Regression Analysis of a Bivariate Function

Aron Jansson Nordberg

(Dated: December 29, 2019)

We have studied the application of the three machine learning methods Ordinary Least Squares, Ridge and LASSO on the Franke function. We have approximated them using polynomials, and tuned the hyperparameters to find the best fit. We have also employed resampling techniques when determining the hyperparameters. Ordinary Least Squares yielded the best results, with polynomials of degree 5.

I. INTRODUCTION

All the code and material for this report is to be found in the github repository:

In machine learning, a major topic of interest is supervised learning. We are given a data set which consists of inputs/predictors X and outputs/responses Y. The goal of supervised learning is to determine a function/model that maps the input to the output, that is finding an f such that $Y = f(X)$. When the output is a continuous function, the we call it a regression problem.

In this project, we shall be concerned with fitting models to bivariate functions, $z = f(x, y)$, also called terrain functions. We will consider the Franke function, and test three regression methods: Ordinary Least Squares, Ridge, and LASSO. We will also employ the resampling methods train-test split and K-fold cross validation. We will test the results of each method by computing the Mean Squared Errors and the R^2 scores.

The model that we will study, and use to approximate the bivariate functions, are linear combinations of polynomials in x and y.

II. THEORY

A. Regression Methods

1. *Ordinary Least Squares*

Consider a data set, where the N responses are collected in the N vector $\boldsymbol{z}$. We will assume that each entry can be expressed as a linear combination of polynomials in x and y. Let $\hat{\boldsymbol{z}}$ be our approximation, and let $\boldsymbol{\beta}$ be a p vector, which contains the weights to each of the p polynomials for each of the N entries. The design matrix X is an $N \times p$ matrix, where each row corresponds to an entry in $\boldsymbol{z}$ and contains each of the p polynomials as columns. We can now write out approximation as the matrix equation:

$$\hat{\boldsymbol{z}} = X\boldsymbol{\beta} \tag{1}$$

We will determine quality of our approximation by computing the Mean Squared Error (MSE), given by

$$MSE = \frac{1}{N}\left(\boldsymbol{z} - \hat{\boldsymbol{z}}\right)^T\left(\boldsymbol{z} - \hat{\boldsymbol{z}}\right) \tag{2}$$

This is our cost function. We want to minimize 2, so we take its derivative with respect to the weights $\boldsymbol{\beta}$, using 1.

$$\begin{aligned} MSE &= \frac{1}{N}\left(\boldsymbol{z} - X\boldsymbol{\beta}\right)^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right) \\ \frac{\partial}{\partial\boldsymbol{\beta}}MSE &= \frac{1}{N}\left(\mathbf{0} - X\right)^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right) + \left(\boldsymbol{z} - X\boldsymbol{\beta}\right)^T\left(\mathbf{0} - X\right) \\ &= \frac{1}{N}\left[-X^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right) - \left(\boldsymbol{z} - X\boldsymbol{\beta}\right)^T X\right] \\ &= \frac{1}{N}\left[-X^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right) - X^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right)\right] \\ &= \frac{1}{N}\left[-2X^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right)\right] \end{aligned}$$

We now set this equal to $\mathbf{0}$ and solve for $\boldsymbol{\beta}$.

$$\begin{aligned} \frac{\partial}{\partial\boldsymbol{\beta}}MSE &= \mathbf{0} \\ \frac{1}{N}\left[-2X^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right)\right] &= \mathbf{0} \\ X^T\left(\boldsymbol{z} - X\boldsymbol{\beta}\right) &= \mathbf{0} \\ X^T X\boldsymbol{\beta} &= X^T\boldsymbol{z} \end{aligned}$$

Assuming that $X^T X$ is non-singular, meaning it is invertible, we have an explicit solution for the values of $\boldsymbol{\beta}$ that minimizes 2.

$$\boldsymbol{\beta} = \left(X^T X\right)^{-1} X^T\boldsymbol{z} \tag{3}$$

2. *Ridge*

In cases where $X^T X$ is near-singular, to where it might cause computational problems, we can add a penalty term to its diagonal. This will increase the bias of our model.

