# Regression Analysis of the Franke Function

Eirik Nordgård
Geophysical Institute, University of Oslo

2019
September

**Abstract**

Ordinary Least Squares, Ridge regression and Lasso regression have been used on data generated by the Franke Function. MSE and R2 score have been evaluated for each model. Ordinary Least Squares seems to be the most accurate regression method on this dataset. Ridge also performs quite well, while the lasso is very dependent on the penalty parameter to provide good results.

All material for this project may be found on `https://github.com/eirikngard/Project1`

## 1 Introduction

The main aim of this project is to study various linear regression methods in detail. Ordinary Least Squares (OLS), Ridge regression and Lasso regression are the methods which will be investigated. Models using linear regression assumes linear inputs X1, X2,...,Xp. Even though there are many other, no-linear models you might think outperforms the linear models, they actually have a couple of advantages compared to the non-linear models. They are simple to use and provide interpretable descriptions of how the inputs affect the outputs. In situations with small numbers of training cases or if the signal-to-noise ratio is small, linear models can outperform non-linear models in prediction. Also, applying linear methods to transformations of the inputs expands their scope tremendously[1].

In this project I have used data generated by the Franke function. This is a smooth, terrain-like function with two Gaussian peaks and a small dip often used in interpolation problems.

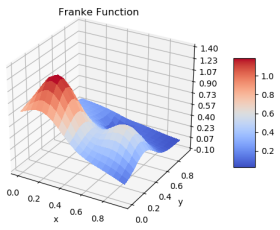


Figure 1: 3D plot of the Franke Function

# 2 Theory

## 2.1 Linear Regression Models

The goal is to make a model that predicts a real-valued output $y$ with an input vector $x$. We can make a linear model $y(x_i)$ on the form

$$y = \tilde{y}_i + \epsilon_i \tag{1}$$

where $\tilde{y}$ is a function of some variable and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is the normally distributed error with mean, $\mu = 0$ and variance $= \sigma^2$. The function $y$ can then be written as

$$y = \beta_0 + \sum_{i=1}^{n} X_j \beta_j + \epsilon_i \tag{2}$$

where $\beta$ are unknown coefficients and $X$ is the so called Vandermonde design-matrix with data input vector $X^T = (X_1, X_2, ..., X_p)$.

Eq. (2) can be rewritten as

$$y = X\beta + \epsilon \tag{3}$$

and thus the approximation can be written as

$$\tilde{y} = X\beta \tag{4}$$

.

To obtain the optimal set of $\beta_i$ a cost function is defined. This function gives a measure of the spread between the exact values $y_i$ and the parametrized values $\tilde{y}_i$,

$$C(\beta) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 \tag{5}$$

or

$$C(\beta) = \frac{1}{n} \{(y - X^T\beta)^T (y - X^T\beta)\} \tag{6}$$

in terms of design matrix $X$ and coefficients $\beta$.

To obtain the the optimal set of $\beta$ it is necessary to minimize $C(\beta)$ for all $\beta$. [2]

### 2.1.1 Ordinary Least Squares (OLS)

Solving the minimization problem of (6) require

$$\frac{dC(\beta)}{d\beta_j} = \frac{d}{d\beta} [\frac{1}{n} \sum_{i=0}^{n-1} (y_i - \beta_0 x_{i,0} - \beta_1 x_{i,1} - ... - \beta_{n-1} x_{i,n-1})^2] = 0 \tag{7}$$

resulting in

$$\frac{dC(\beta)}{d\beta} = X^T(y - X\beta) = 0 \tag{8}$$

.

Rewriting this as

$$X^T y = X^T X\beta \tag{9}$$

gives the solution

$$\hat{\beta}^{OLS} = (X^T X)^{-1} X^T y \tag{10}$$

if the matrix $X^T X$ is invertible.[2]

Here the hat-rematk indikates this is a predicted value for the true $\beta$. The tilde remark, $\tilde{*}$, has the same meaning.

