UiO : Department of Physics
University of Oslo

Regression analysis and resampling methods

Erik Skaar
Sondre Torp
Mikael Kiste

Contents

1 Introduction **2**

2 Theory **2**

3 Method **3**
3.1 Implementation of OLS, Ridge and Lasso 3
3.2 Implementation of MSE, R2 and β variance 3

4 Implementation **4**
4.1 Scikit vs. manually implementation 4
4.2 Time evolution 5
4.3 Noise - MSE & R2 evolution 5

5 Result & Discussion **6**

6 Conclusion **7**

7 Appendix **7**

Abstract

In this report the following regression methods were implemented; OLS, ridge and Lasso. To understand how well our methods worked, we tested it against Scikit's solutions, checked time dependance for different order of fitting and checked how noise affected the resulting polynomial. Then our implementation, OLS, Ridge and Lasso, were used on the Franke function. MSE,R2 score and VAR was calculated for all of the methods and how k-fold cross validation affected to resulting polynomial is shown. After this had been done on the Franke function, we repeated the proccedure for terrain data in Norway. The resulting polynomial were a good approximation for general features for the data, but fell short to describe the details. [1]

1 Introduction

2 Theory

3 Method

3.1 Implementation of OLS, Ridge and Lasso

The OLS method was implemented as shown in equation **??** and the Ridge method as shown in the equation **??**. The $\hat{x}$ matrix for both the methods, OLS and Ridge, is made as shown in the equation below:

$$\hat{x} = [x^0y^0, ..., x^0y^n, ..., x^1y^0, x^1y^{n-1}, ..., x^ny^0]$$

Scikit makes the $\hat{x}$ in a different way, but it does not matter for the result. As shown in the next section, Implementation.

Lasso is a bit harder to implement, so we used scikit's version for this. In order to understand which coefficient that corrosponded to which polynomial degree, we compared scikits coefficient with ours from the OLS implementation. This becomes important in the Result section.

3.2 Implementation of MSE, R2 and β variance

The R2 score and MSE was implemented as shown in equation eq: R squared and eq: mse. All the methods just simple send their predictions to a function that knows the solution. For the β variance it is a different story; The implementation for variance was used with the k-fold algorithm. For each part, the expected value of β and β^2 were added to a sum, that was then divided by $k - 1$. This was used in the equation below to obtain variance for the beta's:

$$\text{VAR} = \text{E}(\beta^2) - \text{E}(\beta)^2$$

4 Implementation

The three different algorithms discussed in section XXX was implemented in our script. It is a few different versions, but the ëversion contains all you need. All the scripts discussed in this report can be found at our github.

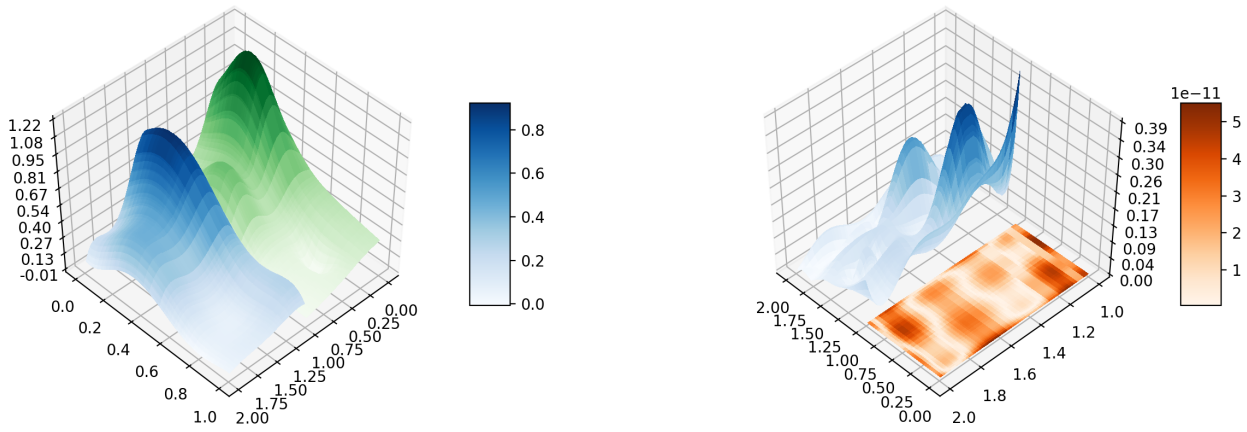
This part of the report is meant as a part where we show that our program works as we expected. We will look at how our implementation compare to Scikit's, how time increase as we increase the order we fit and how the modul crumbles as we add noise to our data.