Our cost function now looks like this

$$C = \frac{1}{N}\left(\boldsymbol{z} - \hat{\boldsymbol{z}}\right)^T\left(\boldsymbol{z} - \hat{\boldsymbol{z}}\right) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \tag{4}$$

where λ is a so-called hyperparameter. Just like with OLS, our goal now is to minimize 4 by taking its derivative, setting it equal to $\mathbf{0}$ and solving for $\boldsymbol{\beta}$. Here we state the result

$$\boldsymbol{\beta} = \left(X^T X + \lambda I_p\right)^{-1} X^T\boldsymbol{z} \tag{5}$$

where I_p is the $p \times p$ identity matrix. We see that it is quite similar to 3, but we now have an additional parameter λ that must be determined. This is usually done with a grid search, where a range of values for λ are used to generate models, which are tested using MSE and/or R^2. The adding of terms to the model is called regularization.

3. *LASSO*

The method Least Absolute Shrinkage and Selection Operator (LASSO) is similar to Ridge, but the cost function is now

$$C = \frac{1}{N} \left(\boldsymbol{z} - \hat{\boldsymbol{z}}\right)^T \left(\boldsymbol{z} - \hat{\boldsymbol{z}}\right) + \lambda \sqrt{\boldsymbol{\beta}^T \boldsymbol{\beta}} \tag{6}$$

This time, our previous strategy will no longer work, and we cannot achieve an analytical expression for $\boldsymbol{\beta}$. Instead we must perform the computations in a grid search, and determine the $\boldsymbol{\beta}$ that minimizes 6.

B. Resampling Methods

In order to avoid overfitting, that is that our model is fit only to predict the outcomes of the training data used to make it, and not general enough to be applied to new data sets, where the outcome may not be known in advance, we must perform resampling techniques for quality checks.

1. *Train-Test Split*

In this method, the training data is split into data used for training the model (X_{train} and $\boldsymbol{z}_{train}$), and data used for testing (X_{test} and $\boldsymbol{z}_{test}$). The given regression method then uses the training data to estimate a $\boldsymbol{\beta}$, which is then used to estimate out approximation

$$\hat{\boldsymbol{z}} = X_{test}\boldsymbol{\beta}$$

Which is then evaluated against $\boldsymbol{z}_{test}$ using, for example, MSE.

2. *K-Fold Cross Validation*

This method is quite similar to train-test split, but instead of splitting the data set just once, we iterate through all possible divisions (called folds). For example, given $K = 10$, we would split the data set into 10 equal folds, use 9 of them to train our model and test on the remaining 1. The results would be evaluated with MSE. Then another one is used as test data while the rest is trained, and so it continues until all combinations have been done. The average MSE is then computes, and is a measure for how good the model did, and it makes sure it is not prone to overfitting.

Figure 1 illustrates the process.

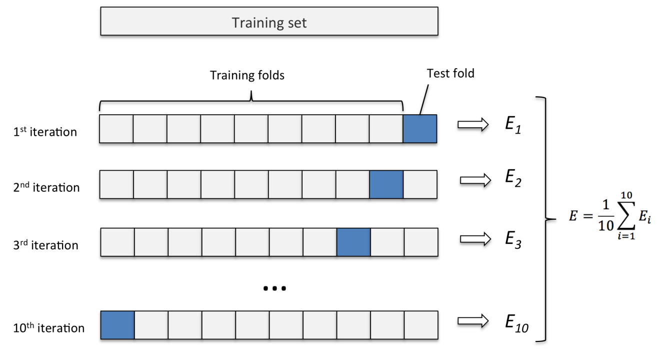

Figure 1. Illustration of K-fold cross validation. Source: https://miro.medium.com/max/6542/1*me-aJdjnt3ivwAurYkB7PA.png

C. Model Evaluation

We will in this project evaluate the results of the methods using two methods: MSE and R^2.

$$MSE = \frac{1}{N}\sum_{i=0}^{N-1}\left(y_i - \hat{y}_i\right)^2 \tag{7}$$

$$R^2 = 1 - \frac{\sum_{i=0}^{N-1}\left(y_i - \hat{y}_i\right)^2}{\sum_{i=0}^{N-1}\left(y_i - \langle y\rangle\right)^2} \tag{8}$$

D. Bias-Variance Tradeoff

We will in this section explain how one can decompose the 7 into terms which are the variance and bias, and how they relate to one another. The following identity is very useful in that regard, and it is a decomposition of the MSE:

$$MSE = bias^2 + var \tag{9}$$

What follows is a proof for 9.

$$\begin{aligned} E\left[(z-\hat{z})^2\right] &= E\left[(z^2 - 2z\hat{z} + \hat{z})^2\right] \\ &= E\left[z^2\right] - 2E\left[z\hat{z}\right] + E\left[\hat{z}^2\right] \end{aligned}$$