To estimate the confidence interval of the $\beta$ obtained by the OLS we need to calculate the variance, $\sigma^2$, of $\hat{\beta}$. Here we must assume uncorrelated observations of $y_i$ and that the variance is constant. Additionally the $x_i$ must be non-random. Derived from equation of $\hat{\beta}$ above we then have[1]

$$Var(\hat{\beta}) = (X^T X)^{-1} \sigma^2 \tag{11}$$

where,

$$\sigma^2 = \frac{1}{N - p - 1} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{12}$$

$Var(\hat{\beta})$ is called a covariance matrix of the $\beta$ parameter. The denominator in Eq. (12) makes it an unbiased estimator of $\sigma^2$.

The 95% confidence interval for the $\hat{\beta}$ is

$$\beta \pm STD * z^{1-\alpha} \tag{13}$$

Here $z^{1-\alpha}$ is the 1-$\alpha$ percentile of a normal distribution. $STD$ is the standard deviation of $\beta$, or the diagonal elements of $(X^TX)^{-1}$. For the 95% confidence interval we have $\alpha = 0,025$ and $z^{1-0,025} = 1,96$[1].

A visualization of how far off the ols method is, is displayed in Fig.2.

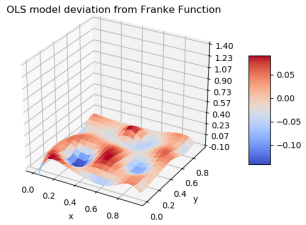


Figure 2: OLS deviation from the Franke Function

### 2.1.2 RIDGE Regression

Ridge regression is a shrinkage method very similar to OLS. The goal of ridge regression is to shrink the regression coefficients. This is done by imposing a penalty on their size, minimizing the penalized residual sum of squares. The penalty, $\lambda$, is often called a hyper-parameter. Large value of $\lambda$ results in large amount of shrinkage. The ridge shrinking problem can be written as

$$\hat{\beta}^{RIDGE} = \underset{\beta}{argmin}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2\} \tag{14}$$

with solution

$$\beta^{RIDGE} = (X^TX + \lambda I)^{-1}X^Ty \tag{15}$$

Here $I$ is the $pxp$ identity matrix.

A visualization of how far off the ridge method is, is displayed in Fig.3.

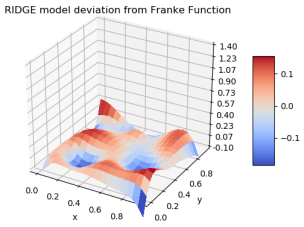


Figure 3: RIDGE deviation from the Franke Function

### 2.1.3 LASSO Regression

Like ridge regression, the lasso is a shrinkage method. The lasso shrinkage problem in its Lagrangian form can be written

$$\hat{\beta}^{LASSO} = \underset{\beta}{argmin}\{\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p} \mid \beta_j \mid\} \tag{16}$$

which is different from the ridge problem only in the penalty $\sum \mid \beta_j \mid$. Since this term makes the solution for $\beta$ nonlinear in $y_i$, there are no definite expression for the solution as in the ridge regression. The solution is obtained by solving a programming problem using the functionalities in the scikit-learn library. Similarly to the ridge method, the penalty parameter $\lambda$ must be chosen to minimize the expected prediction error for the problem. The difference from ridge is that in stead of doing a proportional shrinkage, each $\beta$ is shrunk by a constant penalty parameter $\lambda$[1]

A visualization of how far off the lasso method is, is displayed in Fig.4.

### 2.1.4 Cross-Validation

When creating a model you want to estimate the prediction error (performance) of the model. To test the prediction performance we spit the data into two sets, a training set and