The program was tested on the Frank-function, see equation 1. With an known solution we did a k-fold test and an degree and λ/α test. Both tested was done with the script descriped earlier. The tables below shows the different results.

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

4.1 Scikit vs. manually implementation

Our implementation was verified with Scikit's OLS solution. The figur below shows the result from the comparison.



(a) (b)

Figure 1: a)Shows the Franke function, equation 1, as it is. The OLS implementation was used on the Franke function with noise. The result is shown as the blue graph. b)The blue graph is the error compared to the actual Franke function, not the dataed that OLS were trained on. The orange graph is the absolute error for our OLS compared to Scikit's.

4.2 Time evolution

Table 1: This tables shows how time develops as a function of order fitted. One should not be suprised that Scikit has a faster algorithm, than us. The $_{relative}$ means that it is as fraction of the second order fit time. For lasso and ridge the λ/α was set to 1e-5. It is expected for higher order fits to have longer calculation time, because the matrix $\mathbf{X}$ will get bigger. This is exactly what we see with our program.

degree ↓	method →	OLS	SCIKIT	RIDGE	SCIKIT LASSO
2		0.01517s	0.25830s	0.00516s	0.00543s
$2_{relative}$		1.00	1.00	1.00	1.00
$3_{relative}$		2.42	1.58	2.45	2.38
$4_{relative}$		3.63	2.45	5.11	4.88
$5_{relative}$		4.98	3.61	8.77	8.31

4.3 Noise - MSE & R2 evolution

Table 2: This tables shows how the MSE evoles for different amount of noise. As described in section XXX, the Z data is normalized, so the MSE is 0.00127, when the highest values are 1. Which means that we have at least an error of 0.1% for the implementation on Franke function without noise. The $_{relative}$ means that it is as fraction of the zero noise data's MSE. For lasso and ridge the λ/α was set to 1e-5 and it was a fifth order fitting for all of the methods. The MSE grows as the noise increase to 50% of the data, which is expected!

Noise level ↓	method →	OLS	SCIKIT	RIDGE	SCIKIT LASSO
0		0.00127	0.00127	0.00514	0.00127
$0_{relative}$		1.00	1.00	1.00	1.00
$0.01_{relative}$		1.03	1.03	1.00	1.03
$0.2_{relative}$		12.84	12.84	3.68	12.84
$0.5_{relative}$		42.04	42.04	10.84	42.04

Table 3: This tables shows how the R2 score evoles for different amount of noise. For lasso and ridge the λ/α was set to 1e-5 and it was a fifth order fitting for all of the methods. Ideally we would want an R2 score of 1 for zero noise. The R2 score shrinks as the noise increase to 50% of the data, which is expected!

Noise level ↓	method →	OLS	SCIKIT	RIDGE	SCIKIT LASSO
0		0.98	0.98	0.91	0.98
0.01		0.98	0.98	0.91	0.98
0.2		0.68	0.68	0.62	0.68
0.5		0.28	0.28	0.25	0.28