We will now consider each term in turn.

$$\begin{aligned} E\left[z^2\right] &= E\left[(f+\epsilon)^2\right] \\ &= E\left[f^2 + 2f\epsilon + \epsilon^2\right] \\ &= E\left[f^2\right] + 2E\left[f\epsilon\right] + E\left[\epsilon^2\right] \end{aligned}$$

Now, f is deterministic so $E\left[f^2\right] = f^2$, also per definition $E\left[\epsilon^2\right] = \sigma^2$, and $E\left[f\epsilon\right] = 0$. Sp

$$E\left[z^2\right] = f^2 + \sigma^2$$

The 9 shows that by introducing bias, one might decrease the variance of the model by quite a bit. This is what happens in the Ridge and LASSO methods, where the penalty term adds bias, decreases variance.

III. METHOD

The entire project was done using the programming language `Python`.

A. Franke Function

The first data set was generated using the Franke function, which is a function widely used in applications of approximation schemes. We let $x, y \in [0, 1]$. In order for it to be a more realistic data set taken from real-life measurement, we added stochastic noise, specifically random numbers drawn from a Gaussian distribution $0.1 \times N(0, 1)$. It is shown in figure 2.

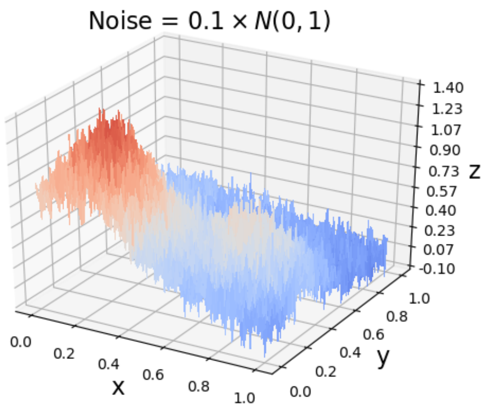

Figure 2. The Franke function with added noise. The noise are random numbers drawn from a Gaussian distribution $N(0, 1)$.

1. *Regression: OLS*

We studied OLS by varying the maximum degree of the polynomials used in the model. Degrees 0 to 5 were used, and their results evaluated using 7 and 8. First without resampling, and then with both train-test split and k-fold cross validation.

Both the resampling methods were done using methods from `ScikitLearn`, specifically `train_test_split()` and `KFold`. The reaos for this was that they proved difficult to implement from scratch.

The confidence interval was computed using the following formula from (citaiton)

$$\beta_j \pm z^{1-\alpha} v_j^{1/2} \sigma \tag{10}$$

where $z^{1-\alpha}$ is the $1-\alpha$ percentile of the normal/Gaussian distribution, thus we used $z^{1-0.025} = 1.96$. v_j is the jth diagonal of $(X^t X)^{-1}$ and σ is the standard deviation of the values themselves.

2. *Regression: Ridge*

We did the same procedure for Ridge, but we only used k-fold cross validation as resampling. Also, since Ridge contains a hyperparameter λ, we performed a grid search, and we produce a heat map with MSE as a function of polynomial degree (model complexity) and λ. We let the degree vary between 0 and 5, and $\lambda \in [10^{-4}, 10]$.

3. *Regression: LASSO*

We used `ScikitLearn` for this method, specifically the `Lasso` method. Once again we did a grid search for the hypermarameter, and let $\lambda \in [10^{-2}, 10^{-1}]$. Note that the hyperparameter is called α in the program.

IV. RESULTS

Figures 6 to 11 shows the results of OLS without the use of any resampling techniques. Table I shows the β vector with the associated confidence intervals for each entry. Figure 3 shows the MSE as a function of model complexity, that is the degree of the polynomial.

Figure 4 shows the result of the Ridge method. The heat map shows MSE as a function of λ and polynomial degree.

Figure 5 shows the result of the Ridge method. The heat map shows MSE as a function of λ and polynomial degree.

Table II displays the lowest MSE score for the three methods with k-fold cross validation, along with the best value for their respective hyperparameters.

V. CONCLUSION

It is clear from table II that OLS is the method that best fits the kind of data we look at in this project. It has the lowest MSE score, although not by much. Ridge also performs nearly as well, but it should be pointed out that the hyperparameter is very small, in fact the

Table I. The $\boldsymbol{\beta}$ that minimizes the cost function, using OLS. No resampling.