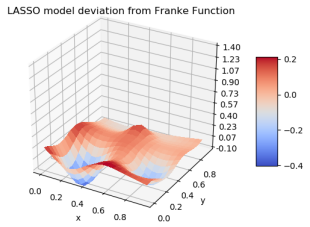


Figure 4: LASSO deviation from the Franke Function

a test set. The training set plays the role of the original data which the model it self is built upon, while the test data is novel data which is used to evaluate the prediction performance of the model. This splitting procedure can be done for a collection of penalty parameter choices, $\lambda$. Performing the splitting many times allows us to pick the $\lambda$ that yields the model with the best prediction performance on average. Additionally we would like to set aside a validation set to evaluate the predictive performance of our model. This set is picked to allow the model to test on data it has not seen before.[2] $K-foldcrossvalidation$ is one method that handles this problem, since data sets are normally too small to allow us to set aside a validation set. Another problem arising when doing this repetitive splitting is that some data samples may end up in a vast majority of the splits in the training or test sets. This may influence the prediction evaluation. K-fold is one method that handles this by structuring the data splitting.[1][2]

#### 2.1.5 *k*-fold Cross-Validation

The data are divided into $k$ more or less equally sized, exclusive subsets. For each split, one of the subsets acts as the test set while all the remaining subsets acts as the training set. This is repeated $k-1$ times for many splits. Such splitting results in a balanced representation of the data points in both training and test set for all the splits. Still the division into the $k$ subsets involves some degree of randomness, which may be fully excluded when choosing $k$=$n$. This particular case is referred to as leave-one-out cross-validation.

#### 2.1.6 Bootstrap

The bootstrap is used for assessing statistical accuracy. The aim of the bootstrap is thus to estimate the expected prediction error $Err$. After splitting the data into a training and a test set, the basic idea is to randomly draw datasets with replacement from the training data with same size as the original training set. This is repeated $B$ times (for instance 100 or 1000), giving $B$ datasets. For each of these bootstraps we refit the model and evaluate the fits over the replications.[1]

#### 2.1.7 Bias-Variance Tradeoff

The cost function

$$C(X,\beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y})^2 = E[(y-\tilde{y})^2] \quad (17)$$

can be rewritten as

$$E[(y-\tilde{y})^2] = \frac{1}{n}\sum_i (f_i - E[\tilde{y}])^2 + \frac{1}{n}\sum_t (\tilde{y}_i - E[\tilde{y}])^2 + \sigma^2 \quad (18)$$

When creating a predictive model it is important to consider the prediction errors such as variance and bias of the model. What we call the bias-variance tradeoff is problem of minimizing both the bias and the variance at the same time. The bias is the difference the average prediction of the model and the true value which the model is trying to predict. High bias corresponds to a oversimplification of the model, thus underfitting the data. This can often lead to high errors on training and test data.

$$Bias = \frac{1}{n}\sum_i (y_i - \tilde{\hat{y}})^2 \tag{19}$$

The variance is the variability of the model prediction for a given data point which indicated the spread of the data. High variance is often a result of overfitting, which means that the model captures too much of the noise along with the underlying datapattern. Thus, models with high variance does not generalize on the data it has not yet seen.

The objective is to find function $\hat{f}(x)$, denoted $\hat{f}$, that in the best possible way approximates some true function $f(x)$, denoted $f$. *The best way possible* is made by minimizing the MSE for $y$ and $\hat{f}(x)$ for all $x$.[3] For any function $\hat{f}$ we can decompose the expected error like this:

$$E[(y-\hat{f}(x))^2] = Bias[\hat{f}(x)])^2 + Var[\hat{f}(x)] + \sigma^2 \tag{20}$$

where

$$Bias[\hat{f}(x)] = E[\hat{f}(x)] - E[f(x)] \tag{21}$$

and

$$Var[\hat{f}(x)] = E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 \tag{22}$$

Since $\epsilon$ and $\hat{f}$ are independent,

$$E[(y - \hat{f})^2] = E[(f + \epsilon - \hat{f})^2] \tag{23}$$

we have

$$= Bias[\hat{f}]^2 + \sigma^2 + Var[\hat{f}] \tag{24}$$

See full derivation[3] in the Appendix.
We can rewrite Eq. (24) to

$$\begin{aligned} E_{rr}(x) = Bias^2 + Variance \\ + Irreducibleerror \end{aligned} \tag{25}$$

The bias term in Eq. (25) is the squared difference between the true mean and the expected value of the estimate. As the complexity $k$ of a model increases, the bias will most likely also increase. The variance term will decrease as the inverse of $k$. The last term is the variance of the irreducible error. This is simply a measure of noise in our the data which is always present and therefore unavoidable.

