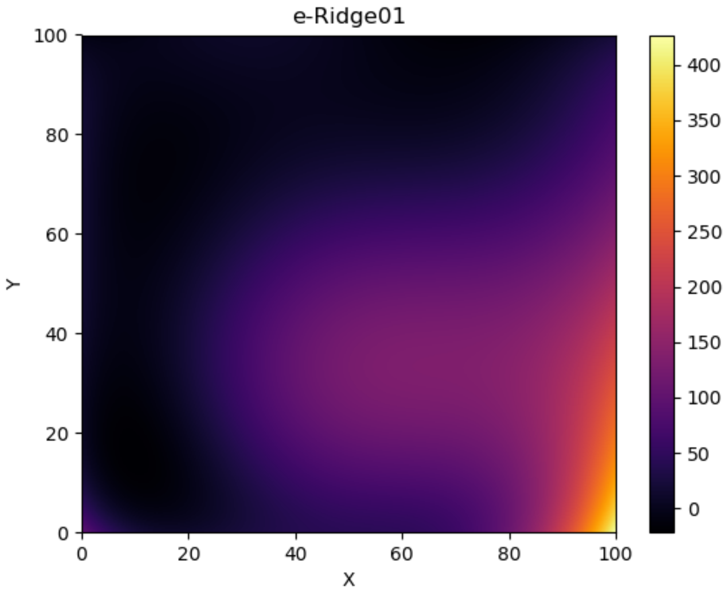
Bootstrap-values from degree of 5 and 100 bootstrap-samples
VAR: 1.287057

BIAS: 791.273865
Bootstrap mean of MSE: 792.5609
Bootstrap mean of r2Score: 0.8291

The MSE and R2-score are really close to our values for MSE and R2.

4.7 Ridge

Plot of the data with the Ridge-method of degree 5, lambda = 0.1:



Ridge Score of the Real Data with lambda = 0.1:

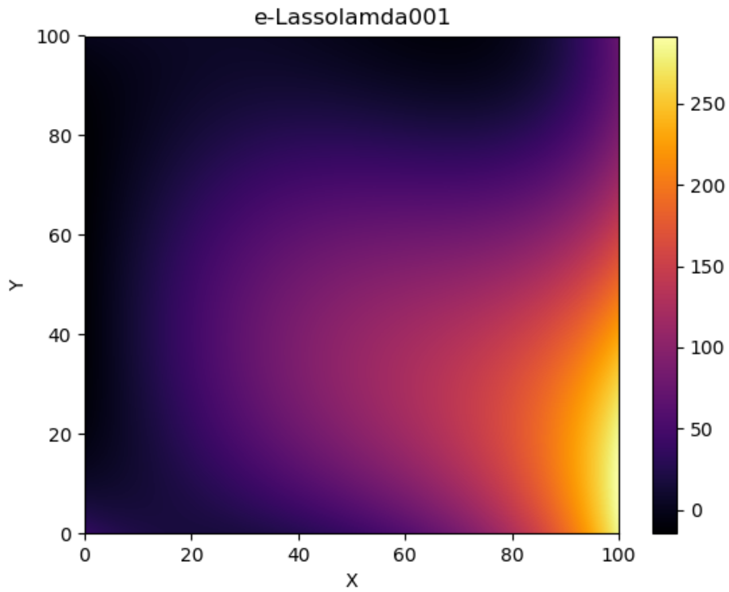
k	MSE	R2
1	1912.190999	0.587579
2	1177.397307	0.746059
3	1023.042624	0.779351
4	824.635824	0.822143
5	791.268452	0.829340

4.7.1 Resampling

Bootstrap-values from degree of 5, lmb = 10 and 100 bootstrap-samples
VAR: 1.297002
BIAS: 791.285081
Bootstrap mean of MSE: 792.5821
Bootstrap mean of r2Score: 0.8291
Can tell that MSE and R2-score is pretty similar to our computations.

4.8 Lasso

Plot of the data with the Lasso-method of degree 5, lambda = 0.01:



Lasso Score of the Real Data with lambda= 0.1:

k	MSE	R2
1	6601.647721	-0.423841
2	1564.545187	0.662560
3	1189.660192	0.743415
4	1155.338990	0.750817
5	974.391256	0.789844

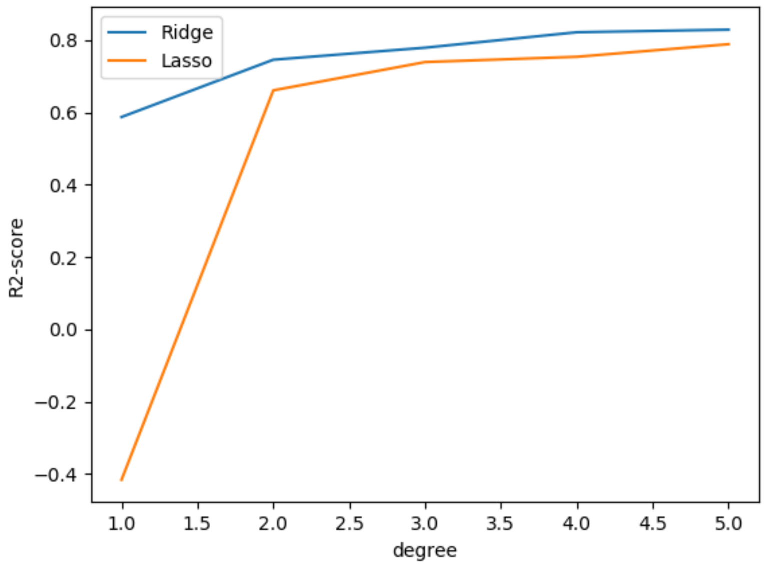
4.8.1 Resampling

Bootstrap-values from degree of 5, lmb = 10 and 100 bootstrap-samples
VAR: 1.689048
BIAS: 1239.400793
Bootstrap mean of MSE: 1241.0898
Bootstrap mean of r2Score: 0.7323

Here the MSE and R2score is a pretty long way from our scores when we use all our data. So the model is probably a little overfitted and we should expect our R2 to actually be lower than it is.

5 Conclusion

In conclusion, from what we can see, the Lasso model fitted our data the worst. Our scores from OLS and Ridge did pretty evenly and had very good scores. The bootstrap also validated our scores. To illustrate we can look at the R2-score between Ridge and Lasso with lambda = 0.1. These were both scores from our bootstrap.



We can tell that both Lasso and Ridge does pretty even, but Ridge beats Lasso consistantly. This is to be expected though, but the benefit of using Lasso is usually the lower variance. We can however see that the variance and the bias is both significantly higher then Ridge, so there is really no benefit of using Lasso in this particular case.