β entry	value	95% confidence
0	0.435072	$\pm$ 0.00378377
1	7.48767	$\pm$ 0.0423837
2	3.52444	$\pm$ 0.0423837
3	-33.1635	$\pm$ 0.204656
4	-13.862	$\pm$ 0.159838
5	-7.96259	$\pm$ 0.204656
6	46.9101	$\pm$ 0.459742
7	41.6446	$\pm$ 0.340723
8	19.0085	$\pm$ 0.340723
9	-9.04219	$\pm$ 0.459742
10	-23.2623	$\pm$ 0.480122
11	-50.6661	$\pm$ 0.366568
12	-5.51051	$\pm$ 0.341091
13	-28.5962	$\pm$ 0.366568
14	29.9243	$\pm$ 0.480122
15	1.64575	$\pm$ 0.188273
16	18.2115	$\pm$ 0.163674
17	9.41273	$\pm$ 0.159634
18	-5.75018	$\pm$ 0.159634
19	16.364	$\pm$ 0.163674
20	-16.7033	$\pm$ 0.188273

Table II. MSE scores for OLS, Ridge and LASSO methods.

Method	MSE	Hypermarameter
OLS	0.0120786	-
Ridge	0.0120865	$\lambda \approx 0.0001$
LASSO	0.2556345	$\lambda \approx 0.0774264$

smallest tested value, suggesting that the penalty term makes the model less accurate.

REFERENCES

- https://www.uio.no/studier/emner/matnat/astro/AST2000/h19/undervisningsmateriell_h2019/ forelesningsnotater/part1b.pdf

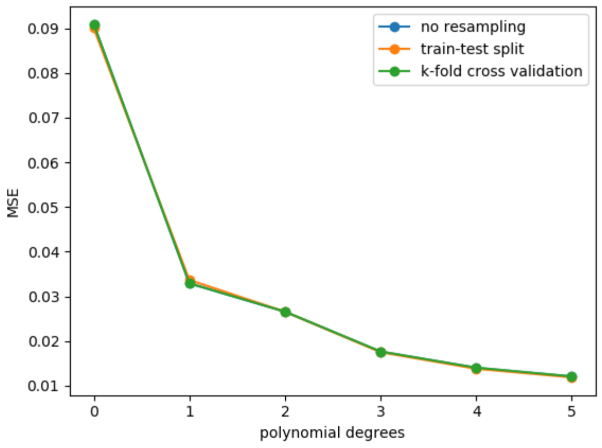

Figure 3. MSE of the OLS method, without resampling, with train-test split and with k-fold cross validation.

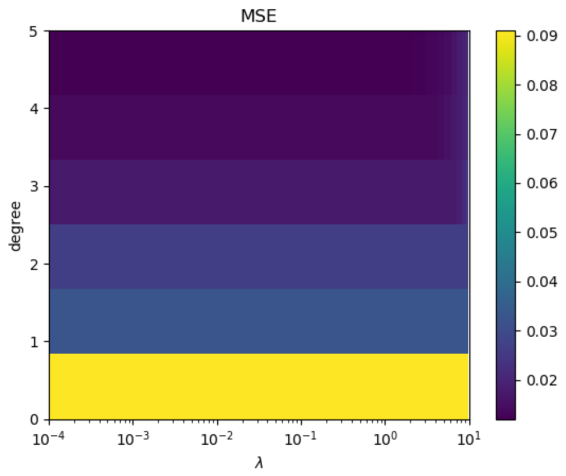

Figure 4. MSE of the Ridge method, with k-fold cross validation.

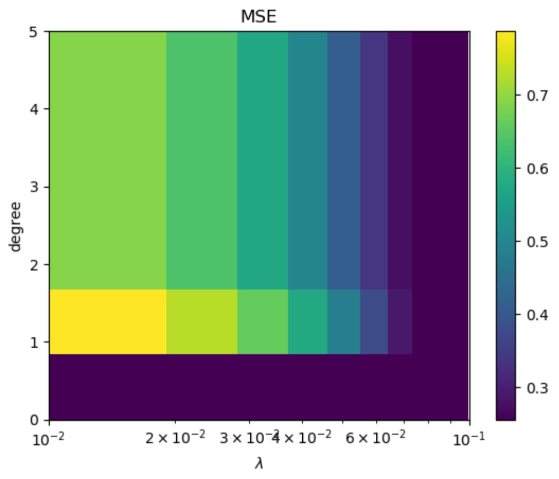

Figure 5. MSE of the LASSO method, with k-fold cross validation.