The bias-variance tradeoff is therefore about balancing the bias and the variance to a point where they both are acceptable, where the test error is at a minimum. Too high model complexity results in high variance, whereas too low complexity results in too high bias and vice versa.[1]

The more complex our model $\hat{f}(x)$ is, the more data points it will capture, and the lower the bias will be. However, complexity will make the model "move" more to capture the data points, and hence its variance will be larger[3].

# 3 Method

Defining the Franke function is done using the code provided in in the Project1 PDF (`https://github.com/CompPhysics/MachineLearning/tree/master/doc/Projects/2019/Project1`).

First off I have defined the basic functions MSE, R2 score, bias and variance defined in the regression slide[2]. These functions have input $(data, model)$ and returns the current value. The confidence interval of the $\beta$ is also calculated in this manner.

For the regression, the design matrix $X$ is needed. This is calculated using x and y flattened creating rows $[1, x, y, x^2, xy, xy^2, etc.]$ depending on the argument n, polynomial degree I want to fit. In the regression itself, this matrix is used to calculate the $\beta$ values, using matrix-inversions. With z beeing the data provided by the Franke Function, this line may look like this:

$$Beta_{ols} =$$

$$np.linalg.inv(X.T.dot(X)).dot(X.T).dot(z)$$

Here, *np* is used to access the *NumPy* library functionality. More information on the *NumPy* functionalities can be found on `https://numpy.org`. Thereafter, the prediction is calculated using the $\beta$s and the desig matrix $X$. This procedure is repeated for ridge regression as well, but due to an inversion problem the *scikit-learn* functionality is used to perform the lasso regression. More information on *scikit-learn* can be found on `https://scikit-learn.org/stable/`. For comparison on the ols and ridge I halve also calculated the MSE using *scikit-learn* as well.

For a simple resample I used *train_test_split*, giving an overview of the model performance. Typically this gives results like this:

$$\textit{The model performance for training set}$$

$$---------------------$$

$$MSE(scikitols)\ is \quad 0.00225729$$

$$MSE(ols)\ is \quad 0.00225729$$

$$MSE(ridge)\ is \quad 0.03072305$$

$$MSE(lasso)\ is \quad 0.02490839$$

$$R2\ (ols)\ is \quad 0.97134498$$

and

$$\textit{The model performance for testing set}$$

$$---------------------$$

$$MSE(scikit\ ols)is \quad 0.00207010$$

$$MSE(ols)\ is \quad 0.00130538$$

$$MSE(ridge)\ is \quad 0.01264897$$

$$MSE(lasso)\ is \quad 0.02770861$$

$$R2\ score\ is \quad 0.97791144$$

Tuning the penalty parameter to get the best MSE is done using two foor-loops. One creating the design matrix looping over the polynomial degree from 1 to 5. Inside this I loop over a logarithmic $\lambda$-scale, calculating MSE and R2 score using ridge regression.

# 4 Results

To find the optimal $\lambda$ value for the ridge regression it was necessary to manually tune parameter to find the lowest MSE possible. For Ridge regression, the MSE calculated by my own regression code seamed to explode at $\lambda$ around $10^{-1}$ in Fig.5 For comparison, the MSE calculated by scikit-learn is also showing a slight increase towards higher values of $\lambda$, but does not spike like my code. The overall optimal parameter is around $10^{-5.3}$, providing a MSE of 0.00220566. Additionally, two local lows exist around $10^{-2.7}$ and $10^{-0.9}$, also providing very low MSE (a number). Equivalently, Fig.6 shows R2 score for the ridge regression is very good (very close to 1) at these $\lambda$ as well. For larger values of $\lambda$ the MSE spikes through the roof, possibly indicating a numerical instability at certain points. To better display the optimal MSE for my model I would ideally create a heatmap, but due to this problem giving very large MSE this was not possible.