5 Result & Discussion

Table 4: LASSO

$x^i y^j$	Coef	VAR	Confidens interval
x^0y^0	0.342129	0.000080	[0.333197,0.351061]
x^0y^1	-0.437389	0.000347	[-0.456013,-0.418765]
x^0y^2	-0.016612	0.000261	[-0.032765,-0.000458]
x^0y^3	0.000000	0.000000	[0.000000,0.000000]
x^0y^4	0.000000	0.000000	[0.000000,0.000000]
x^0y^5	0.000024	0.000000	[-0.000140,0.000188]
x^1y^0	0.558845	0.000299	[0.541543,0.576146]
x^1y^1	-0.389778	0.000394	[-0.409623,-0.369933]
x^1y^2	0.000000	0.000000	[0.000000,0.000000]
x^1y^3	0.000000	0.000000	[0.000000,0.000000]
x^1y^4	0.000000	0.000000	[0.000000,0.000000]
x^2y^0	0.000000	0.000000	[0.000000,0.000000]
x^2y^1	0.000000	0.000000	[0.000000,0.000000]
x^2y^2	0.000000	0.000000	[0.000000,0.000000]
x^2y^3	0.000000	0.000000	[0.000000,0.000000]
x^3y^0	-0.000178	0.000001	[-0.000933,0.000576]
x^3y^1	0.000000	0.000000	[0.000000,0.000000]
x^3y^2	0.000000	0.000000	[0.000000,0.000000]
x^4y^0	-0.120878	0.000436	[-0.141758,-0.099998]
x^4y^1	0.000000	0.000000	[0.000000,0.000000]
x^5y^0	0.000000	0.000000	[0.000000,0.000000]

Table 5: LASSO

$x^i y^j$	Coef	VAR	Confidens interval
x^0y^0	0.342129	0.000080	[0.333197,0.351061]
x^0y^1	-0.437389	0.000347	[-0.456013,-0.418765]
x^0y^2	-0.016612	0.000261	[-0.032765,-0.000458]
x^0y^5	0.000024	0.000000	[-0.000140,0.000188]
x^1y^0	0.558845	0.000299	[0.541543,0.576146]
x^1y^1	-0.389778	0.000394	[-0.409623,-0.369933]
x^3y^0	-0.000178	0.000001	[-0.000933,0.000576]
x^4y^0	-0.120878	0.000436	[-0.141758,-0.099998]

Table 6: RIDGE

$x^i y^j$	Coef	VAR	Confidens interval
x^0y^0	-0.014747	0.000532	[-0.037820,0.008326]
x^0y^1	1.700777	0.262232	[1.188691,2.212863]
x^0y^2	-1.423363	7.920556	[-4.237711,1.390986]
x^0y^3	-7.494369	38.216859	[-13.676348,-1.312390]
x^0y^4	11.580568	35.505840	[5.621890,17.539246]
x^0y^5	-4.247212	4.498345	[-6.368142,-2.126282]
x^1y^0	-0.095820	0.012602	[-0.208080,0.016439]
x^1y^1	-2.779005	0.819186	[-3.684094,-1.873916]
x^1y^2	6.873662	2.015972	[5.453812,8.293511]
x^1y^3	-1.592223	0.802557	[-2.488079,-0.696368]
x^1y^4	-2.951493	0.510070	[-3.665684,-2.237301]
x^2y^0	11.119198	0.363834	[10.516011,11.722384]
x^2y^1	-14.485972	2.483606	[-16.061918,-12.910026]
x^2y^2	0.999862	5.328256	[-1.308439,3.308164]
x^2y^3	3.275747	1.446726	[2.072948,4.478546]
x^3y^0	-22.518257	3.845345	[-24.479212,-20.557302]
x^3y^1	24.970159	1.048842	[23.946030,25.994289]
x^3y^2	-5.075608	0.906148	[-6.027526,-4.123690]
x^4y^0	15.006599	5.479219	[12.665826,17.347372]
x^4y^1	-10.208141	0.083641	[-10.497349,-9.918933]
x^5y^0	-2.719594	0.888911	[-3.662415,-1.776773]

6 Conclusion

References

[1] Morten Hjorth-Jensen. *Computational Physics*. Lecture notes. 2015. URL: `https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf`.

7 Appendix