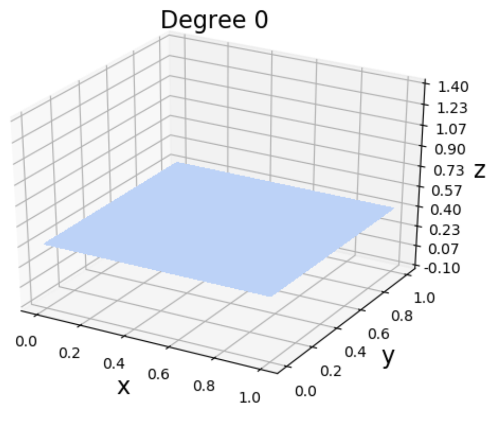

Figure 6. Franke function approximated by polonimials of degree 0.

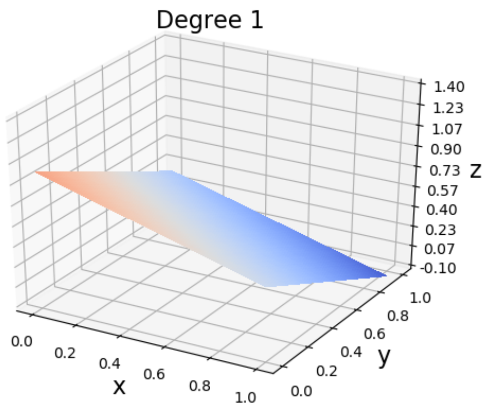

Figure 7. Franke function approximated by polonimials of degree 1.

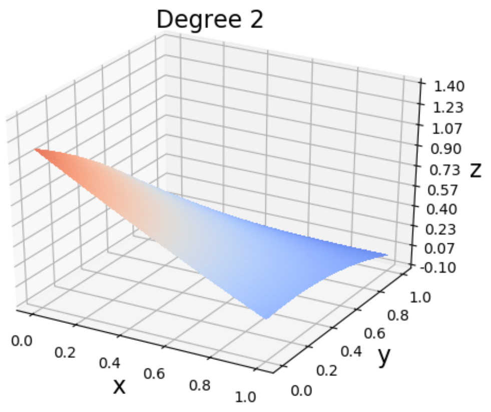

Figure 8. Franke function approximated by polonimials of degree 2.

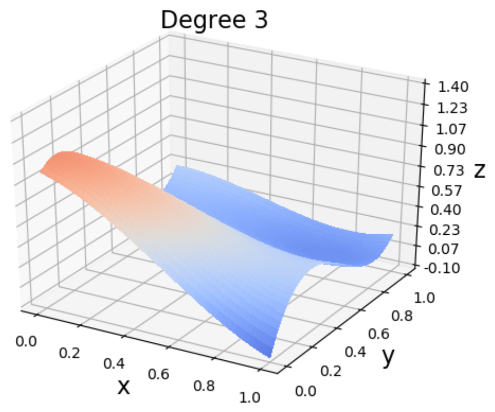

Figure 9. Franke function approximated by polonimials of degree 3.

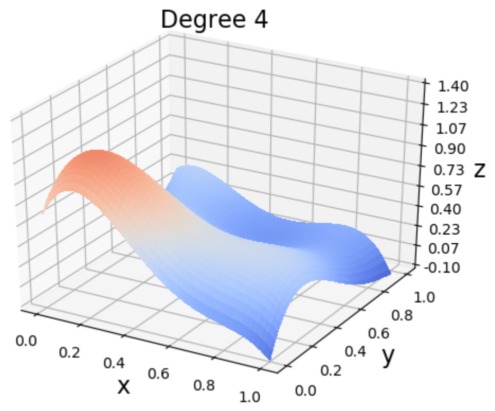

Figure 10. Franke function approximated by polonimials of degree 4.

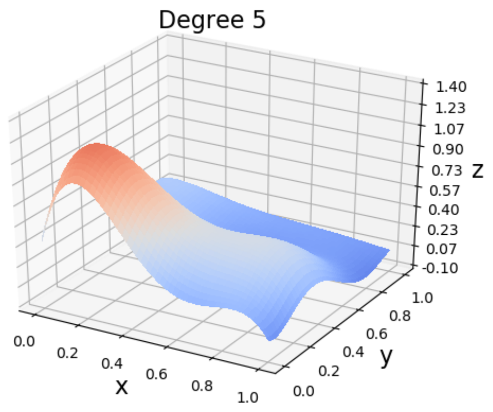

Figure 11. Franke function approximated by polonimials of degree 5.