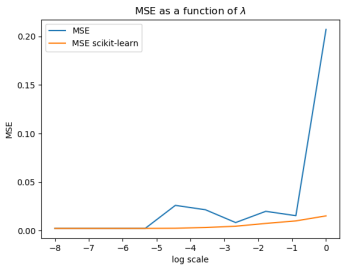


Figure 5: MSE plotted as function of $\lambda$. The x-axis is logarithmic, where $-1$ corresponds to $10^{-1}$

From the regression results it is clear that the OLS provides the lowest MSE of the three methods on the Franke function without noise. This result also hold for the R2 score. The difference in bias for the three methods are marginal. This result is expected (why) since the Franke function is a smooth function. Adding noise rapidly increases the MSE for all

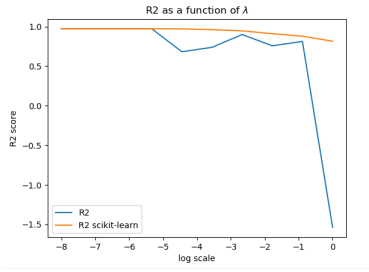


Figure 6: R2 score plotted as function of $\lambda$

three methods. Also the R2 becomes closer and closer to zero.

In Fig.7 the ridge regression performs slightly worse than the OLS with $\lambda$ around order $10^{-4}$. This also holds for higher values of $\lambda$. The smaller the penalty parameter, the closer the lasso comes to the ridge and OLS. As displayed in Fig. 7, lasso i performing fairly well compared to ridge and OLS, but once the penalty parameter increases it performs very bad, see Fig.8.The lasso drives the coefficients to zero for small values of $\lambda$.

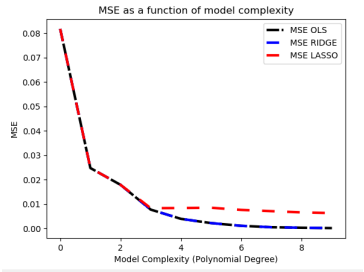


Figure 7: MSE as function of model complexity. $\lambda = 0.0001$

Using the *train_test_split* function provided by scikit-learn the model performance was tested for both the training set and the test set of the data.

The confidence intervals of the $\beta$ was calculated for each of the regression methods, resulting in the plots in Fig. 9 to Fig.13.

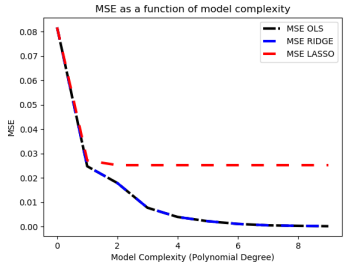


Figure 8: MSE as function of model complexity. $\lambda = 0.01$

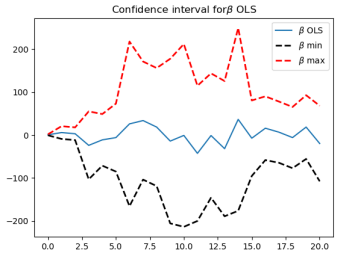


Figure 9: Confidence interval for $\beta_{OLS}$

# 5 Conclusion

Ordinary Least Squares seems to be the most accurate regression method on the data provided by the Franke Function. The MSE computed by the ols i very low, and you are not in need of tuning a penalty parameter as in ridge or lasso. Ridge also performs quite well for given penalty parameters $\lambda$, while the lasso is very dependent on the penalty parameter to provide good results. For small $\lambda$ the lasso performs quite good, although the performance weakens rapidly for increasing $\lambda$.
As stated in the project description, this analysis should also have been done on a real terrain data set. The result of doing so would most likely not have been as clear as for the Franke data.

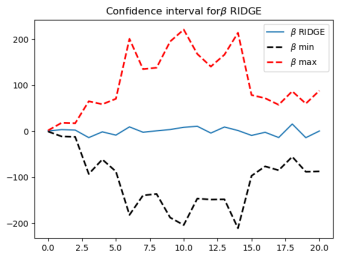


Figure 10: Confidence interval for $\beta_{RIDGE}$, $\lambda = 0.01$

.

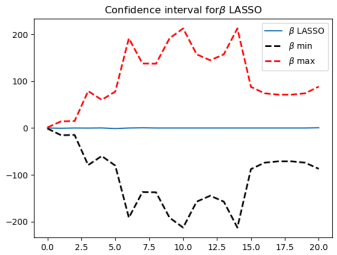


Figure 11: Confidence interval for $\beta_{LASSO}$, $\lambda = 0.001$

.

# 6 Appendix

Derivation of the bias-variance tradeoff [3].

$$E[(y-\hat{f})^2] = E[(f+\epsilon-\hat{f})^2]$$
$$= E[(f+\epsilon-\hat{f}+E[\hat{f}]-E[\hat{f}])^2]$$
$$= E[(f-E[\hat{f}])^2] + E[\epsilon^2] + E[E[\hat{f}-\hat{f})^2] + 2E[(f-E[\hat{f}])\epsilon] + 2E[\epsilon(E[\hat{f}]-\hat{f})] + 2E[(E[\hat{f}]-\hat{f})(f-E[\hat{f}])]$$
$$= (f-E[\hat{f}])^2 + E[\epsilon^2] + E[(E[\hat{f}]-\hat{f})^2] + 2(f-E[\hat{f}])E[\epsilon] + 2E[\epsilon]E[E[\hat{f}]-\hat{f}] + 2E[E[\hat{f}]-\hat{f}](f-E[\hat{f}])$$
$$= (f-E[\hat{f}])^2 + E[\epsilon^2] + E[E[\hat{f}-\hat{f})^2]$$
$$= (f-E[\hat{f}])^2 + Var[y] + Var[\hat{f}]$$
$$= Bias[\hat{f}]^2 + Var[y] + Var[\hat{f}]$$
$$= Bias[\hat{f}]^2 + \sigma^2 + Var[\hat{f}][3]$$

Bootstrap and $k$-fold cross validation calculations using Mortens code.

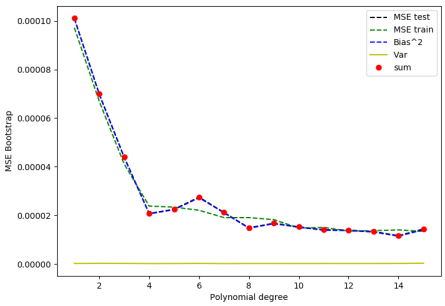


Figure 12: MSE for test and train set, bias and variance calculated using bootstrap.

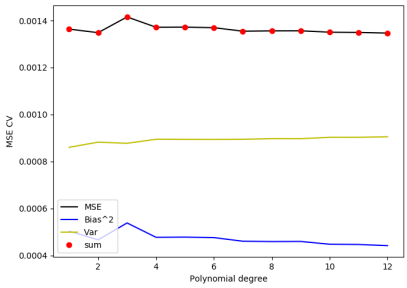


Figure 13: MSE for test and train set, bias and variance calculated using $k$-fold cross validation

# 7 References

Give always references to material you base your work on, either scientific articles/reports or books. Refer to articles as: name(s) of author(s), journal, volume (boldfaced), page and year in parenthesis. Refer to books as: name(s) of author(s), title of book, publisher, place and year, eventual page numbers

# References


[1] Hastie, Trevor. Tibshirani, Robert. Friedman, Jerome. *The Elements of Statistical Learning. Data Mining, Interference, and Prediction.* Second Edition. Springer, 2009. Chapter 3, Chapter 7

[2] M. Hjorth-Jensen Lecture Notes in FYS-STK4155. *Data Analysis and Machine Learning: Linear Regression and more Advanced Regression Analysis.* URL: https://compphysics.github.io/MachineLearning/doc/pub/Regression/html/Regression.html Unpublished, 2019.

[3] Wikipedia: Bias-Variance tradeoff URL: `https://en.wikipedia.org/wiki/Bias\OT1\textendashvariance_tradeoff` Read: 23.09.2019